Goal-Space Planning with Subgoal Models

Chunlok Lo[*]   CHUNLOK@UALBERTA.CA

Kevin Roice[*]   ROICE@UALBERTA.CA

Parham Mohammad Panahi[*]   PARHAM1@UALBERTA.CA

Scott M. Jordan   SJORDAN@UALBERTA.CA

Adam White[†]   AMW8@UALBERTA.CA

Gábor Mihucz   MIHUCZ@UALBERTA.CA

Farzane Aminmansour   AMINMANS@UALBERTA.CA

Martha White[†]   WHITEM@UALBERTA.CA

[†] *Canada CIFAR AI Chair*
Alberta Machine Intelligence Institute (Amii)
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada

Editor: Laurent Orseau

Abstract

This paper investigates a new approach to model-based reinforcement learning using background planning: mixing (approximate) dynamic programming updates and model-free updates, similar to the Dyna architecture. Background planning with learned models is often worse than model-free alternatives, such as Double DQN, even though the former uses significantly more memory and computation. The fundamental problem is that learned models can be inaccurate and often generate invalid states, especially when iterated many steps. In this paper, we avoid this limitation by constraining background planning to a given set of (abstract) subgoals and learning only local, subgoal-conditioned models. This goal-space planning (GSP) approach is more computationally efficient, naturally incorporates temporal abstraction for faster long-horizon planning, and avoids learning the transition dynamics entirely. We show that our GSP algorithm can propagate value from an abstract space in a manner that helps a variety of base learners learn significantly faster in different domains.

Keywords: Model-Based Reinforcement Learning, Temporal Abstraction, Planning

1. Introduction

Planning with learned models in reinforcement learning (RL) is important for sample efficiency. Planning provides a mechanism for the agent to generate hypothetical experience, in the background during interaction, to improve value estimates. This hypothetical experience provides a stand-in for the real-world; the agent can generate many experiences (transitions) in its head (via a model) and learn from those experiences. Dyna (Sutton, 1991) is a classic example of *background planning*. On each step, the agent generates several transitions

---

[*]. Equal contribution

according to its model, and updates with those transitions as if they were real experience. Background planning can also be used to adapt to non-stationarity, because continually updating the model and re-planning allows the agent to adapt to the current situation.

The promise of background planning is that we can to learn and adapt value estimates efficiently, but many open problems remain to make it more widely useful. These include that 1) long rollouts generated by one-step models can diverge or generate invalid states, 2) learning probabilities over outcome states can be complex, especially for high-dimensional tasks and 3) planning itself can be computationally expensive for large state spaces.

One way to overcome these issues is to construct an abstract model of the environment and plan at a higher level of abstraction. In this paper, we construct abstract MDPs using both state abstraction as well as temporal abstraction. State abstraction is achieved by simply grouping states. Temporal abstraction is achieved using *options*—a policy coupled with a termination condition and initiation set (Sutton et al., 1999). A temporally-abstract model based on options allows the agent to jump between abstract states potentially alleviating the need to generate long rollouts.

An abstract model can be used to directly compute a policy in the abstract MDP, but there are issues with this approach. This idea was explored with an algorithm called Landmark-based Approximate Value Iteration (LAVI) (Mann et al., 2015). Though planning can be shown to be provably more efficient, the resulting policy is suboptimal, restricted to going between landmark states. This suboptimality issue forces a trade-off between increasing the size of the abstract MDP (to increase the policy's expressivity) and increasing the computational cost to update the value function. In this paper, we investigate abstract model-based planning methods that have a small computational cost, can quickly propagate changes in value over the entire state space, and do not limit the optimality of learned policy.

An alternative strategy that we explore in this work is to use the policy computed from the abstract MDP to guide the learning process in solving the original MDP. More specifically, the purpose of the abstract MDP is to propagate value quickly over an abstract state space and then transfer that information to a value function estimate in the original MDP. This approach has two main benefits: 1) the abstract MDP can quickly propagate value with a small computational cost, and 2) the learned policy is not limited to the abstract value function's approximation.

Specifically, we introduce Goal-Space Planning (GSP), a new background planning formalism for the general online RL setting. The key novelty is designing the framework to leverage *subgoal-conditioned models*: temporally-extended models that condition on subgoals. These models output predictions of accumulated rewards and discounts for state-subgoal pairs, which can be estimated using standard value-function learning algorithms. The models are designed to be simple to learn, as they are only learned for states local to subgoals and they avoid generating entire next state vectors. We use background planning on transitions between a given set of subgoals, to quickly propagate (suboptimal) subgoal value estimates. We leverage these quickly computed subgoal values, without suffering from suboptimality, by incorporating them into any standard value-based algorithm via potential-based reward shaping. We layer GSP onto two different algorithms—Sarsa(λ) and Double Deep Q-Network (DDQN)—and show it improves both base learners. We prove that dynamic programming with our subgoal models is sound (Proposition 3) and highlight that using these subgoal values through potential-based shaping does not change the optimal policy.

We then investigate properties of GSP empirically, in a simplified setting where we assume subgoals are given to the agent and the subgoal models—which are actually UVFAs (Schaul et al., 2015)—are learned offline. Our goal is to understand the utility of this planning formalism, without simultaneously solving subgoal discovery and efficient off-policy UVFA learning. We show that 1) it propagates value and learns an optimal policy faster than its base learner, 2) can perform well with somewhat suboptimal subgoal selection, but can harm performance if subgoals are very poorly selected, 3) is quite robust to inaccuracy in its models and 4) that alternative potential-based rewards and alternatives ways to incorporate subgoal values are not as effective as the particular approach used in GSP. We conclude with a discussion on the large literature of related work and a discussion on the benefits of GSP over other background planning approaches, as well as limitations of this work.

2. Problem Formulation

We consider the standard reinforcement learning setting, where an agent learns to make decisions through interaction with an environment, formulated as a Markov Decision Process (MDP) represented by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} \rangle$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and the transition probability $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ describe the expected reward and probability of transitioning to a state, for a given state and action. On each discrete timestep t the agent selects an action A_t in state S_t, the environment transitions to a new state S_{t+1} and emits a scalar reward R_{t+1}.

The agent's objective is to find a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximizes expected *return*, the future discounted reward $G_t \doteq R_{t+1} + \gamma_{t+1} G_{t+1}$ across all states. The state-based discount $\gamma_{t+1} \in [0, 1]$ depends on S_{t+1} (Sutton et al., 2011), which allows us to specify termination. If S_{t+1} is a terminal state, then $\gamma_{t+1} = 0$; else, $\gamma_{t+1} = \gamma_c$ for some constant $\gamma_c \in [0, 1]$. The policy can be learned using algorithms like Sarsa(λ) (Sutton and Barto, 2018), which approximate the action-values: the expected return from a given state and action, $q(s, a) \doteq \mathbb{E}[G_t | S_t = s, A_t = a]$.

Most model-based reinforcement learning systems learn a state-to-state transition model. The transition dynamics model can be either an expectation model $\mathbb{E}[S_{t+1} | S_t, A_t]$ or a probabilistic model $P(S_{t+1} | S_t, A_t)$. If the state space or feature space is large, then the expected next state or distribution over it can be difficult to estimate, as has been repeatedly shown (Talvitie, 2017). Further, these errors can compound when iterating the model forward or backward (van Hasselt et al., 2019; Aminmansour et al., 2024). It is common to use an expectation model, but unless the environment is deterministic or we are only learning the values rather than action-values, this model can result in invalid states and detrimental updates (Wan et al., 2019). The goal in this work is to develop a model-based approach that avoids learning a state-to-state transition model, but still obtains the benefits of model-based learning for faster learning and adaptation.

3. Goal-Space Planning at a High-Level

We consider three desiderata for when a model-based approach should be effective. (1) The model should be feasible to learn: we can get it to a sufficient level of accuracy that makes it beneficial to plan with that model. (2) Planning should be computationally efficient, so
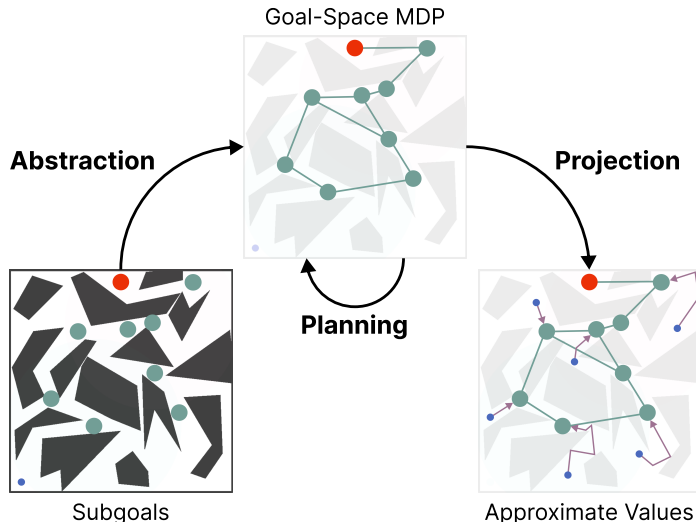
Figure 1: GSP in the PinBall domain. The agent begins with a set of subgoals (denoted in teal) and learns a set of subgoal-conditioned models. (**Abstraction**) Using these models, the agent forms an abstract MDP where the states are subgoals with options to reach each subgoal as actions. (**Planning**) The agent plans in this abstract MDP to quickly learn the values of these subgoals. (**Projection**) Using learned subgoal values, the agent obtains approximate values of states based on nearby subgoals and their values. These quickly updated approximate values are then used to speed up learning.

that the agent's values can be quickly updated. (3) Finally, the model should be modular—composed of several local models or those that model a small part of the space—so that the model can quickly adapt to small changes in environment. These small changes might still result in large changes in the value; planning can quickly propagate these small changes, potentially changing the value function significantly. In this section, we motivate how GSP provides these three benefits by introducing the key concepts in the algorithm; we dive into the formal details of GSP in Section 5.

At a high level, the GSP algorithm focuses planning over a set of given abstract subgoals to provide quickly updated approximate values to speed up learning. To do so, the agent first learns a set of *subgoal-conditioned models*, minimal models focused around planning utility. These models then form a temporally abstract goal-space MDP, with subgoals as states, and options to reach each subgoal as actions. Finally, the agent can update its policy based on these subgoal values to speed up learning. Figure 1 provides a visual overview of this process. We visualize this is an environment called PinBall, which we also use in our experiments. PinBall is a continuous state domain where the agent must navigate a ball through a set of obstacles to reach the main goal, with a four-dimensional state space consisting of $(x, y, \dot{x}, \dot{y})$ positions and velocities.

Abstraction: In Figure 1, the set of subgoals $\mathcal{G}$ are the teal dots, a finite space of 9 subgoals. The subgoals are abstract states, in that they correspond to many states: a subgoal is any (x, y) location in a small ball, at any velocity. In this example, the subgoals are randomly distributed across the space. Subgoal discovery—identifying this set of subgoals $\mathcal{G}$—is an important part of this algorithm, as we show empirically in Section 6.3. For this

paper, however, we focus on this planning formalism assuming these subgoals are given, already discovered by the agent.

Planning in Goal-Space: In the planning step, we treat $\mathcal{G}$ as our finite set of states and do value iteration. The precise formula for this update is given later, in (3). In words, we compute the subgoal values $\tilde{v} : \mathcal{G} \to \mathbb{R}$, using the models $\tilde{r}_\gamma(g, g')$ and $\tilde{\Gamma}(g, g')$ where

$$\tilde{r}_\gamma(g, g') = \text{ discounted return when trying to reach } g' \text{ from } g \qquad \text{(see (2))}$$

$$\tilde{\Gamma}(g, g') = \text{discounted probability of reaching } g' \text{ from } g \qquad \text{(see (2))}$$

$$\tilde{v}(g) = \max_{g'} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')\tilde{v}(g') \qquad \text{(see (3))}$$

where these equations are formally defined in Equations (2) and (3) referenced on the rhs.

Projection Step 1 (Computing Projected Subgoal Values): The projection step involves updating values for states using the subgoal values. To connect between the abstract space and the state space, we need the models

$$r_\gamma(s, g) = \text{discounted return when trying to reach } g \text{ from } s$$

$$\Gamma(s, g) = \text{discounted probability of reaching } g \text{ from } s.$$

We can then compute the projected subgoal values $v_{g^\star}(s)$

$$v_{g^\star}(s) = \max_{\text{nearby subgoals } g} r_\gamma(s, g) + \Gamma(s, g)\tilde{v}(g) \qquad \text{(see (4))}.$$

Projection Step 2 (Using Projected Subgoal Values): There are several ways we could use this value estimate, with two obvious ideas being to use them inside an actor-critic architecture or as a bootstrap target. For example, for a transition (s, a, r, s'), we could update action-values $q(s, a)$ using $r + \gamma v_{g^\star}(s')$. This naive approach, however, can result in significant bias, as we discuss further in Section 5.4.

Instead, we propose to use $v_{g^\star}$ as a potential function for reward shaping (Ng et al., 1999). Potential-based reward shaping defines a new MDP with a modified reward function, $\tilde{R}_{t+1} \doteq R_{t+1} + \gamma\Phi(S_{t+1}) - \Phi(S_t)$, where $\Phi \colon \mathcal{S} \to \mathbb{R}$ is any state-dependent function. Ng et al. (1999) show that such a reward transformation preserves the optimal policies from the original MDP. We propose using $\Phi = v_{g^\star}$ to modify any TD learning algorithm to be compatible with GSP. For example, in the Sarsa(λ) algorithm, the update for the weights $\mathbf{w}$ of the function $q \colon \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$ would use the TD-error

$$\delta_t \doteq \underbrace{R_{t+1} + \gamma_{t+1}v_{g^\star}(S_{t+1}) - v_{g^\star}(S_t)}_{\tilde{R}_{t+1}} + \gamma_{t+1}q(S_{t+1}, A_{t+1}; \mathbf{w}) - q(S_t, A_t; \mathbf{w}). \qquad (1)$$

Potential-based shaping rewards the agent for selecting actions that result in transition that increase Φ. Consider the case when Φ represents the *negative* distance to a goal state. When $\Phi(S_{t+1}) > \Phi(S_t)$, then the agent has made progress towards getting to the goal, and it receives a positive addition to the reward. When Φ is an estimate of the value function, one can interpret the additive reward as rewarding the agent for taking actions that increase the value function estimate and penalizes actions that decrease the value. In this way, using $\Phi = v_{g^\star}$, the agent can leverage immediate feedback on the quality of its actions using the information from the abstract value function about what an optimal policy might look like.

Learning the models: A key part of this algorithm is learning the subgoal models, r_γ and Γ, which are UVFAs (Schaul et al., 2015). These models are quite different from standard models in RL, in that most models in RL input a state (or abstract state) and action and output an expected next state (or expected next abstract state). Here, the models take as inputs both the source and destination, and output only scalars (accumulated reward and discounted probability). Further, these models only consider local regions for each subgoal: the initiation sets for the options that reach those subgoals. Effectively, the subgoal models correspond to a set of local models, which allow for updating only parts of the space that change and focus function approximation capacity on only what we need to model. The design of GSP is built around using these types of models, that avoid outputting predictions about entire state vectors and restrict what is modeled to local regions.

4. Motivating Experiments for GSP

This section motivates the utility of GSP in propagating value and speeding up learning. We run experiments in three environments: FourRooms, PinBall (Konidaris and Barto, 2009), and GridBall (a version of PinBall without velocities). All learning curves are averaged over 30 runs, with shaded regions representing one standard error. We use a discount factor of $\gamma = 0.99$ and Sarsa($\lambda = 0.9$) or Sarsa(0) for the experiments in this section. We learn the subgoal models and compute $v_{g^\star}$ offline, to focus on the utility of the planning formalism; see Appendix E for details on learning these subgoal models.

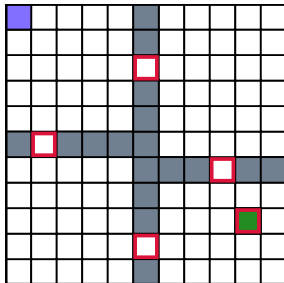### 4.1 GSP helps Propagate Value Quicker

Figure 2: The Four-Rooms domain. The blue square is the initial state, green square the goal state, and red boxes the subgoals. Grey squares are walls.

In this section we test the hypotheses that GSP can accelerate value propagation and accelerate learning in a simple tabular episodic environment: FourRooms. Transitions are deterministic, with a discrete action space $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$. The agent starts at the top left (the blue state in Figure Figure 2) and navigates to a goal state (green square), with 0 reward per step and +1 at the goal. Episodes timeout after 1000 timesteps.

We test the effect of using GSP with pre-trained models on a Sarsa($\lambda = 0.9$) base learner in the tabular setting (i.e., one-hot state encoding). We set the four hallway states plus the goal state as subgoals (outlined in red), with their initiation sets being the two rooms they connect. The full details of the learning algorithms and their hyperparameters are described in Appendix E.3.

Hypothesis 1 *GSP changes the value for more states with the same set of experience as the base learner.*

To test this hypothesis, we generate data from a uniform random policy. This represents an initial episode, where all approaches similarly randomly explore until seeing the goal. Figure 3 shows the action-value function using this single episode of data for four different algorithms: Sarsa(0), Sarsa(λ), Sarsa(0)+GSP, and Sarsa(λ)+GSP.

Figure 3 shows the action-value function after a single episode for four different algorithms: Sarsa(0), Sarsa(λ), Sarsa(0)+GSP, and Sarsa(λ)+GSP. We can see that Sarsa(0) updates
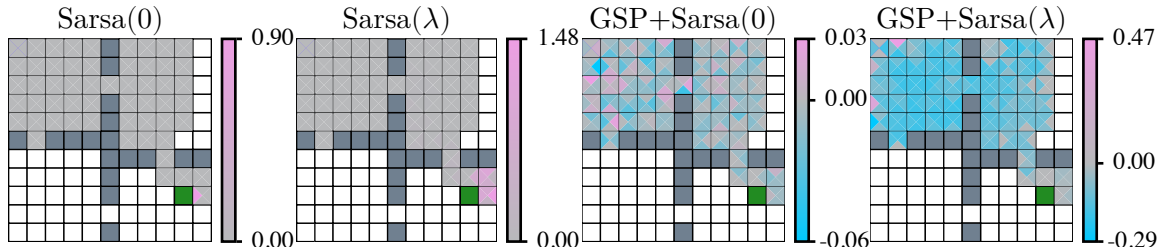
6

Figure 3: Action values after a single episode of updates for Sarsa(λ) with and without GSP, for $\lambda = 0.9$. All algorithms use the same data collected from a uniform random policy. We depict the values of each of the four actions in each state (square). Squares that are not visited are white.

the value of the state-action pair that immediately preceded the $+1$ reward at the goal state. Sarsa(λ) has more states updated near the goal, due to the use of traces. For the GSP variants, all sampled state-action pairs received instant feedback on the quality of their actions. Note that, even though rewards are non-negative, the values for GSP can be both positive or negative based on if the agent makes progress towards the goal state or not. The potential-based reward shaping rewards/penalizes transitions based on whether $\gamma_{t+1} v_{g^\star}(S_{t+1}) > v_{g^\star}(S_t)$. It is clear that projecting subgoal values from the abstract MDP leads to action-value updates over more of the visited states, even without memory mechanisms such as eligibility traces.

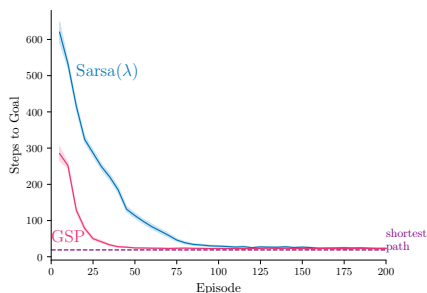## 4.2 GSP results in Faster Learning

Figure 4: The average number of steps to goal, smoothed over five episodes in the FourRooms domain. Shaded region represents 1 standard error across 100 runs.

The above result shows GSP updates more values, making it more likely to learn to get to the goal sooner. We test the following hypothesis.

Hypothesis 2 *GSP enables a TD base-learner to learn faster than with unshaped rewards.*

Figure 4 shows the performance of Sarsa(λ) with and without GSP in FourRooms. GSP-augmented Sarsa(λ) is able to reach the optimal policy much faster. The GSP learner also starts at a much lower steps-to-goal, because the potential-based shaping impacts action selection in the first episode.

4.3 Testing the Hypotheses in Continuous-State Environments

We test these two hypotheses again, but now in two continuous-state environments: PinBall and GridBall. In PinBall, the agent navigates a ball through a set of obstacles to reach the main goal. The state space consists of positions and velocities, $(x, y, \dot{x}, \dot{y}) \in [0, 1] \times [0, 1] \times [-2, 2] \times [-2, 2]$. There are five actions $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}, \texttt{nothing}\}$, where the

`nothing` action adds no change to the ball's velocity, and the other actions add an impulse force in one of the four cardinal directions. All collisions are elastic and use a drag coefficient of 0.995. The task is episodic, with a fixed starting state (with zero initial velocity), rewards 0 everywhere except +1 at the goal. An episode ends when the agent reaches the main goal, with timeouts after 1,000 time steps.

GridBall is like PinBall, but modified to remove velocity to make the state two-dimensional to facilitate visualization. The state consists of (x, y) locations, and the action space is changed to displace the ball by a fixed amount in each cardinal dimension. We keep the same obstacle collision mechanics and calculations from PinBall. Since GridBall does not have any velocity components, we can plot heatmaps of value propagation without having to consider the velocity at which the agent arrived at a given position.

For both environments, agents use linear function approximation with tile-coding (Sutton and Barto, 2018). We selected subgoals somewhat randomly, simply ensuring they are not inside walls and that they are somewhat spread out. We visualize the subgoals and initiation sets for the subgoals in Figure 5. It should be noted that, unlike in FourRooms, there exist states which are not in the initiation set of any subgoal; this is a likely occurrence for GSP, where subgoals are iteratively chosen and unlikely to cover the space. The design of GSP accounts for this lack of coverage by simply not using the projected subgoal values for these non-covered regions and defaulting to the base learner's update. Full details on the option policies and subgoal models are in Appendix E.
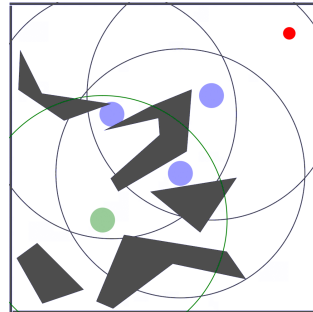

Figure 5: Obstacles and subgoals for GridBall and PinBall. The red dot is the start state, the green the goal. The larger circles show the initiation set boundaries.

For Hypothesis 1, we collect a single episode of experience from Sarsa(0)+GSP to use as the fixed dataset for all learners. The results are similar to those on FourRooms, shown in Figure 6. The Sarsa(0) algorithm only updates the value of the tiles activated by the state preceding the goal. Sarsa(λ) has a decaying trail of updates to the tiles activated preceding the goal, and the GSP learners update values at all states in the initiation set of a subgoal.

For Hypothesis 2), we measure performance (steps to goal) in both GridBall and PinBall domains, shown in Figure 7. As before, GSP significantly improves the rate of learning, and

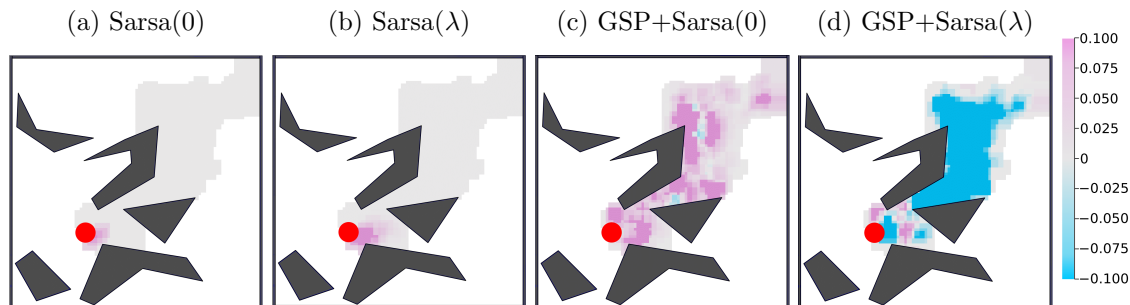| (a) Sarsa(0)	(b) Sarsa(λ)	(c) GSP+Sarsa(0)	(d) GSP+Sarsa(λ)

Figure 6: The value function after one episode in GridBall for the four algorithms. The red circle is the goal. The gray region is the visited states which were not updated.

converges to the same high-quality solution. Variability is quite low across 30 runs, with convergence to a similar number of steps to goal for all runs. Even though the obstacles remain unchanged from GridBall, it takes roughly 50 episodes longer for even the GSP variant to reach a good policy in PinBall. This is likely due to the continuous 4-dimensional state space and ball momentum making the task harder.
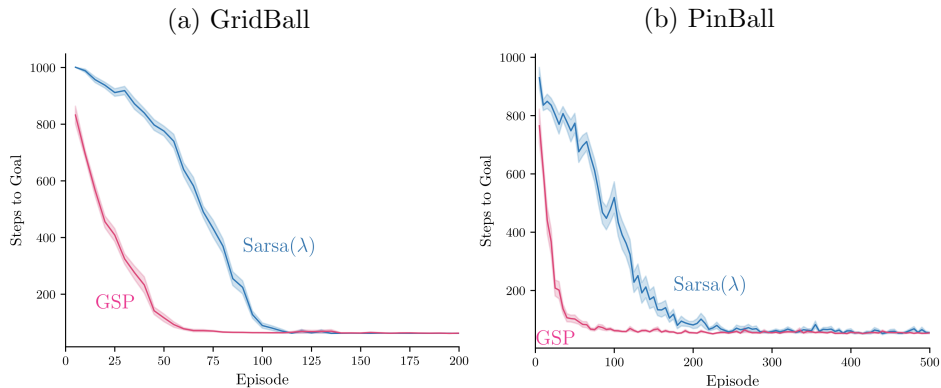
(a) GridBall   (b) PinBall

Figure 7: Five-episode moving average of return (steps to goal) in GridBall and PinBall, averaged over 30 runs with 1 standard error as the shaded region.

5. Goal-Space Planning in More Detail

In this section we provide the technical definitions and details for GSP. We first discuss the definition of subgoals and then the corresponding subgoal-conditioned models. We then discuss how to use these models for planning, particularly how to do value iteration to compute the subgoal values and then how to use those values to influence values of states (the projection step). We conclude the section summarizing the overall goal-spacing planning framework, including how we can layer GSP into a standard algorithm called Double DQN.

5.1 Defining Subgoals

Similar to options (Sutton et al., 1999), we define a subgoal as having two components: a set of goal states, and a set of initiation states. These two sets are defined by the indicator functions: m specifies the states in the goal set, and d specifies states in an initiation set. We say that a state s is a member of subgoal g if $m(s, g) = 1$. We only reason about reaching a subgoal g from a state s in the initiation set, namely such that $d(s, g) = 1$. This constraint is key for locality, to learn and reason about a subset of states for a subgoal.

   While subgoals could represent a single state, they can also describe more complex conditions that are common to a group of states. For example, g could correspond to a situation where both the front and side distance sensors of a robot report low readings—what a person would call being in a corner. If the first two elements of the state vector s consist of the front and side distance sensor, $m(s, g) = 1$ for any states where s_1, s_2 are less than some threshold ϵ. As another example, in Figure 5 for PinBall, we encode a subgoal as a small ball (depicted in teal); any s in this ball satisfies $m(s, g) = 1$. In Figure 5, we also visualize

9

the initiation set d as the larger circles, which restrict reasoning about how to reach the subgoal from relatively nearby (local) states.

For the rest of this paper, we assume we are given a finite set of subgoals $\mathcal{G}$. We develop algorithms to learn and use models, given those subgoals. We expect a complete agent to discover these subgoals on its own, including how to represent these subgoals to facilitate generalization and planning. We discuss some approaches to learn this initiation function in Appendix D. In this work, though, we first focus on how the agent can leverage a reasonably well-specified subgoals.

5.2 Defining Subgoal-Conditioned Models

For planning and acting to operate in two different spaces, we define four models: two used in planning over subgoals (subgoal-to-subgoal) and two used to project these subgoal values back into the underlying state space (state-to-subgoal). The state-to-subgoal models are $r_\gamma : \mathcal{S} \times \bar{\mathcal{G}} \to \mathbb{R}$ and $\Gamma : \mathcal{S} \times \bar{\mathcal{G}} \to [0,1]$, where $\bar{\mathcal{G}} = \mathcal{G} \cup \{s_{\text{terminal}}\}$ if there is a terminal state (episodic problems) and otherwise $\bar{\mathcal{G}} = \mathcal{G}$. An option policy $\pi_g : \mathcal{S} \times \mathcal{A} \to [0,1]$ for subgoal g starts from any s in the initiation set, and terminates in g—in $\tilde{s}$ where $m(\tilde{s}, g) = 1$. The reward-model $r_\gamma(s, g)$ is the discounted rewards under option policy π_g:

$$r_\gamma(s, g) = \mathbb{E}_{\pi_g}[R_{t+1} + \gamma_g(S_{t+1})r_\gamma(S_{t+1}, g)|S_t = s],$$

where the discount is zero upon reaching subgoal g,

$$\gamma_g(S_{t+1}) \doteq \begin{cases} 0 & \text{if } m(S_{t+1}, g) = 1, \text{ namely if subgoal } g \text{ is achieved by being in } S_{t+1}, \\ \gamma_{t+1} & \text{else.} \end{cases}$$

The discount-model $\Gamma(s, g)$ reflects the discounted probability of reaching subgoal g starting from s, in expectation under option policy π_g,

$$\Gamma(s, g) = \mathbb{E}_{\pi_g}[m(S_{t+1}, g)\gamma_{t+1} + \gamma_g(S_{t+1})\Gamma(S_{t+1}, g)|S_t = s].$$

These state-to-subgoal models will only be queried for (s, g) where $d(s, g) > 0$: they are local models. They can also be written as General Value Functions (GVFs) (Sutton et al., 2011), as shown in Appendix D.2.

To define subgoal-to-subgoal models,[1] $\tilde{r}_\gamma : \mathcal{G} \times \bar{\mathcal{G}} \to \mathbb{R}$ and $\tilde{\Gamma} : \mathcal{G} \times \bar{\mathcal{G}} \to [0,1]$, we use the state-to-subgoal models and aggregate over all s where $m(s, g) = 1$.

$$\tilde{r}_\gamma(g, g') \doteq \frac{1}{z(g)} \sum_{s:m(s,g)=1} r_\gamma(s, g') \quad \text{and} \quad \tilde{\Gamma}(g, g') \doteq \frac{1}{z(g)} \sum_{s:m(s,g)=1} \Gamma(s, g') \tag{2}$$

for normalizer $z(g) \doteq \sum_{s:m(s,g)=1} m(s, g)$. This definition assumes a uniform weighting over the states s where $m(s, g) = 1$. We could allow a non-uniform weighting, potentially based on visitation frequency in the environment. For this work, however, we assume that $m(s, g) = 1$ for a smaller number of states s with relatively similar $r_\gamma(s, g')$, making a uniform weighting reasonable. For continuous states, these sums are replaced by integrals.

---

1. The first input is any $g \in \mathcal{G}$, the second is $g' \in \bar{\mathcal{G}}$, which includes s_{terminal}. We need to reason about reaching any subgoal or s_{terminal}. But s_{terminal} is not a real state: we do not reason about starting from it to reach subgoals.

We can similarly extract $\tilde{d}(g, g')$ from $d(s, g')$ and only reason about g' nearby or relevant to g: $\tilde{d}(g, g') \doteq \max_{s \in \mathcal{S}:m(s,g)>0} d(s, g')$. This definition indicates that if there is a state s that is in the initiation set for g' and has membership in g, then g' is also relevant to g.

Let us consider an example, in Figure 8. The red states are members of g (namely $m(A, g) = 1$) and the blue members of g' (namely $m(X, g') = 1$ and $m(Y, g') = 1$). For all s in the diagram, $d(s, g') > 0$ (all are in the initiation set): the policy $\pi_{g'}$ can be queried from any s to get to g'. The green path in the left indicates the trajectory under $\pi_{g'}$ from A, stochastically reaching either X or Y, with accumulated reward $r_\gamma(A, g')$ and discount $\Gamma(A, g')$ (averaged over reaching X and Y). The subgoal-to-subgoal models, on the right, indicate g' can be reached from g, with $\tilde{r}_\gamma(g, g')$ averaged over both

Figure 8: Original and abstract state spaces.

$r_\gamma(A, g')$ and $r_\gamma(B, g')$ and $\tilde{\Gamma}(g, g')$ over $\Gamma(A, g')$ and $\Gamma(B, g')$, as described in (2).

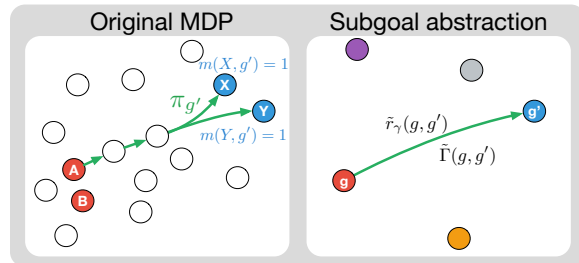### 5.3 Goal-Space Planning with Subgoal-Conditioned Models

Planning in GSP involves learning $\tilde{v}(g)$: the value for different subgoals. This can be achieved using an update similar to value iteration, for all $g \in \mathcal{G}$,

$$\tilde{v}(g) = \max_{g' \in \bar{\mathcal{G}}:\tilde{d}(g,g')>0} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')\tilde{v}(g'). \tag{3}$$

The value of reaching g' from g is the discounted rewards along the way, $\tilde{r}_\gamma(g, g')$, plus the discounted value in g'. If $\tilde{\Gamma}(g, g')$ is very small, it is difficult to reach g' from g—or takes many steps—and so the value in g' is discounted by more. With a relatively small number of subgoals, we can sweep through them all quickly compute $\tilde{v}(g)$. With a larger set of subgoals, we can sample many g instead.

We can interpret this update as value iteration in a new MDP, where 1) the set of states is $\mathcal{G}$, 2) the actions from $g \in \mathcal{G}$ are state-dependent, corresponding to choosing which $g' \in \bar{\mathcal{G}}$ to go to in the set where $\tilde{d}(g, g') > 0$ and 3) the rewards are $\tilde{r}_\gamma$ and the discounted transition probabilities are $\tilde{\Gamma}$. It is straightforward to show that the above converges to the optimal values in this new Goal-Space MDP, shown in **Proposition 3** in Appendix B.

Planning in goal space has two benefits. We have a temporally extended model that does not to be iterated over multiple timesteps, avoiding compounding error. Additionally, we do not need to predict entire state vectors—or distributions over them—like a usual state transition model. This is because the outcome g' from Equation 3 is passed into our model $\tilde{\Gamma}$, rather than being the output of a model. This may feel like a false success as it potentially requires restricting ourselves to a smaller number of subgoals. If we want to use a larger number of subgoals, then we may need a function to generate these subgoal vectors anyway—bringing us back to the problem of predicting vectors. However, this generation need only produce potentially *relevant* subgoals, which requires lower precision—a larger set can be generated—than generating the true distribution over outcomes.

5.4 Using Subgoal Values to Update the Policy

Now let us examine how to use $\tilde{v}(g)$ to update our main policy. The simplest way to decide how to behave from a state is to cycle through the subgoals, and pick the one with the highest value. In other words, we can define

$$v_{g^\star}(s) \doteq \begin{cases} \max\limits_{g\in\bar{\mathcal{G}}:d(s,g)>0} r_\gamma(s,g) + \Gamma(s,g)\tilde{v}(g) & \text{if } \exists\, g \in \bar{\mathcal{G}} : d(s,g) > 0, \text{ (projection step)} \\ \text{undefined} & \text{otherwise,} \end{cases} \tag{4}$$

and take action a that corresponds to the action given by π_g for this maximizing g as shown in Figure 9. Note that some states may not have any nearby subgoals, and $v_{g^\star}(s)$ is undefined for that state. This is not the only problem with this naive approach.
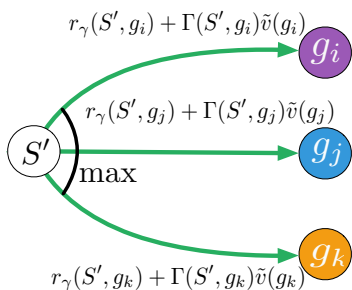

Figure 9: Computing $v_{g^\star}(S')$.

There are two other critical issues with this approach. Policies are restricted to go through subgoals, which might result in suboptimal policies. From a given state s, the set of relevant subgoals g may not be on the optimal path. This was a key limitation of Landmark Value Iteration (LAVI), which was developed for the setting where models are given (Mann et al., 2015), and is one we explicitly wish to avoid. Second, the learned models themselves may have inaccuracies, or planning may not have been completed in the background, resulting in $\tilde{v}(g)$ that are not yet fully accurate.

We propose to instead leverage $v_{g^\star}$ using potential-based reward shaping (Ng et al., 1999), as described in Section 3. This approach avoids incurring significant bias, instead simply guiding the main policy with a modification to the TD-error given in Equation (1): $\delta_t = \tilde{R}_{t+1} + \gamma_{t+1}q(S_{t+1}, A_{t+1}; \mathbf{w}) - q(S_t, A_t; \mathbf{w})$ for $\tilde{R}_{t+1} = R_{t+1} + \gamma_{t+1}v_{g^\star}(S_{t+1}) - v_{g^\star}(S_t)$. For intuition as to why potential-based reward shaping does not bias the optimal policy, notice that $\sum_{t=0}^{\infty} \gamma^t \left(\gamma\Phi(S_{t+1}) - \Phi(S_t)\right) = -\Phi(S_0)$, which means the relative values of each action remain the same.[2] Under linear value function approximation, we show that such shaping does not bias the optimal policy when Φ is also linear in the features (Appendix C.1). Another benefit of potential-based reward shaping is its equivalence to initializing q to Φ in the tabular setting (Wiewiora, 2003). We show this for Sarsa(λ) in Appendix C.2.

It is important to note that if $v_{g^\star}$ can help improve learning, it can also make learning harder if its guidance makes it less likely for an agent to sample optimal actions. This increase in difficulty is likely if the models used to construct the abstract MDP and $v_{g^\star}$ have substantial errors. In this case, the agent has to learn to overcome the bad "advice" provided by $v_{g^\star}$. We investigate this further with non-stationary environments and inaccurate models in Sections G.1 and 6.2 respectively.

---

2. The cancellations of these intermediate terms mean that algorithms like REINFORCE (Williams, 1992), Proximal Policy Optimization (Schulman et al., 2017) or Monte Carlo methods will see little benefit when combined with potential-based reward shaping (as they use the discount sum of all rewards to update the policy). For these algorithms, one could instead estimate a $q_{g^\star}$ and leverage trajectory-wise control variates (Pankov, 2018; Cheng et al., 2019; Huang and Jiang, 2020). We leave the investigation of this approach to future work and focus on TD learning algorithms in this paper.

5.5 Putting it All Together: The Full Goal-Space Planning Algorithm

The remaining piece is to learn the models and put it all together. Learning the models is straightforward, as we can leverage the large literature on general value functions (Sutton et al., 2011) and UVFAs (Schaul et al., 2015). There are nuances involved in 1) restricting updating to relevant states according to $d(s, g)$, 2) learning option policies that reach subgoals, but also maximize rewards along the way and 3) considering ways to jointly learn d and Γ. We include these details in Appendix D. In this subsection, we summarize the higher-level steps.

The algorithm is visualized in Figure 10. Each agent-environment step includes:

1. take action A_t in state S_t, to get S_{t+1}, R_{t+1} and γ_{t+1}

2. query the model for $r_\gamma(S_{t+1}, g), \Gamma(S_{t+1}, g),$ $\tilde{v}(g)$ for all g where $d(S_{t+1}, g) > 0$

3. compute projection $v_{g^\star}(S_{t+1})$, using (4)

4. update the main policy with the transition and $v_{g^\star}(S_{t+1})$, using (1).

All background computation is used for model learning using a replay buffer and for planning to obtain $\tilde{v}$, so that they can be queried at any time on step 2.

To be more concrete, Algorithm 1 shows the GSP algorithm layered on DDQN (van Hasselt et al., 2016). DDQN is a variant of DQN—and so relies on replay—that additionally incorporates the idea behind Double Q-learning to avoid maximization bias in the Q-learning update (van Hasselt, 2010). All new parts relevant to GSP are colored; without these parts, we recover the standard DDQN algorithm. The primary change is the addition of the potential to the action-value weights $\mathbf{w}$, with the other magenta lines primarily around learning the model and doing planning. GSP should improve on replay because it simply augments replay with a potential difference that more quickly guides the agent to take promising actions.

Figure 10: Goal-Space Planning.

6. Experiments to understand GSP deeper

In Section 4, we investigated the role of GSP in propagating value and speeding up learning under linear function approximation. In this section, we investigate how this utility of GSP is affected by: 1) deep non-linear function approximation, 2) subgoal placement, 3) sensitivity to models, and 4) the $v_{g^\star}$ potential used. Throughout the experiments, we learn the subgoal models upfront, to focus the investigation on the utility of the planning formalism; see Appendix E for a description of this procedure.
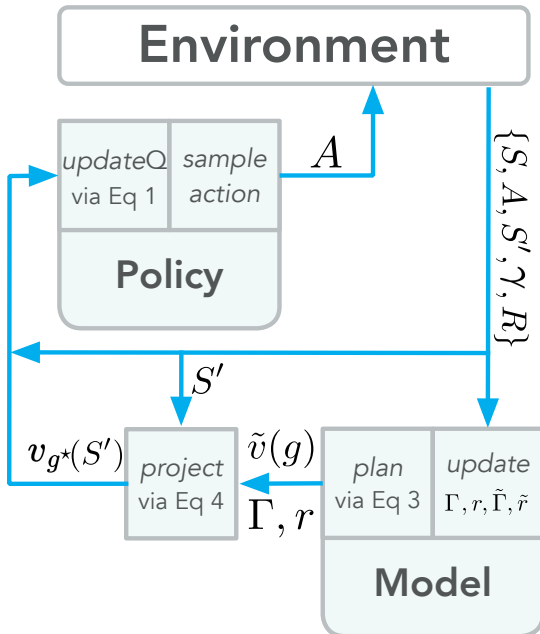

Algorithm 1 Goal Space Planning with DDQN as a base learner

Initialize base learner parameters $\mathbf{w}, \mathbf{w}_{\text{targ}} = \mathbf{w}_0$, $n_{\text{updates}} = 0$, target refresh rate τ,

set of subgoals $\bar{\mathcal{G}}$, relevance function d, model parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}^r, \boldsymbol{\theta}^\Gamma, \boldsymbol{\theta}^\pi), \tilde{\boldsymbol{\theta}} = (\tilde{\boldsymbol{\theta}}^r, \tilde{\boldsymbol{\theta}}^\Gamma)$

Sample initial state s_0 from the environment

for $t \in 0, 1, 2, ...$ **do**

    Take action a_t using q (e.g., ϵ-greedy), observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$

    Add experience $(s_t, a_t, r_{t+1}, s_{t+1}, \gamma_{t+1})$ to replay buffer D

    Update_Models() (see Algorithm 6)

    Planning() (see Algorithm 2)

    for n mini-batches **do**

        Sample batch $B = \{(s, a, r, s', \gamma)\}$ from D

        $\tilde{r} \leftarrow r$

        // Add reward shaping if projected subgoal values defined for both s and s'

        if $d(s, g) > 0$ for some $g \in \mathcal{G}$ and $d(s', g) > 0$ for some $g \in \mathcal{G}$ **then**

            $v_{g^\star}(s) = \max_{g \in \bar{\mathcal{G}}:d(s,g)>0} r_\gamma(s, g; \boldsymbol{\theta}^r) + \Gamma(s, g; \boldsymbol{\theta}^\Gamma)\tilde{v}(g)$

            $v_{g^\star}(s') = \max_{g \in \bar{\mathcal{G}}:d(s',g)>0} r_\gamma(s', g; \boldsymbol{\theta}^r) + \Gamma(s', g; \boldsymbol{\theta}^\Gamma)\tilde{v}(g)$

            $\tilde{r} \leftarrow \tilde{r} + \gamma v_{g^\star}(s') - v_{g^\star}(s)$

        $Y(s, a, r, s', \gamma) = \tilde{r} + \gamma q(s', \text{argmax}_{a'} q(s', a'; \mathbf{w}); \mathbf{w}_{\text{targ}})$

        $L = \frac{1}{|B|} \sum_{(s,a,r,s',\gamma) \in B} (Y(s, a, r, s', \gamma) - q(s, a; \mathbf{w}))^2$

        $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L$

        if $\text{modulo}(n_{\text{updates}}, \tau) == 0$ **then** $\mathbf{w}_{\text{targ}} \leftarrow \mathbf{w}$

        $n_{\text{updates}} = n_{\text{updates}} + 1$

Algorithm 2 Planning()

---

for n iterations, for each $g \in \mathcal{G}$ **do**

    $\tilde{v}(g) \leftarrow \max_{g' \in \bar{\mathcal{G}}:d(g,g')>0} \tilde{r}_\gamma(g, g'; \tilde{\boldsymbol{\theta}}^r) + \tilde{\Gamma}(g, g'; \tilde{\boldsymbol{\theta}}^\Gamma)\tilde{v}(g')$

6.1 GSP with Deep Reinforcement Learning

We test GSP layered on DDQN as in Algorithm 1 in the PinBall domain. The base learner's complete hyper-parameter specifications can be found in Appendix E. All other experiment settings were the same as Section 4.3, including using 30 runs.

Unlike the previous experiments, using GSP out of the box resulted in the base learner converging to a sub-optimal policy. This is despite the fact that we used the same $v_{g^\star}$ as the previous PinBall experiments. We investigated the distribution of shaping terms added to the environment reward and observed that they were occasionally an order of magnitude greater than the environment reward. Though the linear and tabular methods handled these spikes in potential difference gracefully, these large displacements seemed to causes issues when using neural networks and a DDQN base learner.

We tested two variants of GSP that better control the magnitudes of the raw potential differences $(\gamma \Phi(S_{t+1}) - \Phi(S_t))$. We adjusted for this by either clipping or down-scaling the potential difference added to the reward. The scaled reward multiplies the potential difference by 0.1. Clipped GSP clips the potential difference into the $[-1, 1]$ interval. It
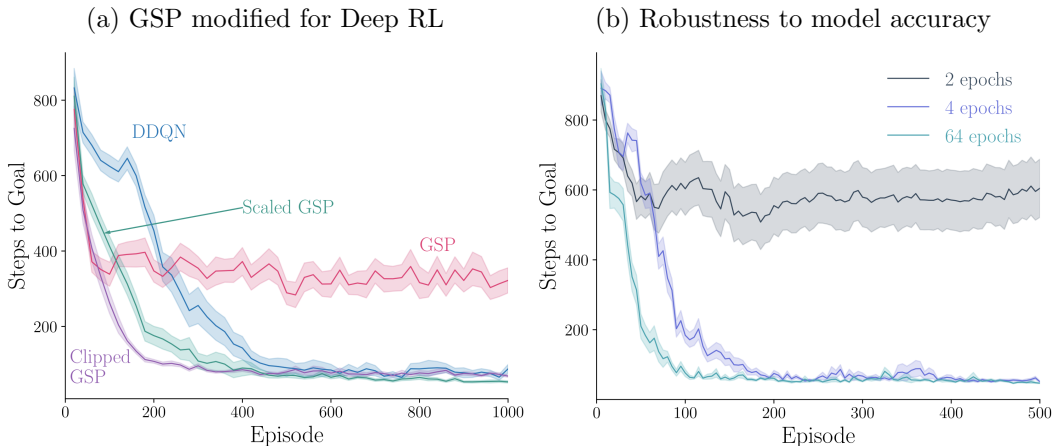
Figure 11: GSP+DDQN performance in PinBall. Results are averaged over 30 runs, with shaded region corresponding to one standard error. (a) The performance of GSP without any clipping or scaling of the potential-based shaping (red), versus adding clipping or scaling to control the magnitudes of the potentials. (b) The performance of GSP with models trained with differing numbers of epochs. We use a 20-episode moving average of the steps to goal for (a), and a five-episode moving average for (b).

should be noted that clipping the potential difference no longer guarantees the optimal policy will be preserved. With these basic magnitude controls, GSP again learns significantly faster than its base learner, as shown in Figure 11a.

6.2 Robustness to Accuracy of the Learned Models

In this section, we investigate how robust GSP is to inaccuracy of its models. When examining the accuracy of the learned models, we found the the errors in r_γ and Γ could be as high as 20% in some parts of the state space (see Appendix G for more information). Despite this level of inaccuracy in some states, GSP still learned effectively, as seen in Sections 4.1 and 4.3. We conducted a targeted experiment controlling the level of accuracy to better understand this robustness and test the following hypothesis.

Hypothesis 3 *GSP can learn faster with more accurate models, but can still improve on the base learner even with partially learned models.*

We varied the number of epochs to obtain models of varying accuracy. Our models were fully connected artificial neural networks, and we learn the models for each subgoal by performing mini-batch stochastic gradient descent on a dataset of trajectories that end in a member state of that subgoal g. Full implementation details for this mini-batch stochastic gradient descent can be found in Appendix E.

As expected, Figure 11b shows that more epochs over the same dataset of transitions improves how quickly the base learner reaches the main goal. Within 4 epochs of model training, the learner is able to reach a good policy to the main goal. However, if the model is very inaccurate (2 epochs), the GSP update will bias the base learner to a sub-optimal policy. There is a trend of diminishing improvement when iterating over the same dataset of

experience: doubling the number of epochs from two to four results in a policy that reaches the main goal $10\times$ quicker, but a learner which used a further $16\times$ the number of epochs attains a statistically identical episode length by episode 500. While more accurate models lead to faster learning, relatively few epochs are required to propagate enough value to help the learner reach a good policy. Note that we also report the state-to-subgoal model errors in Table 3. We can see there is a more notable decrease in error from 2 to 4 epochs, as compared with 4 to 10.

6.3 The Impact of Subgoal Selection

In this section we investigate how the selection of subgoals impacts value propagation in GSP. We consider a setting where the world changes and the agent needs to quickly updates its policy (action-values). After the change, the state-to-subgoal and subgoal-to-subgoal models are updated online and we measure how much $v_{g^\star}$ changes, along with how quickly the base learner can change its policy given different subgoal configurations.

We use a modified variant, where a dangerous region (a lava pool) arises partway through learning, as shown in Figure 12. Before seeing the lava pool, the agent learns the optimal policy: taking the shorter of the two paths to the goal (green square). Then we introduce a lava pool along the optimal path that gives the agent a large negative reward for entering it. This negative reward means the initial path is no longer optimal and that the agent needs to switch to the alternate path. The FourRooms environment uses -1 reward per step, and each state in the lava pool has a reward of -20. GSP and Sarsa are initialized with $q^\star$ for the original FourRooms environment and then run for 100 episodes in the lava pool variant. GSP is run with the four different subgoal configurations shown in Figure 12. We use a tabular setting and report results for 200 runs for each algorithm.

Hypothesis 4 *The placement of subgoals along the initial and alternate optimal paths are essential for fast adaptation.*
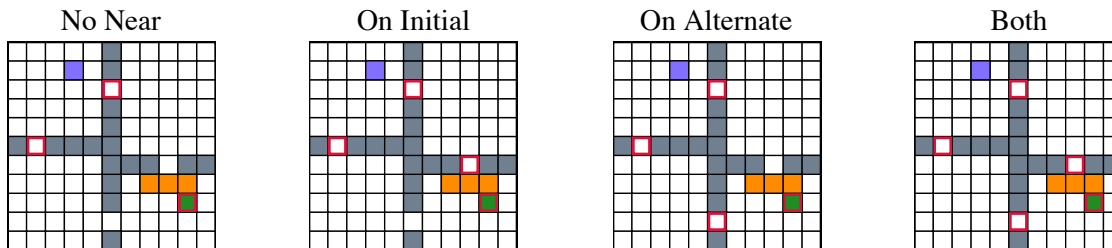

Figure 12: Different subgoal configurations in FourRooms with a lava pool. The purple square is the start location, the gray squares the walls, the orange squares the lava pool, and the green square the goal. The only difference between these figures are the red boxes, which indicate the subgoals. A subgoal's initiation set is the states in the two adjacent rooms.

For this experiment, the state-to-subgoal models need to be updated online. However, since only the reward function is changing, we only need to update the reward models r_γ and $\tilde{r}_\gamma$. We can represent r_γ using successor features so that the agent only needs to estimate the
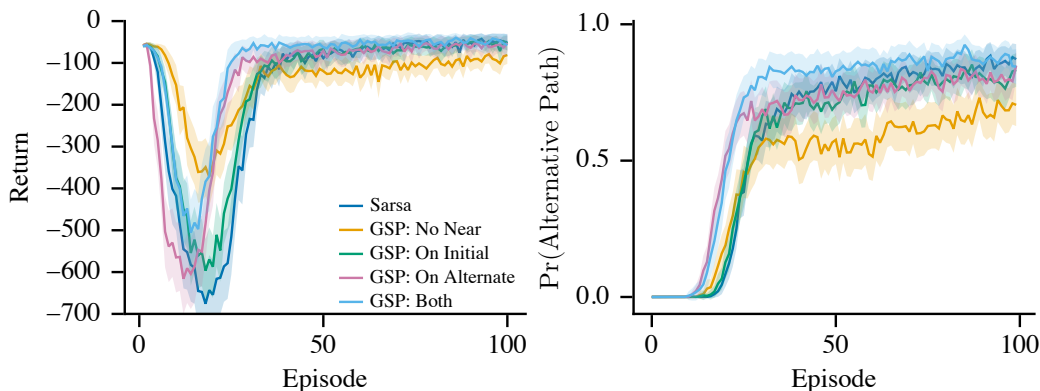
Figure 13: The average return (left) and average probability the agent will take the alternative path (right) from each episode. Shaded regions represent (0.05,0.9) tolerance intervals (Patterson et al., 2020) over 200 trials.

reward function (Barreto et al., 2017). Let $\boldsymbol{\psi}^{\pi_g}(s) \approx \mathbb{E}_{\pi_g}\left[\sum_{k=0}^{\infty}\prod_{k'=0}^{k}\gamma_{t+k'}\boldsymbol{\phi}(S_{t+k})|S_t = s\right]$, where $\boldsymbol{\phi}(S_t) \in \mathbb{R}^n$ is a vector of features for state S_t and actions are selected according to option policy π_g. Then $r_\gamma(s,g) = \mathbf{w}^\top \boldsymbol{\psi}^{\pi_g}(s)$, where $\mathbf{w} \in \mathbb{R}^n$. The learner can then update r_γ by estimating the reward function with stochastic gradient descent, i.e., $\mathbf{w} \leftarrow \mathbf{w} + \eta[R_t - \mathbf{w}^\top\boldsymbol{\phi}(S_t)]\boldsymbol{\phi}(S_t)$ for some scalar step size η.

To understand how learning is impacted by the subgoal configuration we show the return and probability the agent takes the alternative path in Figure 13. The first thing that is apparent is that all configurations are able to change the policy so that the probability of taking the alternative path increases. The main differences come from how quickly each configuration is able to change the policy to have a high probability of taking the alternate path. The Both and On Alternate subgoal configurations have the quickest change in the policy on average, while the other methods are slower. The No Near configuration also seems to, on average, have the smallest increase in probability of taking the alternate path. These results suggest that for GSP to be most impactful, there needs to be a path through the subgoals that represents the desirable path.

To better understand these results, we measure how $v_{g^\star}$ changes over learning. The top row of Figure 14 show the values of $v_{g^\star}$ before the introduction of the lava pool. For the No Near subgoals configuration, $v_{g^\star}$ has a disconnected graph, so all but the room with the goal state has a large negative value. For both On Initial and On Alternate configurations, $v_{g^\star}$ is the smallest in the room that is furthest from the goal state according to the abstract MDP. For example, in On Alternate, the value for the North subgoal is low, because the abstract MDP (erroneously) indicates the agent must go through the two other subgoals (West and South) to reach the goal, rather than going through the East doorway. In the Both subgoal configuration $v_{g^\star}$ closely represents the optimal value function in each state.

We then look at the change in $v_{g^\star}$ after the lava pool is introduced, i.e. $v_{g^\star,t} - v_{g^\star,0}$, where $v_{g^\star,i}$ is the value of $v_{g^\star}$ after episode i. The middle row in Figure 14 show the changes after one episode and the bottom row after 100 episodes. The value in the No Near subgoal configuration does not propagate information from the lava pool to rooms outside the bottom left room. For the On Initial configuration, the value decreases quickly in the top right room,
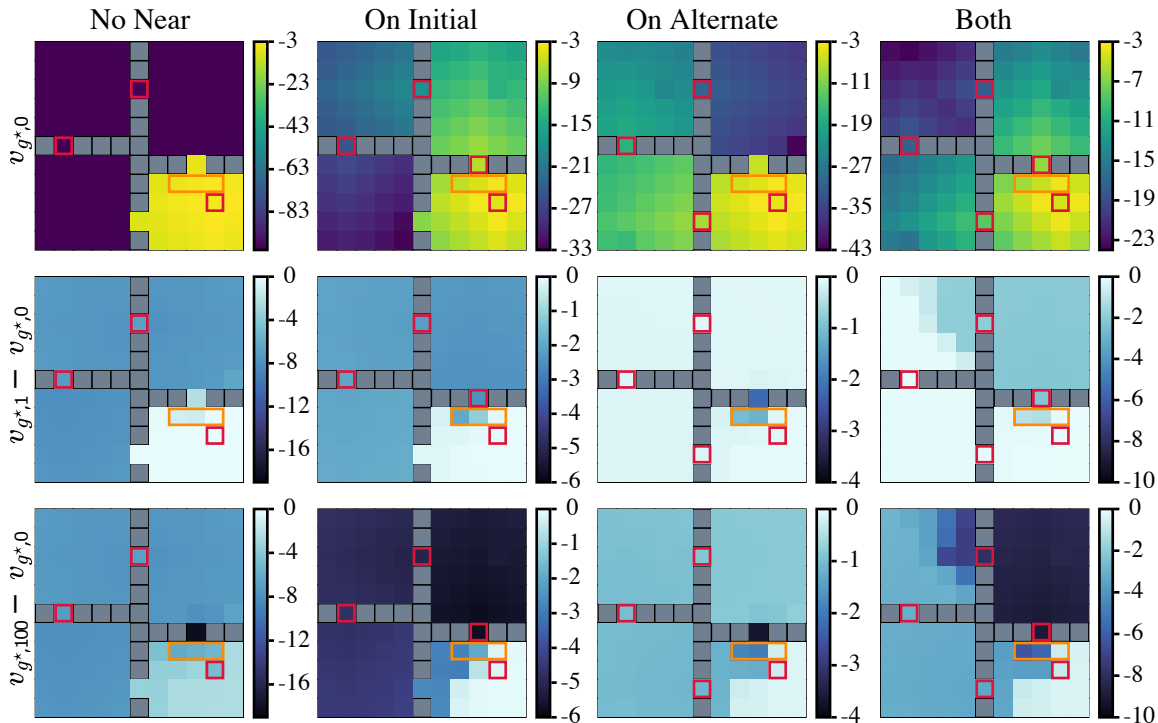
17

Figure 14: The top row of this figure shows the value of $v_{g^\star}$ for each state before the lava pool, for each subgoal configuration. The second and third rows show the change in $v_{g^\star}$ after the first and 100th episode, after the lava pool is introduced.

but also the other two rooms as well. After 100 episodes the value is decreased in most states but the top right room sees the largest decrease. For the On Alternate configuration value is not quickly propagated after discovering the lava pool because there is no connected region from the path the agent took to the lava pool. However, small changes are propagated over time due to the small probability of hitting the lava pool on the alternate path. With the Both subgoal configuration, value is quickly decreased in the states that would take the initial path, but not the alternate path. This indicates the desirable path through subgoals changes in the abstract MDP. Over time the decrease in value is largely isolated to the top right room with the decreases in the other rooms coming from small chances of hitting the lava pool on the alternate path.

Remark: We also examined the utility of these subgoals for learning before the lava pool was introduced. Here we found that the On Alternate subgoal placement actually caused the agent to learn a suboptimal policy, because it biased it towards the alternate path initially. You can see a visualization of this $v_{g^\star}$ in Figure 14 (top row, third column). The base learner does not use a smart exploration strategy to overcome this initial bias, and so settles on a suboptimal solution—namely, to take the slightly longer alternate path. See Appendix G.1 for the full details and results for this experiment. Note that this suboptimality did not arise in the above experiment, because the lava pool made one path significantly worse than the other.

6.4 Comparison to Other Potentials

We now investigate how the above improvements in learning are simply due to using potential-based reward shaping. We test this by comparing $v_{g^\star}$ with two other potentials—an informative and an uninformative one—in the PinBall domain. The first potential function is the negative L_2 distance in position space (scaled) to the main goal, (x_g, y_g),

$$\Phi(S_t) = -100 \times \left\| \begin{bmatrix} x_g \\ y_g \end{bmatrix} - \begin{bmatrix} x(S_t) \\ y(S_t) \end{bmatrix} \right\|_2$$

where $x(S_t)$ and $y(S_t)$ are functions that return the x and y coordinates of the agent's state respectively. This potential function captures a measure of closeness to the goal state, but does not consider obstacles or the velocity component. It should provide some learning benefit but should not be as helpful as $v_{g^\star}$. We scale this potential by a factor of 100 to make it comparable in magnitude to $v_{g^\star}$. Reward shaping with the unscaled L_2 distance did not have any significant effect on the base learner. The second potential is created by randomly assigning a potential value for each state, i.e., $\forall s \in \mathcal{S}, \ \Phi(s) \leftarrow \mathcal{U}[-100, 0]$. This potential does not encode any useful information about the environment on average. It should not help learning and could even make it harder if it encourages the agent to take sub-optimal actions.

Hypothesis 5 *In PinBall, a potential based on L_2 distance also speeds up learning, but not as much as $v_{g^\star}$ which better reflects transition dynamics in the given MDP.*
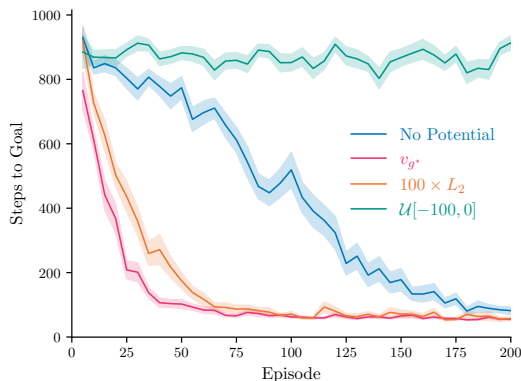

Figure 15: Five episode moving average of steps to goal in PinBall with different potential functions for $\Phi(s)$.

We test the impacts of these potentials by comparing a Sarsa(λ) base learner in PinBall with the same subgoal configuration and settings as in Section 4.3 and same format as Figure 11a. We can see in Figure 15 that using $v_{g^\star}$ for the potential reaches the main goal fastest, though using L_2 also resulted in significant speed-ups over the base learner (no potential). The L_2 heuristic, however, is specific to navigation environments, and finding such general purpose heuristics is difficult. The random potential harms performance, likely because it skews the reward and impacts exploration negatively.

6.5 Comparing to an Alternative Way of using $v_{g^\star}$

We used $v_{g^\star}$ through potential-based reward shaping, but other approaches are possible. For example, another approach is to solely bootstrap off of the prediction from $v_{g^\star}$, instead of the base learner's q estimate,

$$R_{t+1} + \gamma_{t+1} v_{g^\star}(S_{t+1}) - q(S_t, A_t; \mathbf{w}).$$

The update with this TD error is reminiscent of an algorithm called Landmark Approximate Value Iteration (LAVI) (Mann et al., 2015). LAVI is designed for the setting
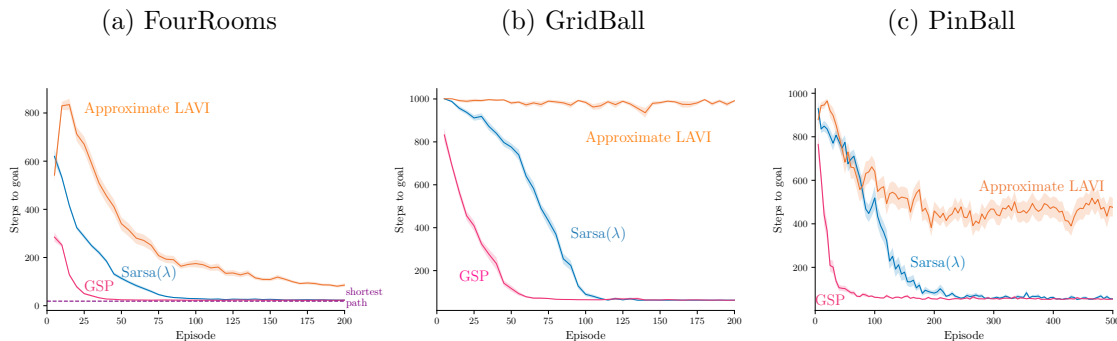
Figure 16: Five episode moving average of return in FourRooms, GridBall and PinBall. Curves are averaged over 30 runs where the shaded region is one standard error.

where a model, or simulator, is given. Similar to GSP, the algorithm plans only over a set of landmarks (subgoals). They assume that they have options that terminate near the landmarks, and do value iteration with the simulator by executing options from only the landmarks. The greedy policy for a state uses the computed values for landmark states by selecting the option that takes the agent to the best landmark state, and using options to move only between landmark states from there. The planning is much more efficient, because the number of landmark states is relatively small, but the policies are suboptimal.

We could similarly use $v_{g^\star}$, by running the option to bring the agent to the best nearby subgoal. However, a more direct comparison in our setting is to use the modified TD error update above. We call this update Approximate LAVI, to recognize the similarity to this elegant algorithm. In all environments, the approximate LAVI learner either learns much slower or converges to a sub-optimal policy instead, as shown in Figure 16.

In our preliminary experiments, we had investigated an update rule that partially bootstraps off $v_{g^\star}$. Namely, we used a TD error of $R_{t+1} + \gamma_{t+1}(\beta v_{g^\star}(S_{t+1}) + (1-\beta)q(S_{t+1}, A_{t+1})) - q(S_t, A_t)$, where $\beta \in [0, 1]$. Potential based reward shaping with $v_{g^\star}$ was found to outperform this technique. We discuss this more in Appendix F.

7. Relationships to Other Model-based Approaches

Now that we have detailed the GSP algorithm, we contrast it to other approaches for background planning. In this section we first provide an overview of related work to better place GSP amongst the large literature of related ideas, beyond background planning. Then we contrast GSP to Dyna and Dyna with options, which are two natural approaches to background planning. Finally, we provide a short discussion around efficient planning—a key property of GSP—and why it is desirable for background planning approaches.

7.1 Related Work

A variety of approaches have been developed to handle issues with learning and iterating one-step models. Several papers have shown that using forward model simulations can produce simulated states that result in catastrophically misleading values (van Hasselt et al., 2019; Lambert et al., 2022; Aminmansour et al., 2024). This problem has been tackled by

using reverse models (Pan et al., 2018; van Hasselt et al., 2019; Aminmansour et al., 2024); primarily using the model for decision-time planning (van Hasselt et al., 2019; Silver et al., 2008; Chelu et al., 2020); and improving training strategies to account for accumulated errors in rollouts (Talvitie, 2014; Venkatraman et al., 2015; Talvitie, 2017). An emerging trend is to avoid approximating the true transition dynamics, and instead learn dynamics tailored to predicting values on the next step correctly (Farahmand et al., 2017; Farahmand, 2018; Ayoub et al., 2020; Rakhsha et al., 2022). This trend is also implicit in the variety of techniques that encode the planning procedure into neural network architectures that can then be trained end-to-end (Tamar et al., 2016; Silver et al., 2017; Oh et al., 2017; Weber et al., 2017; Farquhar et al., 2018; Schrittwieser et al., 2020). We similarly attempt to avoid issues with iterating models, but do so by considering a different type of model.

Current deep model-based RL techniques plan in a lower-dimensional abstract space where the relevant features from the original high-dimensional experience are preserved, often refered to as a *latent space.* MuZero (Schrittwieser et al., 2020), for example, embeds the history of observations to then use predictive models of values, policies and one-step rewards. Using these three predictive models in the latent space guides MuZero's Monte Carlo Tree Search without the need for a perfect simulator of the environment. Most recently, DreamerV3 demonstrated the capabilities of a discrete latent world model in a range of pixel-based environments (Hafner et al., 2023). There is growing evidence that it is easier to learn accurate models in a latent space.

Temporal abstraction has also been considered to make planning more efficient, through the use of hierarchical RL and/or options. MAXQ Dietterich (2000) introduced the idea of learning hierarchical policies with multiple levels, breaking up the problem into multiple subgoals. A large literature followed, focused on efficient planning with hierarchical policies (Diuk et al., 2006) and using a hierarchy of MDPs with state abstraction and macro-actions (Bakker et al., 2005; Konidaris et al., 2014; Konidaris, 2016; Gopalan et al., 2017). See Gopalan et al. (2017) for an excellent summary.

Rather than using a hierarchy and planning only in abstract MDPs, another strategy is simply to add options as additional (macro) actions in planning, still also including primitive actions. Similar ideas were explored before the introduction of options (Singh, 1992; Dayan and Hinton, 1992). There has been some theoretical characterization of the utility of options for improving convergence rates of value iteration (Mann and Mannor, 2014) and sample efficiency (Brunskill and Li, 2014), though also hardness results reflecting that the augmented MDP is not necessarily more efficient to solve (Zahavy et al., 2020) and hardness results around discovering options efficient for planning (Jinnai et al., 2019). Empirically, incorporating options into planning has largely only been tested in tabular settings (Sutton et al., 1999; Singh et al., 2004; Wan et al., 2021). Recent work has considered mechanisms for identifying and learning option policies for planning under function approximation (Sutton et al., 2022), but as yet did not consider issues with learning the models.

There has been some work using options for planning that is more similar to GSP, using only one-level of abstraction and restricting planning to the abstract MDP. Hauskrecht et al. (2013) proposed to plan only in the abstract MDP with macro-actions (options) and abstract states corresponding to the boundaries of the regions spanned by the options, which is like restricting abstract states to subgoals. Bagaria et al. (2021) discover skills to construct discrete graph abstractions of continuous state and action spaces with subgoal nodes and

option policy edges. The most similar to our work is LAVI, which restricts value iteration to a small subset of landmark states (Mann et al., 2015).[3] These methods also have similar flavors to using a hierarchy of MDPs, in that they focus planning in a smaller space and (mostly) avoid planning at the lowest level, obtaining significant computational speed-ups. The key distinction to GSP is that we are not in the traditional planning setting where a model is given; in our online setting, the agent needs to learn the model from interaction.

The use of landmark states has also been explored in *goal-conditioned RL*, where the agent is given a desired goal state or states. This is a problem setting where the aim is to learn a policy $\pi(a|s,g)$ that can be conditioned on different possible goals. The agent learns for a given set of goals, with the assumption that at the start of each episode the goal state is explicitly given to the agent. After this training phase, the policy should generalize to previously unseen goals. Naturally, this idea has particularly been applied to navigation, having the agent learn to navigate to different states (goals) in the environment. The first work to exploit the idea of landmark states in GCRL was for learning and using universal value function approximators (UVFAs) (Huang et al., 2019). The UVFA conditions action-values on both state-action pairs as well as landmark states. The agent can reach new goals by searching on a learned graph between landmark states, to identify which landmark to moves towards. A flurry of work followed, still in the goal-conditioned setting (Nasiriany et al., 2019; Emmons et al., 2020; Zhang et al., 2020, 2021; Aubret et al., 2021; Hoang et al., 2021; Gieselmann and Pokorny, 2021; Kim et al., 2021; Dubey et al., 2021).

Some of this work focused on exploiting landmark states for planning in GCRL. Huang et al. (2019) used landmark states as interim subgoals, with a graph-based search to plan between these subgoals (Huang et al., 2019). The policy is set to reach the nearest goal (using action-values with cost-to-goal rewards of -1 per step) and learned distance functions between states and goals and between goals. These models are like our reward and discount models, but tailored to navigation and distances. Nasiriany et al. (2019) built on this idea, introducing an algorithm called Latent Embeddings for Abstracted Planning (LEAP), that using gradient descent to search for a sequence of subgoals in a latent space.

The idea of learning models that immediately apply to new subtasks using successor features is like GCRL, but does not explicitly use landmark states. The option keyboard involves encoding options (or policies) as vectors that describe the corresponding (pseudo) reward (Barreto et al., 2019). This work has been expanded more recently, using successor features (Barreto et al., 2020). New policies can then be easily obtained for new reward functions, by linearly combining the (basis) vectors for the already learned options. However no planning is involved in that work, beyond a one-step decision-time choice amongst options.

7.2 Contrasting GSP to Dyna and Dyna with Options

In this section we contrast GSP to *Dyna* and *Dyna with Options* (Mihucz, 2022), which are two canonical approaches to do background planning. Dyna involves learning a transition model and updating it with simulated experience in the background. The original version of Dyna simply uses one-step transitions from observed states, making it look quite similar to

---

3. A similar idea to landmark states has been considered in more classical AI approaches, under the term bi-level planning (Wolfe et al., 2010; Hogg et al., 2010; Chitnis et al., 2022). These techniques are quite different from Dyna-style planning—updating values with (stochastic) dynamic programming updates—and so we do not consider them further here.

Algorithm 3 Dyna with Options using the DDQN update

Initialize base learner parameters $\mathbf{w}, \mathbf{w}_{\text{targ}} = \mathbf{w}_0$, $n_{\text{updates}} = 0$, model parameters $\boldsymbol{\theta}$, search-control queue P, target refresh rate τ, set of options Π

Sample initial state s_0 from the environment and store s_0 in P

for $t \in 0, 1, 2, ...$ **do**

    Take action a_t (e.g., ϵ-greedy on q, where if select $\pi \in \Pi$, then choose $a_t \sim \pi(\cdot | s_t)$)

    Observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$ and store s_{t+1} in P

    Update_Models$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$ (see Algorithm 6)

    for n_{main} mini-batches **do**

        Sample batch of states B from P

        For each $s \in B$, pick a random $\tilde{a}$ from $\mathcal{A} \cup \Pi$

        // if $\tilde{a}$ is an option, s' is an outcome state after many steps,

        // r is a discounted sum of rewards under the option until termination

        // and γ is the discount raised to the number of steps that option executes

        Query model at each $(s, \tilde{a})$ to get outcome s', r, γ and corresponding target

        $Y = r + \gamma q(s', \text{argmax}_{a' \in \mathcal{A} \cup \Pi} q(s', a'; \mathbf{w}); \mathbf{w}_{\text{targ}})$

        $L = \frac{1}{|B|} \sum_{(s,\tilde{a},Y) \in B} (Y - q(s, \tilde{a}; \mathbf{w}))^2$

        $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L$

        if modulo$(n_{\text{updates}}, \tau) == 0$ **then** $\mathbf{w}_{\text{targ}} \leftarrow \mathbf{w}$

        $n_{\text{updates}} = n_{\text{updates}} + 1$

---

experience replay. Experience replay can actually be viewed as a limited, non-parametric version of Dyna, and often Dyna and replay perform similarly (Pan et al., 2018; van Hasselt et al., 2019), without more focused *search control* (the process of selecting which states to query the model from). To truly obtain the benefits of the model with Dyna, it is key to consider which (s, a) is the most useful to update from, which may even be a hypothetical (s, a) never observed. Querying the model from such an unseen (s, a) leverages the generalization capabilities of the model much more than simply querying the model from an observed (s, a). A clever search control strategy could likely significantly improve Dyna, but it is also a hard problem. Very few search-control techniques have been proposed in the literature (Moore and Atkeson, 1993; Wingate et al., 2005; Pan et al., 2019).

If we go beyond one-step transitions, then we further deviate from replay and can benefit from having an explicit learned model. As mentioned above, Dyna with rollouts can suffer from model iteration error. An alternative approach is to incorporate options into Dyna. This extension was first proposed for the tabular setting (Singh et al., 2004), with little follow-up work beyond a recent re-investigation still in the tabular setting (Sutton et al., 2022). The idea behind Dyna with options is to treat options like macro-actions in the planning loop. Let us consider the one-step transition dynamics model, for a given π, where the model outputs $\tilde{s}', \tilde{r}, \tilde{\gamma}$ from (s, π). The model outputs a possible outcome state $\tilde{s}'$ after executing the option from s. The outputted reward $\tilde{r}$ from (s, π) is the discounted cumulative sum of rewards of the option π when starting from s, until termination. The outputted $\tilde{\gamma}$ is the discounted probability of terminating. For example, if the option always terminated after n steps, then the model would output $\tilde{\gamma} = \gamma^n$. We show a possible variant of Dyna and Dyna with options, again using a similar update to DDQN, in Algorithm 3.
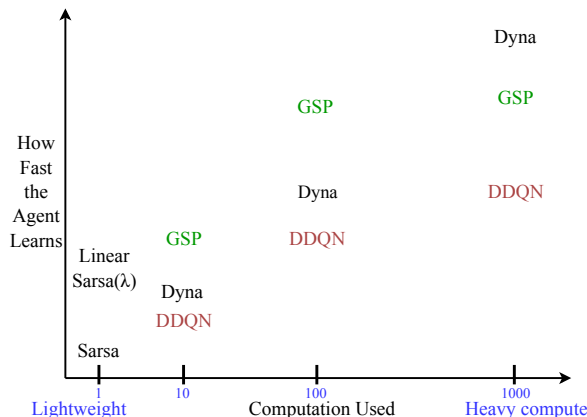
Figure 17: A visualization of the hypothetical trade-off between computation and how quickly the agent learns for different algorithms. This plot is focused on computation, rather than dealing with model errors, so Dyna means Dyna with a highly accurate model. Lightweight algorithms like Sarsa, that update only with the most recent sample, cannot leverage more computation to improve learning. DDQN, GSP and Dyna can leverage more computation by increasing the amount of replay and planning steps. Of course, this diagram is completely hypothetical, but reflects the thinking that guides this work as well as anticipated behavior of these algorithms. GSP should be more effective than Dyna with less compute, but is likely to plateau at a slightly suboptimal point. With a lot of compute, Dyna with an accurate model is effectively doing dynamic programming and extracting a policy from the model. But with much less compute, it does not efficiently focus that compute to improve the policy. DDQN is limited by the limited data that it has in its buffer, and unlike Dyna, cannot reason about possible outcomes outside of this dataset.

Dyna with Options should allow for faster value propagation than Dyna alone. It effectively uses multi-step updates rather than single-step updates. However, it actually requires learning an even more complex model than Dyna, since it must learn the transition-dynamics for the options as well as the transition dynamics for the primitive actions. Moreover, it does still plan over all states and actions; again without smarter search-control, planning is likely to still be inefficient. In this sense, the variants of Dyna and Dyna with options presented here do not satisfy two key desiderata: feasible model learning and efficient planning. We discuss the importance of efficient planning in more depth in the next section.

Remark: The well-versed reader may be confused why we consider Dyna on states, rather than on a latent state, also called agent state. Such a change is likely to make it more feasible to learn the model and make planning more efficient. Nonetheless, we are still stuck planning in a continuous latent space that is likely to have 32 dimensions, or more, based on typical design choices. It reduces, but does not remove, these issues.

7.3 The Importance of Efficient Planning

GSP is designed to allow for efficient planning. We want changes in the environment to quickly propagate through the value function. This is achieved by focusing planning over a small set of subgoals. The local subgoal models can be updated efficiently, and value

iteration can be used to get the new subgoal values. Value iteration for a small set of states is efficient: the agent can perform many value iteration updates per step to keep these subgoal values accurate. Replay then propagates these subgoal values to the state values.

Standard replay, Dyna, and even Dyna with Options do not have the same computational efficiency due to the lack of higher-level planning. In practice, with a bounded agent, poor computational efficiency can result in poor sample efficiency. The learned model might even be perfectly accurate, and with unlimited computation per step, the agent could obtain the perfect value function. But with a computational budget—for example with a budget of ten planning steps per step—it may fail to transfer its (immense) knowledge about the world into the policy. Eventually, over many steps (environment interactions), it will get an accurate value function. An algorithm, on the other hand, that can more quickly transfer knowledge from its model to the value function will get closer to the true action-values in a smaller number of environment steps. We visualize this conceptual trade-off in Figure 17, for DDQN (namely replay), Dyna, and GSP (layered on top of DDQN).

8. Discussion and Limitations

In this paper, we introduced a new planning framework, called Goal-Space Planning (GSP). This new approach uses background planning to improve value propagation, with minimalist, local models and computationally efficient planning. The primary focus of our experiments was to understand the utility of this planning framework, but there remain several open technical questions.

Limitations of the formalism: The biggest limitation, highlighted upfront and throughout the paper, is our reliance on a reasonable subgoal discovery mechanism. In this paper, we provided the GSP algorithm with subgoals. In some settings, GSP could be used today, where expert knowledge makes it straightforward to specify subgoals for a problem. Ultimately, though, we aim to design learning systems that discover appropriate subgoals themselves. We did find that subgoal placement was important for performance, and this algorithmic gap is the largest to fill to make GSP practically useful.

Limitations of the empirical study: In our experiments we made use of pretrained subgoal models to focus the empirical study on the utility of GSP planning. The first key question we wanted to answer was, given suitable discovery and subgoal model learning approaches, would we even gain benefit from how GSP plans in abstract space and projects to the low-level space. We found a clear affirmative to this question. Moving to the next step will likely require some improvements to our subgoal model learning algorithms, because learning UVFAs off-policy and in parallel remains an ongoing research question.

In this work, we highlight the inherent limitations with Dyna and Dyna with options, to motivate the design of GSP, but do not empirically compare to these algorithms. We do point out that a replay-based algorithm, like DDQN, can be seen as a non-parametric version of Dyna, and often model inaccuracies cause the model-based variants to perform worse than simply using replay. However, explicitly comparing to Dyna (with options) and contrasting model learnability would make this argument even stronger.

Despite these limitations, this work provides a promising approach to using partial models for planning which is robust to model inaccuracy. The result from (abstract) planning is only used to define the potential function for reward shaping, guiding the main policy learner

with minimal bias, even under model inaccuracy. Partial models—our subgoal conditioned models—allow us to avoid modeling everything, allowing for more judicious choices in how we use our function approximation capacity. Further, there is a wealth of literature on learning value functions off-policy, which are precisely the types of algorithms needed to learn our subgoal-conditioned algorithms.

Acknowledgements

Author Contributions

All empirical results in this work were run by Kevin Roice, Parham Mohammad Panahi, and Scott Jordan. The theoretical results and algorithm development for GSP were by Chunlok Lo, Martha White, Adam White, Farzane Aminmansour, and Gábor Mihucz. The reward-shaping theory was a product of a discussion between Scott Jordan and Kevin Roice. Martha White and Adam White helped with the discussion, and experimental design and advised the project. Kevin Roice, Scott Jordan, Adam White, and Martha White are responsible for the writing.

References

Farzane Aminmansour, Taher Jafferjee, Ehsan Imani, Erin Talvitie, Michael Bowling, and Martha White. Mitigating Value Hallucination in Dyna-Style Planning via Multistep Predecessor Models. *Journal of Artificial Intelligence Research*, 2024.

Arthur Aubret, Laetitia Matignon, and Salima Hassas. DisTop: Discovering a Topological representation to learn diverse and rewarding skills. *arXiv:2106.03853*, 2021.

Alex Ayoub, Zeyu Jia, Csaba Szepesvári, Mengdi Wang, and Lin Yang. Model-Based Reinforcement Learning with Value-Targeted Regression. In *International Conference on Machine Learning*, 2020.

Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, 2021.

Bram Bakker, Zoran Zivkovic, and Ben Krose. Hierarchical dynamic programming for robot path planning. In *International Conference on Intelligent Robots and Systems*, 2005.

André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver, and Hado van Hasselt. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

Andre Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan Hunt, Shibl Mourad, David Silver, and Doina Precup. The

Option Keyboard: Combining Skills in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2019.

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48), 2020.

Emma Brunskill and Lihong Li. PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *International Conference on Machine Learning*, 2014.

Veronica Chelu, Doina Precup, and Hado P van Hasselt. Forethought and hindsight in credit assignment. In *Advances in Neural Information Processing Systems*, 2020.

Ching-An Cheng, Xinyan Yan, and Byron Boots. Trajectory-wise Control Variates for Variance Reduction in Policy Gradient Methods. In *3rd Annual Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, 2019.

Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning. In *International Conference on Intelligent Robots and Systems*. IEEE, 2022.

Peter Dayan and Geoffrey E Hinton. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 1992.

Thomas G Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 2000.

Carlos Diuk, Alexander L Strehl, and Michael L Littman. A hierarchical approach to efficient reinforcement learning in deterministic domains. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2006.

Rohit K. Dubey, Samuel S. Sohn, Jimmy Abualdenien, Tyler Thrash, Christoph Hoelscher, André Borrmann, and Mubbasir Kapadia. SNAP:Successor Entropy based Incremental Subgoal Discovery for Adaptive Navigation. In *Motion, Interaction and Games*, 2021.

Scott Emmons, Ajay Jain, Misha Laskin, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse Graphical Memory for Robust Planning. In *Advances in Neural Information Processing Systems*, 2020.

Amir-massoud Farahmand. Iterative Value-Aware Model Learning. In *Advances in Neural Information Processing Systems*, 2018.

Amir-massoud Farahmand, Andre M S Barreto, and Daniel N Nikovski. Value-Aware Loss Function for Model-based Reinforcement Learning. In *International Conference on Artificial Intelligence and Statistics*, 2017.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2018.

Robert Gieselmann and Florian T. Pokorny. Planning-Augmented Hierarchical Reinforcement Learning. *IEEE Robotics and Automation Letters*, 6(3), 2021.

Nakul Gopalan, Michael Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, Lawson Wong, et al. Planning with abstract markov decision processes. In *International Conference on Automated Planning and Scheduling*, 2017.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering Diverse Domains through World Models. *arXiv:2301.04104*, 2023.

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Uncertainty in Artificial Intelligence*, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, 2015.

Christopher Hoang, Sungryull Sohn, Jongwook Choi, Wilka Carvalho, and Honglak Lee. Successor Feature Landmarks for Long-Horizon Goal-Conditioned Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2021.

Chad Hogg, U. Kuter, and Hector Muñoz-Avila. Learning Methods to Generate Good Plans: Integrating HTN Learning and Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2010.

Jiawei Huang and Nan Jiang. From Importance Sampling to Doubly Robust Policy Gradient. In *International Conference on Machine Learning*, 2020.

Zhiao Huang, Fangchen Liu, and Hao Su. Mapping State Space using Landmarks for Universal Goal Reaching. In *Advances in Neural Information Processing Systems*, 2019.

Yuu Jinnai, David Abel, David Hershkowitz, Michael Littman, and George Konidaris. Finding Options that Minimize Planning Time. In *International Conference on Machine Learning*, 2019.

George H John. When the best move isn't optimal: Q-learning with Exploration. In *AAAI conference on Artificial Intelligence*, 1994.

Khimya Khetarpal, Zafarali Ahmed, Gheorghe Comanici, David Abel, and Doina Precup. What can I do here? A Theory of Affordances in Reinforcement Learning. In *International Conference on Machine Learning*, 2020.

Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-Guided Subgoal Generation in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2021.

George Konidaris. Constructing abstraction hierarchies using a skill-symbol loop. In *International Joint Conference on Artificial Intelligence*, 2016.

George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Constructing symbolic representations for high-level planning. In *AAAI Conference on Artificial Intelligence*, 2014.

George D. Konidaris and Andrew G. Barto. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In *Advances in Neural Information Processing Systems*, 2009.

Nathan Lambert, Kristofer Pister, and Roberto Calandra. Investigating Compounding Prediction Errors in Learned Dynamics Models. *arXiv:2203.09637*, 2022.

Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *International Conference on Machine Learning*, pages 127–135, 2014.

Timothy A. Mann, Shie Mannor, and Doina Precup. Approximate Value Iteration with Temporally Extended Actions. *Journal of Artificial Intelligence Research*, 53, 2015.

Amy McGovern and Andrew G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *International Conference on Machine Learning*, 2001.

Gábor Mihucz. Dyna with Options: Incorporating Temporal Abstraction into Planning. 2022.

Andrew W. Moore and Christopher G. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time. *Machine learning*, 13(1), 1993.

Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with Goal-Conditioned Policies. In *Advances in Neural Information Processing Systems*, 2019.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In *International Conference on Machine Learning*, 1999.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value Prediction Network. In *Advances in Neural Information Processing Systems*, 2017.

Yangchen Pan, Muhammad Zaheer, Adam White, Andrew Patterson, and Martha White. Organizing Experience: A Deeper Look at Replay Mechanisms for Sample-Based Planning in Continuous State Domains. In *International Joint Conference on Artificial Intelligence*, 2018.

Yangchen Pan, Hengshuai Yao, Amir-Massoud Farahmand, and Martha White. Hill Climbing on Value Estimates for Search-Control in Dyna. In *International Joint Conference on Artificial Intelligence*, 2019.

Sergey Pankov. Reward-Estimation Variance Elimination in Sequential Decision processes. *arXiv:1811.06225*, 2018.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning. *arXiv:2304.01315*, 2020.

Roger Penrose. A Generalized Inverse for Matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3), 1955.

Amin Rakhsha, Andrew Wang, Mohammad Ghavamzadeh, and Amir-massoud Farahmand. Operator Splitting Value Iteration. *Advances in Neural Information Processing Systems*, 2022.

Gavin A. Rummery. *Problem Solving with Reinforcement Learning*. PhD thesis, University of Cambridge, 1995.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *International Conference on Machine Learning*, 2015.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839), 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347*, 2017.

David Silver, Richard S. Sutton, and Martin Müller. Sample-Based Learning and Search with Permanent and Transient Memories. In *International Conference on Machine Learning*, 2008.

David Silver, Hado 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. In *International Conference on Machine Learning*, 2017.

Satinder Singh. Scaling Reinforcement Learning algorithms by learning Variable Temporal Resolution Models. In *Machine Learning Proceedings 1992*, pages 406–415. Elsevier, 1992.

Satinder Singh, Andrew Barto, and Nuttapong Chentanez. Intrinsically Motivated Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2004.

Martin Stolle and Doina Precup. Learning Options in Reinforcement Learning. In *Abstraction, Reformulation, and Approximation*, 2002.

Richard S. Sutton. Integrated Modeling and Control based on Reinforcement Learning and Dynamic Programming. In *Advances in Neural Information Processing Systems*, 1991.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1-2), 1999.

Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *International Conference on Autonomous Agents and Multiagent Systems*, 2011.

Richard S. Sutton, Rupam A. Mahmood, and Martha White. An Emphatic Approach to the Problem of Off-Policy Temporal-Difference Learning. *The Journal of Machine Learning Research*, 2016.

Richard S. Sutton, Marlos C. Machado, G. Zacharias Holland, David Szepesvári, Finbarr Timbers, Brian Tanner, and Adam White. Reward-Respecting Subtasks for Model-Based Reinforcement Learning. *Artificial Intelligence*, 324, 2022.

Erin Talvitie. Model Regularization for Stable Sample Roll-Outs. In *Uncertainty in Artificial Intelligence*, 2014.

Erin Talvitie. Self-Correcting Models for Model-Based Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2017.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In *Advances in Neural Information Processing Systems*, 2016.

Hado van Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, 2010.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

Hado van Hasselt, Matteo Hessel, and John Aslanides. When to use Parametric Models in Reinforcement Learning? In *Advances in Neural Information Processing Systems*, 2019.

Arun Venkatraman, Martial Hebert, and J. Andrew Bagnell. Improving Multi-Step Prediction of Learned Time Series Models. In *AAAI Conference on Artificial Intelligence*, 2015.

Yi Wan, Muhammad Zaheer, Adam White, Martha White, and Richard S. Sutton. Planning with Expectation Models. In *International Joint Conference on Artificial Intelligence*, 2019.

Yi Wan, Abhishek Naik, and Richard S. Sutton. Average-Reward Learning and Planning with Options. In *Advances in Neural Information Processing Systems*, 2021.

Theophane Weber, Sebastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2017.

Martha White. Unifying Task Specification in Reinforcement Learning. In *International Conference on Machine Learning*, 2017.

Eric Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 1992.

David Wingate, Kevin D. Seppi, and Cs Byu Edu. Prioritization Methods for Accelerating MDP Solvers. *Journal of Machine Learning Research*, 2005.

Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined Task and Motion Planning for Mobile Manipulation. *International Conference on Automated Planning and Scheduling*, 2010.

Tom Zahavy, Avinatan Hasidim, Haim Kaplan, and Yishay Mansour. Planning in Hierarchical Reinforcement Learning: Guarantees for Using Local Policies. In *International Conference on Algorithmic Learning Theory*, 2020.

Lunjun Zhang, Ge Yang, and Bradly C. Stadie. World Model as a Graph: Learning Latent Landmarks for Planning. In *International Conference on Machine Learning*, 2021.

Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2020.

Appendix A. Starting Simpler: Goal-Space Planning for Policy Evaluation

To highlight the key idea for efficient planning, we provide an example of GSP in a simpler setting: policy evaluation for learning v^π for a fixed policy π assuming access to the true models. For this setting, we do not have subgoal policies, since we are not trying to learn to get to subgoals quickly, we are just trying to reason about values when executing π. This setting highlights the key idea of propagating values quickly across the space by updating between *subgoals*, $g \in \mathcal{G} \subset \mathcal{S}$, as visualized in Figure 18. (In general, note that $\mathcal{G}$ need not be a subset of $\mathcal{S}$, we can have abstract subgoal vectors that need not correspond to any state.) To do so, we need temporally extended models between pairs g, g' that may be further than one-transition apart. For policy evaluation, these models are the accumulated rewards $r_{\pi,\gamma} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ and discounted probabilities $P_{\pi,\gamma} : \mathcal{S} \times \mathcal{S} \to [0,1]$ under π:

$$r_{\pi,\gamma}(g, g') \doteq \mathbb{E}_\pi[R_{t+1} + \gamma_{g',t+1} r_{\pi,\gamma}(S_{t+1}, g')|S_t = g]$$
$$P_{\pi,\gamma}(g, g') \doteq \mathbb{E}_\pi[1(S_{t+1} = g')\gamma_{t+1} + \gamma_{g',t+1} P_{\pi,\gamma}(S_{t+1}, g')|S_t = g]$$

where $\gamma_{g',t+1} = 0$ if $S_{t+1} = g'$ and otherwise equals γ_{t+1}, the environment discount. If we cannot reach g' from g under π, then $P_{\pi,\gamma}(g, g')$ will simply accumulate many zeros and be zero.

We first give an example for the deterministic setting, where both the environment and policy are deterministic, since it more clearly provides the desired intuition. For completeness, we do also show how these arguments extend to the stochastic setting, and provide proofs of convergence for both.

A.1 GSP for Policy Evaluation with Deterministic Environments

Assume π is deterministic and the MDP is deterministic. We can treat $\mathcal{G}$ as our new state space and plan in this space, to get value estimates v for all $g \in \mathcal{G}$

$$v(g) = r_{\pi,\gamma}(g, g') + P_{\pi,\gamma}(g, g')v(g') \quad \text{where } g' = \text{argmax}_{g' \in \bar{\mathcal{G}}} P_{\pi,\gamma}(g, g')$$

where $\bar{\mathcal{G}} = \mathcal{G} \cup \{s_{\text{terminal}}\}$ if there is a terminal state (episodic problems) and otherwise $\bar{\mathcal{G}} = \mathcal{G}$. It is straightforward to show this converges, because $P_{\pi,\gamma}$ is a substochastic matrix (see Appendix A.3). We use $g' = \text{argmax}_{g' \in \bar{\mathcal{G}}} P_{\pi,\gamma}(g, g')$ since it is the closest subgoal under π; because we have a deterministic policy and environment, this is the first subgoal we will see when executing π from g.

Once we have these values, we can propagate these to other states, locally, again using the closest g to s. We can do so by noticing that the above definitions can be easily extended to $r_{\pi,\gamma}(s, g')$ and $P_{\pi,\gamma}(s, g')$, since for a pair (s, g) they are about starting in the state s and reaching g under π.

$$v(s) = r_\gamma(s, g) + P_{\pi,\gamma}(s, g)v(g) \quad \text{where } g = \text{argmax}_{g \in \bar{\mathcal{G}}} P_{\pi,\gamma}(s, g).$$

Because the rhs of this equation is fixed, we only cycle through these states once to get their values.

All of this might seem like a lot of work for policy evaluation; indeed, it will be more useful to have this formalism for control. But, even here goal-space planning can be beneficial.
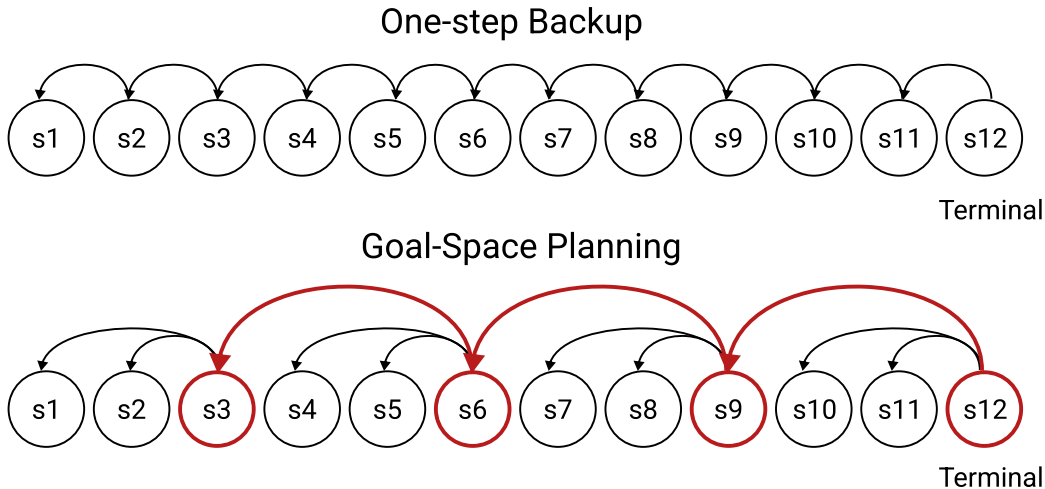
Figure 18: Comparing one-step backup with Goal-Space Planning when subgoals are concrete states. GSP first focuses planning over a smaller set of subgoals (in red), then updates the values of individual states.

Let assume a chain $s_1, s_2, \ldots, s_n$, where $n = 1000$ and $\mathcal{G} = \{s_{100}, s_{200}, \ldots, s_{1000}\}$. Planning over $g \in \mathcal{G}$ only requires sweeping over 10 states, rather than 1000. Further, we have taken a 1000 horizon problem and converted it into a 10 step one.[4] As a result, changes in the environment also propagate faster. If the reward at s' changes, locally the reward model around s' can be updated quickly, to change $r_{\pi,\gamma}(g, g')$ for pairs g, g' where s' is along the way from g to g'. This local change quickly updates the values back to earlier $\tilde{g} \in \mathcal{G}$.

A.2 Extension to Policy Evaluation Setting for Stochastic Environments

All the above used a deterministic policy, since the equations actually end up better matching the control setting we ultimately care about, so help with our primary goal of providing intuition. But we can redo the above with stochastic environments and stochastic policies. We simply need to also have P_π as well as a $P_{\pi,\gamma}$. Conveniently, P_π can actually be extracted from $P_{\pi,\gamma}$, assuming we have a fixed $\gamma_{t+1} = \gamma_c$ for our environment, (except for at termination where it is zero). Specifically, we want to take the expectation over the outcome g' we could reach under π, which means we need to define

$$r_{\pi,\gamma}(g) = \sum_{g' \in \bar{\mathcal{G}}} P_\pi(g, g') r_{\pi,\gamma}(g, g') \quad \text{where } P_\pi(g, g') = \frac{P_{\pi,\gamma}(g, g')}{\sum_{g' \in \bar{\mathcal{G}}} P_{\pi,\gamma}(g, g')}$$

where we overload the notation $r_{\pi,\gamma}$ since it is clear from the inputs which variant we are refering to. Then we have the more standard policy evaluation update, for stochastic

---

4. In this simplified example, we can plan efficiently by updating the value at the end in s_n, and then updating states backwards from the end. But, without knowing this structure, it is not a general purpose strategy. For general MDPs, we would need smart ways to do search control: the approach to pick states from one-step updates. In fact, we can leverage search control strategies to improve the goal-space planning step. Then we get the benefit of these approaches, as well as the benefit of planning over a much smaller state space.

environments and stochastic policies

$$v(g) = r_{\pi,\gamma}(g) + \sum_{g' \in \bar{\mathcal{G}}} P_{\pi,\gamma}(g, g') v(g')$$

It is straightforward to show that this converges, by showing that $P_{\pi,\gamma}$ is a substochastic matrix. We assume throughout that the environment discount γ_{t+1} is a constant $\gamma_c \in [0, 1)$ for every step in an episode, until termination when it is zero. The below results can be extended to the case where $\gamma_c = 1$, using the standard strategy for the stochastic shortest path problem setting.

Proposition 1 *Assume we are given a (potentially stochastic) policy π, $\gamma_c < 1$, a discrete set of subgoals $\mathcal{G} \subset \mathcal{S}$, and that we iteratively update $v_t \in \mathbb{R}^{|\bar{\mathcal{G}}|}$ with the dynamic programming update*

$$v_t(g) = r_{\pi,\gamma}(g) + \sum_{g' \in \bar{\mathcal{G}}} P_{\pi,\gamma}(g, g') v_{t-1}(g')$$

for all $g \in \mathcal{G}$, starting from an arbitrary (finite) initialization $v_0 \in \mathbb{R}^{|\bar{\mathcal{G}}|}$, with $v_t(s_{terminal})$ fixed at zero. Then v_t converges to a fixed-point v.

Proof To analyze this as a matrix update, we need to extend $P_{\pi,\gamma}(g, g')$ to include an additional row transitioning from $g = s_{\text{terminal}}$ to other subgoals. This row is all zeros, because the value in the terminal state is always fixed at zero. Without this additional row, for $n = |\mathcal{G}|$, we have that $P_{\pi,\gamma} \in [0, 1]^{n \times (n+1)}$, because $|\bar{\mathcal{G}}| = n + 1$. With the addition of the row, $P_{\pi,\gamma} \in [0, 1]^{(n+1) \times (n+1)}$. Note that there are ways to avoid introducing terminal states, using transition-based discounting (White, 2017), but for this work it is actually simpler to explicitly reason about them and reaching them from subgoals.

To show this we simply need to ensure that $P_{\pi,\gamma}$ is a substochastic matrix. Recall that

$$P_{\pi,\gamma}(g, g') \doteq \mathbb{E}_\pi[1(S_{t+1} = g')\gamma_{t+1} + \gamma_{g',t+1} P_{\pi,\gamma}(S_{t+1}, g') | S_t = g]$$

where $\gamma_{g',t+1} = 0$ if $S_{t+1} = g'$ and otherwise equals γ_{t+1}, the environment discount. If it is substochastic, namely it satisfies $\|P_{\pi,\gamma}\|_2 < 1$, then the Bellman operator

$$(Bv)(g) = r_{\pi,\gamma}(g) + P_{\pi,\gamma}(g, g')\tilde{v}(g')$$

is a contraction, because $\|Bv_1 - Bv_2\|_2 = \|P_{\pi,\gamma}v_1 - P_{\pi,\gamma}v_2\|_2 \le \|P_{\pi,\gamma}\|_2 \|v_1 - v_2\|_2 < \|v_1 - v_2\|_2$.

Because $\gamma_c < 1$, then either g immediately terminates in g', giving $1(S_{t+1} = g')\gamma_{t+1} + \gamma_{g',t+1}P_{\pi,\gamma}(S_{t+1}, g') = \gamma_{t+1} + 0 \le \gamma_c$. Or, it does not immediately terminate, and $1(S_{t+1} = g')\gamma_{t+1} + \gamma_{g',t+1}P_{\pi,\gamma}(S_{t+1}, g') = 0 + \gamma_c P_{\pi,\gamma}(S_{t+1}, g') \le \gamma_c$ because $P_{\pi,\gamma}(S_{t+1}, g') \le 1$. Therefore, if $\gamma_c < 1$, then $\|P_{\pi,\gamma}\|_2 \le \gamma_c$. ∎

A.3 Proof for the Deterministic Policy Evaluation Setting

We provide proof here for the policy evaluation update we used for a deterministic policy and deterministic MDP.

Corollary 2 *Assume that we have a deterministic MDP, deterministic policy π, $\gamma_c < 1$, a discrete set of subgoals $\mathcal{G} \subset \mathcal{S}$, and that we iteratively update $v_t \in \mathbb{R}^{|\bar{\mathcal{G}}|}$ with the dynamic programming update*

$$v_t(g) = r_{\pi,\gamma}(g, g') + P_{\pi,\gamma}(g, g')v_{t-1}(g') \quad \text{where } g' = \underset{g' \in \bar{\mathcal{G}}}{\text{argmax}} \, P_{\pi,\gamma}(g, g')$$

for all $g \in \mathcal{G}$, starting from an arbitrary (finite) initialization $v_0 \in \mathbb{R}^{|\bar{\mathcal{G}}|}$, with $v_t(s_{terminal})$ fixed at zero. Then then v_t converges to a fixed-point.

Proof This follows from Proposition 1, by defining a new substochastic matrix $\bar{P}(g, g') = P_{\pi,\gamma}(g, g')$ for $g' = \text{argmax}_{g' \in \bar{\mathcal{G}}} P_{\pi,\gamma}(g, g')$ and $\bar{P}(g, g') = 0$ for all other g'. This new $\bar{P}(g, g')$ simply zeros out the parts that are not used in the update, and only reflects the path the policy takes through the subgoals. Because π and the environment are deterministic, we know that this does not change the connectivity structure and the policy will still reach all the parts of the environment that it otherwise would, including the terminal state. This new matrix remains substochastic, because we only modified it by zeroing out entries, ensuring all row sums are no larger than before. ■

Remark: Both the stochastic and deterministic updates described for policy evaluation result in a $v(g) = v_\pi(g)$ for $g \in \mathcal{S}$ and the corresponding $v_{g^\star}(s) = v_\pi(s)$. This is because in the stochastic setting, we did not use initiation sets and option policies and so did not restrict connectivity between subgoals, nor between states and subgoals. In the deterministic setting, the subgoals simply break up the deterministic trajectory followed by the policy, to show how we propagate value. This simplified GSP approach here results in optimal values, which is not reflective of the control setting.

Appendix B. Proofs for the General Control Setting

In this section we assume that $\gamma_c < 1$, to avoid some of the additional issues for handling proper policies. The same strategies apply to the stochastic shortest path setting with $\gamma_c = 1$, with additional assumptions.

Proposition 3 *[Convergence of Value Iteration in Goal-Space] Assuming that $\tilde{\Gamma}$ is a substochastic matrix, with $v_0 \in \mathbb{R}^{|\bar{\mathcal{G}}|}$ initialized to an arbitrary value and fixing $v_t(s_{terminal}) = 0$ for all t, then iteratively sweeping through all $g \in \mathcal{G}$ with update*

$$v_t(g) = \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')v_{t-1}(g')$$

convergences to a fixed-point.

Proof We can use the same approach typically used for value iteration. For any $v_0 \in \mathbb{R}^{|\bar{\mathcal{G}}|}$, we can define the operator

$$(B^g v)(g) \doteq \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')\tilde{v}(g')$$

First we can show that B^g is a γ_c-contraction. Assume we are given any two vectors v_1, v_2. Notice that $\tilde{\Gamma}(g, g') \leq \gamma_c$, because for our problem setting the discount is either equal to γ_c or equal to zero at termination. Then we have that for any $g \in \bar{\mathcal{G}}$

$$
\begin{aligned}
&|(B^g v_1)(g) - (B^g v_2)(g)| \\
&= \left| \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')v_1(g') - \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')v_2(g') \right| \\
&\leq \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} |\tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')v_1(g') - (\tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')v_2(g'))| \\
&= \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} |\tilde{\Gamma}(g, g')(v_1(g') - v_2(g'))| \\
&\leq \max_{g' \in \bar{\mathcal{G}}: \tilde{d}(g,g')>0} \gamma_c |v_1(g') - v_2(g')| \\
&\leq \gamma_c \|v_1 - v_2\|_\infty
\end{aligned}
$$

Since this is true for any g, it is true for the max over g, giving

$$
\|B^g v_1 - B^g v_2\|_\infty \leq \gamma_c \|v_1 - v_2\|_\infty.
$$

Because the operator B^g is a contraction, since $\gamma_c < 1$, we know by the Banach Fixed-Point Theorem that the fixed-point exists and is unique. ∎

Remark: Note here that even though we call this the general control setting, there does not seem to be any actions! In typical value iteration, we would have models $r(s, a, s')$ and $P(s'|s, a)$. To do value iteration, we have to maximize over the action, which dictates which s' we reach. Here, implicitly our actions are actually the option policies for reaching subgoals. Each action is specialized to reach a subgoal, and so we can directly reason about reaching g' in our update.

Appendix C. Theoretical Results with Potential-Based Reward Shaping

This section extends the theoretical results of potential-based reward shaping. In Section C.1, we show settings where potential-based reward shaping preserves the same optimal policy with function approximation, and settings that change the optimal policy. In Section C.2, we show how potential based reward shaping on Sarsa(λ) is equivalent to initializing its action-value function to the potential.

C.1 Preservation of Optimal Policies

Potential-based reward shaping is a useful tool to modify reward functions because it does not change the optimal policy. However, this result is only true of the set of all policies and not necessarily true for the set of functions that can be represented by a class of approximators. When we apply potential-based reward shaping to GSP we want to know if we are changing the optimal policy the agent can represent. In this section, we prove that when the potential function Φ is in the same linear function class as q, that there is no change in optimal policy within that class. When Φ is in a different (potentially arbitrary) class, then we cannot make any statements to guarantee the optimum will not be changed.

First recall that the TD fixed-point for linear function approximation is

$$\mathbf{w}_{\mathrm{TD}} = \underbrace{\mathbb{E}\left[\mathbf{x}(S_t, A_t)\left(\gamma\mathbf{x}(S_{t+1}, A_{t+1}) - \mathbf{x}(S_t, A_t)\right)^{\top}\right]^{-1}}_{=\mathbf{A}_{\mathrm{TD}}}\mathbb{E}\left[\mathbf{x}(S_t, A_t)R_{t+1}\right],$$

where $\mathbf{x}(s, a) \in \mathbb{R}^n$, $q(s, a; \mathbf{w}) = \mathbf{w}^{\top}\mathbf{x}(s, a)$, γ is a constant discount factor, and the expectation is taken with respect to the on policy state distribution (Sutton and Barto, 2018, Chapter 9.4). Similarly, the optimal weights for minimizing mean squared error to Monte Carlo returns is

$$\mathbf{w}_{\mathrm{MC}} = \underbrace{\mathbb{E}\left[\mathbf{x}(S_t, A_t)\mathbf{x}(S_t, A_t)^{\top}\right]^{-1}}_{=\mathbf{A}_{\mathrm{MC}}}\mathbb{E}\left[\mathbf{x}(S_t, A_t)G_{t+1}\right].$$

In both these cases, the matrices $\mathbf{A}_{\mathrm{TD}}, \mathbf{A}_{\mathrm{MC}} \in \mathbb{R}^{n\times n}$ are independent of the rewards. So the fixed-point, $\mathbf{w}_{\mathrm{TD}}'$, and optimal weights, $\mathbf{w}_{\mathrm{MC}}'$, using the shaped rewards $R_{t+1} + \gamma\Phi(S_{t+1}) - \Phi(S_t)$ are

$$\mathbf{w}_{\mathrm{TD}}' = \underbrace{\mathbf{A}_{\mathrm{TD}}^{-1}\mathbb{E}\left[\mathbf{x}(S_t, A_t)R_{t+1}\right]}_{=\mathbf{w}_{\mathrm{TD}}} + \mathbf{A}_{\mathrm{TD}}^{-1}\mathbb{E}\left[\mathbf{x}(S_t, A_t)\left(\gamma\Phi(S_{t+1}) - \Phi(S_t)\right)\right],$$

$$\mathbf{w}_{\mathrm{MC}}' = \underbrace{\mathbf{A}_{\mathrm{MC}}^{-1}\mathbb{E}\left[\mathbf{x}(S_t, A_t)G_{t+1}\right]}_{=\mathbf{w}_{\mathrm{MC}}} - \mathbf{A}_{\mathrm{MC}}^{-1}\mathbb{E}\left[\mathbf{x}(S_t, A_t)\Phi(S_t)\right].$$

These fixed-points are different than the ones without potential-based shaping, but we only care if the policy derived from $q(s, a; \mathbf{w}')$ is different than $q(s, a; \mathbf{w})$ for both TD and MC. Without knowing something about $\mathbf{x}(s, a)$ or Φ we cannot, in general say if the policy will be better or worse using potential-based shaping. For the specific case where Φ can be written as a linear function of $\mathbf{x}(s, a)$, we can say the set of policies at the fixed-point or local minimum remain unchanged with potential-based reward shaping.

Theorem 4 *If $\exists \boldsymbol{\theta} \in \mathbb{R}^n$ such that $\forall s, a$ $\Phi(s) = \mathbf{x}(s, a)^{\top}\boldsymbol{\theta}$, then the set of policies expressed by q at a fixed-point remain unchanged under potential-based reward shaping, i.e.,*

$$\forall s, \ \forall a, a', \ q(s, a; \mathbf{w}_{\mathrm{TD}}') \geq q(s, a'; \mathbf{w}_{\mathrm{TD}}') \implies q(s, a; \mathbf{w}_{\mathrm{TD}}) \geq q(s, a'; \mathbf{w}_{\mathrm{TD}}), \ and$$

$$\forall s, \ \forall a, a', \ q(s, a; \mathbf{w}_{\mathrm{MC}}') \geq q(s, a'; \mathbf{w}_{\mathrm{MC}}') \implies q(s, a; \mathbf{w}_{\mathrm{MC}}) \geq q(s, a'; \mathbf{w}_{\mathrm{MC}}).$$

Proof We first show that at the TD fixed-points, action values are only offset by $-\Phi(s)$.

$$
\begin{aligned}
q(s, a; \mathbf{w}_{\mathrm{TD}}') &= \mathbf{x}(s, a)^\top \mathbf{w}_{\mathrm{TD}}' \\
&= \mathbf{x}(s, a)^\top \left(\mathbf{w}_{\mathrm{TD}} + \mathbf{A}_{\mathrm{TD}}^{-1} \mathbb{E} \left[\mathbf{x}(S_t, A_t) \left(\gamma \Phi(S_{t+1}) - \Phi(S_t) \right) \right] \right) \\
&= \mathbf{x}(s, a)^\top \mathbf{w}_{\mathrm{TD}} + \mathbf{x}(s, a)^\top \mathbf{A}_{\mathrm{TD}}^{-1} \mathbb{E} \left[\mathbf{x}(S_t, A_t) \left(\gamma \Phi(S_{t+1}) - \Phi(S_t) \right) \right] \\
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \mathbf{x}(s, a)^\top \mathbf{A}_{\mathrm{TD}}^{-1} \mathbb{E} \left[\mathbf{x}(S_t, A_t) \left(\gamma \Phi(S_{t+1}) - \Phi(S_t) \right) \right] \\
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \mathbf{x}(s, a)^\top \mathbf{A}_{\mathrm{TD}}^{-1} \mathbb{E} \left[\mathbf{x}(S_t, A_t) \left(\gamma \mathbf{x}(S_{t+1}, A_{t+1})^\top \boldsymbol{\theta} - \mathbf{x}(S_t, A_t)^\top \boldsymbol{\theta} \right) \right] \\[2mm]
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \mathbf{x}(s, a)^\top \mathbf{A}_{\mathrm{TD}}^{-1} \mathbb{E} \left[\mathbf{x}(S_t, A_t) \left(\gamma \mathbf{x}(S_{t+1}, A_{t+1}) - \mathbf{x}(S_t, A_t) \right)^\top \boldsymbol{\theta} \right] \\
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \mathbf{x}(s, a)^\top \mathbf{A}_{\mathrm{TD}}^{-1} \underbrace{\mathbb{E} \left[\mathbf{x}(S_t, A_t) \left(\gamma \mathbf{x}(S_{t+1}, A_{t+1}) - \mathbf{x}(S_t, A_t) \right)^\top \right]}_{=\mathbf{A}_{\mathrm{TD}}} \boldsymbol{\theta} \\
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \mathbf{x}(s, a)^\top \mathbf{A}_{\mathrm{TD}}^{-1} \mathbf{A}_{\mathrm{TD}} \boldsymbol{\theta} \\
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \mathbf{x}(s, a)^\top \boldsymbol{\theta} \\
&= q(s, a; \mathbf{w}_{\mathrm{TD}}) + \Phi(s).
\end{aligned}
$$

By a similar process, we show the same is true for the Monte Carlo value function estimate.

$$
\begin{aligned}
q(s, a; \mathbf{w}_{\mathrm{MC}}') &= \mathbf{x}(s, a)^\top \mathbf{w}_{\mathrm{MC}}' \\
&= \mathbf{x}(s, a)^\top \left(\mathbf{w}_{\mathrm{MC}} - \mathbf{A}_{\mathrm{MC}}^{-1} \mathbb{E} \left[\mathbf{x}(S_t, A_t) \Phi(S_t) \right] \right).
\end{aligned}
$$

∎

Remark 5 *Note that these statements are only about the fixed-points under the evaluation setting. When applied to the control setting, potential-based reward shaping can drastically change how the agent explores; see Sections 6.3 and G.1.*

The above results show that if both q and Φ come from the same linear function space, that the optimal policy via q-estimate if unchanged. However, if Φ comes from a different space (possibly nonlinear), then there is no guarantee that the optimal policy-based on q is unchanged. We illustrate this property with an example below.

Consider the four state MDP (shown in Figure 19) with states s_1, s_2, s_3, s_4, and actions a_1 and a_2. The initial state distribution is: $d_0(s_1) = 0.5$, $d_0(s_2) = 0.5$. The rewards are zero except for $r(s_3, a_1) = 1$, $r(s_3, a_2) = 2$, $r(s_4, a_1) = 2$, and $r(s_4, a_2) = 1$. The transition probabilities are displayed in Table 1. The function approximator is unable to distinguish any states, i.e., $\forall s, s\prime, a, \ q(s, a, \mathbf{w}) = q(s', a, \mathbf{w})$. We consider two potential functions: Φ' and Φ'', both can observe the state. The values for these potential functions are shown in Table 2. The policies for this function approximator is the limited to having the same distribution of actions in every state. The optimal policy is to take action a_1 in every state; see Figure 20 (left).

We examine the TD fixed-points of the function approximator for different policies and potential functions. We display the difference of q for each action using the TD fixed-point
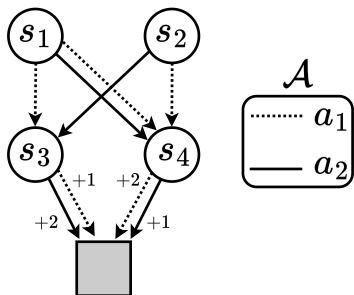
Figure 19: Illustration of MDP, where the dotted and solid lines represent transitions from taking actions a_1 and a_2 respectively. Numbers indicate nonzero rewards.

$$p(s_3|s_1, a_1) = 0.5 \quad p(s_4|s_1, a_1) = 0.5$$
$$p(s_4|s_1, a_2) = 1.0$$
$$p(s_3|s_2, a_2) = 1.0 \quad p(s_4|s_2, a_1) = 1.0$$
$$p(s_\infty|s_3, \cdot) = 1 \quad p(s_\infty|s_4, \cdot) = 1.0.$$

Table 1: Transition dynamics of the MDP

$$\Phi'(s_1) = +0 \quad \Phi''(s_1) = +0$$
$$\Phi'(s_2) = +0 \quad \Phi''(s_2) = +0$$
$$\Phi'(s_3) = +3 \quad \Phi''(s_3) = -1$$
$$\Phi'(s_4) = -1 \quad \Phi''(s_4) = -3$$

Table 2: The two potential functions consider for this MDP.

for different policies in Figure 20 (right). This example shows that we can construct Φ that make the optimal action preferable for some policies and not others, or make it so the optimal action is not preferable for any policy. This means we cannot directly apply potential-based reward shaping to function approximation and know we will not change the resulting policy found with TD methods.
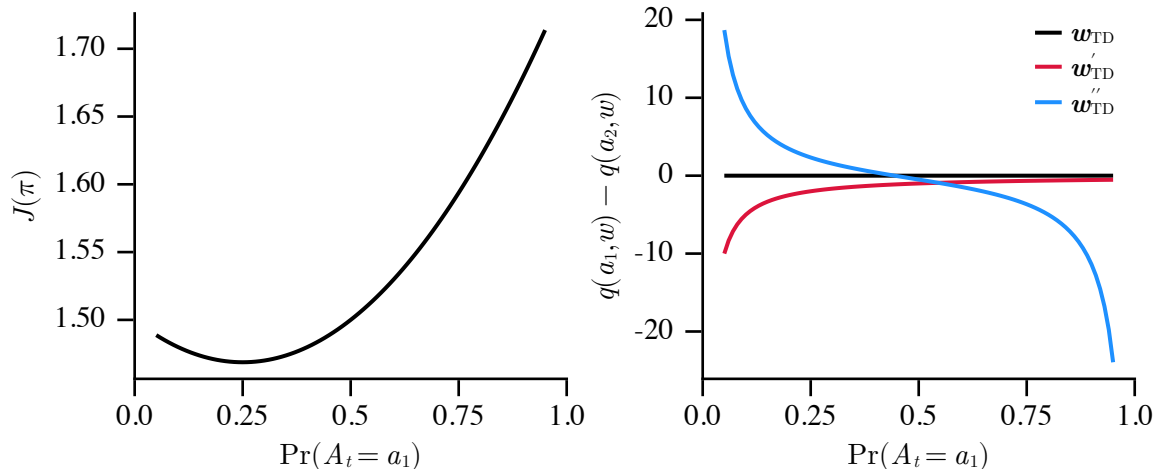

Figure 20: (left) This plot shows $J(\pi) \doteq \mathbb{E}[G_0]$ for each policy having the same action distribution in every state. (right) This figure shows the difference in q approximation for actions a_1 and a_2, when using the TD-fixed-point for each policy. The weights $\mathbf{w}_{\text{TD}}$ are the fixed-point when no potential shaping is used. Weights $\mathbf{w}_{\text{TD}}'$ and $\mathbf{w}_{\text{TD}}''$ correspond to the fixed-points when using Φ' and Φ'' respectively. When $q(a_1, \mathbf{w}) - q(a_2, \mathbf{w}) > 0$ then action a_1 (the optimal action) is preferred. Notice that Φ' and Φ'' can make it so a_1 is not preferred.

C.2 Shaping and Q-Function Initialization

Besides the convergence point, in the tabular setting, it is known that using PBRS is equivalent to initializing Q to Φ and then performing updates on the same set of experience (Wiewiora, 2003). While Wiewiora explicitly showed this for tabular Q-learning and stated this shaping-initialization equivalence extends to all TD learners, in this paper we explicitly show it for tabular Sarsa(λ).

Proposition 6 *Given the same sequence of experience, performing TD(λ) updates with potential-based reward shaping is equivalent to adding the potential to the learner's initial action values and updating using the unshaped rewards, in the tabular setting.*

Proof We first explicitly show this result for Sarsa(λ), one of the algorithms we use to empirically analyze GSP with shaping, and then show how it extends to all TD learners.

We start with two Sarsa(λ) learners L and L', with Q-tables Q_t and Q'_t. L will perform Sarsa(λ) updates with PBRS, whereas L' will have its Q-table initialized as $Q'_0(s,a) = Q_0(s,a) + \Phi(s)$ and it will use unshaped rewards. $\Phi : \mathcal{S} \mapsto \mathbb{R}$ is the potential function. Our experiences are stored as a list of 5-tuples $\mathcal{D} = \{\langle S_i, A_i, R_{i+1}, S_{i+1}, \gamma_{i+1}\rangle\}_{i=0}^{n-1}$. Both learners will use this same list of experiences.

For tabular Sarsa(λ), the update rule for an experience $\langle s, a, r, s', \gamma \rangle$ is:

$$\mathbf{z}_t(s,a) = 1 \quad \text{(replacing trace)}$$
$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha \delta_t \mathbf{z}_t$$
$$\mathbf{z}_{t+1} \leftarrow \lambda\gamma\mathbf{z}_t,$$

and we use

$$\delta_t = r + \gamma\Phi(s') - \Phi(s) + \gamma Q_t(s',a') - Q_t(s,a),$$
$$\delta'_t = r + \gamma Q'_t(s',a') - Q'_t(s,a)$$

as the TD errors for L, and L' respectively. $\mathbf{z}_t \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ is the eligibility trace vector [5]. We denote the change in the Q-tables after k updates from initialization as $\Delta Q_k = \sum_{t=0}^{k-1} \alpha \delta_t \mathbf{z}_t$ and $\Delta Q'_k = \sum_{t=0}^{k-1} \alpha \delta'_t \mathbf{z}_t$. Since both learners use the same list of experience, γ and λ, they would have the same eligibility trace vector $\mathbf{z}_t \,\forall\, t$. We initialise $\mathbf{z}_{-1} = \mathbf{0}$.
For the theorem to be true, we require

$$\Delta Q_t = \Delta Q'_t \,\forall\, t.$$

We show this using a proof by induction.
Base Case: When $t = 1$,

$$\Delta Q_1 = \alpha\delta_0\mathbf{z}_0$$
$$= \alpha \begin{bmatrix} R_1 + \gamma\Phi(S_{t+1}) - \Phi(S_t) \\ +\gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \end{bmatrix} \mathbf{z}_0,$$

---

5. It is a memory mechanism that, on each learning update, gives a decaying amount of credit for state-action pairs that occurred previously. The λ in Sarsa(λ) determines the rate of this decay with time (Sutton and Barto, 2018).

$$\Delta Q_1' = \alpha \delta_0' \mathbf{z}_0$$
$$= \alpha[R_1 + \gamma Q'(S_{t+1}, A_{t+1}) - Q'(S_t, A_t)]\mathbf{z}_0$$
$$= \alpha \begin{bmatrix} R_1 \\ +\gamma(Q(S_{t+1}, A_{t+1}) + \Phi(S_{t+1})) \\ -(Q(S_t, A_t) + \Phi(S_t)) \end{bmatrix} \mathbf{z}_0$$
$$= \alpha \begin{bmatrix} R_1 + \gamma\Phi(S_{t+1}) - \Phi(S_t) \\ +\gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \end{bmatrix} \mathbf{z}_0$$
$$= \Delta Q_1.$$

The changes to the Q tables are equivalent after 1 update.

Assumption : $\exists \ k \in \mathbb{N}$ s.t. $\Delta Q_k = \Delta Q_k'$.

Inductive Step: When $t = k + 1$, the learner L updates with experience $\langle s, a, r, s', \gamma \rangle$.

$$\Delta Q_{k+1} = \Delta Q_k + \alpha \delta_k \mathbf{z}_k$$
$$= \Delta Q_k + \alpha[r + \gamma\Phi(s') - \Phi(s) + \gamma Q_k(s', a') - Q_k(s, a)]\mathbf{z}_k$$
$$= \Delta Q_k + \alpha \begin{bmatrix} r + \gamma\Phi(s') - \Phi(s) \\ +\gamma(Q_0(s', a') + \Delta Q_k(s', a')) \\ -Q_0(s, a) - \Delta Q_k(s, a) \end{bmatrix} \mathbf{z}_k.$$

The third line was possible because $Q_k = Q_0 + \sum_{t=0}^{k-1} \alpha \delta_t \mathbf{z}_t$ (i.e. Q_k is the initalization plus $k - 1$ updates). Whereas learner L' updates as:

$$\Delta Q_{k+1}' = \Delta Q_k' + \alpha \delta_k' \mathbf{z}_k'$$
$$= \Delta Q_k' + \alpha[r + \gamma Q_k'(s', a') - Q_k'(s, a)]\mathbf{z}_k'$$
$$= \Delta Q_k' + \alpha \begin{bmatrix} r \\ +\gamma(Q_0(s', a') + \Phi(s') + \Delta Q_k'(s', a')) \\ -Q_0(s, a) - \Phi(s) - \Delta Q_k'(s, a) \end{bmatrix} \mathbf{z}_k'$$
$$= \Delta Q_k' + \alpha \begin{bmatrix} r + \gamma\Phi(s') - \Phi(s) \\ +\gamma(Q_0(s', a') + \Delta Q_k'(s', a')) \\ -Q_0(s, a) - \Delta Q_k'(s, a) \end{bmatrix} \mathbf{z}_k'.$$

By our assumption, $\Delta Q_k = \Delta Q_k'$. Furthermore as $\mathbf{z}_k = \mathbf{z}_k'$, we see that $(3.2) = (3.3)$.

$$\implies \Delta Q_{k+1} = \Delta Q_{k+1}'.$$

So if $\Delta Q_k = \Delta Q_k'$, we have shown that $\Delta Q_{k+1} = \Delta Q_{k+1}'$. Since we have shown that $\Delta Q_0 = \Delta Q_0'$, by induction $\Delta Q_t = \Delta Q_t' \ \forall \ t \in \mathbb{N}$.

More generally, this holds for any TD-learner. This can be seen if we consider the TD errors,

$$\delta_t = r + \gamma\Phi(s') - \Phi(s) + \gamma \mathcal{C}Q_t(s', \cdot) - Q_t(s, a) \text{ and}$$
$$\delta_t' = r + \gamma \mathcal{C}Q_t(s', \cdot) - Q_t'(s, a)$$

Where $\mathcal{C}Q_t(s', \cdot) = \sum_{a' \in \mathcal{A}} \alpha_{a'} Q_t(s', a')$ denotes a convex combination over actions. This includes learners like Expected Sarsa, $\alpha_{a'} = \Pr(A_{t+1} = a')$, (John, 1994) and Q-learning,

$$\alpha_{a'} = \begin{cases} 1 & a' \in \text{argmax}_{a \in \mathcal{A}} Q(s', a), \\ 0 & \text{otherwise.} \end{cases} \qquad \blacksquare$$

Algorithm 4 Goal-Space Planning for Any Base Learner

Assume given subgoals $\mathcal{G}$ and relevance function d

Initialize base learner (i.e. $\mathbf{w}, \mathbf{z} = \mathbf{0}, \mathbf{0}$ for Sarsa(λ) [6]), model parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}^r, \boldsymbol{\theta}^\Gamma, \boldsymbol{\theta}^\pi), \tilde{\boldsymbol{\theta}} = (\tilde{\boldsymbol{\theta}}^r, \tilde{\boldsymbol{\theta}}^\Gamma)$

Sample initial state s_0 from the environment

for $t \in 0, 1, 2, ...$ **do**

    Take action a_t using q (e.g., ϵ-greedy), observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$

    Choose a' from s_{t+1} using q (e.g. ϵ-greedy)

    `Update_Models`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$ (see Algorithm 6)

    `Planning`() (see Algorithm 2)

    `MainPolicyUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1}, a')$ // Changes with base learner

---

Appendix D. GSP Algorithm Details and Pseudocode

This section includes more details on the GSP algorithm, including pseudocode. The overall pseudocode for GSP is in Algorithm 4 for any base learner. The Sarsa base learner update is given in Algorithm 5, the DDQN variant is given in the main text in Algorithm 1. The subgoal model learning pseudocode is in Algorithm 6. We explain in detail where this pseudocode comes from in the next two subsections.

Note that we overload the definitions of the subgoal models. We learn action-value variants $r_\gamma(s, a, g; \boldsymbol{\theta}^r)$, with parameters $\boldsymbol{\theta}^r$, to avoid importance sampling corrections, as discussed in Section D.2. We learn the option-policy using action-values $\tilde{q}(s, a; \boldsymbol{\theta}^\pi)$ with parameters $\boldsymbol{\theta}^\Gamma$, and so query the policy using $\pi_g(s; \boldsymbol{\theta}^\pi) \doteq \operatorname{argmax}_{a \in \mathcal{A}} \tilde{q}(s, a, g; \boldsymbol{\theta}^\pi)$. The policy π_g is not directly learned, but rather defined by $\tilde{q}$. Similarly, we do not directly learn $r_\gamma(s, g)$; instead, it is defined by $r_\gamma(s, a, g; \boldsymbol{\theta}^r)$. Specifically, for model parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}^r, \boldsymbol{\theta}^\Gamma, \boldsymbol{\theta}^\pi)$, we set $r_\gamma(s, g; \boldsymbol{\theta}) \doteq r_\gamma(s, \pi_g(s; \boldsymbol{\theta}^\pi), g; \boldsymbol{\theta}^r)$ and $\Gamma(s, g; \boldsymbol{\theta}) \doteq \Gamma(s, \pi_g(s; \boldsymbol{\theta}^\pi), g; \boldsymbol{\theta}^\Gamma)$. We query these derived functions in the pseudocode.

We provide a potential implementation of model learning—which includes option policy and subgoal model learning—in Algorithm 6, by using DDQN to learn the GVFs. Because the models are composed of options and GVFs, we could have used any number of different off-policy learning algorithms. In our own experiments, we learned these models offline, and so did not use this pseudocode in Algorithm 6 (the approach we used is detailed in Section E). An important next step for GSP is to better understand which algorithms to use to learn the options and GVFs, off-policy, from one stream of data. We provide the pseudocode in Algorithm 6 as a concrete suggestion, even though it was not tested in this paper.

Finally, recall we assume access to a given set of subgoals, which is why Algorithm 2 takes these as inputs. But there have been several natural ideas already proposed for option discovery, that nicely apply in our more constrained setting. An early, and often cited idea, is to use bottleneck states (McGovern and Barto, 2001). For our setting, a simpler idea might be a great place to start: using subgoals that are often visited by the agent (Stolle and Precup, 2002). We also discuss how we could use a notion of reachability, below when discussing learning the relevance function d.

---

6. Sarsa(λ) has two sets of parameters to initialize: its action-value function weights $\mathbf{w}$, and its eligibility trace vector $\mathbf{z}$ (Rummery, 1995).

Algorithm 5 `MainPolicyUpdate for Sarsa`$(s, a, s', r, \gamma, a')$

// For a Sarsa(λ) base learner

$v_{g^\star} \leftarrow \max_{g \in \bar{\mathcal{G}}:d(s,g)>0} r_\gamma(s, g; \boldsymbol{\theta}) + \Gamma(s, g; \boldsymbol{\theta})\tilde{v}(g)$

$\delta \leftarrow r + \gamma v_{g^\star}(s') - v_{g^\star}(s) + \gamma q(s', a'; \mathbf{w}) - q(s, a; \mathbf{w})$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}\nabla_{\mathbf{w}}q(s, a; \mathbf{w})$

$\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla_{\mathbf{w}}q(s, a; \mathbf{w})$

Algorithm 6 `Update_Models`(s, a, s', r, γ)

Add new transition (s, a, s', r, γ) to buffer D_{model}

for $g' \in \bar{\mathcal{G}}$ **do**

    for n_{model} mini-batches **do**

        Sample batch $B_{\mathrm{model}} = \{(s, a, r, s', \gamma)\}$ from D_{model} where $d(s, g) > 0$

        $\gamma_{g'} \leftarrow \gamma(1 - m(s', g'))$

        // Update option policy

        $a' \leftarrow \operatorname{argmax}_{a' \in \mathcal{A}} \tilde{q}(s', a', g'; \boldsymbol{\theta}^\pi)$

        $\delta^\pi(s, a, s', r, \gamma) \leftarrow \frac{1}{2}(r - 1) + \gamma_{g'}\tilde{q}(s', a', g'; \boldsymbol{\theta}^\pi_{\mathrm{targ}}) - q(s, a, g'; \boldsymbol{\theta}^\pi)$

        $\boldsymbol{\theta}^\pi \leftarrow \boldsymbol{\theta}^\pi - \alpha^\pi\nabla_{\boldsymbol{\theta}^\pi}\frac{1}{|B_{\mathrm{model}}|}\sum_{(s,a,r,s',\gamma)\in B_{\mathrm{model}}}(\delta^\pi)^2$

        // Update reward model and discount model

        $\delta^r(s, a, r, s', \gamma) \leftarrow r + \gamma_{g'}r_\gamma(s', a', g'; \boldsymbol{\theta}^r{}_{\mathrm{targ}}) - r_\gamma(s, a, g'; \boldsymbol{\theta}^r)$

        $\delta^\Gamma(s, a, r, s', \gamma) \leftarrow m(s', g)\gamma + \gamma_{g'}\Gamma(s', a', g'; \boldsymbol{\theta}^\Gamma{}_{\mathrm{targ}}) - \Gamma(s, a, g'; \boldsymbol{\theta}^\Gamma)$

        $\boldsymbol{\theta}^r \leftarrow \boldsymbol{\theta}^r - \alpha^r\nabla_{\boldsymbol{\theta}^r}\frac{1}{|B_{\mathrm{model}}|}\sum_{(s,a,r,s',\gamma)\in B_{\mathrm{model}}}(\delta^r)^2$

        $\boldsymbol{\theta}^\Gamma \leftarrow \boldsymbol{\theta}^\Gamma - \alpha^\Gamma\nabla_{\boldsymbol{\theta}^\Gamma}\frac{1}{|B_{\mathrm{model}}|}\sum_{(s,a,r,s',\gamma)\in B_{\mathrm{model}}}(\delta^\Gamma)^2$

        if $n_{\mathrm{updates}}\%\tau == 0$ **then**

            $\boldsymbol{\theta}^\pi{}_{\mathrm{targ}} \leftarrow \boldsymbol{\theta}^\pi$

            $\boldsymbol{\theta}^r{}_{\mathrm{targ}} \leftarrow \boldsymbol{\theta}^r$

            $\boldsymbol{\theta}^\Gamma{}_{\mathrm{targ}} \leftarrow \boldsymbol{\theta}^\Gamma$

        $n_{\mathrm{updates}} = n_{\mathrm{updates}} + 1$

    // Update goal-to-goal models using state-to-goal models

    for each g such that $m(s, g) > 0$ **do**

        $\tilde{\boldsymbol{\theta}}^r \leftarrow \tilde{\boldsymbol{\theta}}^r + \tilde{\alpha}^r(r_\gamma(s, g'; \boldsymbol{\theta}) - \tilde{r}_\gamma(g, g'; \tilde{\boldsymbol{\theta}}^r))\nabla_{\boldsymbol{\theta}^r}\tilde{r}_\gamma(g, g'; \tilde{\boldsymbol{\theta}}^r)$

        $\tilde{\boldsymbol{\theta}}^\Gamma \leftarrow \tilde{\boldsymbol{\theta}}^\Gamma + \tilde{\alpha}^\Gamma(\Gamma(s, g'; \boldsymbol{\theta}) - \tilde{\Gamma}(g, g'; \tilde{\boldsymbol{\theta}}^r))\nabla_{\boldsymbol{\theta}^\Gamma}\tilde{\Gamma}(g, g'; \tilde{\boldsymbol{\theta}}^\Gamma)$

D.1 Option Policy Learning

The option policies solve subproblems that can be seen as episodic tasks. The set of start states are $s \in \mathcal{S}$ s.t. $d(s, g) > 0$, which in this case is the whole region from which the policy could be started, not just a small set of start states. Termination occurs when $m(s, g) > 0$. Our goal is to find policies that reach termination, while also maximizing reward along the way.

However, there is one important difference to the standard episodic setting: maximizing environment reward may be at odds with reaching the subgoal in a reasonable number of steps (or at all). For example, in environments where the reward is always positive,

maximizing environment reward might encourage the option policy not to terminate. It is not always the case that positive rewards result in option policies that do not terminate. If there is a large, positive reward at the subgoal in the environment, and $\gamma_c < 1$, then the agent may get a higher return by reaching this subgoal sooner, even if there are positive rewards along the way. On the other hand, if the rewards are always negative, then the option policy will terminate, trying to find the path with the best (but still negative) return.

At the same time, we do not just want to focus on reaching the subgoal; we want π_g to reach g, while also obtaining the best return along the way to g. For example, if there is a lava pit along the way to a goal, even if going through the lava pit is the shortest path, we want the learned option to get to the goal by going around the lava pit. We therefore want to be reward-respecting, as introduced for reward-respecting subtasks (Sutton et al., 2022).

As yet, we do not have a complete approach for this issue. But, here we provide a simple one as a reasonable starting point. We can consider a spectrum of option policies that range from the policy that reaches the goal as fast as possible to one that focuses on environment reward. We can specify a new reward for learning the option: $\tilde{R}_{t+1} = cR_{t+1} + (1-c)(-1)$. When $c = 0$, we have a cost-to-goal problem, where the learned option policy should find the shortest path to the goal, regardless of reward along the way. When $c = 1$, the option policy focuses on environment reward, but may not terminate in g. We can start by learning the option policy that takes the shortest path with $c = 0$, and the corresponding $r_\gamma(s, g), \Gamma(s, g)$. The constant c can be increased until π_g stops going to the goal, or until the discounted probability $\Gamma(s, g)$ drops below a specified threshold.

For this work, we propose a simple default, where we fix $c = 0.5$. Adaptive approaches, such as the idea described above, are left to future work. The resulting algorithm to learn π_g involves learning a separate value function for these rewards. We can learn action-values (or a parameterized policy) using the above reward. For example, we can learn a policy with the Q-learning update to action-values $\tilde{q}$

$$\delta = cR_{t+1} + c - 1 + \gamma_{g,t+1} \max_{a'} \tilde{q}(S_{t+1}, a', g) - \tilde{q}(S_t, A_t, g)$$

Then we can set π_g to be the greedy policy, $\pi_g(s) = \text{argmax}_{a \in \mathcal{A}} \tilde{q}(s, a, g)$.

D.2 Learning the Subgoal Models

Now we need a way to learn the state-to-subgoal models, $r_\gamma(s, g)$ and $\Gamma(s, g)$. These can both be expressed as General Value Functions (GVFs) (Sutton et al., 2011),

$$\Gamma(s, g) = \mathbb{E}_{\pi_g} \left[\sum_{k=0}^{\infty} \left(\prod_{k'=0}^{k} \gamma_{t+k'+1} \right) m(S_{t+1}, g) \Big| S_t = s \right],$$

$$r_\gamma(s, g) = \mathbb{E}_{\pi_g} \left[\sum_{k=0}^{\infty} \left(\prod_{k'=0}^{k} \gamma_{t+k'+1} \right) R_{t+k+1} \Big| S_t = s \right],$$

and we leverage this form to use standard algorithms in RL to learn them.

These GVFs (the models) need to be learned off-policy, from one stream of data according to a behavior policy b. We can either use importance sampling or we can learn the action-value variants of these models to avoid importance sampling. We describe both options here.

Off-policy Model Update using Importance Sampling   We can update $r_\gamma(\cdot, g)$ with an importance-sampled temporal difference (TD) learning update $\rho_t \delta_t \nabla r_\gamma(S_t, g)$ where $\rho_t = \frac{\pi_g(a|S_t)}{b(a|S_t)}$ and

$$\delta_t^r = R_{t+1} + \gamma_{g,t+1} r_\gamma(S_{t+1}, g) - r_\gamma(S_t, g)$$

The discount model $\Gamma(s, g)$ can be learned similarly, because it is also a GVF with cumulant $m(S_{t+1}, g)\gamma_{t+1}$ and discount $\gamma_{g,t+1}$. The TD update is $\rho_t \delta_t^\Gamma$ where

$$\delta_t^\Gamma = m(S_{t+1}, g)\gamma_{t+1} + \gamma_{g,t+1}\Gamma(S_{t+1}, g) - \Gamma(S_t, g)$$

All of the above updates can be done using any off-policy GVF algorithm, including those using clipping of IS ratios and gradient-based methods, and can include replay.

Off-policy Model Update without Importance Sampling   Overloading notation, let us define the action-value variants $r_\gamma(s, a, g)$ and $\Gamma(s, a, g)$. We get similar updates to above, now redefining

$$\delta_t^r = R_{t+1} + \gamma_{g,t+1} r_\gamma(S_{t+1}, \pi_g(S_{t+1}), g) - r_\gamma(S_t, A_t, g)$$

and using update $\delta_t \nabla r_\gamma(S_t, A_t, g)$. For Γ we have

$$\delta_t^\Gamma = m(S_{t+1}, g)\gamma_{t+1} + \gamma_{g,t+1}\Gamma(S_{t+1}, \pi_g(S_{t+1}), g) - \Gamma(S_t, A_t, g)$$

We then define $r_\gamma(s, g) \doteq r_\gamma(s, \pi_g(s), g)$ and $\Gamma(s, g) \doteq \Gamma(s, \pi_g(s), g)$ as deterministic functions of these learned functions.

Restricting the Model Update to Relevant States   Recall, however, that we need only query these models where $d(s, g) > 0$. We can focus our function approximation resources on those states. This idea has previously been introduced with an interest weighting for GVFs (Sutton et al., 2016), with connections made between interest and initiation sets (White, 2017). For a large state space with many subgoals, using goal-space planning significantly expands the models that need to be learned, especially if we learn one model per subgoal. Even if we learn a model that generalizes across subgoal vectors, we are requiring that model to know a lot: values from all states to all subgoals. It is likely such a models would be hard to learn, and constraining what we learn about with $d(s, g)$ is likely key for practical performance.

The modification to the update is simple: we simply do not update $r_\gamma(s, g), \Gamma(s, g)$ in states s where $d(s, g) = 0$.[7] For the action-value variant, we do not update for state-action pairs (s, a) where $d(s, g) = 0$ and $\pi_g(s) \neq a$. The model will only ever be queried in (s, a) where $d(s, g) = 1$ and $\pi_g(s) = a$.

---

7. More generally, we could use *emphatic weightings* (Sutton et al., 2016) that allow us to incorporate such interest weightings $d(s, g)$, without suffering from bootstrapping off of inaccurate values in states where $d(s, g) = 0$. Incorporating this algorithm would likely benefit the whole system, but we keep things simpler for now and stick with a typical TD update.

Learning the relevance model d We assume in this work that we simply have $d(s, g)$, but we can at least consider ways that we could learn it. One approach is to attempt to learn Γ for each g, to determine which are pertinent. Those with $\Gamma(s, g)$ closer to zero can have $d(s, g) = 0$. In fact, such an approach was taken for discovering options (Khetarpal et al., 2020), where both options and such a relevance function are learned jointly. For us, they could also be learned jointly, where a larger set of goals start with $d(s, g) = 1$, then if $\Gamma(s, g)$ remains small, then these may be switched to $d(s, g) = 0$ and they will stop being learned in the model updates.

Learning the Subgoal-to-Subgoal Models   Finally, we need to extract the subgoal-to-subgoal models $\tilde{r}_\gamma, \tilde{\Gamma}$ from r_γ, Γ. These models were defined as means of the GVFs taken over member states of each subgoal, as specified in Equation 2. The strategy involves updating towards the state-to-subgoal models, whenever a state corresponds to a subgoal. In other words, for a given s, if $m(s, g) = 1$, then for a given g' (or iterating through all of them), we can update $\tilde{r}_\gamma$ using

$$(r_\gamma(s, g') - \tilde{r}_\gamma(g, g'))\nabla\tilde{r}_\gamma(g, g'),$$

and update $\tilde{\Gamma}$ using

$$(\Gamma(s, g') - \tilde{\Gamma}(g, g'))\nabla\tilde{\Gamma}(g, g').$$

Note that these updates are not guaranteed to uniformly weight the states where $m(s, g) = 1$. Instead, the implicit weighting is based on sampling s, such as through which states are visited and in the replay buffer. We do not attempt to correct this skew, as mentioned in the main body, we presume that this bias is minimal. An important next step is to better understand if this lack of reweighting causes convergence issues, and how to modify the algorithm to account for a potentially changing state visitation.

Appendix E. Experiment Details

The following sections will cover the low level experimental details and design choices for each empirical result we present in this paper.

E.1 Offline Model Learning and Planning

For our experiments, we learned the models offline and computed $\tilde{v}$ offline. Therefore, learning options, subgoal models, and $\tilde{v}$ for each environment[8] were done once and re-used across many of our plots. We highlight this procedure in the Figure 21.

Procedure for learning the option policies in our experiments   For each subgoal g, we learn its corresponding option model π_g by initialising the base learner in the initiation set of g, and terminating the episode once the learner is in a state that is a member of g. We used a reward of -1 per step and save the option policy once we reach a 90% success

---

8. We implemented our own FourRoom environment. The PinBall configuration that we used is based on the easy configuration found at `https://github.com/DecisionMakingAI/BenchmarkEnvironments.jl`, which was released under the MIT license. We have modified the environment to support additional features such as changing terminations, visualizing subgoals, and various bug fixes.
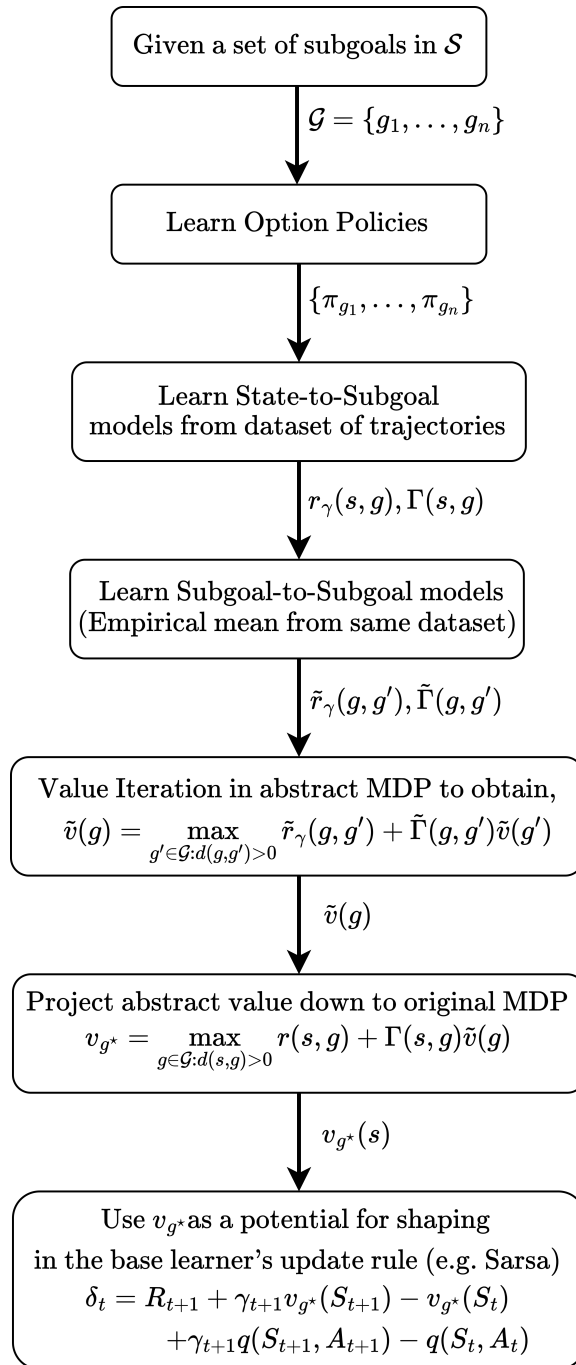
Figure 21: Our procedure for learning and using pre-trained models in our experiments.

rate, and the last 100 episodes are within some domain-dependent cut off. This cut off was 10 steps for FourRooms, and 50 steps for GridBall and PinBall.
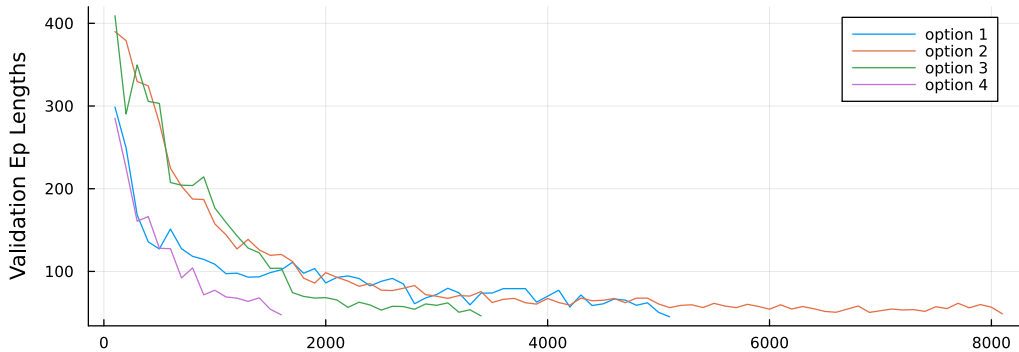
Figure 22: Evaluation of PinBall option policies by average trajectory length. Policies were saved once they were able to reach their respective subgoal in under 50 steps, averaged across 100 trajectories. Subgoal 2 was the hardest to learn an option policy for, due to its proximity to obstacles.

Procedure for learning the subgoal models in our experiments In our experiments, the data is generated offline according to each π_g. A different dataset is generated for each π_g. We then use this on-policy episodic dataset from each π_g to learn the subgoal models for that subgoal g. This is done by ordinary least squares regression to fit a linear model in Four Rooms, and by stochastic gradient descent with neural network models in GridBall and PinBall.

More formally, we first collect a dataset of n episodes generated from π_g, $\mathcal{D}_g = \{\langle S_{i,1}, A_{i,1}, R_{i,1}, S_{i,1}, \ldots, S_{i,T_i}\rangle\}_{i=1}^n$. $S_{i,t}, A_{i,t}, R_{i,t}$ represent the state, action and reward at timestep t of episode i. T_i is the length of episode i. $S_{i,0}$ is a randomised starting state within the initiation set of g, and S_{i,T_i} is a state that is a member of subgoal g. For each g, we use $\mathcal{D}_g$ to generate a matrix of all visited states, $\mathbf{X} \in \mathbb{R}^{l \times |\mathcal{S}|}$, and a vector of all reward model returns, $\mathbf{g}_r \in \mathbb{R}^l$, and transition model returns $\mathbf{g}_\gamma \in \mathbb{R}^l$,

$$\mathbf{X} = \begin{pmatrix} S_{i,1} \\ S_{i,2} \\ \vdots \\ S_{n,T_n} \end{pmatrix}, \mathbf{g}_r = \begin{pmatrix} R_{i,2} + \gamma r_\gamma(S_{i,2}, g) \\ R_{i,3} + \gamma r_\gamma(S_{i,3}, g) \\ \vdots \\ R_{n,T_n} \end{pmatrix}, \mathbf{g}_\gamma = \begin{pmatrix} \gamma^{T_i - 0} \\ \gamma^{T_i - 1} \\ \vdots \\ \gamma^{T_n - T_n} \end{pmatrix},$$

where $l = \sum_{i=1}^n T_i$ is the total number of visited states in $\mathcal{D}_g$.

This creates a system of linear equations, whose weights we can solve for numerically in the four-room domain,

$$\mathbf{X}\boldsymbol{\theta}^r = \mathbf{g}_r \implies \boldsymbol{\theta}^r = \mathbf{X}^+ \mathbf{g}_r,$$
$$\mathbf{X}\boldsymbol{\theta}^\Gamma = \mathbf{g}_\gamma \implies \boldsymbol{\theta}^\Gamma = \mathbf{X}^+ \mathbf{g}_\gamma,$$

where $\boldsymbol{\theta}^r, \boldsymbol{\theta}^\Gamma \in \mathbb{R}^{|\mathcal{S}|}$ and $\mathbf{X}^+$ is the Moore-Penrose pseudoinverse of $\mathbf{X}$ (Penrose, 1955).

For GridBall and PinBall, we used fully connected artificial neural networks for r_γ and Γ, and performed mini-batch stochastic gradient descent to solve $\boldsymbol{\theta}^r$ and $\boldsymbol{\theta}^\Gamma$ for that subgoal g. We use each mini-batch of m states, reward model returns and transition model returns

to perform the update:

$$\boldsymbol{\theta}^r \leftarrow \boldsymbol{\theta}^r - \eta_r \sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}^r} (\boldsymbol{\theta}^{r\top} \mathbf{X}_{j,:} - \mathbf{g}_{r,j})^2,$$

$$\boldsymbol{\theta}^\Gamma \leftarrow \boldsymbol{\theta}^\Gamma - \eta_\Gamma \sum_{j=1}^{m} \nabla_{\boldsymbol{\theta}^\Gamma} (\boldsymbol{\theta}^{\Gamma\top} \mathbf{X}_{j,:} - \mathbf{g}_{\gamma,j})^2,$$

where η_r and η_Γ are the learning rates for the reward and discount models respectively. $\mathbf{X}_{j,:}$ is the jth row of $\mathbf{X}$. $\mathbf{g}_{r,j}$ and $\mathbf{g}_{\gamma,j}$ are the jth entry of $\mathbf{g}_r$ and $\mathbf{g}_\gamma$ respectively.

In our experiments, we had a fully connected artificial neural network with two hidden layers of 128 units and ReLU activation for each subgoal. The network took a state $s = (x, y, \dot{x}, \dot{y})$ as input and outputted both $r_\gamma(s, g)$ and $\Gamma(s, g)$. All weights were initialised using Kaiming initialisation (He et al., 2015). We use the Adam optimizer with $\eta = 0.001$ and the other parameters set to the default ($b_1 = 0.9, b_2 = 0.999, \epsilon = 10^{-8}$), mini-batches of 1024 transitions and 100 epochs.
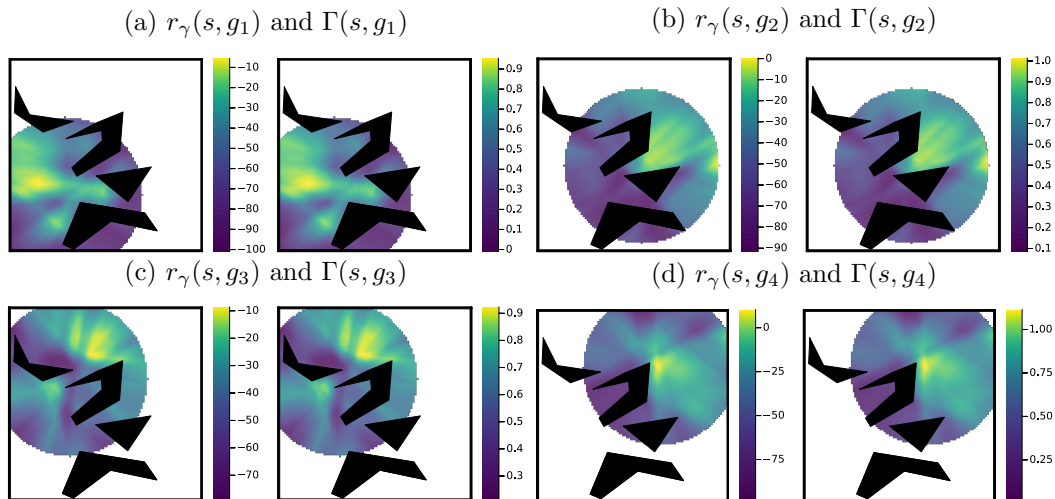
(a) $r_\gamma(s, g_1)$ and $\Gamma(s, g_1)$ (b) $r_\gamma(s, g_2)$ and $\Gamma(s, g_2)$

(c) $r_\gamma(s, g_3)$ and $\Gamma(s, g_3)$ (d) $r_\gamma(s, g_4)$ and $\Gamma(s, g_4)$

Figure 23: State-to-Subgoal models learnt by neural models after 100 epochs.

Procedure for computing $\tilde{v}$ in our experiments We compute $\tilde{v}$ by value iteration in the abstract MDP with a tolerance of $\epsilon = 10^{-8}$ and maximum of 10,000 iterations. The resulting $\tilde{v}$ from these subgoal models was used in the projection step to obtain $v_{g^\star}$, by iterating over relevant subgoals as described in Equation (4). Because we do not change the model online, we only need to compute $\tilde{v}$ once offline and then use these fixed values for our set of subgoals when learning online.

E.2 Optimizations for GSP when using Fixed Models

It is possible to reduce computation cost of GSP when learning with a fixed model. When the subgoal models are fixed, $v_{g^\star}$ for an experience sample does not change over time as all components that are used to calculate $v_{g^\star}$ are fixed. This means that the agent can calculate

$v_{g^\star}$ when it first receives the experience sample and save it in the buffer, and use the same calculated $v_{g^\star}$ whenever this sample is used for updating the main policy. When doing so, $v_{g^\star}$ only needs to be calculated once per sample experienced, instead of with every update. This is beneficial when training neural networks, where each sample is often used multiple times to update network weights.

An additional optimization possible on top of caching of $v_{g^\star}$ in the replay buffer is that we can batch the calculation of $v_{g^\star}$ for multiple samples together, which can be more efficient than calculating $v_{g^\star}$ for a single sample every step. To do this, we create an intermediate buffer that stores up to some number of samples. When the agent experiences a transition, it adds the sample to this intermediate buffer rather than the main buffer. When this buffer is full, the agent calculates $v_{g^\star}$ for all samples in this buffer at once and adds the samples alongside $v_{g^\star}$ to the main buffer. This intermediate buffer is then emptied and added to again every step. We set the maximum size for the intermediate buffer to 1024 in our experiments.

E.3 Value Propagation

This subsection covers Figures 3 and 6. For Sarsa(λ), we swept its learning rate over $[0.001, 0.01, 0.05\ 0.01, 0.5, 0.9]$. 0.01 and 0.05 were found to work best for FourRooms and GridBall respectively. We used an exploration rate of $\epsilon = 0.02$ in FourRooms and $\epsilon = 0.1$ in GridBall. ϵ was decayed by 0.05% each timestep. All learners used $\gamma_c = 0.99$ and $\lambda = 0.9$. For Figure 3 we ran the first episode until the agent reached the main goal (essentially a uniform policy), and studied how the +1 reward at the main goal was propagated back through the regular Sarsa base learner, one with an eligibility trace, and one using GSP. For Figure 6, a random policy took too long to reach the main goal, so we ran Sarsa+GSP for one episode to collect a successful episode's trajectory. The learners used the same tile coder with 16 tilings and 4 tiles across each of the two dimensions of $\mathcal{S}$.

E.4 Faster Learning

This subsection covers Figures 4, 7 and 11. For the FourRooms and GridBall curves, we used the same best learning rates of 0.01 and 0.05. For PinBall, our sweep showed 0.1 worked best. γ_c and λ were still kept at 0.99 and 0.9. In the linear value function approximation setting, we used the same tilecoder as before, just extending it to operate across the 4-dimensional $\mathcal{S}$ of PinBall. In this setting, the reward was -1 per timestep. For the DDQN base learner, we use $\alpha = 0.004$, $\gamma_c = 0.99$, $\epsilon = 0.1$, a buffer that holds up to $10,000$ transitions a batch size of 32, and a target refresh rate of every 100 steps. The Q-Network weights used Kaiming initialisation (He et al., 2015). We swept its learning rate α over $[5 \times 10^{-4}, 1 \times 10^{-3}, 2 \times 10^{-3}, 4 \times 10^{-3}, 5 \times 10^{-3}]$ and target refresh rate τ over $[1, 50, 100, 200, 1000]$ as shown in Figure 26.

Appendix F. An Alternative way of using $v_{g^\star}$

As mentioned in section 6.5, this work also looked at an alternative way of incorporating $v_{g^\star}$ into the base learner's update rule. We do so by biasing the target of the TD error towards $v_{g^\star}$. This modifies the TD error,

$$R_{t+1} + \gamma_{t+1}(\beta v_{g^\star}(S_{t+1}) + (1-\beta)q(S_{t+1}, A_{t+1})) - q(S_t, A_t),$$

where $\beta \in [0,1]$ is a hyper-parameter. We can recover the base learner's update rule by setting $\beta = 0$, whereas $\beta = 1$ completely biases the updates towards the model's prediction (as in our approximation to LAVI in Section 6.5). While this allows us to control the extent of our model's influence on the learning update, we found using $v_{g^\star}$ as a potential to out perform all β in Four-rooms, GridBall and PinBall. However, biasing the TD target in this manner does give the update a faster convergence as we reduce the effective horizon. We shall show this by analyzing the update to the main policy.

We assume we have a finite number of state-action pairs n, with parameterized action-values $q(\cdot; \mathbf{w}) \in \mathbb{R}^n$ represented as a vector with one entry per state-action pair. Value iteration to find q^* corresponds to updating with the Bellman optimality operator

$$(Bq)(s,a) \doteq r(s,a) + \sum_{s'} P(s'|s,a)\gamma(s') \max_{a' \in \mathcal{A}} q(s',a')$$

On each step, for the current $q_t \doteq q(\cdot; \mathbf{w}_t)$, if we assume the parameterized function class can represent Bq_t, then we can reason about the iterations of $\mathbf{w}_1, \mathbf{w}_2, \ldots$ obtain when minimizing distance between $q(\cdot; \mathbf{w}_{t+1})$ and Bq_t, with

$$q(s,a; \mathbf{w}_{t+1}) = (Bq(\cdot; \mathbf{w}_t))(s,a)$$

Under function approximation, we do not simply update a table of values, but we can get this equality by minimizing until we have zero Bellman error. Note that $q^\star = Bq^\star$, by definition.

In this *realizability* regime, we can reason about the iterates produced by value iteration. The convergence rate is dictated by γ_c, as is well known, because

$$\|Bq_1 - Bq_2\|_\infty \leq \gamma_c \|q_1 - q_2\|_\infty$$

Specifically, if we assume $|r(s,a)| \leq r_{\max}$, then we can use the fact that 1) the maximal return is no greater than $G_{\max} \doteq \frac{r_{\max}}{1-\gamma_c}$, and 2) for any initialization q_0 no larger in magnitude than this maximal return we have that $\|q_0 - q^\star\|_\infty \leq 2G_{\max}$. Therefore, we get that

$$\|Bq_0 - q^\star\|_\infty = \|Bq_0 - Bq^\star\|_\infty \leq \gamma_c \|q_0 - q^\star\|_\infty$$

and so after t iterations we have

$$\|q_t - q^\star\|_\infty = \|Bq_{t-1} - Bq^\star\|_\infty \leq \gamma_c \|q_{t-1} - q^\star\|_\infty \leq \gamma_c^2 \|q_{t-2} - q^\star\|_\infty \ldots \leq \gamma_c^t \|q_0 - q^\star\|_\infty = \gamma_c^t G_{\max}$$

We can use the exact same strategy to show convergence of value iteration, under our subgoal-value bootstrapping update. Let $r_{g^\star}(s,a) \doteq \sum_{s'} P(s'|s,a)v_{g^\star}(s')$, assuming $v_{g^\star} : \mathcal{S} \to [-G_{\max}, G_{\max}]$ is a given, fixed function. Then the modified Bellman optimality operator is

$$(B^\beta q)(s,a) \doteq r(s,a) + \beta r_{g^\star}(s,a) + (1-\beta) \sum_{s'} P(s'|s,a)\gamma(s') \max_{a' \in \mathcal{A}} q(s',a').$$

52

Proposition 7 (Convergence rate of tabular value iteration for the biased update)
The fixed-point $q_\beta^\star = B^\beta q_\beta^\star$ exists and is unique. Further, for q_0, and the corresponding $\mathbf{w}_0$, initialized such that $|q_0(s, a; \mathbf{w}_0)| \le G_{max}$, the value iteration update with subgoal bootstrapping $q_t = B^\beta q_{t-1}$ for $t = 1, 2, \ldots$ satisfies

$$\|q_t - q_\beta^\star\|_\infty \le \gamma_c^t (1 - \beta)^t \frac{r_{max} + \beta G_{max}}{1 - \gamma_c(1 - \beta)}$$

Proof First we can show that B^β is a $\gamma_c(1 - \beta)$-contraction. Assume we are given any two vectors q_1, q_2. Notice that $\gamma(s) \le \gamma_c$, because for our problem setting it is either equal to γ_c or equal to zero at termination. Then we have that for any (s, a)

$$
\begin{aligned}
|(B^\beta)q_1(s, a) - (B^\beta q_2)(s, a)| &= \left| (1 - \beta) \sum_{s'} P(s'|s, a)\gamma(s')[\max_{a' \in \mathcal{A}} q_1(s', a') - \max_{a' \in \mathcal{A}} q_2(s', a')] \right| \\
&\le \gamma_c(1 - \beta) \sum_{s'} P(s'|s, a) \left| \max_{a' \in \mathcal{A}} q_1(s', a') - \max_{a' \in \mathcal{A}} q_2(s', a') \right| \\
&\le \gamma_c(1 - \beta) \sum_{s'} P(s'|s, a) \max_{a' \in \mathcal{A}} |q_1(s', a') - q_2(s', a')| \\
&\le \gamma_c(1 - \beta) \sum_{s'} P(s'|s, a) \max_{s' \in \mathcal{S}, a' \in \mathcal{A}} |q_1(s', a') - q_2(s', a')| \\
&\le \gamma_c(1 - \beta) \sum_{s'} P(s'|s, a) \|q_1 - q_2\|_\infty \\
&= \gamma_c(1 - \beta) \|q_1 - q_2\|_\infty
\end{aligned}
$$

Since this is true for any (s, a), it is true for the max, giving

$$\|B^\beta q_1 - B^\beta q_2\|_\infty \le \gamma_c(1 - \beta)\|q_1 - q_2\|_\infty.$$

Because the operator is a contraction, since $\gamma_c(1 - \beta) < 1$, we know by the Banach Fixed-Point Theorem that the fixed-point exists and is unique.

Now we can also use contraction property for the convergence rate. Notice first that we can consider $\tilde{r}(s, a) \doteq r(s, a) + r_{g^\star}(s, a)$ as the new reward, with maximum value $r_{\max} + \beta G_{\max}$. Taking discount as $\gamma_c(1 - \beta)$, the maximal return is $\frac{r_{\max} + \beta G_{\max}}{1 - \gamma_c(1 - \beta)}$.

$$
\begin{aligned}
\|q_t - q_\beta^\star\|_\infty = \|B^\beta q_{t-1} - B^\beta q^\star\|_\infty &\le \gamma_c(1 - \beta)\|q_{t-1} - q^\star\|_\infty \ldots \le \gamma_c^t(1 - \beta)^t \|q_0 - q^\star\|_\infty \\
&\le \gamma_c^t(1 - \beta)^t \frac{r_{\max} + \beta G_{\max}}{1 - \gamma_c(1 - \beta)} \qquad \blacksquare
\end{aligned}
$$

This rate is dominated by $(\gamma_c(1 - \beta))^t$. We can determine after how many iterations this term overcomes the increase in the upper bound on the return. In other words, we want to know how big t needs to be to get

$$\gamma_c^t(1 - \beta)^t \frac{r_{\max} + \beta G_{\max}}{1 - \gamma_c(1 - \beta)} \le \gamma_c^t G_{\max}.$$

Rearranging terms, we get that this is true for

$$t > \log\left(\frac{r_{\max} + \beta G_{\max}}{G_{\max}(1 - \gamma_c(1 - \beta))}\right) \bigg/ \log\left(\frac{1}{1 - \beta}\right).$$

For example if $r_{\max} = 1$, $\gamma_c = 0.99$ and $\beta = 0.5$, then we have that $t > 1.56$. If we have that $r_{\max} = 10$, $\gamma_c = 0.99$ and $\beta = 0.5$, then we get that $t \geq 5$. If we have that $r_{\max} = 1$, $\gamma_c = 0.99$ and $\beta = 0.1$, then we get that $t \geq 22$.

While this increased convergence rate is present for the biased update, it does not show up when using $v_{g^\star}$ as a potential-based shaping reward. Although using $v_{g^\star}$ as a potential does not increase the convergence rate to $v^\star$, it can help quickly identify $\pi^\star$. Specifically, when $v_{g^\star}$ is $v^\star$, and the value function is constant, e.g., initialized to 0, it only takes one application of the bellman operator in each state to find the optimal policy. We formalize this in the proposition below.

Proposition 8 *For $v_{g^\star} = v^\star$ and $v_0 = c$, for $c \in \mathbb{R}$, then the policy, π_1 derived after a single bellman update at all states will be optimal, i.e.,*

$$\forall s \; \pi_1(s) \in \operatorname*{argmax}_a q^\star(s, a).$$

Proof Let the q estimate for the kth iteration be

$$q_k(s, a) = R(s, a) + \sum_{s'} P(s, a, s')\left(\gamma_c v_{g^\star}(s') - v_{g^\star}(s) + \gamma_c v_{k-1}(s')\right).$$

The value function for iteration k is $v_k = \max_a q_k(s, a)$ and the policy for the kth iteration is $\pi_k(s) \in \operatorname{argmax}_a q_k(s, a)$. The value of q_1 is

$$\begin{aligned}
q_1(s, a) &= R(s, a) + \sum_{s'} P(s, a, s')\left(\gamma_c v_{g^\star}(s') - v_{g^\star}(s) + \gamma_c v_0(s')\right) \\
&= R(s, a) + \sum_{s'} P(s, a, s')\left(\gamma_c v^\star(s') - v^\star(s) + \gamma_c v_0(s')\right) \\
&= \underbrace{R(s, a) + \sum_{s'} P(s, a, s')\gamma_c v^\star(s')}_{=q^\star(s,a)} + \underbrace{\sum_{s'} P(s, a, s')\gamma_c v_0(s')}_{=\gamma_c c} - v^\star(s) \\
&= q^\star(s, a) + \gamma_c c - v^\star(s)
\end{aligned}$$

where the last line follows because $v_0(s') = c$ for all s'. Then plugging this expression into π_1 yields

$$\pi_1(s) \in \operatorname*{argmax}_a q^\star(s, a) + \gamma_c c - v^\star(s) = \operatorname*{argmax}_a q^\star(s, a).$$

∎

Remark 9 *While having $v_{g^\star} = v^\star$ is not realistic, Proposition 8 means that the policy will quickly align with what is preferable under $v_{g^\star}$ before finding what is optimal for the MDP without the shaping reward.*

Appendix G. Errors in Learned Subgoal Models

(a) Absolute error in $r_\gamma(s, g)$ (b) Absolute error in $\Gamma(s, g)$
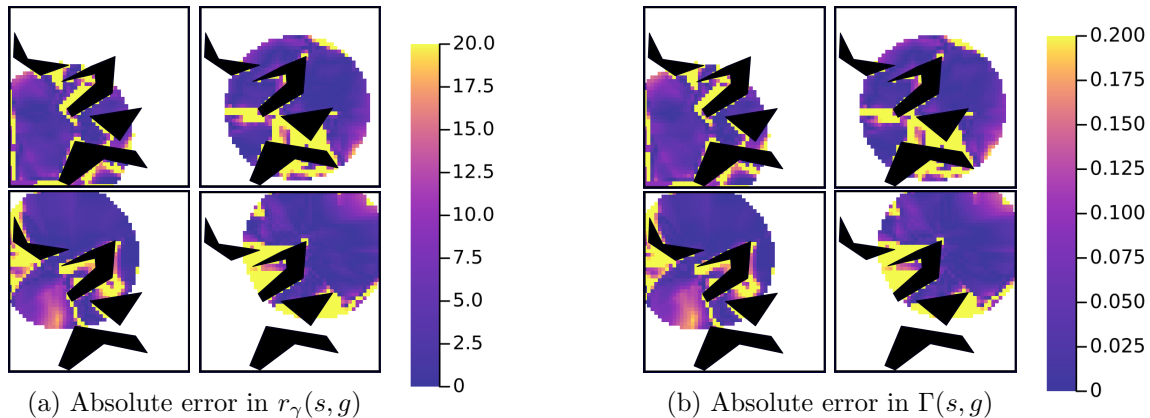
Figure 24: Model errors in State-to-Subgoal models used in GridBall.

To better understand the accuracy of our learned subgoal models, we performed roll-outs of the learned option policy at different (x, y) locations on GridBall and compared the true r_γ and Γ with the estimated values. Figure 24 shows a heatmap of the absolute error of the model compared to the ground truth, with the mapping of colors on the right. The error in each pixel was computed by rolling out episodes from that state and logging the actual reward and discounted probability of reaching the subgoal. The models tend to be more accurate in regions that are clear of obstacles, and less near these obstacles or near the boundary of the initiation set. The distribution of error over the initiation set is very similar for both r and Γ models. While the magnitudes of errors are not unreasonable, they are also not very near zero. This results is encouraging in that inaccuracies in the model do not prevent useful planning.

Epochs	Mean Squared Error across models
2	0.608
4	0.464
10	0.334

Table 3: Mean squared error across state-to-subgoal models used in PinBall.

G.1 Subgoal Placement and Region of Attraction

A counter intuitive observation from the experiments in Section 6.3 was that the On Alternate path helped the agent quickly change its policy but $v_{g^\star}$ did not quickly change. In this section, we investigate this reason and put forth the following hypothesis:

Hypothesis 6 *GSP creates a region of attraction so that the agent follows the optimal path as determined by the abstract MDP.*
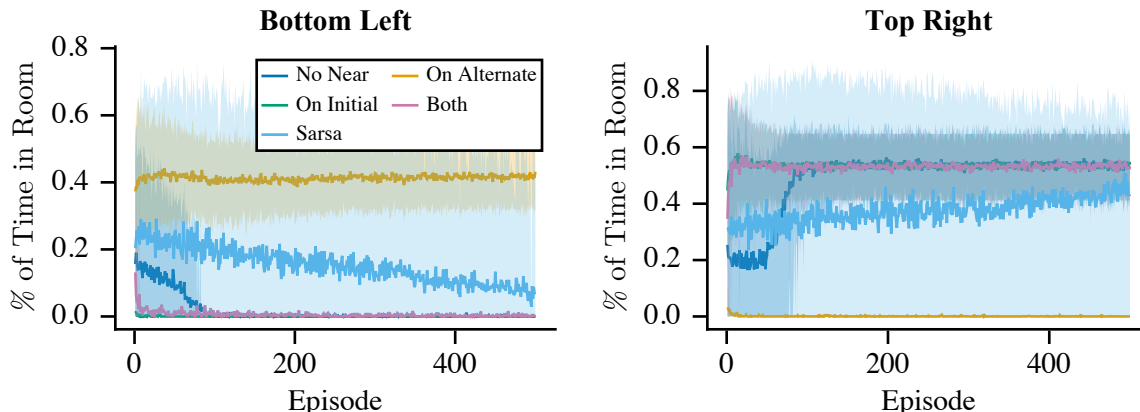
Figure 25: This figure shows the time the agent spends per episode in the bottom left and top right rooms. The lines convey the average % of time the agent spend and the shaded lines represent $(0.05, 0.9)$ tolerance intervals computed from 100 trials.

That is to say, if a single chain of subgoals is represented in the abstract MDP, then the learner will initially try and closely follow this path even if it is not the optimal path. To test this hypothesis, we want to see that the agent will occupy states similar to what is specified by the optimal path in the abstract MDP. For this experiment, we ran GSP on FourRooms (without the lava pools) with each subgoal configuration defined in the previous section. We measured how much time the agent spends in the bottom left room and the top right room. The agent should, as it learns about the environment, spend more time in the top right room and less time in the bottom left room. We would expect all agents to follow this trend, except for the one that is missing a subgoal to go through the top right room. We show the results for each configuration and Sarsa(0) with no GSP in Figure 25.

The results in Figure 25 are clear. All methods learn to go through the top right room except for the subgoal configuration missing a subgoal on that path to the goal state. This supports our hypothesis that the agent will learn to follow the optimal path as specified by the abstract MDP. This also means that while potential-based shaping (used to propagate value information from the abstract MDP to the base learner) does not change the optimal policy, it can make it harder for the learner to find the optimal policy.

A key insight from this result, and results in Section 6.3, are that exploration is impacted by the choice of subgoals. With the basic ϵ-greedy exploration policy that GSP currently uses, GSP will quickly follow and refine the best policy found within the abstract MDP. If the optimal policy is near to the policy found by the abstract MDP, then GSP will be able to quickly discover it. However, if the optimal policy is very different than the one found by the abstract MDP (for example, if the best abstract MDP policy follows an alternate sub-optimal path), this will make the agent explore around its sub-optimal policy, and thus possibly slowing down the discovery of the optimal policy. An important next step is to consider better exploration strategies with GSP, including leveraging the planning structure to do even better exploration. For example, one may consider leveraging an existing subgoal formulation for more directed exploration by introducing reward bonuses at other subgoals, once we know the environment has changed.
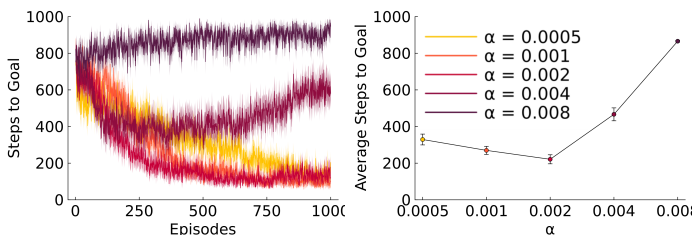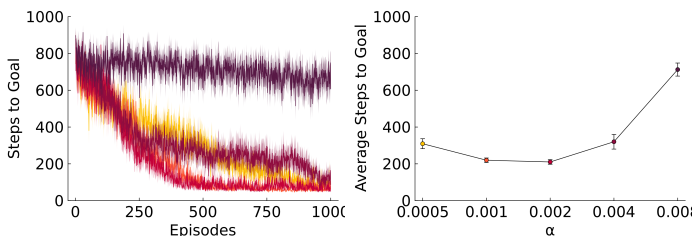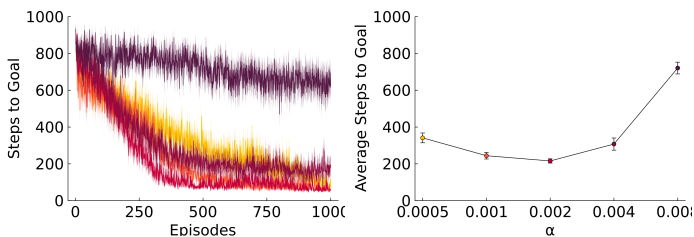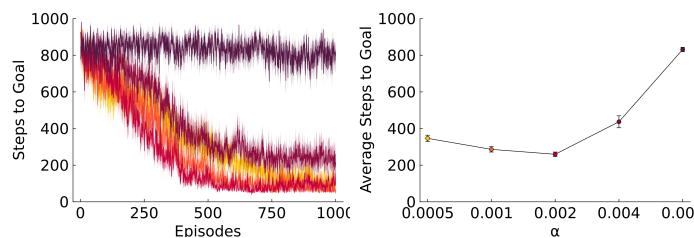
(a) $\tau = 1$
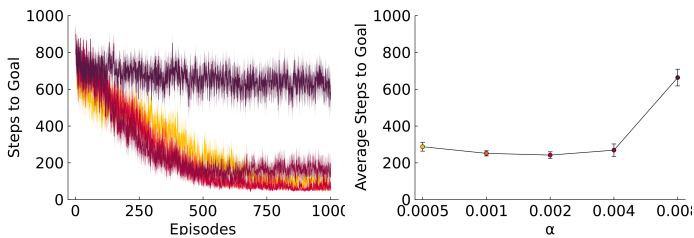
(b) $\tau = 50$

(c) $\tau = 100$

(d) $\tau = 200$

(e) $\tau = 1000$

Figure 26: Left Column: each figure show the learning curves for five different step sizes, α, averaged over 30 runs. Right Column: sensitivity of the DDQN base learner to different step sizes. Each dot represents the steps to goal for that learner, averaged over 30 runs and 1000 episodes. The error bars show one standard error. The refresh rate τ increases with each row.