# Learning State Reachability as a Graph in Translation Invariant Goal-based Reinforcement Learning Tasks

**Anonymous submission**

**Abstract**

Deep Reinforcement Learning proved efficient at learning universal control policies when the goal state is close enough to the starting state, or when the value function features few discontinuities. But reaching goals that require long action sequences in complex environments remains difficult. Drawing inspiration from the cognitive process which reuses learned atomic skills in a global planning procedure, we propose an algorithm which encodes reachability between abstract goals as a graph, and produces plans in this goal space. Transitions between goals rely on the exploitation of a learned policy which enjoys a property we call *translation invariant local optimality*, which encodes the intuition that goal-reaching skills can be reused throughout the state space. Overall, our contribution permits solving large and difficult navigation tasks, outperforming related methods from the literature.

## 1 Introduction

Model-free Reinforcement Learning (RL) has demonstrated an outstanding ability to learn complex optimal policies from raw interaction data, for well-defined atomic tasks with relatively short time and state space outreach, such as balancing a pendulum (Barto, Sutton, and Anderson 1983), learning to walk for a quadruped (Kimura, Yamashita, and Kobayashi 2002), or learning to balance a bicycle (Randløv and Alstrøm 1998). But when it comes to solving more structured, long-term tasks, such as navigating through a building or a maze, baking a cake, or assembling furniture, humans seem to rely more on learned models, which they use for planning, instead of performing trial-and-error learning.

Such a decomposition was inherent to seminal RL agents like the Dyna architecture (Sutton 1991), and was later one of the core intuitions behind hierarchical RL (Sutton, Precup, and Singh 1999). It is notable that often (although not always), atomic tasks enjoy a property which we call *translational invariance*. Balancing a bicycle, for instance, implies in practice an optimal policy that recommends the same sequences of actions regardless of the geographical position, mostly because gravity does not change too much across the globe and that we ride bicycles on surfaces that have close enough friction properties. Similarly, when navigating in a reasonably homogeneous environment, reaching position $B$ from position $A$, can be achieved by applying the same policy than reaching position $B + \Delta$ from position $A + \Delta$, provided there are no obstacles in the way. The optimal policies might somehow differ, but are close enough in many practical cases. In this paper, we consider such environments which enjoy this translational invariance property for local, atomic goal-reaching tasks. We exploit this property to efficiently learn an abstract model that is used by the agent to plan its course of action.

Our contribution is threefold.

- We propose a generic framework linking goal spaces and state spaces for goal-reaching policy optimization.
- We formalize the notion of re-usability of a goal-reaching policy throughout the state space as one of translation invariance.
- We propose a complete graph-based model learning method, which relies on planning in the goal space, and chains local application of translation invariant goal-reaching policies. By combining planning and RL, this method permits solving tasks over long horizons, a common pitfall for classical RL methods.

As such, the proposed algorithm belongs to the family of goal-based RL methods. Since it couples planning and RL, it also connects with hierarchical RL. Finally, it presents many similarities with the *Search on the Replay Buffer* (SoRB) algorithm (Eysenbach, Salakhutdinov, and Levine 2019) and subsequent works, with several key differences which can be seen as a generalization of SoRB and permit better applicability. The next section sets the necessary background and puts our contribution in perspective of the current literature. Section "Learning a reachability graph" introduces key ingredients, namely a formal definition of goals as state abstractions, a characterization of policy translation invariance, and finally the reachability graph learning (RGL) procedure. In the following section, we evaluate RGL empirically and discuss its properties. Finally, we draw some conclusions about the present approach.

## 2 Background and related work

**Goals in Reinforcement Learning (RL).** RL (Sutton and Barto 2018) considers the problem of learning an optimal decision making policy for an agent interacting over multiple time steps with a dynamic environment, modeled as a Markov Decision Process (Puterman 2014) of unknown transition and

reward models. At each time step, the agent and the environment are described through a state $s \in S$. When an action $a \in A$ is performed, the system then transitions to a new state $s'$, while receiving a reward $r(s, a)$. Stochastic Shortest Path problems are a particular class of MDPs which aim at reaching a terminal goal state as quickly as possible. Such problems can be encoded as MDPs featuring $-1$ rewards for all transitions but those to a terminal goal state. One can quantify the efficiency of a policy $\pi : S \to A$ in every state $s \in S$ via its value function $V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))$, with $\gamma \in [0, 1)$ a discount factor on future rewards (which can also be interpreted as a stepwise probability of non-termination).[1] Training an RL agent consists in finding a policy with the highest possible value function. A long-standing goal in RL is to design multi-purpose agents, able to achieve various goals through a single goal-conditioned policy $\pi(s, g)$ (Kaelbling 1993), where the goal $g$ is either a single state in $S$ or an abstraction for a set of states. The ability of deep neural networks to approximate complex functions has triggered a renewal of interest in learning universal value function and policy approximators (Schaul et al. 2015), $V(s, g)$ and $\pi(s, g)$ respectively. Among the many approaches developed to learn goal-based policies and value functions, Hindsight Experience Replay (Andrychowicz et al. 2017, HER) proposes a seminal method which defines goal-based reward functions by re-labelling states collected in past trajectories as goals.

**Hierarchical RL (HRL).** Combining local goal-reaching sequences of actions in order to achieve a more general goal is the core idea of HRL (Sutton, Precup, and Singh 1999; Precup 2000; Konidaris and Barto 2009). Notably, among recent works, Kulkarni et al. (2016) define a bi-level hierarchical policy, using a DQN (Mnih et al. 2015) agent to select high-level goals, that define options which make use of a low-level goal-based DQN agent. Nachum et al. (2018) specializes this idea to the case when the lower-level policy learns to achieve goals that encode relative changes to the current state. Levy et al. (2019) couples HER with a three-level hierarchy into an architecture called Hierarchical Actor-Critic (HAC). McClinton, Levy, and Konidaris (2021) enhance HAC with a separate higher-level goal generator which drives the exploration process during learning. Overall, these approaches all aim at designing a global neural-network-based controller, able to solve the tasks at hand.

**Planning and learning.** An alternative to crafting a hierarchy of learned policies is to rely on RL for producing "lower level" option policies, and on some model of how these options affect the environment. The aim is then to optimize a sequence of options, or skills, in a global plan. The key to such approaches hence relies on how the model is built. Silver et al. (2017) train a "predictron" which, for a given task, predicts $n$-step returns and long term values from any state, using a network that builds a consistent internal representation of the environment's dynamics and rewards. Similarly, several approaches (Ha and Schmidhuber 2018; Hafner et al. 2020, 2021) build models that emulate the dynamics and rewards related to a task, and permit planning by simulating this surrogate model, but without a hierarchy of options and for a single task. In contrast, Nasiriany et al. (2019) optimize a sequence of reachable intermediate goal states (represented in the latent space of a variational auto-encoder on states) in order to reach a final goal (single task), using a pre-computed reachability metric for a given goal-based lower-level policy. Parascandolo et al. (2020) optimize online a similar curriculum of sub-goals between a starting state and a given goal. They implement a divide-and-conquer approach by building an AND/OR search tree. Each node corresponds to a new subgoal in the sequence. They explore this tree with a Monte Carlo tree search strategy, which exploits the value function of a pre-trained goal-based policy. Some methods store explicitly these links between subgoals by constructing a reachability graph. In turn, this graph can be used for higher-level goal-based planning. Savinov, Dosovitskiy, and Koltun (2018) build this graph by randomly exploring the environment, and add a node for every encountered state, which yields a very dense graph. For a given goal, a shortest path in this graph is computed. Then a sequence of landmark subgoals is extracted so that each landmark is far enough from the previous one according to a pre-trained neural network. Eysenbach, Salakhutdinov, and Levine (2019) introduce Search on the Replay Buffer (SoRB), which supposes the availability of a replay buffer of states and defines a graph where each state in a random subset of the replay buffer is a node. Then it uses the goal-reaching policy's value function to estimate edge weights between these nodes and finds a shortest path of state waypoints to the goal. SGM (Emmons et al. 2020) improve SoRB's results by pruning useless nodes in the graph, and edges that cannot been traversed by the control policy. Pruning useless nodes enables a reduction in the number of graph edges and permits a faster convergence to a close-to-optimal graph (ie. representative of actual reachability with a minimal number of nodes and edges). Chaplot et al. (2020) learn a reachability graph in a robotics navigation environment. For each new location in his graph, the agent uses its camera to estimate promising exploration directions. Aubret, Hassas et al. (2021) and Ruan et al. (2022) incrementally grow a graph representing reachability, where nodes are abstractions of sets of states, using a neural network as a surrogate of the similarity between states.

**Originality of the present work.** With respect to this general body of work, our contribution has several key features. We formalize a context which alleviates the need to train the lower-level goal-conditioned policy on all states and goals. Similarly to SoRB, we exploit the policy's value function as a local reachability measure, while introducing a level of abstraction since we clearly distinguish between goals and states. As developed in the next sections, this provides a sparser, abstract planning graph, closer to a hierarchy of options. Also in contrast to SoRB and SGM, we do not rely on a pre-existing replay buffer and avoid defining nodes over an arbitrary subset of sampled states; instead we incrementally grow a reachability graph to cover the attainable goal space.

[1] SSPs are well-defined for $\gamma = 1$ but this is not the case for all MDPs so we keep this discount factor for the sake of genericity in further developments.

## 3 Learning a reachability graph

The rationale for the method we propose below goes as follows. Because they are *continuous* universal approximators, neural networks are intrinsically unsuited to approximate discontinuous functions such as the value functions arising in some difficult RL environments (e.g. mazes, non-holonomous robots, etc.). Because their optimization assumes that samples are obtained independently and identically from a *stationary* distribution, they are also unsuited to retain local information: either because the distribution (and hence the training set) is unbalanced or because of distributional shift which causes catastrophic forgetting. The sequential nature of RL decisions makes it crucial to make good decisions in infrequently-visited states, to retain local information even when facing distributional shift, and to approximate some functions that can easily be discontinuous. Neural networks are great at learning complex continuous functions such as navigation or movement primitives, or local goal-reaching policies. Elaborating on this statement, we turn to a hierarchy of approximators, coupling planning in a graph of goal space waypoints, with local goal-reaching skills learned with deep neural networks. When requiring to achieve a goal $g^* \in G$ from a state $s_0 \in S$, we link $g^*$ and $s_0$ to their closest graph nodes. Specifically, we find vertices $v^*$ and $v_0$ whose waypoints $g_{v^*}$ and $g_{v_0}$ minimize some measure of proximity $d(g^*, g_{v^*})$ and $d(P(s_0), g_{v_0})$ respectively, with $P(s_0)$ an abstraction of $s_0$ in the goal space. Then we find a shortest path between them in the graph, which defines an *execution curriculum* of waypoints, and the local policy is used to reach each waypoint's vicinity in sequence.

The core of our contribution lies hence in the graph expansion and pruning method, its ability to represent accurately an abstraction of the environment dynamics despite unbalanced samples and discontinuous properties, and finally its use to design goal-conditioned policies over large and complex state spaces. To present the method in a well-defined framework, we restrict the set of MDPs we consider to those enjoying a property we call *translation invariance of local optimal policies* which we discuss later on. We also discuss therein to what extent this assumption is a strong constraint. Then, given such a goal-reaching policy $\pi$, we grow and prune a graph $\mathcal{G}$ which encodes an abstract notion of reachability and distance over the state space. The pair $(\pi, \mathcal{G})$ can then be used jointly to encode a policy that benefits from the best of both worlds and allows one to exploit planning algorithms over $\mathcal{G}$ in order to define an *execution curriculum* of waypoints for $\pi$; resulting in a global agent that can reliably learn to reach distant goals in complex environments.

### Goals as state abstractions

In the general sense, a goal $g$ is an abstraction for a set of states. For instance, a goal for a robotic ant might be "reach this room, regardless of orientation, legs configuration, or precise final position". In this paper, for the sake of genericity, we assume that goals live in a goal space $G$, that both $S$ and $G$ are normed vector spaces, and that there exists a projection $P(s) = g$ which projects states into the (lower dimensional) goal space. Consequently, we can define $K_0 = \ker P$ as the set of states corresponding to the null goal $0_G \in G$. Let $\bar{P}$ be a mapping from goals to states such that $P \circ \bar{P}$ is the identity function on $G$. There are many possible such mappings if the dimension of $G$ is smaller than that of $S$. Conversely, when $\dim G = \dim S$, one can take $\bar{P} = P^{-1}$, although in this case it is practical to straightforwardly identify goals and states, which means $P$ and $\bar{P}$ are the identity function. When $S$ and $G$ differ, we assume such a $\bar{P}$ mapping is provided. Then, $K_g = \{\bar{P}(g) + \delta,\ \delta \in K_0\}$ is the set of states whose projection by $P$ is $g$. In what follows, we retain the $P$ and $\bar{P}$ notations for genericity, but the reader is encouraged to discard them as the identity function in order to catch the key intuitions. Finally, when the goal and state spaces differ, we introduce the strong assumption that for a given goal $g \in G$, any $s_2 \in K_g$ is reachable for a negligible cost from any other $s_1 \in K_g$. In plain words, moving between any two states which correspond to the same goal (same state abstraction) is supposed feasible and costless. Note that this is immediately verified when $S = G$.

### Translation invariance of local optimal policies

Intuition indicates that a four-legged robot should not have to learn to walk again when it is moved from a room of the lab to another. Of course, this assumption might be questionable if one moves the robot from a room to the hills outside the lab. We formalize this notion of re-usability of learned policies as one of translational invariance. We say an MDP admits translation invariant local optimal policies (TILO policies) if there exists a goal-conditioned optimal policy $\pi^*$ such that

$$\forall s, \delta \in S \times S, \exists \rho \in \mathbb{R}, \text{ such that}$$
$$\forall g \in \mathcal{B}(P(s), \rho), \pi^*(s, g) = \pi^*(s + \delta, g + P(\delta)), \quad (1)$$

where $\mathcal{B}(P(s), \rho)$ is a ball, centered in $P(s)$ and of radius $\rho$. In plain words, such a policy guarantees that whatever close enough starting states $s$ and $s'$ we consider, we can always find *local* goals for which the first action recommended by the policy will be the same. A corollary of this property is that in deterministic MDPs, all actions taken to reach $g$ from $s$ are the same as those necessary to reach $g + P(s' - s)$ from $s'$, for goals that are close enough to $P(s)$.

To set ideas and illustrate the notion of TILO policies, one can consider a continuous state space maze, or a problem of navigation in an environment with isotropic movement properties, but cluttered with obstacles. In this example, we assume $G = S$ and $P$ is the identity function. Then, given two states $s$ and $s'$, there exists a vicinity of $s$ and $s'$ where picking goals $g$ and $g + P(s' - s)$ will induce the same sequence of optimal distributions over actions. This vicinity is constrained by the presence of obstacles close to $s$ and $s'$ and might shrink to very small balls, but it exists nonetheless and this property captures the notion of reusability of goal-reaching policies across the state space. As a consequence, a TILO policy which has learned to reach goals around $s$ needs not be trained again in other regions of $S$, which marks a notable difference with the relative goal policies introduced by Nachum et al. (2018). In turn, TILO policies need only be trained to reach goals from a fixed starting state, and the TILO property enables their re-usability throughout the state space to reach local goals.

Arguably, MDPs that admit TILO policies do not represent the full span of MDPs. However we argue that with an appropriate choice of the metric on $G$, defining $\mathcal{B}(P(s), \rho)$, this property actually applies to many common control problems, including, in particular, navigation problems. Moreover, one can extend the reasoning to $\epsilon$-optimal policies, hence defining $\epsilon$-TILO policies. An MDP admits $\epsilon$-TILO policies if there exists a policy that is $\epsilon$-optimal and obeys (1). When one takes the four-legged robot that has only been trained on the lab's concrete grounds, to the grassy hills, it is reasonable to assume its translated goal-conditioned policy will not be optimal. However, this policy is still likely to perform better than most other policies and hence to be $\epsilon$-TILO.

The method we develop herein applies to MDPs which admit $\epsilon$-TILO policies that are pre-trained. Practically, given a starting state $s$, we directly train a translation invariant goal-conditioned policy $\pi(s, g)$. Training of this policy is done before directed exploration and graph learning takes place. We also define a goal-proximity quasi-metric $d^\pi(g, g') = (V_{max} - V^\pi(\bar{P}(g), g'))/(V_{max} - V_{min})$, indicating how close two goals are under policy $\pi$, with $V_{max}$ and $V_{min}$ chosen so that, on the training domain, $d^\pi(g, g') \in [0, 1]$ and $d^\pi(g, g) = 0$. This quasi-metric $d^\pi(g, g')$ can be interpreted as a measure of how long it takes for policy $\pi$ to reach $K_g$ from $K_{g'}$ on average. The goal-conditioned policy training method is any algorithm that trains a universal value function approximator; it trains $\pi(s, g)$ and $V^\pi(s, g)$ within a playground state space with no obstacles. Specifically, starting with a random exploration strategy, a replay buffer is filled, and the goal-conditioned policy is trained to reach goals sampled from it. This yields a $\pi(s, g)$ policy that is also an $\epsilon$-TILO policy, along with the corresponding $d^\pi(g, g')$ quasi-metric. We emphasize that this policy is not required to be able to reach any possible goal from $s$, even in the playground environment (Levy et al. (2019) and Nachum et al. (2018) have illustrated how RL algorithms struggle when the goals become too distant). Instead, its performance and goal outreach is as good as the training procedure can make it, and we rely on the graph learning procedure to encode the reachability between states, based on this policy.

## Learning a reachability topology

To ease the presentation of ideas, we present the proposed Reachability Graph Learning algorithm (RGL, Algorithm 1) in the context of deterministic MDPs, and defer the discussion of the stochastic case to the end of this section. Given a pre-trained $\epsilon$-TILO policy $\pi$, we wish to construct an oriented graph $\mathcal{G} = (V, E)$ which will represent the reachability between sub-goals, using $\pi$. Each vertex $v \in V$ of such a graph is associated with a given goal $g_v$, and directed edges $e \in E$ indicate reachability of the successor node's goal from the states corresponding to the source node's goal. In other words, if an edge exists between $v$ and $w$, then $\pi$ successfully reaches $g_w$ from states in $K_v = K_{g_v}$. The edge linking $v$ and $w$ is weighted with a traversal cost of $d^\pi(g_v, g_w)$. Knowledge of this weighted graph permits running a planning algorithm to find an execution curriculum of waypoint vertices (intermediate goals $g_v$) which eventually link any start state and final goal. This is very similar in spirit to SoRB (although our graph is defined on goals, not states). The (other) key difference lies in the fact that graph nodes are not built on an arbitrary set of sampled states, which might be rather sensitive to the distribution of these sampled states, and graph edges do not rely solely on evaluating the policy's value function, which might poorly account for discontinuities (walls) or rarely visited states. Instead we grow and prune the graph dynamically so that it actually encodes reachability between goals.

During an iteration of the RGL procedure, a starting state $s_0$ is first sampled from an initial state distribution. Note that RGL does not suppose a fixed starting state. If $s_0$ is the first sampled starting state ever, or if the closest goal to $P(s_0)$ lies far from $P(s_0)$ in the goal space, this means $s_0$ does not correspond to any previously explored goal and we add a node in the graph at $P(s_0)$. Then, a node $v^*$ in the graph is selected for exploration. This selection relies on a count-based criterion which influences the progressive coverage of the goal space. Although one could design heuristics or refined exploration strategies (Bellemare et al. 2016; Ecoffet et al. 2019; Badia et al. 2019; Burda et al. 2019; Domingues et al. 2021) for this criterion, we rely on a simple count of the number of times a node has been selected for exploration, hence promoting uniform visits to every node in the graph. A finite horizon plan $(v_i)_{i \in [0,H]}$ is computed by finding a shortest path in the graph from the starting state's node $v_0$ to the selected node $v^* = v_H$. Note that there may not exist a path between $v_0$ and $v^*$ in the graph, in which case $H = 0$ and the `shortestPath` procedure returns the single node $\{v_0\}$. Let $(g_i)_{i \in [0,H]}$ denote the corresponding sequence of waypoint subgoals. Then $\pi$ is used to sequentially reach each goal. Specifically, when trying to reach $g_v$, $\pi(\cdot|g_v)$ is run until a $\texttt{reached}(s, v) := d^\pi(P(s), g_v) \leq \eta_{reach}$ condition becomes true, or a maximum number of steps $T_r$ is exceeded. If applying $\pi$ allowed the agent to reach the $\eta_{reach}$-neighborhood of $g_v$, then the next waypoint $w$ in the plan is selected and the procedure is repeated until the node $v^*$ is reached.

We interpret not reaching the neighborhood of $g_w$ when applying $\pi$ as a mismatch between the notion of reachability encoded in the graph and the actual reachability in the environment using $\pi$. As a consequence, we set the cost of edge $e$ between $v$ and $w$ to $+\infty$ to account for this non-reachability. Consequently, if the graph is learned without errors, the existence of an edge $e$ between two nodes $v$ and $w$ indicates that $\pi$ permits reaching the $\eta_{reach}$-neighborhood of $g_w$ from states $s$ whose $P(s)$ are in the $\eta_{reach}$-neighborhood of $g_v$ in less than $T_r$ time steps (or that this edge is never selected by the shortest path planning procedure). This pruning[2] procedure keeps the graph free of mis-identified edges. In mazes, it deletes edges that cross walls, and hence accounts for the discontinuities we wished to represent within the policy.

Conversely, if applying $\pi$ throughout the sequence of waypoints actually fulfills the goal $g^*$ of node $v^*$ selected for exploration, then a generic exploration procedure is performed from the reached state $s^*$ in the $\eta_{reach}$-neighborhood of $K_{v^*}$,

[2]Calling this operation pruning is somewhat inaccurate since we do not delete edges but rather set their weight to $+\infty$.

**Algorithm 1:** Reachability graph learning (RGL)

Input: $\pi$, $d^\pi$, $\eta_{reach}$, $\eta_{node}$, $\eta_{edge}$, $T_r$, $T_e$
Initialize: $V = \emptyset$, $E = \emptyset$
**repeat**
  $s_0 = \texttt{env.init}()$
  $g_0$ = goal associated to closest node to $P(s_0)$
  **if** $d^\pi(P(s_0), g_0) > \eta_{edge} \vee V = \emptyset$ **then**
    $V \leftarrow V \cup \{\texttt{node}(P(s_0))\}$
    $g_0 = P(s_0)$
  $v^* = \texttt{selectExplorationNode}(V)$
  $(v_i)_{i\in[0,H]} = \texttt{shortestPath}(V, E, g_0, v^*)$
  $s = s_0, t = 0$
  **for** $i \in [1, H]$ **do**
    **while** $\neg reached(s, v_i) \wedge t \leq T_r$ **do**
      $s \leftarrow \texttt{env.step}(s, \pi(s, g_{v_i}))$
      $t \leftarrow t + 1$
    **if** $\neg reached(s, v_i)$ **then**
      $\texttt{setWeight}(E, v_{i-1}, v_i, +\infty)$
      $\texttt{break}$
  **if** $reached(s, v^*) \vee H = 0$ **then**
    $\{s_t\}_{t\in[1,T_e]} \leftarrow \texttt{explore}(s, T_e)$
    $(V, E) = \texttt{growGraph}\left(V, E, \{s_t\}_{t\in[1,T_e]}\right)$

Function $\texttt{growGraph}(V, E, \{s_t\}_{t\in[1,T_e]})$:
**for** $t \in [1, T_e]$ **do**
  $\texttt{addNode} = True$, $E_{in} = E_{out} = \emptyset$, $w = \texttt{node}(P(s_t))$
  **for** $v \in V$ **do**
    $l_{in} = d^\pi(g_v, g_w)$
    $l_{out} = d^\pi(g_w, g_v)$
    **if** $l_{in} \leq \eta_{node} \wedge l_{out} \leq \eta_{node}$ **then**
      $\texttt{addNode} = False$; $\texttt{break}$
    **if** $l_{in} \leq \eta_{edge}$ **then**
      $E_{in} \leftarrow E_{in} \cup \{\texttt{edge}(v, w)\}$
      $\texttt{setWeight}(E_{in}, v, w, l_{in})$
    **if** $l_{out} \leq \eta_{edge}$ **then**
      $E_{out} \leftarrow E_{out} \cup \{\texttt{edge}(w, v)\}$
      $\texttt{setWeight}(E_{out}, w, v, l_{out})$
  **if** $\texttt{addNode} = True \wedge E_{in} \neq \emptyset$ **then**
    $V \leftarrow V \cup \{w\}$, $E \leftarrow E \cup E_{in} \cup E_{out}$
**return** $V, E$

during $T_e$ time steps. The intention of such an exploration procedure is to discover states $s$ whose $P(s)$ permit expansion of the graph. We randomly sample a goal within a certain radius of $g^*$ and try to reach it using $\pi$. If we succeed, we sample another random goal and repeat this exploration until we obtain a complete exploration trajectory of length $T_e$. Note that this exploration strategy could be replaced by any other, which is why we refer to it generically as the `explore` procedure in algorithm 1. The states visited along the trajectory are collected in a buffer. We wish to expand the graph so that its nodes induce a good coverage of the buffer states' goals and its edges indicate proximity (but not necessarily reachability at this stage). To that end, we cycle through the buffer and incrementally add vertices to $V$ whenever a goal is $d^\pi$-further away from all nodes than a threshold $\eta_{node}$. Edges are created from this new vertex to all nodes within $\eta_{edge} > \eta_{node}$. We differentiate between incoming and outgoing edges from the new candidate node: if there is no incoming edge, then the node is not added to the graph. This greedy procedure expands the graph to create new nodes that complete the goal space coverage wherever required, with limited connectivity between nodes. At this stage, some newly created edges might not account for reachability, e.g. in a maze, this might happen if the closest existing graph node to the newly created node is behind a wall. We rely on future explorations to prune the graph as presented in the previous paragraph.

Overall, this growth and pruning RGL procedure results in a graph $\mathcal{G} = (V, E)$ which encodes goal space reachability when using $\pi$ in the state space. The pair $(\mathcal{G}, \pi)$ implicitly defines a general goal-reaching policy which requires computing a shortest path in $\mathcal{G}$ to chain local executions of $\pi$ between subgoals.

At execution time, determining the action to undertake in $s$ in order to reach $g$ requires solving a shortest path problem in $\mathcal{G}$. This can be typically implemented using Dijkstra's algorithm (Dijkstra 1959), which has complexity $\mathcal{O}(E + V \log V)$.[3] It is important to note however that in deterministic MDPs (or MDPs with limited noise) this shortest path needs only be computed once per goal-reaching task and can be carried over to the next time step of the task, thus strongly dampening the overall computational cost. During learning, the pruning phase of an iteration of RGL has complexity $\mathcal{O}(E + V(\log V + T_r))$. The exploratory collection of new samples runs in $\mathcal{O}(T_e)$, while the `growGraph` function has complexity $\mathcal{O}(T_e V)$.[4] This results in an overall time complexity of $\mathcal{O}(E + V(\log V + T_r + T_e))$ for each iteration of RGL, which involves $\mathcal{O}(VT_r + T_e)$ iteration steps with the environment.

In the general case of MDPs with stochastic transitions, the pruning procedure of RGL needs to be adapted to account for the stochastic outcomes when trying to reach $g'$ from $K_g$. Note that, in this case, $d^\pi(g, g')$ captures a broader notion than the number of required time steps for $\pi$ to reach $g'$ from $g$: it captures the overall probability to reach $g'$ from $K_g$, given the transition model and a probability of termination of $1 - \gamma$ at each time step. Thus, reachability can be redefined as the probability of reaching $g'$ from $K_g$ being actually equal (or close to) $d^\pi(g, g')$. Verifying this with high confidence requires running several trials between $K_g$ and $g'$, which can be implemented by enhancing the algorithm with a memory of trial outcomes for each edge in the graph. Introducing such a delay in updates is similar in spirit to the practice of RMAX (Brafman and Tennenholtz 2002) or Delayed Q-learning (Strehl et al. 2006), which introduce an $N_{known}$

[3] For the sake of simplicity we adopt the notation $\mathcal{O}(V)$ in place of $\mathcal{O}(|V|)$.

[4] This can be amortized to $\mathcal{O}(T_e \log V)$ with efficient data structures for storing $V$ and finding nearest neighbors.

number of samples which are necessary to correctly identify a transition. Note that in most practical implementations of such algorithms, $N_{known}$ is arbitrarily set to a small value to preserve computational efficiency. An alternative, which we do not explore here and reserve for future work, is to let the weight of an edge adapt to the observed trial durations between $g$ and $g'$.

## 4 Empirical evaluation

To highlight the behavior of RGL, and provide a fair and interpretable benchmark against comparable methods, we consider a set of navigation tasks in mazes.

**Environments.** In each maze, an agent should be able to reach *any position* from its starting point. We consider mazes of different complexities, with various map sizes and heterogeneous corridor widths, as illustrated in figure 1. Namely, "four-rooms" is a $41 \times 41$-size maze resembling the classical "four-rooms" benchmark, "medium" is a more challenging maze of the same size, "hard" is an even more challenging $57 \times 57$-size maze and "mixed" has the same size as "hard" and mixes corridors and rooms of different sizes. Note that compared to mazes used in the literature (e.g. those of Eysenbach, Salakhutdinov, and Levine (2019)), here the walls are thin, inducing sharper discontinuities in the value function across a wall. For each map, we consider three different dynamics and state spaces for the navigating agent, which we refer to as grid-maze, point-maze and ant-maze (Table 1). Grid-maze features a discrete $\{N, S, E, W\}$ action space and deterministic transitions which perform unit-length moves, hence emulating navigation on a grid. Point-maze emulates a point mass moving freely in the maze. It has a continuous, two-dimensional action space of position increments in $[-1, 1]$ on the $x$ and $y$ axes. The transitions are stochastic due to an added Gaussian noise $\mathcal{N}(0, 1)$. Contrarily to grid-maze which has a fixed starting state, point-maze randomly draws the starting point at every episode. In grid-maze and point-maze environments, the state space is simply described by the geographical position of the agent, as in the benchmarks of SoRB (Eysenbach, Salakhutdinov, and Levine 2019) or SGM (Emmons et al. 2020), and $S = G$. Ant-maze environments build upon the MuJoCo Ant simulator (Todorov, Erez, and Tassa 2012) and sets the ant in one of the navigation maps. Actions belong to the standard 8-dimensional action space of the Ant simulator, and the state space is the 29-dimensional space whose first two coordinates are the position of the ant's torso, as in the benchmarks of HAC (Levy et al. 2019) or Distop (Aubret, Hassas et al. 2021). The transitions follow the dynamics of the Ant simulator. In ant-maze environments, the goal space $G$ describes only the torso's $x$ and $y$ coordinates. In all environments, agents receive a $-1$ reward at each time step, unless they reach the goal which terminates the episode. In all evaluations, every agent is independently trained 10 times. To enable reproducibility, hyperparameters for all algorithms are summarized in Appendix 5, and our code and results are available at [anonymous URL].

**Baselines.** To illustrate the behavior of RGL, we compare against a plain DQN agent (Mnih et al. 2015) with HER in grid-maze environments, and SAC (Haarnoja et al. 2018) with HER in point-maze and ant-maze environments. As illustrated by previous works (Nachum et al. 2018; Levy et al. 2019), such a combination can efficiently learn a goal-reaching policy for goals lying a few actions away from the starting state, but struggles to reach goals that require turning around walls. This provides a baseline for performance. Another baseline consists in passing the policy learned by this base agent along with its final replay buffer to SoRB, to extend its outreach throughout the goal space via planning in a random subset of size $N_{init}$ of the replay buffer. Since SGM is more efficient than SoRB (due to their pruning method), we directly compare with SGM.[5] We also implement a variant of RGL which we call TC-RGL, inspired by the STC method (Ruan et al. 2022), where we replace the $d^{\pi}$ pseudo-metric by a so-called temporal correlation network, which is an additional network trained to measure reachability between states, based on their temporal proximity during training trajectories. This variant permits evaluating the core feature of STC as an alternative to using the value function as a reachability metric between goals.

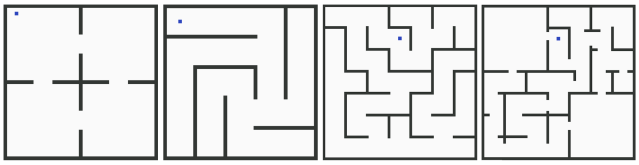


Figure 1: Mazes and starting points. From left to right: "four-rooms", "medium" ($41 \times 41$), "hard", "mixed" ($57 \times 57$).

**Pretraining.** To ensure SGM builds on a sufficiently good pre-trained policy, and following the practice of SoRB, we let the base agent learn a goal-reaching policy over 300 (resp. 500) episodes in grid-maze (resp. point-maze). RGL's lower level goal-reaching TILO policy is trained for 100 (resp. 150) episodes in grid-maze (resp. point-maze) environments. Because training the temporal consistency network of TC-RGL required more samples, it was trained for 200 (resp. 600) episodes in grid-maze (resp. point-maze). To account for pre-training durations, all figures below (e.g. Figure 2) report them using vertical lines. RGL's pre-training is performed in a playground environment of size $40 \times 40$ with no walls. Because ant-maze environments required specific pre-training, we defer their discussion to the end of this section.

**Visualizing graph growth and pruning.** We start by assessing separately the influence of the growth and pruning procedures on the properties of the final reachability graph. To isolate the effect of pruning, we artificially generate waypoints by using a generative model to draw states from the full state space, which yields a graph with the same number of nodes $N_{init}$ as the SGM agents (edges weights are also initialized with $d^{\pi}$), but with better state space coverage since the drawn states are not constrained by the exploration

[5] Note that SoRB and SGM were introduced with tailor-made, goal-based, distributional DQN and DDPG agents. We found this was not necessary for finite-length trajectories and we retain the names even though we slightly change the base agents.

| | $dim(S)$ | $dim(G)$ | actions | dynamics | starting state |
|---|---|---|---|---|---|
| grid-maze | 2 | 2 | discrete (number = 4) | deterministic | fixed single state |
| point-maze | 2 | 2 | continuous ($dim(A) = 2$) | stochastic | uniform distribution on $S$ |
| ant-maze | 29 | 2 | continuous ($dim(A) = 8$) | deterministic | fixed single state |

Table 1: Summary of environments

of the pre-trained DQN+HER agent. This permits defining *Prune-only RGL* (PO-RGL) as the algorithm which prunes this graph as it successively tries to reach random goals, but without performing exploration and graph expansion. Appendix 5 illustrates the evolution of the graphs produced by PO-RGL and RGL in the "four-rooms" and "hard" grid-maze environment.

**RGL agents can reach any goal.** Figure 2 reports the ability of each agent to reach any goal in the maze, along training. The vertical lines correspond to the time at which RGL, TC-RGL, or SGM start exploiting the pre-trained policy. Every 1,000 interactions with the environment, we randomly draw 30 goals across the full goal space, and report the fraction of these goals the agent managed to reach. We call this metric the agent's *accuracy*. As expected, since exploration in mazes is difficult, the pre-training replay buffers do not cover the full state space and the baselines fail to reach all goals. Interestingly, despite the low performance of the pre-trained DQN+HER and SAC+HER agents, RGL is still able to leverage their ability to reach local goals and manages to quickly grow a goal graph which eventually covers the full maze. PO-RGL displays a clear jumpstart effect in the "four-rooms" maze since its initial graph requires little pruning and most goals are readily reachable. Conversely, early planning graphs of RGL and TC-RGL contain few nodes and require expansion (and thus interaction steps) before their accuracy reaches 1. After 1,000 interactions, even though the planning graph of RGL contains only a few nodes (Figure 10, Appendix 5), it already reaches more goals than the baseline agents. As the mazes become more difficult, many more edges need to be pruned from PO-RGL's initial graph before it effectively represents graph reachability and the plans reliably lead to goals. This need for extended pruning is completely compensated by the sparse growth of the graph of RGL and TC-RGL, and PO-RGL presents no advantage in terms of learning curve. In the most difficult "hard" and "mixed" mazes, the set of $N_{init}$ initial nodes of PO-RGL is just insufficient to properly cover the full goal space with feasible edges and PO-RGL's accuracy is capped around 0.5 and 0.8, while the dynamic growth of RGL permit reaching close to 1 accuracy. Also, the extra temporal consistency network of TC-RGL seems detrimental to the training process compared to RGL. Since this network only approximates the notion of reachability instead of directly using the value function, it induces a graph expansion and pruning phase with more errors or missed nodes (which were actually reachable). In turn, as TC-RGL's graph does not accurately represent reachability, some goals are eventually missed. In all environments, RGL reaches an accuracy of 1 and dominates over all variants.

**Graph size.** Overall, RGL produces sparse graphs with little variance in number of nodes within an environment. Due to lack of space, we refer the reader to Appendix 5 for a more detailed discussion.

**Limit case: "reset anywhere".** Point-maze environments feature a stochastic transition model and random resets anywhere in the state space at the begining of each episode, as in the benchmarks of Eysenbach, Salakhutdinov, and Levine (2019); Emmons et al. (2020). This induces diversity in the replay buffers by triggering easier exploration, and somehow departs from the more constrained RL framework with a fixed (or a limited set of) starting state. Consequently, these environments are more favourable to SGM since their replay buffer covers a larger portion of the state space, and SGM performs better in these environments than in grid-maze ones (Figure 2). Even in this case the graph growth of RGL eventually outperforms competing methods, as it progressively discovers new goal waypoints to better map the state space.

**Limit case: stochastic transitions.** As mentioned earlier, RGL in its presented version is designed for deterministic dynamics and would require some adaptations to account for transition uncertainty. Point-maze environments feature a rather high level of action noise ($\sigma = 1$ for action values in $[-1, 1]$). This makes the pruning procedure stochastic, as it will prune out edges depending of a single trial's success. Despite this rather naive behavior, RGL still manages to find paths (possibly sub-optimal) to goals and reaches a high level of accuracy (Figure 2) demonstrating a reasonable level of robustness to transition stochasticity.

**Limit case: high-dimensional state spaces and $G \neq S$ in ant-maze tasks.** Training a goal-reaching policy in ant-maze environments, even in an obstacle-free playground, is already a challenging task. Appendix 5 expands on the pre-training procedure set in place. HAC is the reference method for ant-maze environments but, as discussed in Appendix 5, its efficiency appears brittle and despite our best efforts and the use of the original HAC implementation, it could not solve any of the ant-maze tasks. Hence, Figure 3 only reports results for RGL. Ant-maze tasks, on top of being highly challenging for the baseline agent, also violate the assumption of section "Goals as state abstractions" that any two states within $K_g$ are reachable from each other for a negligible cost. For instance, some ant orientations, velocities and leg configurations are rather complex to reach from others. Consequently, an edge between $g$ and $g'$ in RGL's graph only represents reachability of $g'$ from a subset of states in $K_g$, which can lead to plan failure (extended discussion in Appendix 5). Despite this, RGL manages to achieve large values of accuracy almost as high as those obtained on point-maze tasks on the "four-rooms", "medium" and "mixed" mazes. The most challenging setting remains the "hard" maze, which

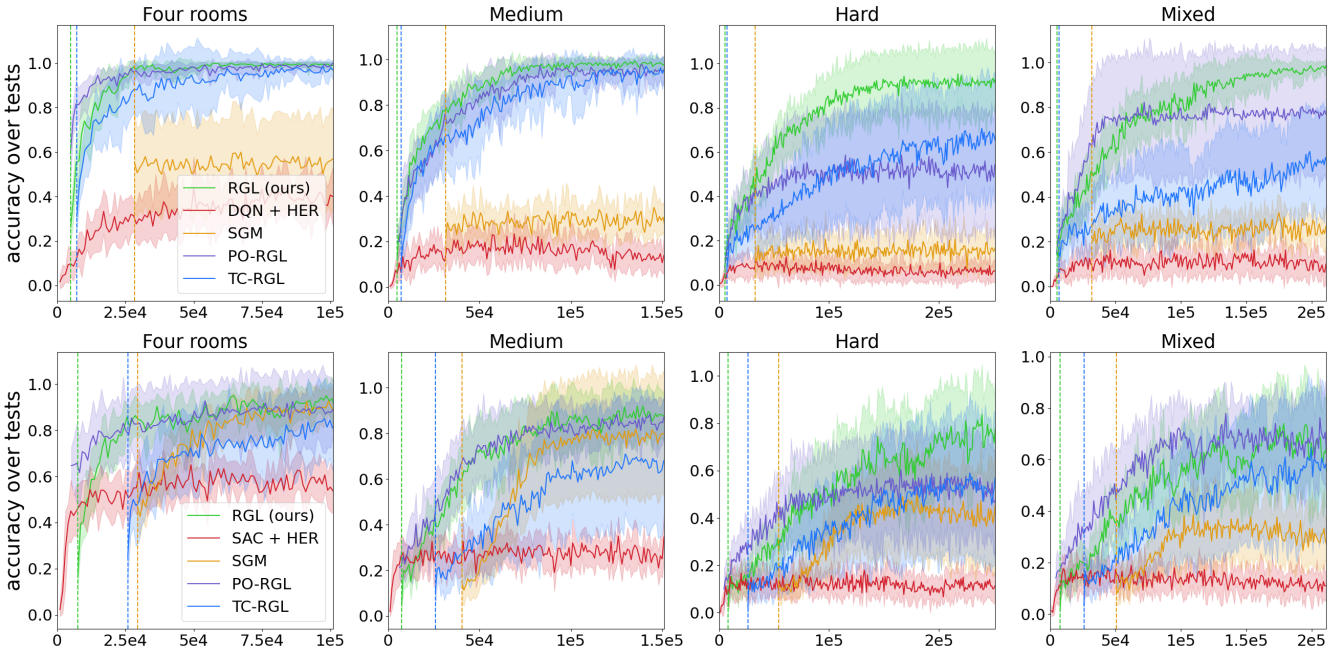



Figure 2: Accuracies of all agents on grid-maze (top) and point-maze (bottom) environments, versus interaction steps.

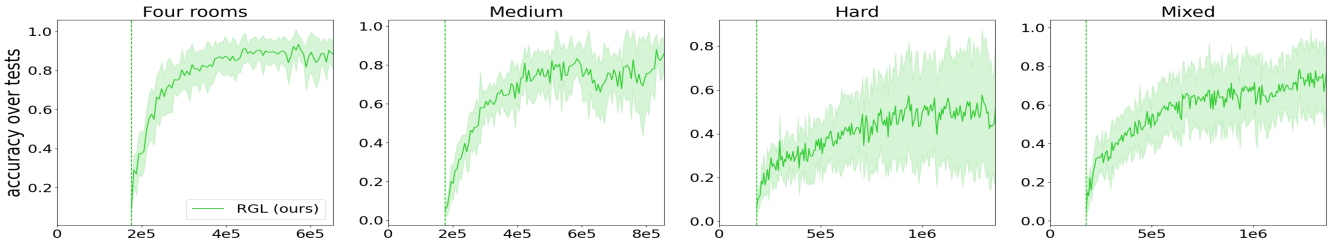



Figure 3: RGL accuracy in ant-maze tasks, as a function of training time steps.

requires fine motor skills to efficiently navigate through narrow corridors and requires turning around many corners to navigate to far goals.

## 5 Conclusion

Efficient coupling of planning and learning in complex MDPs with temporally extended goals is an active field of research. In this work we defend the idea that efficient mechanisms rely on two implicit hypotheses: planning agents should plan in the *goal space* and learned policies are often *re-usable* throughout the state space. We propose a formal framework accounting for these two notions, defining goals as state abstractions and re-usability as translation invariance. This permits deriving an algorithm which performs planning over a graph of goal waypoints, reachable by a lower level goal-reaching policy. This agent is named RGL (*reachability graph learning*) after its training mechanism, which incrementally explores its environment using a pre-trained lower level translation invariant goal-reaching policy, expands and prunes a graph encoding goal reachability. This approach can be seen either as a more grounded version of STC (Ruan et al. 2022), or a generalization of SoRB (Eysenbach, Salakhutdinov, and Levine 2019) or SGM (Emmons et al. 2020) to a hierarchical setting with translation invariance. Empirical evaluation confirms the relevance of RGL agents and their key features. This contribution also forms a basis for future research directions. As is, RGL agents build a somewhat uniformly dense graph, as illustrated in Figure 10 (Appendix 5). This might not be necessary and further sparsity can be achieved in the open rooms of the "four-rooms" or the "mixed" environments. Similarly, weight learning in the graph is currently rather naive and could better exploit interaction data during exploration, in particular in stochastic environments. Finally, RGL requires an $\epsilon$-TILO policy for agent's control. Appendix 5 proposes a discussion on whether such policies are easy to obtain in the general case, beyond navigation tasks. We conjecture such policies also exist in more complex contexts, like vision-based navigation (PO)MDPs, since humans seem to exploit such invariances in daily life. Formalizing how these policies can be discovered and how their definition affects the properties derived in the present work is an exciting avenue for research.

# Appendix

## Hyperparameters and computational setup

| SAC | | |
|---|---|---|
| | Point-Maze | Ant-Maze |
| critic hidden layers | 250, Relu, 150, Relu | |
| actor layers | | |
| optimiser | Adam(lr=5e-4, betas=(0.9, 0.999), eps=1e-08, weight_decay=0) | |
| replay buffer size | 1e5 | 1e6 |
| batch size | 100 | 500 |
| $\gamma$ | 0.99 | 0.99 |
| $\tau$ | 5e-3 | 5e-3 |
| critic alpha | 0.6 | 0.6 |
| actor alpha | 0.05 | 0.1 |

Table 2: SAC hyper-parameters. SAC+HER serves as a control policy for RGL, PO-RGL, and TC-RGL, as well as a baseline, in the "point-maze" and "ant-maze" environments.

Tables 3 to 8 summarize the hyperparameters used when training the different algorithms. The actor network used for the lower level goal-reaching policy takes a state and a goal as input (the dimension varies depending on the task) and processes them through a 2 hidden layer MLP. The output layer depends on the algorithm. Training follows the procedure of DQN and HER with discount factor $\gamma$ and exponential smoothing on the target network (factor $\tau$), and an Adam (Kingma and Ba 2015) optimizer with default parameters. These parameters are the same for pretraining lower level policies for all algorithms. All RGL agents share the same $T_e$, $\eta_{node}$, $\eta_{edge}$, and $\eta_{reach}$ when applicable (for instance, PO-RGL uses $\eta_{edge}$ but not $\eta_{node}$ since it does not create new nodes). TC-RGL uses specific values $\eta_{node}$ and $\eta_{edge}$ for thresholds on node and edge creation, since it uses STC's temporal consistency network to measure node distance instead of our $d^{\pi}$ pseudo-metric; the scale of this network's output is unrelated to that of $d^{\pi}$ (hence the different values of $\eta_{node}$ and $\eta_{edge}$).

| DQN | |
|---|---|
| model hidden layers | 64, ReLU, 64, ReLU |
| optimiser | Adam(lr=1e-3, betas=(0.9, 0.999), eps=1e-08, weight_decay=0) |
| replay buffer size | 1e5 |
| batch size | 100 |
| discount factor $\gamma$ | 0.95 |
| exponential smoothing factor $\tau$ | 1e-3 |

Table 3: DQN hyperparameters. DQN+HER is used in grid-maze tasks to compute goal reaching policies.

The results on grid-maze and point-maze were run on a desktop machine (Intel i9, 10th generation processor, 64GB RAM) with no GPU usage. The results on ant-maze were obtained with single node computations. Each of these nodes was composed of 2 12-core Skylake Intel(R) Xeon(R) Gold 6126 2.6 GHz CPUs with 96 Go of RAM (no GPU hardware).

| C51 | |
|---|---|
| output distribution size | 20 |
| models layers | 64, ReLU, 64, ReLU |
| optimiser | Adam(lr=1e-3, betas=(0.9, 0.999), eps=1e-08, weight_decay=0) |
| replay buffer size | 1e5 |
| batch size | 100 |
| $\gamma$ | 0.95 |
| $\tau$ | 1e-3 |

Table 4: C51 hyper-parameters, which serves as a control policy for SGM in the "grid-maze" environment.

| Distributional DDPG | |
|---|---|
| output distribution size | 20 |
| models layers | 64, ReLU, 64, ReLU |
| optimiser | Adam(lr=1e-4, betas=(0.9, 0.999), eps=1e-08, weight_decay=0) |
| replay buffer size | 1e6 |
| batch size | 64 |
| $\gamma$ | 0.99 |
| $\tau$ | 0.05 |

Table 5: Distributional DDPG hyper-parameters, which serves as a control policy for SGM in the "point-maze" environment.

| RGL | | | |
|---|---|---|---|
| | Grid-maze | Point-maze | Ant-Maze |
| $\eta_{edges}$ | 0.2 | 0.045 | 0.3 |
| $\eta_{nodes}$ | 0.1 | 0.017 | 0.1 |
| nodes reachability threshold | 1 | 0.8 | 0.7 |
| max time-steps to reach next node | 50 | 50 | 150 |
| Exploration goal range | 2 | 4 | 6 |
| interactions per exploration | 90 | 90 | 150 |

Table 6: RGL hyper-parameters.

| PO-RGL | | |
|---|---|---|
| | Grid-maze | Point-maze |
| $\eta_{edges}$ | 0.2 | 0.03 |
| reachability threshold of the nodes | 1 | 0.8 |
| max time-steps to reach next node | 50 | |
| nb nodes | four rooms: 400<br>medium: 600<br>hard: 600<br>mixed: 900 | four rooms: 400<br>medium: 500<br>hard: 700<br>mixed: 900 |

Table 7: PO-RGL hyper-parameters.

Our code is available at [Anonymous URL].

## Visualization of graph growth

Figures 10, 11 and 12 present the reachability graphs evolution for all mazes in, respectively, grid-maze, point-maze and ant-maze environments. Blue dots in some figures correspond to the current selected goal at the time the graph was printed and should be discarded.

In all these figures, red edges are those whose weights have been set to $+\infty$ by the pruning procedure. We observe that (as anticipated in the previous section) only erroneous edges which were selected in a shortest path are pruned, and some remain in the graph, especially in grid-maze, which features a fixed unique starting state. This bears little consequences in terms of goal reachability since these are never selected in shortest paths from the initial state, but still result in a rather dense reachability graph. To avoid misinterpretations, it is important to note that since the graph is oriented, each green edge in these figures actually stands for two edges in the graph. If only one has been pruned and rendering of the other happens afterwards, the segment appears green while only one edge in the graph has non-infinite weight. Overall, the incremental growth of RGL's graph yields an efficient coverage of the state space, avoiding the clusters of unnecessary nodes we can observe using PO-RGL, and reducing the need for pruning.

PO-RGL was created purely for didactical reasons in order to illustrate the pruning process independently of the incremental graph growth. Besides this illustration itself, these figures underline two features. First, the fact that RGL creates the graph incrementally makes it much sparser and avoids clusters of really close, redundant nodes. In turn, this sparse graph is much easier to prune than that of PO-RGL. Secondly, in environments with a fixed initial state (grid-maze, ant-maze), some edges never participate in the shortest path to any goal and hence are never pruned. Even if the sparse growth of the RGL graph limits this phenomenon, some impassable edges remain; e.g. some edges at the far right of Figure 10p. Randomly resetting the starting state at each episode permits a more complete and easier exploration of all shortest paths, and hence results in a slightly more accurate pruning; e.g. the unpruned edges in grid-maze are better pruned on Figure 11p.

| SGM | | |
|---|---|---|
| | Grid-maze | Point-maze |
| node pruning threshold | four rooms: 2<br>medium: 3<br>hard: 3<br>mixed: 2 | four rooms: 3<br>medium: 3<br>hard: 3<br>mixed: 3 |
| max edges length | four rooms: 5<br>medium: 6<br>hard: 6<br>mixed: 5 | four rooms: 7<br>medium: 7<br>hard: 7<br>mixed: 7 |
| nb initial nodes | four rooms: 1400<br>medium: 1400<br>hard: 1800<br>mixed: 1600 | |
| reachability threshold | 1 | |
| max interactions per sub task | 20 | |

Table 8: SGM hyper-parameters.

Figure 12, Appendix 5, (ant-maze environments) deserves a few additional comments. On this graph, to ease the readability and account for directed edges, whenever a directed edge exists between $v$ and $w$, we plot the edge's segment closest to $v$ in green. Orange then means the reverse edge has not been created. Red means the edge has been pruned. Some pruned edges appear in areas which seem passable. To explain this phenomenon, one needs to recall that the state space is 29-dimensional and a waypoint in the goal space (a geographical position of the ant's center of mass) can stand for a wide variety of configurations, as discussed in the empirical evaluation section. For any two nodes $g$ and $g'$, it is possible that $g'$ was reachable from $\bar{P}(g)$ but is not reachable from some other states in $K_g$, since ant-maze environments violate the hypothesis that all states in $K_g$ are reachable from each other for a negligible cost given the pre-trained policy. This leads to some edges being legitimately pruned while a "naive eye" laid on the reachability graph might conclude there was a mistake.

Finally, the graphs grown by RGL in ant-maze environments feature very few edges crossing walls. This is a side effect of the default values of $\eta_{node}$ and $\eta_{edge}$ (kept the same throughout all environments and mazes), and the fact that the ant's geometry prevents its center of mass to get close to the wall. This sometimes happens nonetheless when the ant randomly "tries" to climb over the wall (and systematically fails), which also places a few nodes that appear to be inside the walls.

**Graph size.** Figures 4 to 9 report the number of nodes and edges for RGL agents as their graph grows in the grid-maze, point-maze, and ant-maze environments. Note that graphs on point-maze environments required a log-scale on the $y$-axis for readability since TC-RGL spanned an order of magnitude more nodes than RGL (and two to three orders of magnitude more edges). Recall that instead of deleting edges that need to be pruned, their traversal weights are set to $+\infty$ (to avoid creating them again later). This is why the number of edges of PO-RGL does not decrease. Dotted curves in Figures 7 to 9 indicate the number of edges with a non-infinite weight. Overall, RGL and TC-RGL create just enough nodes to accurately represent the reachability graph given their underlying $d^\pi$ and temporal consistency network. The relative number of node and edges between RGL and TC-RGL cannot be directly compared as the former uses $d^\pi$ as a distance metric while the latter uses a reachability representation, on a different (uncontrolled) scale. Still, the number of nodes is similar across mazes. Interestingly, RGL produces graphs with less connectivity, which can be interpreted as a better ability to create meaningful connections between goal waypoints for navigation. Additionally, TC-RGL features a large variance in the number of nodes and edges developed in the graph. This seems to stem from the training of the temporal consistency network which is very sensitive to the distribution of trajectories during pre-training. In turn, this strongly affects the estimation of reachability when learning the graph

| TC-RGL | | | |
|---|---|---|---|
| | | Grid-maze | Point-maze |
| $\eta_{edges}$ | | 0.4 | 0.1 |
| $\eta_{nodes}$ | | 0.2 | 0.03 |
| TC-Network | layers | 125, ReLU, 100, ReLU, 1, Sigmoid | |
| | batch size | 250 | |
| | buffer max size | 1e9 | |
| | optimizer | Adam(lr=1e-3, betas=(0.9, 0.999), eps=1e-08, weight_decay=0) | |
| | targeted edge length | 20 | |

Table 9: TC-RGL hyper-parameters. Hyper-parameters that are not reported here are the same that the ones in Table 6 for RGL. "targeted edge length" is the minimum number of interactions that must separate two states of the same trajectory, so that they can form a positive pair (distant states) in the TC-network training data.

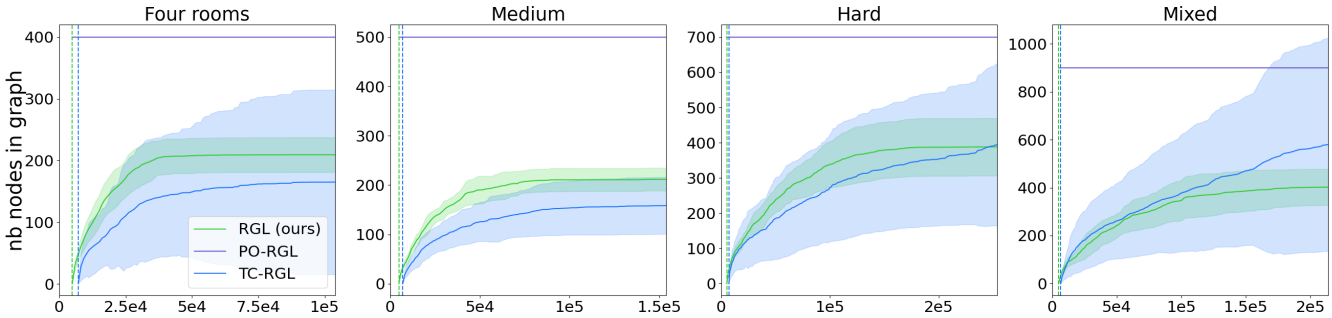



Figure 4: Number of graph nodes in grid-maze versus interaction steps. Shaded area is the $1\sigma$ confidence interval.

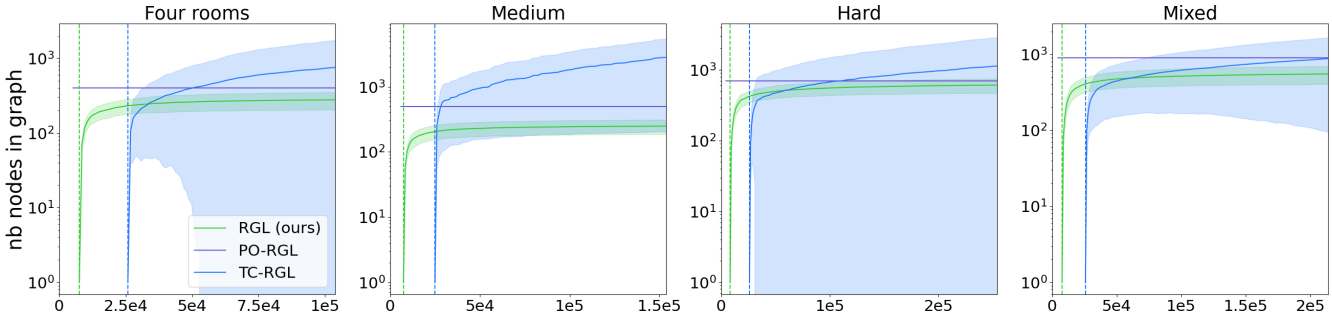



Figure 5: Number of graph nodes in point-maze versus interaction steps. Shaded area is the $1\sigma$ confidence interval.

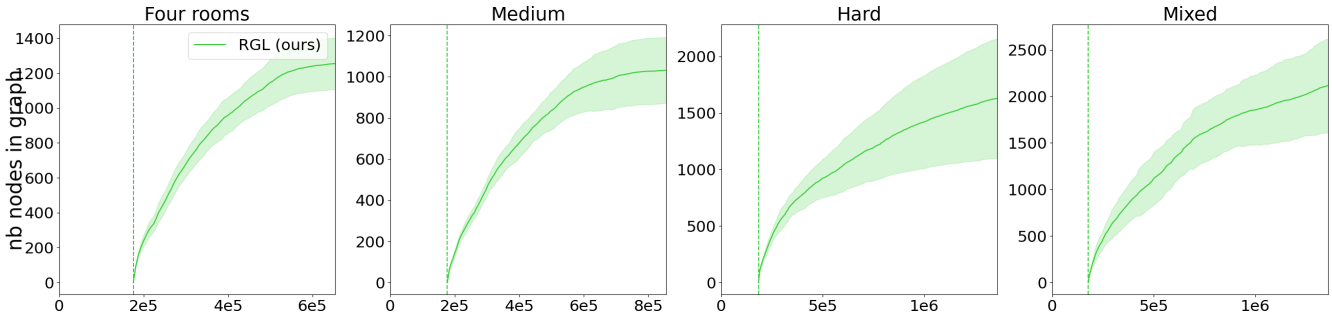



Figure 6: Number of graph nodes in ant-maze versus interaction steps. Shaded area is the $1\sigma$ confidence interval.

and induces this variance in graph density. Appendix 5 provides further discussion on the impact of the graph density's hyperparameters ($\eta_{node}$ and $\eta_{edge}$) on RGL's behavior.

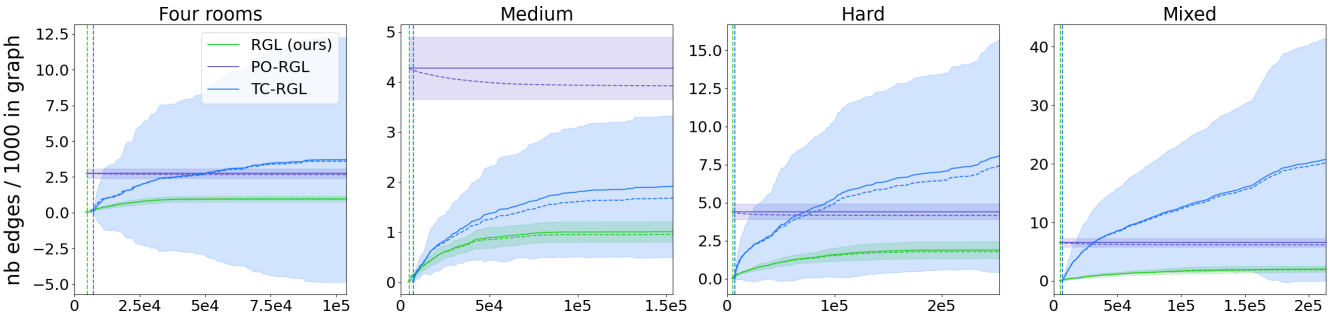



Figure 7: Number of graph edges in grid-maze versus interaction steps. Shaded area is the $\sigma$ confidence interval.

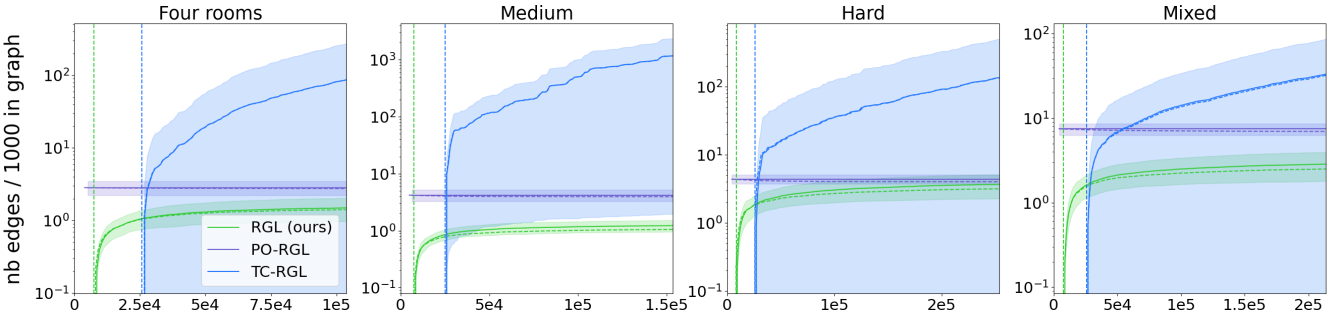



Figure 8: Number of graph edges in point-maze versus interaction steps. Shaded area is the $\sigma$ confidence interval.

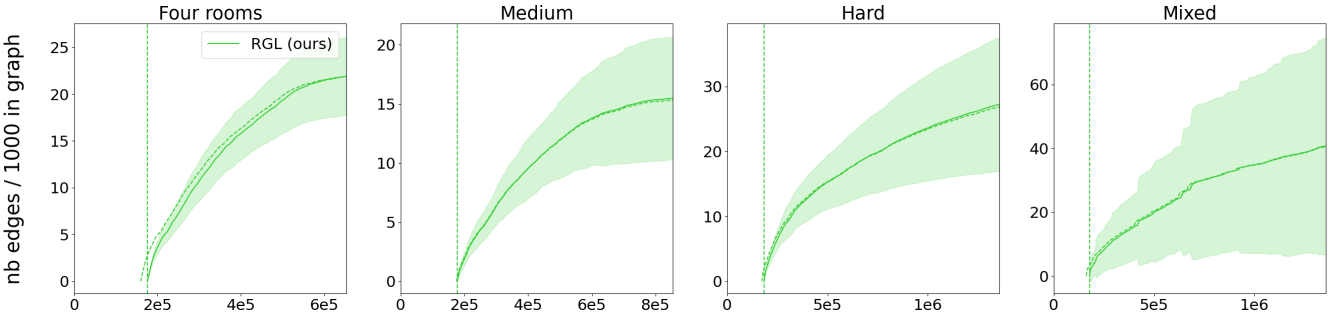
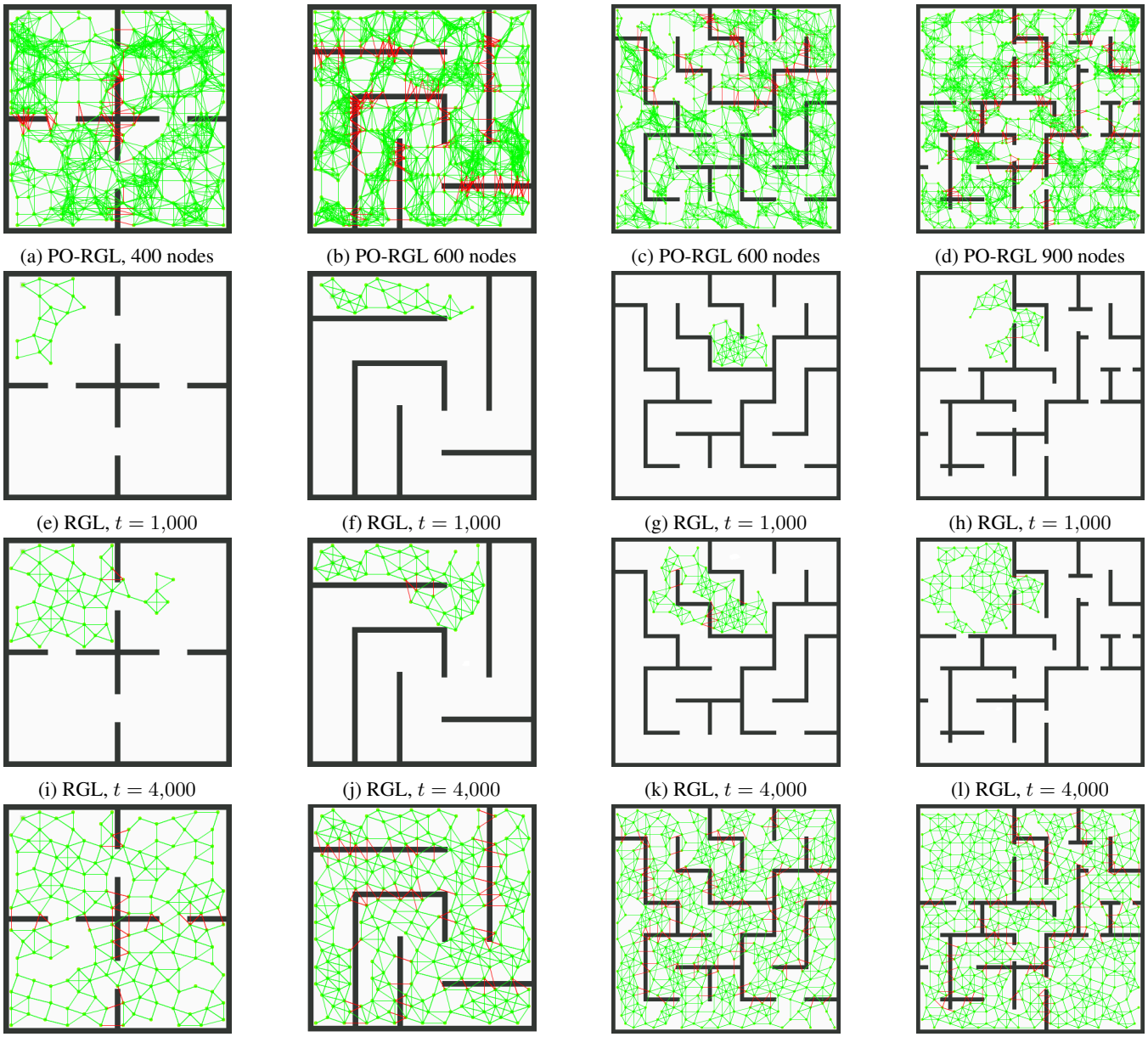



Figure 9: Number of graph edges in ant-maze versus interaction steps. Shaded area is the $\sigma$ confidence interval.

## Are $\epsilon$-TILO policies common?

In the present work, an important assumption is the existence of an $\epsilon$-TILO policy. Thus it seems important to discuss how restrictive this assumption is, and how commonly such policies might occur.

In position-based navigation tasks where $S = G$, generalization by translation invariance seems intuitive and easily justified by the translation invariance of the MDP's transition model properties throughout the state space. In navigation tasks where the state space is the agent's full configuration, but with abstract goal spaces (e.g. agent overall position), such as the ant-maze benchmarks, finding TILO policies is closely linked to defining the goal space, and hence the $P : S \to G$ projection. In this specific example, $P$ is defined by simply keeping some variables of $S$ and discarding the others. Here again, the TILO property is intuitive and translation invariance permits generalizing learned policies to unexplored parts of the state and goal spaces. However, when it comes to state spaces with confounding variables, such as visual navigation tasks, then defining $P$ for abstract goal spaces might become more difficult as it links the input image pixels to positions on the navigation map. In a way, $P$ encodes expert knowledge about what abstractions of the state define a useful goal space, as discussed for instance by Forestier et al. (2022). Such abstractions might be learned (Péré et al. 2018) but since they are a pre-requisite for training a goal-based policy, they are generally considered to be provided by some expert. Such a description of goals is sometimes accessible for a minimal cost (as in navigation tasks), but a perspective for future work implies learning relevant goal descriptors from data. One one can draw a parallel with recent work in expressing goals with natural language



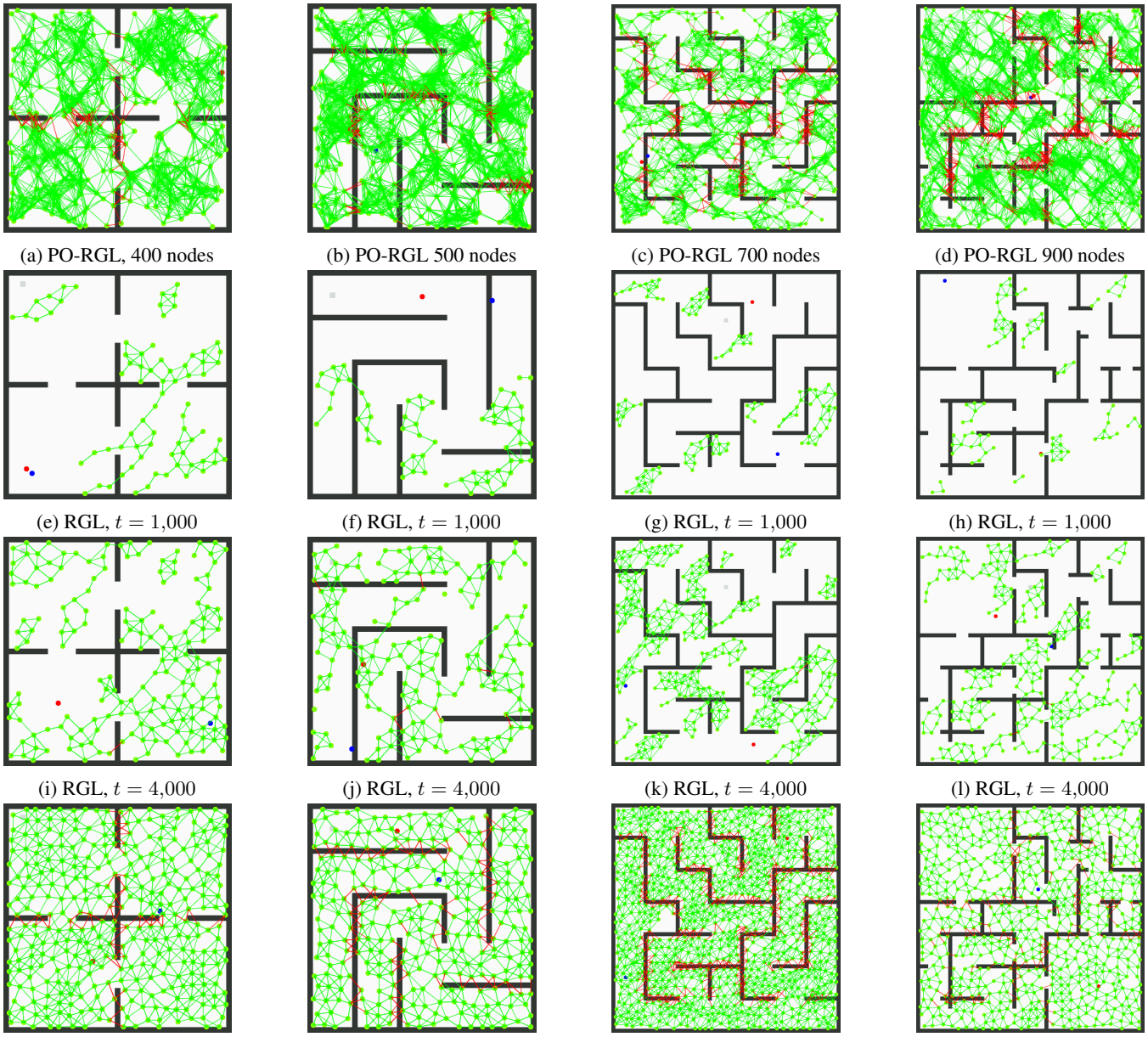
(a) PO-RGL, 400 nodes (b) PO-RGL 600 nodes (c) PO-RGL 600 nodes (d) PO-RGL 900 nodes

(e) RGL, $t = 1{,}000$ (f) RGL, $t = 1{,}000$ (g) RGL, $t = 1{,}000$ (h) RGL, $t = 1{,}000$

(i) RGL, $t = 4{,}000$ (j) RGL, $t = 4{,}000$ (k) RGL, $t = 4{,}000$ (l) RGL, $t = 4{,}000$

(m) RGL, $t = 100{,}000$ (n) RGL, $t = 150{,}000$ (o) RGL, $t = 250{,}000$ (p) RGL, $t = 210{,}000$

Figure 10: Reachability graphs, grid-mazes.



(a) PO-RGL, 400 nodes (b) PO-RGL 500 nodes (c) PO-RGL 700 nodes (d) PO-RGL 900 nodes

(e) RGL, $t = 1{,}000$ (f) RGL, $t = 1{,}000$ (g) RGL, $t = 1{,}000$ (h) RGL, $t = 1{,}000$

(i) RGL, $t = 4{,}000$ (j) RGL, $t = 4{,}000$ (k) RGL, $t = 4{,}000$ (l) RGL, $t = 4{,}000$

(m) RGL, $t = 100{,}000$ (n) RGL, $t = 150{,}000$ (o) RGL, $t = 250{,}000$ (p) RGL, $t = 210{,}000$

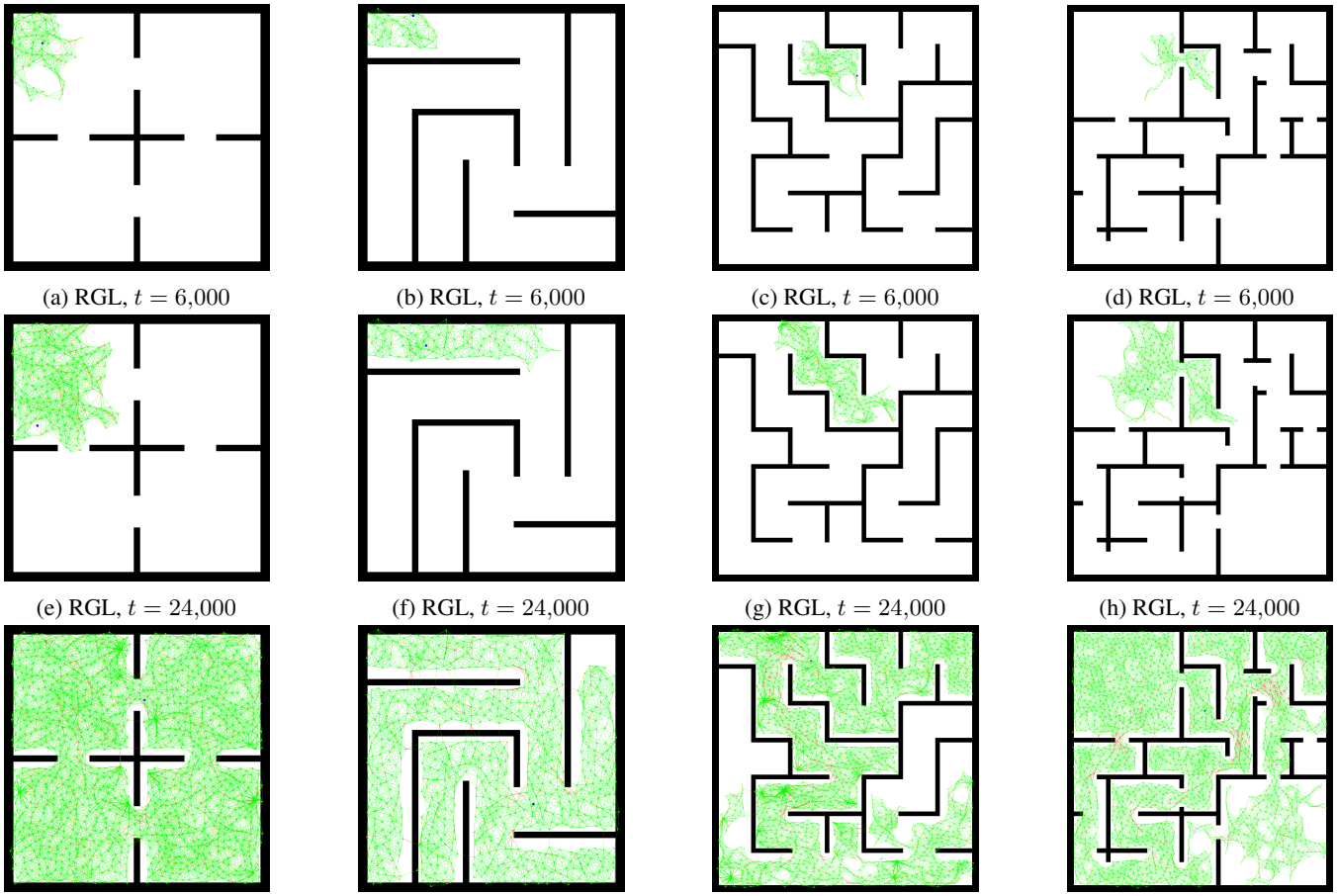
Figure 11: Reachability graphs, point-mazes.

and exploiting (large) language models to embed the goal description. Note, however, that in the general case, even if the corresponding $P$ encoding is given, there is no guarantee that a TILO policy exists.

Besides the considerations above, we argue that the existence of TILO policies is intrinsically linked more to the nature of the task at hand than the definition of the goal space. Navigation is implicitly about finding a (potentially convoluted) path through a terrain with somewhat homogeneous properties. Hence, at least for this family of tasks, the existence of $\epsilon$-TILO policies is a plausible assumption.

**Influence of graph density hyperparameters.**

The thresholds $\eta_{node}$ and $\eta_{edge}$ on node and edge creation condition how coarse the graph is in the goal space. Consequently, they impact the density of the graph, hence the ability to accurately represent transition dynamics. As such, they encode a notion of minimal required granularity to efficiently generate efficient goal-reaching plans in the goal space. Despite RGL's ability to build sparse representative graphs, a poor choice of $\eta_{node}$ and $\eta_{edge}$ parameters can be detrimental to RGL's goal reaching accuracy. Figure 13 reports how sensitive PO-RGL and RGL agents are to these parameters, in the "medium" grid-maze. Figure 13a illus-



(a) RGL, $t = 6{,}000$ (b) RGL, $t = 6{,}000$ (c) RGL, $t = 6{,}000$ (d) RGL, $t = 6{,}000$

(e) RGL, $t = 24{,}000$ (f) RGL, $t = 24{,}000$ (g) RGL, $t = 24{,}000$ (h) RGL, $t = 24{,}000$

(i) RGL, $t = 500{,}000$ (j) RGL, $t = 700{,}000$ (k) RGL, $t = 1200{,}000$ (l) RGL, $t = 1,200{,}000$

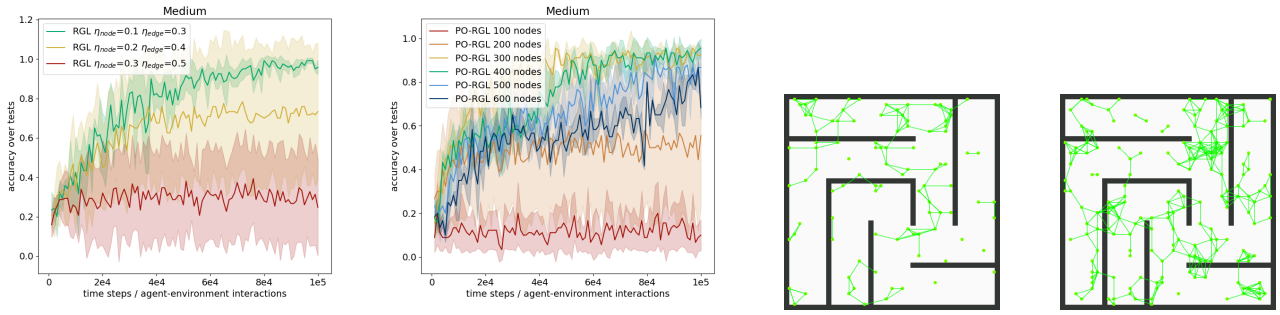
Figure 12: Reachability graphs, ant-mazes.

trates how increasing the values of $\eta_{node}$ to 0.2 (then 0.3) and $\eta_{edge}$ to 0.4 (then 0.5) results in a graph which does not enable reaching distant goals anymore. A similar effect happens for PO-RGL, as choices for $N_{init}$ will have a direct impact on the accuracy of the algorithm. Figure 13b reports how varying $N_{init}$ from 100 to 600 affects the goal reaching accuracy of PO-RGL. With only 100 nodes, the reachability graph of PO-RGL features subgoals which are very distant from each other and rarely reachable between each-other, resulting in a graph with almost no edges (Figure 13c). Hence, no goals besides those reachable by the lower-level policy can be reached. With 200 nodes (Figure 13d), the final goal reaching accuracy of PO-RGL improves to about 50% and keeps improving until $N_{init} = 400$ nodes. For $N_{init} = 500$ and 600, the number of edges to prune in the graph becomes so large that it slows the learning down, resulting in less reachable nodes after 100,000 interaction steps because the graph contains too many misleading edges which have not been pruned yet. Overall, this illustrates how the directed, exploration-driven node and edge creation of RGL yields graphs which are both much sparser and much more representative of reachability, than building a graph over randomly sampled goals (either randomly sampled from a replay buffer as in SoRB, or randomly sampled from an oracle as PO-RGL).

### Pre-train a goal-conditioned TILO policy in Ant-Maze.

Learning goal-based policies for ant-maze environments is challenging, even in the obstacle-free playground. To let SAC+HER converge efficiently, we build a process inspired from curriculum learning (Bengio et al. 2009). We sample goals uniformly in a disc around the agent, starting with a radius of 0. Every time the agent reaches a goal, we increment the radius of 0.1, and decrease it when it fails. If the radius reaches a value of 6, we stop incrementing and let the agent reach an accuracy close to 100% in this pre-training playground. Note that this value of 6 is much larger than that of $\eta_{node}$ and $\eta_{edge}$ (see Appendix 5).

While navigating in the graph, following a sequence of sub-goals, the agent will change its direction many times in an episode. This may lead to more diversity in the states encountered while navigating the maze than those seen during the pre-training. To mitigate this aspect and improve state diversity during pre-training, every 5 episodes, instead of a full agent reset, we reset only the agent's position but retain




(a) Influence of $\eta_{node}$ and $\eta_{edge}$ (b) Influence of $N_{init}$ on PO-RGL (c) PO-RGL, $N_{init} = 100$ (d) PO-RGL, $N_{init} = 200$

Figure 13: Hyperparameter influence on goal-reaching accuracy in the "medium" grid-maze after 100,000 interaction steps.

the orientation, legs configuration and velocities from the last state of the previous episode.

## Hierarchical actor critic on various tasks

Ant-maze tasks have been tackled in previous work, notably in the important HAC (Levy et al. 2019) contribution, on similar tasks to those reported here, in particular the "four-rooms" maze. In order to provide a fair comparison with RGL, we used the reference implementation of HAC provided by the authors at https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-/tree/master/ant_environments/ant_four_rooms_3_levels. This section discusses why this implementation (without modifications) fails on the tasks reported here.

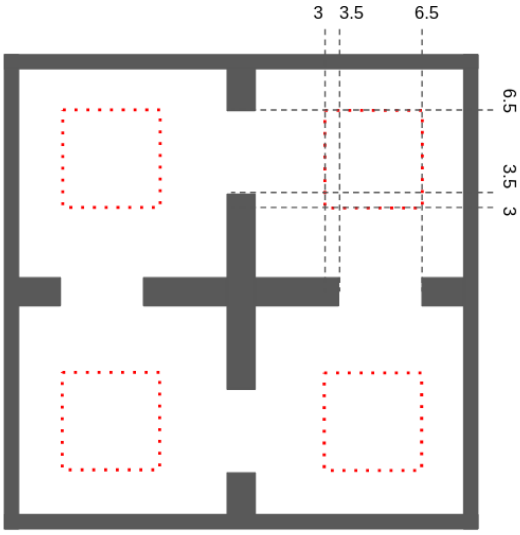



Figure 14: Illustration of goals and initial states sampling areas for HAC.

**Goal and initial state sampling in HAC to promotes diversity.** In the original HAC contribution, during training, goals are sampled uniformly in the center of each room (red areas in Figure 14), then initial states are sampled uniformly in the center of another room. This induces a variety of starting states and insures that starting states and goals are always at least one room away from each other. In turn, this promotes diversity in the replay buffers, which facilitate policy training. In the experiments, we argued that this "reset anywhere" feature was a particularly favourable case for exploration.

**Variations in mazes.** We also investigated whether the loss of efficiency of HAC could be attributed to the difference between the mazes presented here and those of the HAC paper. For this purpose, we tested HAC on three tasks and report results in Figure 16 (averaged over 10 trials).

1. The exact $17 \times 17$ "four-rooms" map used in the HAC paper, with the goal / initial state sampling strategy defined above (labelled *HAC sampling / small "four-rooms"* in Figure 16).
2. The same $17 \times 17$ maze map, but with uniformly sampled goals while keeping the starting state fixed (labelled *Uniform goals / small "four-rooms"* in Figure 16).
3. A larger $41 \times 41$ "four-rooms" map, which is the one used for the reported experiments of RGL, with the HAC goal / initial state sampling strategy (labelled *HAC sampling / large "four-rooms"* in Figure 16). This map features slightly narrower passages between each room (proportionally to the size of the room). Actions remain the same: the ant is not scaled up. Goals are sampled uniformly. HAC's states and goals are scaled to the size of the map.
4. The same $41 \times 41$ "four-rooms" map with a fixed starting state and uniform goal sampling (labelled *Uniform goals / large "four-rooms"* in Figure 16).

The evaluation accuracy of each agent reported in Figure 16 follows the agent's goal / initial state sampling procedure than during training. Specifically, agents that were trained with the HAC sampling strategy are evaluated by the proportion of reached goals when goals and initial states are drawn according to HAC's sampling strategy. Similarly, agents that were trained with a fixed starting state are evaluated on the same setting. Consequently, the only fair comparison with the RGL results in the experimental section is when the starting state is fixed and the goals are uniformly sampled. Recall that RGL reaches an accuracy of 89% on the large "four-

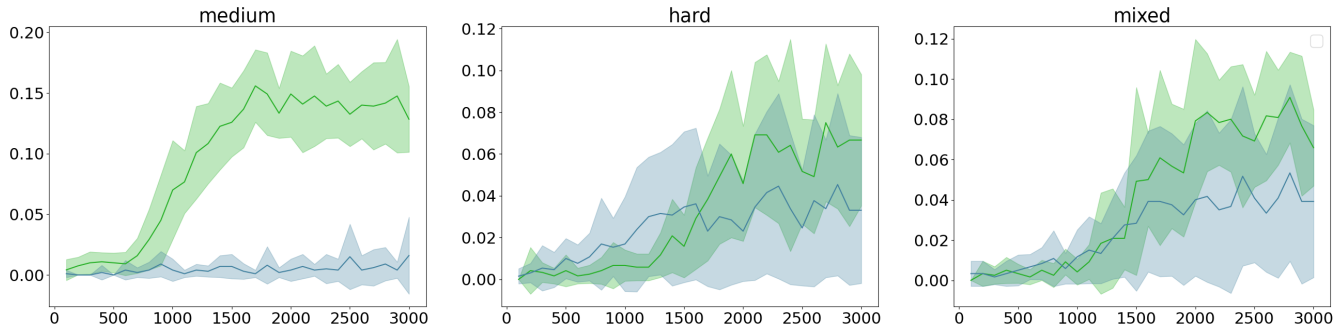



Figure 15: HAC average accuracy on the "medium", "hard", and "mixed" mazes for the ant-maze task, versus number of episodes (episode length is capped at 700 time steps but can be smaller if the goal is reached before). Green curve: uniform initial state sampling. Blue curve: fixed initial state.

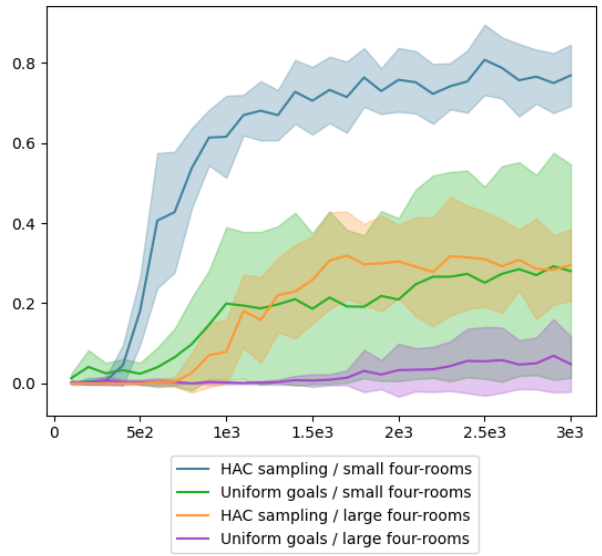



Figure 16: HAC average accuracy on variations of the "four-rooms" ant-maze task, versus number of episodes (episode length is capped at 700 time steps but can be smaller if the goal is reached before).

rooms" environment with fixed initial state and uniform goal sampling (Figure 3).

It appears that the goal / initial state sampling strategy is a crucial feature of HAC in ant-maze. Removing this feature, and sampling goals uniformly, reduces the accuracy of HAC's optimized policy from 76% to 28% in the small "four-rooms" environment, and from 29% to 5% in the large one.

It also appears HAC is rather sensitive to the scale of the map (despite appropriate state scaling in the inputs of the neural networks): even with the HAC initial state / goal sampling strategy, the accuracy of the optimized policy does not exceed 28% (versus the 89% of RGL). More steps are required to cross a room between passages and we hypothesize HAC suffers from this difficulty to span long trajectories between goals and hence struggles to reach a good accuracy in larger mazes.

Note that the HAC sampling strategy is tailor-made for the "four-rooms" maze and is undefined for other mazes, so the comparison above cannot be reproduced for the "medium", "hard", and "mixed" mazes. Instead (and this goes beyond what was proposed by the HAC authors), in an attempt to have a comparison baseline, we replaced this HAC sampling strategy by uniform sampling of both the initial state and the goal in these three mazes. We also evaluated the fixed initial state / uniform goal sampling setting. Results, shown in Figure 15, on other maps are similar to those of Figure 16: HAC reaches very small accuracy levels compared to RGL, even with the diversity of initial states and goals induced by uniform sampling. For this reason, we chose not to include these results in the empirical evaluation section.

## References


Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight experience replay. *Advances in neural information processing systems*, 30.

Aubret, A.; Hassas, S.; et al. 2021. DisTop: Discovering a Topological representation to learn diverse and rewarding skills. *arXiv preprint arXiv:2106.03853*.

Badia, A. P.; Sprechmann, P.; Vitvitskyi, A.; Guo, D.; Piot, B.; Kapturowski, S.; Tieleman, O.; Arjovsky, M.; Pritzel, A.; Bolt, A.; et al. 2019. Never Give Up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations*.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 13(5): 834–846.

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

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *International conference on Machine Learning*, 41–48.

Brafman, R. I.; and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3: 213–231.

Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2019. Exploration by random network distillation. In *International Conference on Learning Representations*.

Chaplot, D. S.; Salakhutdinov, R.; Gupta, A.; and Gupta, S. 2020. Neural topological slam for visual navigation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12875–12884.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271.

Domingues, O. D.; Tallec, C.; Munos, R.; and Valko, M. 2021. Density-Based Bonuses on Learned Representations for Reward-Free Exploration in Deep Reinforcement Learning. In *ICML Workshop on Unsupervised Reinforcement Learning*.

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

Emmons, S.; Jain, A.; Laskin, M.; Kurutach, T.; Abbeel, P.; and Pathak, D. 2020. Sparse graphical memory for robust planning. *Advances in Neural Information Processing Systems*, 33: 5251–5262.

Eysenbach, B.; Salakhutdinov, R.; and Levine, S. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. *arXiv preprint arXiv:1906.05253*.

Forestier, S.; Portelas, R.; Mollard, Y.; and Oudeyer, P.-Y. 2022. Intrinsically motivated goal exploration processes with automatic curriculum learning. *The Journal of Machine Learning Research*, 23(1): 6818–6858.

Ha, D.; and Schmidhuber, J. 2018. World models. *arXiv preprint arXiv:1803.10122*.

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

Hafner, D.; Lillicrap, T.; Ba, J.; and Norouzi, M. 2020. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*.

Hafner, D.; Lillicrap, T. P.; Norouzi, M.; and Ba, J. 2021. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*.

Kaelbling, L. P. 1993. Learning to achieve goals. In *IJCAI*, volume 2, 1094–8. Citeseer.

Kimura, H.; Yamashita, T.; and Kobayashi, S. 2002. Reinforcement learning of walking behavior for a four-legged robot. *IEEJ Transactions on Electronics, Information and Systems*, 122(3): 330–337.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*.

Konidaris, G.; and Barto, A. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29.

Levy, A.; Konidaris, G.; Platt, R.; and Saenko, K. 2019. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*.

McClinton, W.; Levy, A.; and Konidaris, G. 2021. HAC Explore: Accelerating Exploration with Hierarchical Reinforcement Learning. *arXiv preprint arXiv:2108.05872*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Nachum, O.; Gu, S. S.; Lee, H.; and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31.

Nasiriany, S.; Pong, V. H.; Lin, S.; and Levine, S. 2019. Planning with goal-conditioned policies. *arXiv preprint arXiv:1911.08453*.

Parascandolo, G.; Buesing, L.; Merel, J.; Hasenclever, L.; Aslanides, J.; Hamrick, J. B.; Heess, N.; Neitz, A.; and Weber, T. 2020. Divide-and-conquer monte carlo tree search for goal-directed planning. *arXiv preprint arXiv:2004.11410*.

Péré, A.; Forestier, S.; Sigaud, O.; and Oudeyer, P.-Y. 2018. Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration. In *International Conference on Learning Representations*.

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

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Randløv, J.; and Alstrøm, P. 1998. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *ICML*, volume 98, 463–471. Citeseer.

Ruan, X.; Li, P.; Zhu, X.; and Liu, P. 2022. A target-driven visual navigation method based on intrinsic motivation exploration and space topological cognition. *Scientific Reports*, 12(1): 1–22.

Savinov, N.; Dosovitskiy, A.; and Koltun, V. 2018. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–1320. PMLR.

Silver, D.; Hasselt, H.; Hessel, M.; Schaul, T.; Guez, A.; Harley, T.; Dulac-Arnold, G.; Reichert, D.; Rabinowitz, N.; Barreto, A.; et al. 2017. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 3191–3199. PMLR.

Strehl, A. L.; Li, L.; Wiewiora, E.; Langford, J.; and Littman, M. L. 2006. PAC model-free reinforcement learning. In *International Conference on Machine learning*, 881–888.

Sutton, R. S. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4): 160–163.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 5026–5033. IEEE.