LEARNING LATENT LANDMARKS FOR GENERALIZABLE PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Planning - the ability to analyze the structure of a problem in the large and decompose it into interrelated subproblems - is a hallmark of human intelligence. While deep reinforcement learning (RL) has shown great promise for solving relatively straightforward control tasks, it remains an open problem how to best incorporate planning into existing deep RL paradigms to handle increasingly complex environments. This problem seems difficult because planning requires an agent to reason over temporally extended periods. In principle, however, planning can take advantage of the higher-level structure of a complex problem - freeing the lower-level policy to focus on learning simple behaviors. In this work, we leverage the graph structure inherent to MDPs to address the problem of planning in RL. Rather than learning a graph over a collection of visited states, we learn latent landmarks that are scattered - in terms of reachability - across the goal space to provide state and temporal abstraction. On a variety of high-dimensional continuous control tasks, we demonstrate that our method, L^3P, outperforms prior work, and is oftentimes the only method capable of leveraging both the robustness of model-free RL and generalization of graph-search algorithms. We believe our work is an important step towards scalable planning in the reinforcement learning setting.

1 INTRODUCTION

An intelligent agent should be able to solve difficult problems by breaking them down into sequences of simpler problems. Classically, planning algorithms have been the tool of choice for endowing AI agents with the ability to reason over complex long-horizon problems (Doran & Michie, 1966; Hart et al., 1968). Recent years have seen an uptick in monographs examining the intersection of classical planning techniques – which excel at temporal abstraction – with deep reinforcement learning (RL) algorithms – which excel at state abstraction. Perhaps the ripest fruit born of this relationship is the AlphaGo algorithm, wherein a model free policy is combined with an MCTS (Coulom, 2006) planning algorithm to achieve superhuman performance on the game of Go (Silver et al., 2016a).

In the field of robotics, progress on combining planning and reinforcement learning has been somewhat less rapid, although still resolute. Indeed, the laws of physics in the real world are infinitely more complex than the simple rules of Go. Unlike board games such as chess and Go, which have deterministic and known dynamics and discrete action space, robots have to deal with a probabilistic and unpredictable world, and the action space for robots is oftentimes continuous. As a result, planning in robotics presents a much harder problem. One general class of methods (Sutton, 1991) seeks to combine model-based planning and deep RL. These methods can be thought of as an extension of model-predictive control (MPC) algorithms, with the key difference being that the agent is trained over hypothetical experience in addition to the actually collected experience. The primary shortcoming of this class of methods is that, like MCTS in AlphaGo, they resort to planning with action sequences - forcing the robot to plan for each action at each millisecond. Planning on the level of action sequences is fundamentally bottlenecked by the accuracy of the learned dynamics model and the horizon of a task, which limits the scalability of this type of approach. Another general class of methods, Hierarchical RL (HRL), introduces a higher-level learner to address the problem of planning (Dayan & Hinton, 1993; Vezhnevets et al., 2017; Nachum et al., 2018). In this scenario, a goal-based RL agent serves as the worker, and a manager learns what sequences of goals it must set for the worker to achieve a complex task. While this is apparently a sound solution to the problem of planning, having purely parametric hierarchical learners fails to fully take advantage of

the compositionality in the task space, and ultimately only relies on parametric functions to provide generalization power.

On the problem of combining classical planning and reinforcement learning, we choose in this paper to borrow an idea from the navigation literature – the idea of landmarks (Wang et al., 2008). Landmarks are essentially states that an agent can navigate between in order to complete tasks. However, rather than simply using previously seen states as landmarks, as is traditionally done, we will instead develop a novel algorithm to learn the landmarks used for planning. Our key insight is that by mapping previously achieved goals into a latent space that captures the temporal distance between goals, we can perform clustering in the latent space to group together goals that are easily reachable from one another. Subsequently, we can then decode the latent centroids to obtain a set of goals scattered (in terms of reachability) across the goal space. Since our learned landmarks are obtained from latent clustering, we call them *latent landmarks*. The chief algorithmic contribution of this paper is a new method for planning over learned latent landmarks for high-dimensional continuous control domains, which we name Learning Latent Landmarks for Planning (L^3P).

The L^3P algorithm ultimately uses a form of graph search over learned latent landmarks to execute planning. A low-level parametric policy is trained to navigate between these latent landmarks. The idea of reducing planning in RL to a graph search problem has enjoyed some attention recently (Savinov et al., 2018a; Eysenbach et al., 2019; Huang et al., 2019; Liu et al., 2019; Yang et al., 2020; Laskin et al., 2020). A key difference between those works and L^3P is that our use of latent landmarks allows us to substantially reduce the size of the search space. What's more, we use both soft value iterations on latent landmarks and a novel online planning algorithm to improve the robustness and sample efficiency of our method. As a result of those decisions, our algorithm is able to achieve superior performance on a variety of robotics domains involving both navigation and manipulation. In addition to the results presented in Section 5.4, videos of our algorithm's performance, and an analysis of the sub-tasks discovered by the latent landmarks, may be found at `https://sites.google.com/view/latent-landmarks/`.

2 Related Works

The problem of learning landmarks to aid in robotics problems has a long and rich history (Gillner & Mallot, 1998; Wang & Spelke, 2002; Wang et al., 2008). Prior art has been deeply rooted in the classical planning literature. For example, traditional methods would utilize Dijkstra et al. (1959) to plan over generated waypoints, SLAM (Durrant-Whyte & Bailey, 2006) to simultaneously integrate mapping, or the RRT algorithm (LaValle, 1998) for explicit path planning. The A* algorithm (Hart et al., 1968) further improved the computational efficiency of Dijkstra. Those types of methods often heavily rely on a hand-crafted configuration space that provides prior knowledge.

Planning is intimately related to the problem of model-based RL (MBRL), as the core ideas underlying learned models and planners can enjoy considerable overlap. Perhaps the most clear instance of this overlap is Model Predictive Control (MPC), and the related Dyna algorithm (Sutton, 1991). When combined with modern techniques (Kurutach et al., 2018; Luo et al., 2018; Nagabandi et al., 2018), MBRL is able to achieve some level of success, but as discussed in the introduction, planning on action sequences will fundamentally struggle to scale.

Our method will make extensive use of a parametric goal-based RL agent to accomplish low-level navigation between states. This area has seen rapid progress recently, largely stemming from the success of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017). Several improvements to HER augment the goal relabeling and sampling strategies to improve performance (Nair et al., 2018; Pong et al., 2018; 2019; Zhao et al., 2019; Pitis et al., 2020). There have also been attempts at incorporating search as inductive biases within the value function (Silver et al., 2016b; Tamar et al., 2016; Farquhar et al., 2017; Racanière et al., 2017; Lee et al., 2018; Srinivas et al., 2018). The focus of this line of work is to improve the low-level policy and is thus orthogonal to our work.

Recent work in Hierarchical RL (HRL) builds upon goal-based RL by learning a high-level parametric manager that feeds goals to the low-level goal-based agent (Dayan & Hinton, 1993; Vezhnevets et al., 2017; Nachum et al., 2018). This can be viewed as a parametric alternative to classical planning, as discussed in the introduction. Recently, Jurgenson et al. (2020); Pertsch et al. (2020) have derived HRL methods that are intimately tied to tree search algorithms. These papers are

further connected to a recent trend in the literature wherein classical search methods are combined with parametric control (Savinov et al., 2018a; Eysenbach et al., 2019; Huang et al., 2019; Liu et al., 2019; Yang et al., 2020; Laskin et al., 2020). Several of these articles will be discussed throughout this paper. LEAP (Nasiriany et al., 2019) also considers the problem of proposing sub-goals for a goal-conditioned agent: it uses a VAE (Kingma & Welling, 2013) and does CEM on the prior distribution to form the landmarks. Our method constrains the latent space with temporal reachability between goals, a concept previously explored in Savinov et al. (2018b), and uses latent clustering and graph search rather than sampling-based methods to learn and propose sub-goals.

3 BACKGROUND

We consider the problem of Multi-Goal RL under a Markov Decision Process (MDP) that is parameterized by $(S, A, \mathbb{P}, G, \Psi, R, \rho_0)$. S and A are the state and action space. The probability distribution of the initial states is given by $\rho_0(s)$, and $\mathbb{P}(s'|s,a)$ is the transition probability. $\Psi : S \mapsto G$ is a surjective mapping from the state space to the goal space, which assumes that *every* state s can be mapped to a corresponding *achieved* goal g. The reward function R can be defined as $R(s, a, s', g) = -\mathbb{1}\{\Psi(s') \neq g\}$. We further assume that each episode has a fixed horizon T.

The goal-conditioned policy is a probability distribution $\pi : S \times G \times A \to \mathbb{R}^+$. The policy gives rise to trajectory samples with the form $\tau = \{s_0, a_0, g, s_1, \cdots s_T\}$. In this paper, we assume a goal is specified to the agent at the start of each episode, sampled from a static goal proposal distribution $p_g : G \to \mathbb{R}^+$. To test the ability for an agent to generalize, we might choose a goal distribution $p'_g : G \to \mathbb{R}^+$ during test time that is different from p_g. The purpose of the policy π during training is to learn how to reach the goals drawn from p_g, which means maximizing the cumulative rewards for the sampled trajectories conditioned on a sampled goal. To make learning easier, it is common to introduce a discount factor $\gamma \in (0, 1)$, so the final objective of the policy is to maximize $\mathcal{J}(\pi) = \mathbb{E}_{g\sim p_g, \tau\sim\pi(g)}[\sum_{t=0}^{T-1} \gamma^t \cdot R(s_t, a_t, s_{t+1}, g)]$.

While the objective above can be optimized with policy gradient methods, it is often more sample efficient to apply Q-learning, which is able to utilize off-policy data stored in a replay buffer B. A goal-conditioned Q-function takes in a state, an action, and a goal as inputs, and estimates the reward-to-go under the current policy π conditioned upon the given goal, after taking the given action. Q-learning solves this approximate dynamic programming problem by minimizing the Bellman Error. An additional technique, called Hindsight Experience Replay, or HER (Andrychowicz et al., 2017), uses the idea of hindsight relabelling to drastically speed up training. This relabeling crucially relies upon the mapping $\Psi : S \mapsto G$ in the multi-goal MDP setting. Using the converter, we can write the the joint objective of multi-goal Q-learning with HER as minimizing:

$$\min_Q \mathbb{E}_{\substack{\tau\sim B, t\sim\{0\cdots T-1\} \\ (s_t,a_t,s_{t+1})\sim\tau \\ k\sim\{t+1\cdots T\}, g=\Psi(s_k) \\ a'\sim\pi(\cdot|s_{t+1},g)}} \left(Q(s_t, a_t, g) - \Big(R(s_t, a_t, s_{t+1}, g) + \gamma \cdot Q(s_{t+1}, a', g) \Big) \right)^2 \tag{1}$$

4 THE L^3P ALGORITHM

Our overall objective in this section is to derive an algorithm that learns a small number of landmarks scattered across goal space in terms of reachability and use those learned landmarks for planning. There are three chief difficulties we must overcome when considering such an algorithm. First, how can we group together goals that are easily reachable from one another? The answer is to embed goals into a latent space, where the latent representation captures some notion of temporal distance between goals – in the sense that goals that would take many timesteps to navigate in between are further apart in latent space. Second, we need to find a way to learn a sparse set of landmarks used for planning. Our method performs clustering on the constrained latent space, and decodes the learned centroids as the landmarks we seek. Finally, we need to develop a non-parametric planning algorithm responsible for selecting sequences of landmarks the agent must traverse to accomplish its high-level goal. The proposed online planning algorithm is simple, scalable, and robust.

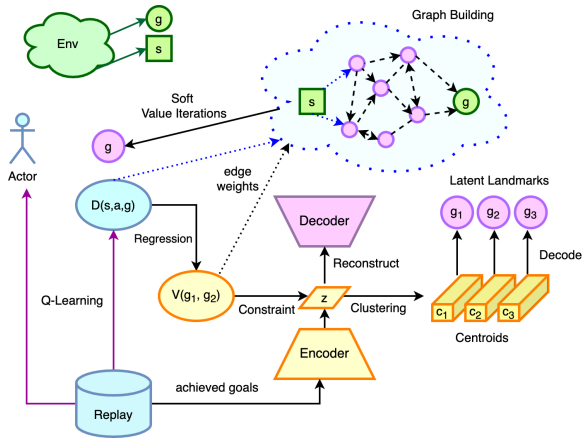

Figure 1: An overview of L^3P, which learns a small number of latent landmarks in the goal space for planning. The main components of our method are: learning distances (via Q-learning and regression), learning a latent space (via an auto-encoder with constraints on the latent space), learning latent landmarks (via clustering in the latent space), and planning with latent landmarks (via soft value iterations over a graph of learned latent landmarks).

4.1 Learning a Latent Space

Let us consider the following question: "How should we go about learning a latent space of goals where the metric reflects reachability?" Suppose we have an auto-encoder (AE) in the agent's goal space, with deterministic encoder f_E and decoder f_D. As usual, the reconstruction loss is given by $\mathcal{L}_{rec}(g) = \left\| f_D\big(f_E(g)\big) - g \right\|_2^2$. We want to make sure that the distance between two latent codes would roughly correspond to the number of steps it would take the policy to go from one goal to another. Concretely, for any pair of goals (g_1, g_2), we optimize the following loss:

$$\mathcal{L}_{latent}(g_1, g_2) = \left(\left\| f_E(g_1) - f_E(g_2) \right\|_2^2 - \frac{1}{2}\Big(V(g_1, g_2) + V(g_2, g_1) \Big) \right)^2 \tag{2}$$

Where $V : G \times G \to \mathbb{R}^+$ is a mapping that estimates how many steps it would take the policy π to go from one goal to another goal on average. By adding this constraint, the encoding-decoding mapping can no longer be arbitrary, giving more structure to the latent space. Goals that are close by in terms of reachability will be naturally clustered in the latent space, and interpolations between latent codes will lead to meaningful results.

Of course, the constraint in Equation 2 is quite meaningless if we do not have a way to estimate the mapping V. We will proceed towards this objective by noting the following interesting connection between multi-goal Q-functions and reachability. In the multi-goal RL framework considered in the background section, the reward is binary in nature. The agent receives a reward of -1 until it reaches the goal, and then 0 when it reaches the desired goal. In this setting, the Q-function is implicitly estimating *the number of steps steps* it takes to reach the goal g from the current state s *after* the action a is taken. Denote this quantity as $D(s, a, g)$, the Q-function can be re-written as:

$$Q(s, a, g) = \sum_{t=0}^{D(s,a,g)-1} \gamma^t \cdot (-1) + \sum_{t=D(s,a,g)}^{T-1} \gamma^t \cdot 0 = -\frac{1 - \gamma^{D(s,a,g)}}{1 - \gamma} \tag{3}$$

Choosing to parameterize Q-functions in this way disentangles the effect of γ on multi-goal Q-learning. It also provides us with access the direct distance estimation function $D(s, a, g)$. We note that this *distance* is not a mathematical distance in the sense of a metric. Instead, we use the word *distance* to refer to the number of steps the policy π needs to take in the environment.

Given our tractable estimate of D, it is now a straightforward matter to estimate the desired quantity V, which approximates how many steps it takes the policy to transition between goals. To get the

desired estimate, we regress V towards D as follows

$$\min_{V} \mathbb{E}_{\substack{\tau \sim B, t \sim \{0 \cdots T-1\} \\ (s_t, a_t, s_{t+1}) \sim \tau \\ k \sim \{t+1 \cdots T\}}} \Big(D\big(s_t, a_t, \Psi(s_k)\big) - V\big(\Psi(s_{t+1}), \Psi(s_k)\big) \Big)^2 \tag{4}$$

One crucial detail is the use of $\Psi(s_{t+1})$ rather than $\Psi(s_t)$ in the inputs to V. This is due to the fact that $D : S \times A \times G \to \mathbb{R}$ outputs the number of steps to go *after* an action is taken, when the state has transitioned into s_{t+1}. The objective above provides an unbiased estimate of the average number of steps between two goals.

The estimates D and V will prove useful beyond helping to optimize the auto-encoder in Equation 2. They will prove essential in weighting and planning over latent landmark nodes in Section 4.3.

4.2 Learning Latent Landmarks

Planning on a graph can be expensive, as the number of edges can grow quadratically with the number of nodes. To battle this issue in scalability, we use the constrained latent space to learn a sparse set of landmarks. A landmark can be thought of as a waypoint that the agent can pass through enroute to achieve a desired goal. Ideally, *goals that are easily reachable from one another should be grouped to form one single landmark*. Since our latent representation captures the temporal reachability between goals, this can be achieved by doing clustering in the latent space. The cluster centroids, when decoded from the decoder, will be precisely the latent landmarks we are seeking.

Clustering proceeds as follows. For N clusters to be learned, we define a mixture of Gaussians in the latent space with N trainable latent centroids, $\{\mathbf{c}_1 \cdots \mathbf{c}_N\}$, and a shared trainable variance vector $\boldsymbol{\sigma}$. We maximize the evidence lower bound (ELBO) with a uniform prior $p(\mathbf{c})$:

$$\log p\Big(z = f_E(g)\Big) \geq \mathbb{E}_{q(\mathbf{c}|z)}\Big[\log p(z \mid \mathbf{c})\Big] - D_{KL}\Big(q(\mathbf{c} \mid z) \parallel p(\mathbf{c})\Big) \tag{5}$$

Ideally, we would like each batch of data given to the latent clustering model to be representative of the whole replay buffer, such that the centroids will quickly learn to scatter out. To this end, we propose to use the Greedy Latent Sparsification (GLS) algorithm (Algorithm 1 in the Appendix) on each batch of data sampled from the replay before taking a gradient step with the batch. GLS is inspired by kmeans++ (Arthur & Vassilvitskii, 2007), with several key differences: this sparsification process is used for both training and initialization, it uses a neural metric for determining the distance between data points, and that it is compatible with mini-batch-style gradient-based training.

4.3 Planning with Latent Landmarks

Having derived a latent encoding algorithm and an algorithm for learning latent landmarks, we at last turn our attention to planning. The core idea here is to construct a weighting matrix that at first provides a raw distance estimate between latent landmarks, and then use soft value iterations to *iteratively refine distance estimations*, before deciding on which landmark to use as the next sub-goal.

We begin by decoding the learned centroids in the latent space into the nodes in the graph $\{f_D(\mathbf{c}_1) \cdots f_D(\mathbf{c}_N)\}$. We first build a densely-connected directed graph, where we add two edges directed in reverse orders for every pair of nodes in the graph. For instance, for an edge going from $f_D(\mathbf{c}_i)$ to $f_D(\mathbf{c}_j)$, the weight on that edge is $-V(f_D(\mathbf{c}_i), f_D(\mathbf{c}_j))$. At the start of an episode, the agent receives a goal g, and we construct the following matrix of size $(N+1) \times (N+1)$:

$$W = \begin{pmatrix} 0 & \dots & -V(f_D(\mathbf{c}_1), f_D(\mathbf{c}_N)) & -V(f_D(\mathbf{c}_1), g) \\ \vdots & \ddots & \vdots & \vdots \\ -V(f_D(\mathbf{c}_N), f_D(\mathbf{c}_1)) & \dots & 0 & -V(f_D(\mathbf{c}_N), g) \\ -\infty & \dots & -\infty & 0 \end{pmatrix} \tag{6}$$

The entry $w_{i,j}$ in W estimates the negative of the distance it would take to go from the i-th landmark to the j-th landmark. The goal of an episode is temporally included in the list of landmarks. However, if the value estimate is too large, our method would not trust the vanilla value estimate and perform an element-wise operation $W \leftarrow W - \texttt{penalty} \cdot \mathbb{1}\big(W < -\texttt{dist_max}\big)$, such that large distances

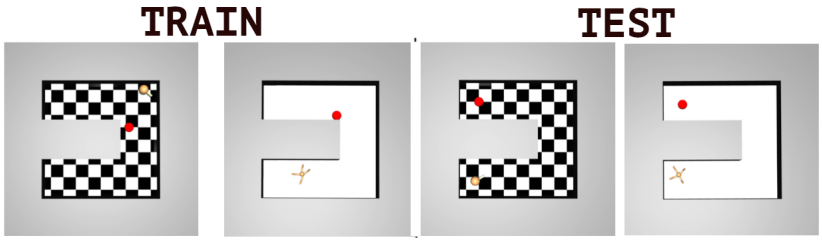

Figure 2: For both Point and Ant, during training, the initialization state distribution and the goal proposal distribution are *uniform* around the maze. During test time, the agent is asked to traverse the longest path in the maze, as shown in the figure. The success rate on the test environment is reported in Figure 3. This environment demonstrates $L3P$'s ability to generalize to longer horizon goals during test time.

in the matrix are penalized to have an even larger negative value. We then apply *soft value iterations* on the graph. For each entry $w_{i,j}$ in the matrix, a soft value iteration does the following operation:

$$w_{i,j} \leftarrow \sum_{k=1}^{N+1} \frac{\exp \frac{1}{\beta}(w_{i,k} + w_{k,j})}{\sum_{k'=1}^{N+1} \exp \frac{1}{\beta}(w_{i,k'} + w_{k',j})} \Big(w_{i,k} + w_{k,j}\Big) \tag{7}$$

where β is the temperature of the value iteration process. Our method repeatedly applies soft value iterations as in equation 7 on the constructed matrix W for S times, in order to offer reliable estimates of the distances between any two landmarks (with the goal included as well). Equation 7 can be implemented in a vectorized (and thus highly optimized) way. We then slice off the last column of W, which provides the estimated number of steps it would take the policy π to go from the learned landmarks to the goal. We denote this vector as $\boldsymbol{d}_{\boldsymbol{c}\to g}$.

For online planning, when the agent receives a goal at the start of an episode, we use the above process to solve for $\boldsymbol{d}_{\boldsymbol{c}\to g}$ (which is fixed throughout an episode). Each time the agent receives an observation state s, the algorithm calculates $\boldsymbol{d}_{s\to\boldsymbol{c}}$ in the following way:

$$\boldsymbol{d}_{s\to\boldsymbol{c}} = \begin{pmatrix} -D\big(s, \pi(s, f_D(\mathbf{c}_1)), f_D(\mathbf{c}_1)\big) \\ \vdots \\ -D\big(s, \pi(s, f_D(\mathbf{c}_N)), f_D(\mathbf{c}_N)\big) \\ -D\big(s, \pi(s, g), g\big) \end{pmatrix} \tag{8}$$

And the chosen landmark would be $\texttt{subgoal} \leftarrow \arg\max(\boldsymbol{d}_{s\to\boldsymbol{c}} + \boldsymbol{d}_{\boldsymbol{c}\to g})$. To further provide *temporal abstraction*, the agent will be asked to pursue $\texttt{subgoal}$ for $-\boldsymbol{d}_{s\to\boldsymbol{c}}[\texttt{subgoal}]$ number of steps, the agent's own estimate of *how many steps it thinks it will need*. After this number of steps, the agent will decide on the next landmark to pursue by re-calculating Equation 8, but the *immediate* previous landmark will not be considered as a candidate landmark. We have found that this simple algorithm helps the agent to avoid getting stuck and improves the overall robustness of the agent. The algorithm for online planning is summarized in Algorithm 2 in the Appendix.

5 Experiments and Evaluation

We investigate the impact of L^3P in a variety of robotic manipulation and navigation environments. These include standard benchmarks such as Fetch-PickAndPlace, and more difficult environments such as AntMaze-Hard and Place-Inside-Box-Hard that have been engineered to have especially demanding test-time generalization requirements. Videos of our algorithm in action are available here: `https://sites.google.com/view/latent-landmarks/`.

5.1 Baselines

We compare our method with a variety of baselines. HER (Andrychowicz et al., 2017) is a model-free RL algorithm. SORB (Eysenbach et al., 2019) is a method that combines RL and graph search by

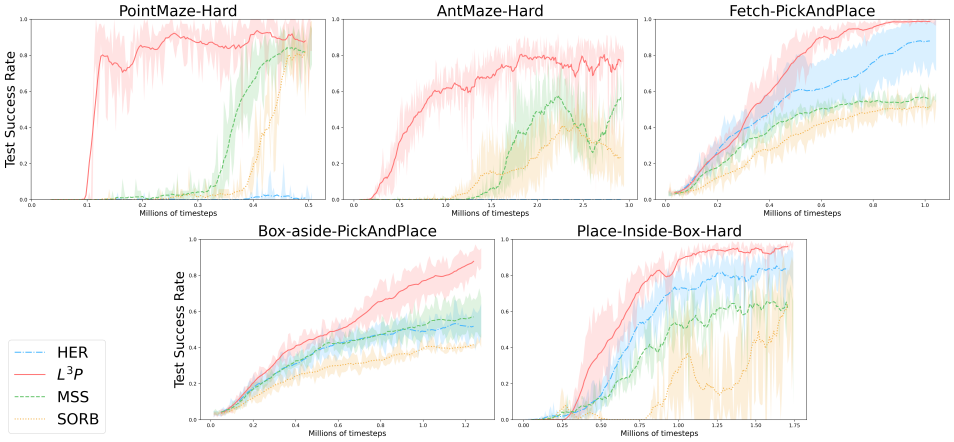

Figure 3: Test time success rate vs. total number of timesteps, on maze and robotic manipulation environments. During test time, new more difficult goals are selected. L^3P shows more robust generalization much more quickly than other methods. For every environment except PointmMaze, L^3P is the only algorithm that consistently solves the task.

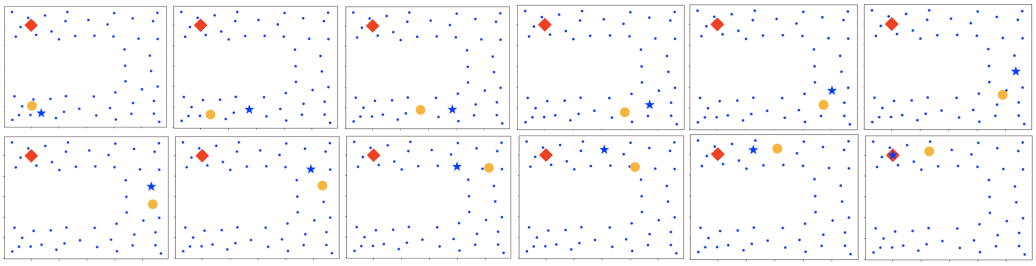

Figure 4: Visualizing planning on AntMaze at test time. Read images from upper left to bottom right. The blue dots are the learned latent landmarks decoded from the latent centroids. The orange dot represents the ant's present location in the maze. The red dot is the final goal that the agent needs to reach. At each step, the blue star indicates the landmark chosen by our planning algorithm. Whereas MSS and SORB sample 400 and hundreds of thousands of landmarks (respectively), our method obtains a lean graph that only contain 50 landmarks. L^3P is the only method capable of achieving over 80% test success rate on AntMaze-Hard within 3M timesteps.

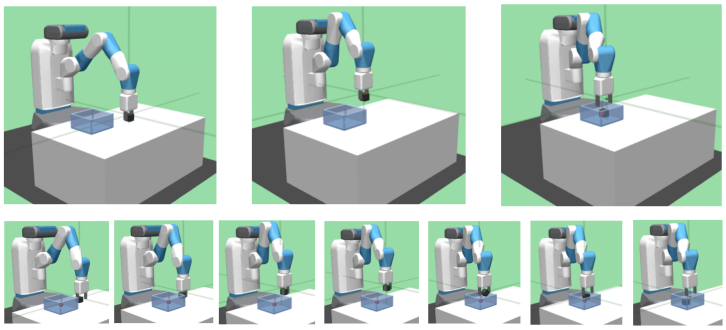

Figure 5: We consider two environments involving a fetch robot, a block, and a box. In Box-aside-PickAndPlace, the fetch must learn to pick and place the block while avoiding collision with the box. In Place-Inside-Box-Hard, the fetch must pick the block and place it inside the box. We visualize the fetch states corresponding to learned landmarks in the second row of images. Note the agent has learned landmarks that help avoid collision with the box.

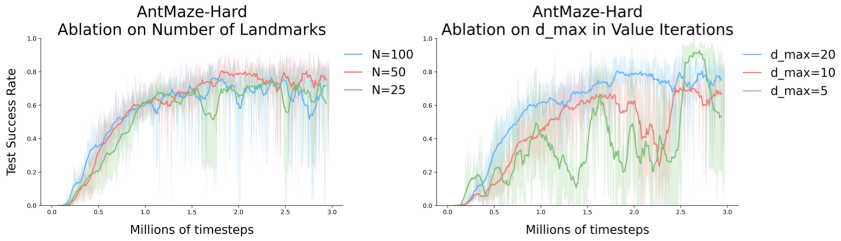

Figure 6: Ablation Studies on the AntMaze-Hard environment. Test-time success rate vs millions of timesteps. Left Figure shows L^3P is robust against a decreasing number of landmarks. Figure right shows that picking the correct maximum distance bound for the soft value iteration process is important, as described in Sec.4.1

using the entire replay buffer. Mapping State Space (MSS Huang et al. 2019) reduces the number of vertices by sub-sampling the replay buffer. L^3P, SORB, and MSS all use the same hindsight relabelling strategy proposed in HER. All of the domains are continuous control tasks, so we adopt DDPG (Lillicrap et al., 2015) as the learning algorithm for the low-level actor.

5.2 Generalization to Longer Horizons

The PointMaze-Hard and AntMaze-Hard environments introduced in Figure 4 are designed to test an agent's ability to generalize to longer horizons. We use a short, 200-timestep time horizon during training and a ρ_0 that is uniform in the maze. At test time, we always initialize the agent on one end of the maze, and set the goal on the other end. The horizon of the test environment is 500 steps. In Figure 3, we see L^3P is the only algorithm capable of solving AntMaze-Hard consistently.

We observe an interesting trend where the success rates for other graph search methods dramatically crash and then slowly recover after making some initial progress. We postulate this occurs because methods that are based on using the entire replay or sub-sampling the replay for landmark selection will always struggle as the buffer size increases. In contrast to these methods, L^3P does not exhibit such undesirable instability. The result convincingly shows that, at least on the navigation tasks considered, **relying solely on parametric generalization of function approximators is an order of magnitude less sample efficient than taking advantage of the problem's inherent graph structure**. Moreover, it suggests that building a graph on the learned latent landmarks is significantly more sample efficient and scalable than directly using or sub-sampling the replay buffer to build the graph.

5.3 Generalization to More Difficult Goals

In Figure 5, we introduce two pick and place tasks involving a box on a table. In particular, Place-Inside-Box-Hard is engineered to test generalization to more difficult goals. During training, the goal distribution has 80% regular pick-and-place goals, enabling the agent to first learn how to fetch in general. Meanwhile, only 20% of the goals are inside the box. During testing, we evaluate the ability of the agent to pick the object from the table and place it inside the box. Our method achieves dominant performance in both learning speed and test-time generalization.

5.4 Some Discussion on Model Parameters

We investigate L^3P's sensitivity to hyper-parameters via an ablation study AntMaze-Hard. We vary both the number of landmarks and the cut-off distance `dist_max` as described in Sec.4.1. Fig.6 shows that L^3P is robust against a decreasing number of landmarks. Whereas picking a suitable cut-off distance is still crutial, as it affects the stability during learning.

6 Closing Remarks

In this work, we introduce the L^3P algorithm, which learns a set of latent landmarks scattered across the goal space to enable scalable planning. We demonstrate that L^3P achieves significantly better sample efficiency, higher asymptotic performance, and better generalization on a range of challenging robotic navigation and manipulation tasks.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pp. 5048–5058, 2017.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

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

James E Doran and Donald Michie. Experiments with the graph traverser program. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 294(1437):235–259, 1966.

Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15246–15257, 2019.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. October 2017. URL `http://arxiv.org/abs/1710.11417`.

Sabine Gillner and Hanspeter A Mallot. Navigation and acquisition of spatial knowledge in a virtual maze. *Journal of cognitive neuroscience*, 10(4):445–463, 1998.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. In *Advances in Neural Information Processing Systems*, pp. 1942–1952, 2019.

Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. Sub-goal trees–a framework for goal-based reinforcement learning. *arXiv preprint arXiv:2002.12361*, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

Michael Laskin, Scott Emmons, Ajay Jain, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning. *arXiv preprint arXiv:2003.06417*, 2020.

Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric Xing, and Ruslan Salakhutdinov. Gated path planning networks. June 2018. URL `http://arxiv.org/abs/1806.06408`.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Kara Liu, Thanard Kurutach, Christine Kit-Ching Tung, Pieter Abbeel, and Aviv Tamar. Hallucinative topological memory for zero-shot visual planning. *ArXiv*, 2019. URL https://www.semanticscholar.org/paper/b8f0417844523d788630bb28a600dfeb74914e2d.

Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pp. 9191–9200, 2018.

Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems*, pp. 14843–14854, 2019.

Karl Pertsch, Oleh Rybkin, Frederik Ebert, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Long-horizon visual planning with goal-conditioned hierarchical predictors. *arXiv preprint arXiv:2006.13205*, 2020.

Silviu Pitis, Harris Chan, Stephen Zhao, Bradly Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. *arXiv preprint arXiv:2007.02832*, 2020.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5690–5701. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7152-imagination-augmented-agents-for-deep-reinforcement-learning.pdf.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *arXiv preprint arXiv:1803.00653*, 2018a.

Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018b.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016a.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. December 2016b. URL http://arxiv.org/abs/1612.08810.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. April 2018. URL http://arxiv.org/abs/1804.00645.

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

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. February 2016. URL http://arxiv.org/abs/1602.02867.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Ranxiao Frances Wang and Elizabeth S Spelke. Human spatial representation: Insights from animals. *Trends in cognitive sciences*, 6(9):376–382, 2002.

Yang Wang, David Mulvaney, Ian Sillitoe, and Erick Swere. Robot navigation by waypoints. *Journal of Intelligent and Robotic Systems*, 52(2):175–207, 2008.

Ge Yang, Amy Zhang, Ari S. Morcos, Joelle Pineau, Pieter Abbeel, and Roberto Calandra. Plan2vec: Unsupervised representation learning by latent plans. In *Proceedings of The 2nd Annual Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pp. 1–12, 2020. arXiv:2005.03648.

Rui Zhao, Xudong Sun, and Volker Tresp. Maximum entropy-regularized multi-goal reinforcement learning. *arXiv preprint arXiv:1905.08786*, 2019.

A APPENDIX

Algorithm 1 Greedy Latent Sparsification (GLS) for Latent Cluster Training

Given: Replay Buffer B, Encoder f_E.
Initialize: `LatentEmbeds` = $\{\}$.

Sample K achieved goals from B.
Sample $k \sim \{0 \cdots K-1\}$.
$\texttt{dist} = [\|f_E(g_1) - f_E(g_k)\|_2^2, \cdots]$
for i = 1 to M **do** ▷ subsampling
  $k \leftarrow \arg\max \texttt{dist}[k]$
  Add $f_E(g_k)$ to `LatentEmbeds`.
  $\texttt{NEWdist} = [\|f_E(g_1) - f_E(g_k)\|_2^2, \cdots]$
  $\texttt{dist} = \min(\texttt{dist}, \texttt{NEWdist})$ ▷ element-wise operation
end for
Optimize equation 5 on `LatentEmbeds`.

Algorithm 2 Online Planning

Given: Environment `env`, initial state s, goal g.

`Cnt` = 0. `SubG` = `None`.
Solve for $\boldsymbol{d}_{\boldsymbol{c}\to g}$ with Eq 7 and Eq 8.
for t = 1 to T **do** ▷ One Episode
  if $\texttt{Cnt} \geq 1.0$ **then**
    $\texttt{Cnt} = \texttt{Cnt} - 1$
  else
    $\boldsymbol{d} \leftarrow \boldsymbol{d}_{s\to\boldsymbol{c}} + \boldsymbol{d}_{\boldsymbol{c}\to g}$
    if $\texttt{SubG} \neq \texttt{None}$ **then**
      $\boldsymbol{d}[\texttt{SubG}] \leftarrow -\infty$
    end if
    $\texttt{SubG}, \texttt{Cnt} \leftarrow \arg\max \boldsymbol{d}, -\max \boldsymbol{d}$
  end if
  $a \sim \pi(s, \texttt{SubG})$ $s \leftarrow \texttt{env.step(a)}$.
end for