A New View on Planning in Online Reinforcement Learning

Kevin Roice[*1], **Parham Mohammad Panahi**[*1], **Scott M. Jordan**[1], **Adam White**[1, 2 †], **Martha White**[1, 2 †]

[1]Department of Computing Science, University of Alberta
[2]Alberta Machine Intelligence Institute (Amii)
[†]Canada CIFAR AI Chair
{roice, parham1, sjordan, amw8, whitem}@ualberta.ca

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 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.

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 (using its model) and learn from those experiences. Dyna (Sutton 1991) is a classic example of this *background planning*. On each step, the agent generates several transitions according to its model, and updates its policy with those transitions as if they were real experience.

Background planning can be used to both adapt to the non-stationarity and exploit aspects of the world that remain constant. In many interesting environments, like the real-world or multi-agent games, the agent will be under-parameterized and thus cannot learn or even represent a stationary optimal policy. The agent can overcome this limitation, however, by using a model to rapidly update its policy. Continually updating the model and replanning allows the agent to adapt to the current situation. In addition, many aspects of the environment remain stationary (e.g., fire hurts and objects fall). The model can capture these stationary facts about how the world works and planning can be used to reason about how the world works to produce a better policy.

The promise of background planning is 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, Precup, and Singh 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 optimize 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, Mannor, and Precup 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 optimized in the abstract MDP to guide the learning process in the original MDP. More specifically, the role 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 has two

[*]These authors contributed equally.

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. Overall, this approach increases the agent's ability to learn and adapt to changes in the environment quickly.

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 subgoals, to quickly propagate (suboptimal) value estimates for subgoals. We then leverage these quickly computed subgoal values, without suffering from suboptimality, by incorporating them into any standard value-based algorithm via potential-based shaping. In fact, we layer GSP algorithm onto two different algorithms—Sarsa(λ) and Double Deep Q-Network (DDQN)—and show that improves on both base learners.

We carefully investigate the components of GSP, particularly showing 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.

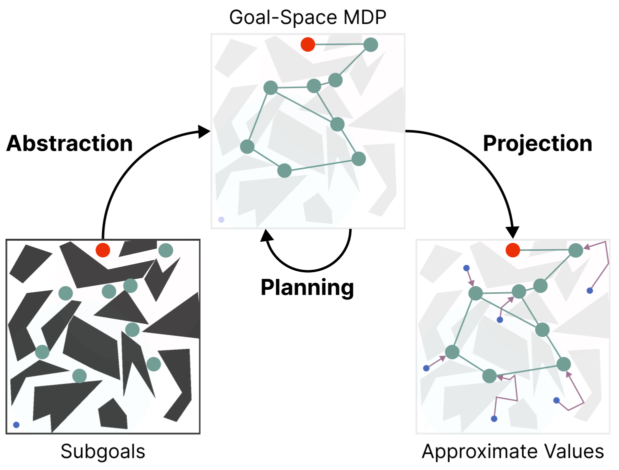

Figure 1: GSP in the PinBall domain. The agent begins with a set of subgoals (denoted in teal) and learns a set of subgoal-conditioned models. **(Abstraction)** Using these models, the agent forms an abstract MDP where the states are subgoals with options to reach each subgoal as actions. **(Planning)** 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.

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 $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$, 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 (Jafferjee et al. 2020; van Hasselt, Hessel, and Aslanides 2019). 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 the state-to-state transition model, but still obtains the benefits of model-based learning for faster learning and adaptation.

3 Goal Space Planning

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 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 the environment. These small changes might still result in large changes in the value function; planning can quickly propagate these small changes, potentially changing the value function significantly.

At a high level, the GSP algorithm focuses on planning over a given set of abstract subgoals to provide quickly updated, approximate values to speed up learning. In order 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 semi-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. In this figure, 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 pre-requisite for this algorithm. For this paper, however, we focus on this planning formalism assuming these subgoals are given, already discovered by the agent.

In the planning step (top central image in Figure 1), we treat $\mathcal{G}$ as our finite set of states and do value iteration. The precise formula for this update is described in Lo et al. (2024), along with the formal definition of the models that we learn for goal-space planning. In words, we compute the subgoal values $\tilde{v} : \mathcal{G} \rightarrow \mathbb{R}$, using $\tilde{r}_\gamma(g, g')$ = discounted return when trying to reach g' from g and $\tilde{\Gamma}(g, g')$ = discounted probability of reaching g' from g,

$$\tilde{v}(g) = \max_{\substack{\text{relevant/nearby} \\ \text{subgoals } g'}} \tilde{r}_\gamma(g, g') + \tilde{\Gamma}(g, g')\tilde{v}(g'). \tag{1}$$

The projection step involves updating values for states, using the subgoal values. The most straightforward way to obtain the value for a state is to find the nearest subgoal s and reason about $r_\gamma(s, g)$ = discounted return when trying to reach g from s and $\Gamma(s, g)$ = discounted probability of reaching g from s,

$$v_{g^\star}(s) = \max_{\substack{\text{relevant/nearby} \\ \text{subgoals } g}} r_\gamma(s, g) + \Gamma(s, g)\tilde{v}(g). \tag{2}$$

Relevance here is defined as s being within the initiation set of the option that reaches that subgoal. We learn an option policy to reach each subgoal, where the initiation set for the option is the set of states from which the option can be executed. The initiation set is a local region around the subgoal, which is why we say we have many local models. Again, we refer the reader to Lo et al. for the formal definitions of these value functions.

There are several ways we can use this value estimate: 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 found in Lo et al. Instead, we propose an approach to use $v_{g^\star}$ as a potential function for potential-based reward shaping (Ng, Harada, and Russell 1999). For example, in the Sarsa(λ) algorithm, the update for the weights $\mathbf{w}$ of the function $q\colon \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \rightarrow \mathbb{R}$ would use the TD-error

$$\begin{aligned}\delta_t := &R_{t+1} + \gamma_{t+1} v_{g^\star}(S_{t+1}) - v_{g^\star}(S_t) + \\ &\gamma_{t+1} q(S_{t+1}, A_{t+1}; \mathbf{w}) - q(S_t, A_t; \mathbf{w}).\end{aligned} \tag{3}$$

A key part of this algorithm is learning the subgoal models, r_γ and Γ. These models will be recognizable to many: they are universal value function approximators (UVFAs) (Schaul et al. 2015). We can leverage advances in learning UVFAs to improve our model learning. 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). Essentially, the model inputs the source and outputs the expected destination, or a distribution over the possible destinations. Here, the models take as inputs both the source and destination, and output only scalars (accumulated reward and discounted probability). The design of GSP is built around using these types of models, that avoids outputting predictions about entire state vectors.

The algorithm is visualized in Figure 2. The steps of agent-environment interaction include:

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 (2)
4. update the main policy with the transition and $v_{g^\star}(S_{t+1})$, using (3).

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.

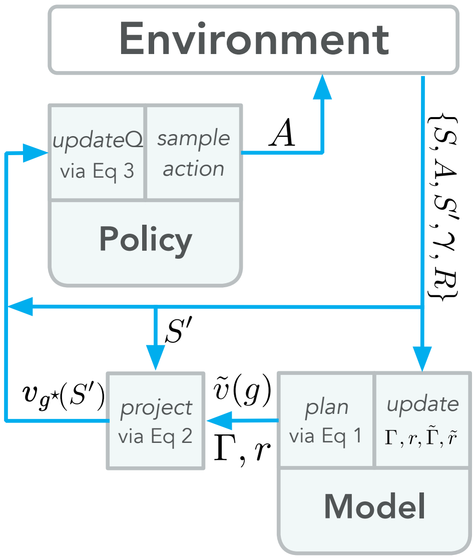

Figure 2: Goal-Space Planning.

To be more concrete, Algorithm 5 shows the GSP algorithm, layered on DDQN (van Hasselt, Guez, and Silver 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

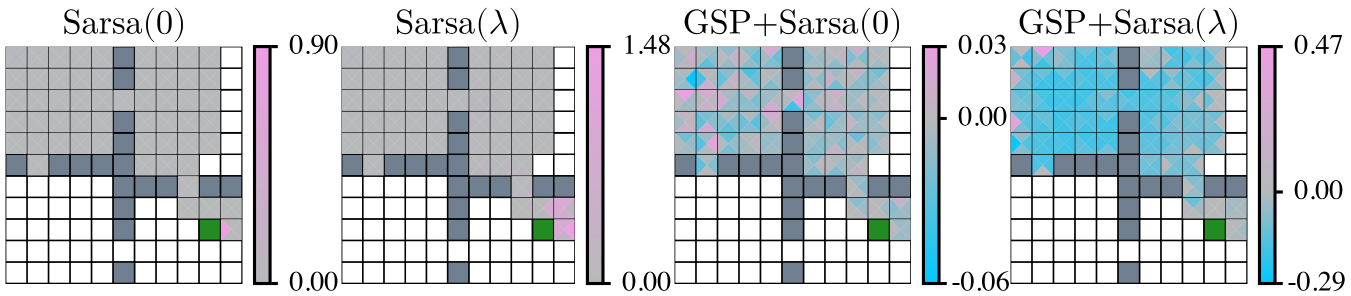

Figure 3: These four plots show the action values after a single episode of updates for Sarsa with and without GSP and eligibility traces, i.e., $\lambda = 0.9$. Each algorithm's update is simulated from the same data collected from a uniform random policy. Each state (square) is made up of four triangles representing each of the four available actions. White squares represent states not visited in the episode.

update (van Hasselt 2010). All new parts relevant to GSP are colored blue; without the blue components, it is a standard DDQN algorithm. The primary change is the addition of the potential to the action-value weights $\mathbf{w}$, with the other blue 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.

Algorithm 1: Goal-Space Planning for Episodic Problems

```
Assume given subgoals $\mathcal{G}$ and relevance function $d$
Initialize base learner (i.e. $\mathbf{w}, \mathbf{z} = \mathbf{0}, \mathbf{0}$ for Sarsa($\lambda$)[1]), 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., $\epsilon$-greedy), observe $r_{t+1}, s_{t+1}, \gamma_{t+1}$
    Choose $a'$ from $s_{t+1}$ using $q$ (e.g. $\epsilon$-greedy)
    ModelUpdate($s_t, a_t, r_{t+1}, s_{t+1}, \gamma_{t+1}$)
    Planning()
    MainPolicyUpdate($s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1}, a'$)
end for
```

4 Experiments

This section motivates the capabilities of the GSP framework through a series of demonstrative results. We investigate the utility of GSP in propagating value and speeding up learning. We do so using learners in three domains: FourRooms, PinBall (Konidaris and Barto 2009) and GridBall (a version of PinBall without velocities). Unless otherwise stated, all learning curves are averaged over 30 runs, with shaded regions representing one standard error.

4.1 GSP on Propagating Value

The central hypothesis of this work is that GSP can accelerate value propagation. By using information from local models in our updates, our belief is that GSP will have a larger change in value to more states, leading to policy changes over larger regions of the state space.

In this section, we consider the effect of our background planning algorithm on value-based RL methods.

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

In order to verify whether GSP helps to quickly propagate value, we first test this hypothesis in a simple grid world environment: the FourRooms domain. The agent can choose from one of 4 actions in a discrete action space $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}\}$. All state transitions are deterministic. The grey squares in Figure 8 indicate walls, and the state remains unchanged if the agent takes an action that leads into a wall. This is an episodic task, where the base learner has a fixed start state and must navigate to a fixed goal state where the episode terminates. Episodes can also terminate by timeout after 1000 timesteps.

In this domain, we test the effect of using GSP with pretrained models on a Sarsa(λ) base learner in the tabular setting (i.e. no function approximation for the value function). Full details on using GSP with this temporal difference (TD) learner can be found in Algorithm 2. We set the four hallway states plus the goal state as subgoals, with their initiation sets being the two rooms they connect. Full details of option policy learning can be found in the Appendix B.

Figure 3 shows the base learner's action-value function after a single episode using four different algorithms: Sarsa(0), Sarsa(λ), Sarsa(0)+GSP, and Sarsa(λ)+GSP. In Figure 3, the Sarsa(0) learner updates the value of the state-action pair that immediately preceded the +1 reward at the goal state. The plot for Sarsa(λ) shows a decaying trail of updates made at the end of the episode, to assign credit to the state-action pairs that led to the +1 reward. The plots for the GSP variants show that all state-action pairs sampled receive instant feedback on the quality of their actions. The updates with GSP can be both positive or negative based on if the agent makes progress towards the goal state or not. This direction of update comes from the potential-based reward shaping rewards or 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

[1]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).

over more of the visited states, even without memory mechanisms such as eligibility traces.

It is evident from these updates over a single episode that the resulting policy from GSP updates should be more likely to go to the goal. We would like to quantify how much faster this propagated value can help our base learner over multiple episodes of experience. More specifically, we want to test the following hypothesis.

Hypothesis 2 *GSP enables a TD base-learner to learn faster.*

We expect GSP to improve a base learner's performance on a task within fewer environment interactions. We shall test whether the value propagation over the state-action space as seen in Figure 3 makes this the case over the course of several episodes (i.e. we are now testing the effect of value propagation over time). Figure 4 shows the performance of a Sarsa(λ) base learner with and without GSP in the FourRooms domain with a reward of -1 per step. Full details on the hyperparameters used can be found in Appendix B. It is evident that the GSP-augmented Sarsa(λ) learner is able to reach the optimal policy much faster. The GSP learner also starts at a much lower steps-to-goal. We *believe* this first episode performance improvement is because the feedback from GSP teaches the agent which actions move towards or away from the goal during the first episode.

4.2 GSP in Larger State Spaces

Many real world applications of RL involve large and/or continuous state spaces. Current planning techniques struggle with such state spaces. This motivates an investigation into how well Hypotheses 1 and 2 hold up when GSP is used in such environments (e.g. the PinBall domain). To better analyse GSP and its value propagation across state-space, we also created an intermediate environment between FourRooms and PinBall called GridBall.

In all our PinBall experiments, the agent is initialised with zero velocity at a fixed start position at the beginning of every episode. It should be noted that, unlike in the FourRooms environment, there exists states which are not in the initiation set of any subgoal - a common occurence when deploying GSP in the state spaces of real-world applications.

GridBall is like PinBall, but change to be more like a gridworld to facilitate visualization. The velocity components of the state are removed, meaning the state only 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 Hypothesis 1, we repeat the experiments on GridBall with base learners that use tile-coded value features (Sutton and Barto 2018), and linear value function approximation. Full details on the option policies and subgoal models used for this are outlined in shown in Appendices B and C. Like in the FourRooms experiment, we set the reward to be 0 at all states and +1 once the agent reaches any state in the main goal to show value propagation. We collect a single episode of experience from the Sarsa(0)+GSP learner and use its trajectory to perform a batch update on all learners. This controls for any variability in trajectories between learners, so we can isolate and study the change in value propagation.

Figure 5 compares the state value function (averaged over the action value estimates) of Sarsa(0), Sarsa(λ), Sarsa(0)+GSP and Sarsa(λ)+GSP learners after a single episode of interaction with the environment. The results are similar to those on FourRooms. 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 preceeding the goal, and the GSP learners updates values at all states in the initiation set of a subgoal.

To examine how GSP translates to faster learning (Hypothesis 2), we measure the performance (steps to goal) over time for each algorithm in both GridBall and PinBall domains. Figure 6 shows that GSP significantly improves the rate of learning in these larger domains too, with the base learner able to reach near its top performance within 75 and 100 epsiodes in GridBall and PinBall respectively. All runs are able to find a similar length path to the goal. As the size of the state space increases, the benefit of using local models in the GSP updates still holds.

Similar to the previous domains, the Sarsa(λ) learner using GSP is able to reach a good policy much faster than the base learner without GSP. In both domains, the GSP and non-GSP Sarsa(λ) learners plateau at the same average steps to goal. 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 making the task harder.

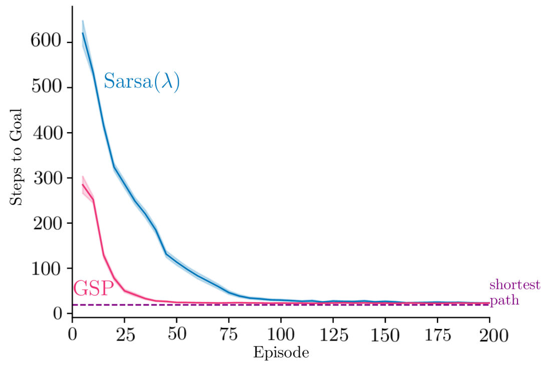

Figure 4: This plot shows the average number of steps to goal smoothed over five episodes in the FourRooms domain. Shaded region represents 1 standard error across 100 runs.

4.3 GSP with Deep Reinforcement Learning

The previous results shed light on the dynamics of value propagation with GSP when a learner is given a representation of it's environment (a look-up table or a tile coding). A natural next step is to look at the whether the reward and

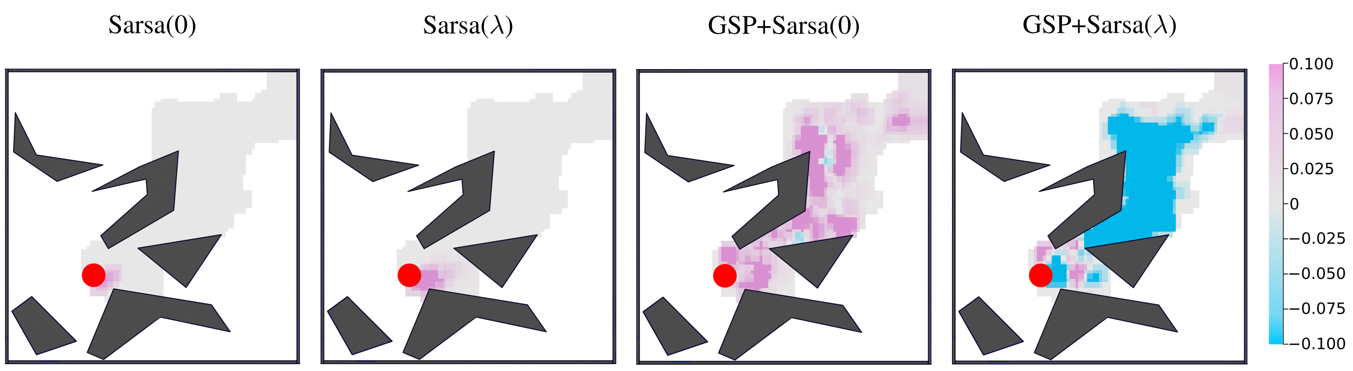

Figure 5: The tile-coded value function after one episode in GridBall. Like Figure 3, the gray regions show the visited states which were not updated. The red circle is the main goal.

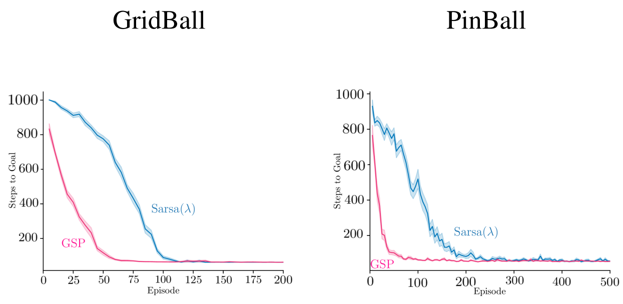

Figure 6: Five episode moving average of return in the GridBall over 200 episodes (left) and PinBall over 500 episodes (right). All learners used linear value function approximation on their tile coded features.

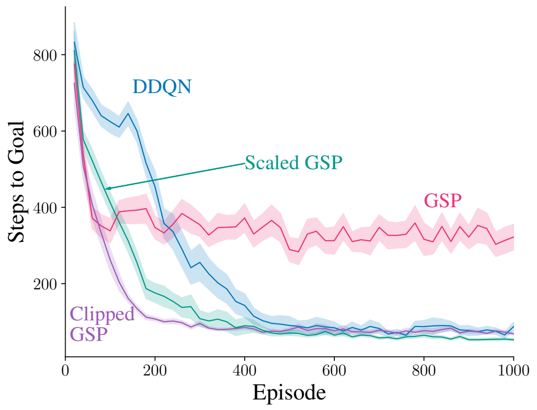

Figure 7: Investigating the behavior of GSP in the deep reinforcement learning setting in PinBall. Following the format of Figure 6, we show the 20 episode moving average of steps to the main goal in PinBall.

transition dynamics learnt in GSP can still propagate value (Hypothesis 2) in the deep RL setting, where the learner must also learn a representation of its environment. We test this by running a DDQN base learner (van Hasselt, Guez, and Silver 2016) in the PinBall domain, with GSP layered on DDQN as in Algorithm 5.

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 ($\gamma v_{g^\star}(S_{t+1}) - v_{g^\star}(S_t)$). We adjusted for this by either clipping or downscaling 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. With these basic magnitude controls, GSP again learns significantly faster than its base learner, as shown in Figure 7.

5 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 (Jafferjee et al. 2020; van Hasselt, Hessel, and Aslanides 2019; Lambert, Pister, and Calandra 2022). This problem has been tackled by using reverse models (Pan et al. 2018; Jafferjee et al. 2020; van Hasselt, Hessel, and Aslanides 2019); primarily using the model for decision-time planning (van Hasselt, Hessel, and Aslanides 2019; Silver, Sutton, and Müller 2008; Chelu, Precup, and van Hasselt 2020); and improving training strategies to account for accumulated errors in rollouts (Talvitie 2014; Venkatraman, Hebert, and Bagnell 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, Barreto, and Nikovski 2017; Farahmand 2018; Ayoub et al. 2020). 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, Singh, and Lee 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, Strehl, and Littman 2006) and using a hierarchy of MDPs with state abstraction and macro-actions (Bakker, Zivkovic, and Krose 2005; Konidaris, Kaelbling, and Lozano-Perez 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, Precup, and Singh 1999; Singh, Barto, and Chentanez 2004; Wan, Naik, and Sutton 2021). Recent work has considered mechanism 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. The most similar to our work is LAVI, which restricts value iteration to a small subset of landmark states (Mann, Mannor, and Precup 2015).[2] 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, Liu, and Su 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; Zhang, Yang, and Stadie 2021; Aubret, Matignon, and Hassas 2021; Hoang et al. 2021; Gieselmann and Pokorny 2021; Kim, Seo, and Shin 2021; Dubey et al. 2021).

Some of this work focused on exploiting landmark states for planning in GCRL. Huang, Liu, and Su (2019) used landmark states as interim subgoals, with a graph-based search to plan between these subgoals (Huang, Liu, and Su 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

[2]A similar idea to landmark states has been considered in more classical AI approaches, under the term bi-level planning (Wolfe, Marthi, and Russell 2010; Hogg, Kuter, and Muñoz-Avila 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.

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.

6 Conclusion

In this paper we analysed a new planning framework, called Goal-Space Planning (GSP). GSP provides a new approach to use background planning to improve value propagation, with minimalist, local models and computationally efficient planning. We showed that these subgoal-conditioned models can be accurately learned using standard value estimation algorithms, and can be used to quickly propagate value through state spaces of varying sizes. We find a consequent learning speed-up on base learners with different types of value function approximation.

This work studies a new formalism, and many new technical questions along with it. We have tested GSP with pretrained models and assumed a given set of subgoals. A critical open question is subgoal discovery. For this work, we relied on hand-chosen subgoals, but in general the agent should discover its own subgoals. In general, though, option and subgoal discovery remain open questions. One utility of this work is that it could help narrow the scope of the discovery question, to that of finding abstract subgoals that help a learner plan more efficiently.

Acknowledgments

We thank the Alberta Machine Intelligence Institute, Alberta Innovates, the Canada CIFAR AI Chairs Program for funding this research, as well as the Digital Research Alliance of Canada for the computation resources.

References

Aubret, A.; Matignon, L.; and Hassas, S. 2021. DisTop: Discovering a Topological Representation to Learn Diverse and Rewarding Skills. *arXiv:2106.03853 [cs]*.

Ayoub, A.; Jia, Z.; Szepesvári, C.; Wang, M.; and Yang, L. 2020. Model-Based Reinforcement Learning with Value-Targeted Regression. In *International Conference on Machine Learning*.

Bakker, B.; Zivkovic, Z.; and Krose, B. 2005. Hierarchical dynamic programming for robot path planning. In *International Conference on Intelligent Robots and Systems*.

Barreto, A.; Borsa, D.; Hou, S.; Comanici, G.; Aygün, E.; Hamel, P.; Toyama, D.; Hunt, J.; Mourad, S.; Silver, D.; and Precup, D. 2019. The Option Keyboard: Combining Skills in Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

Barreto, A.; Hou, S.; Borsa, D.; Silver, D.; and Precup, D. 2020. Fast Reinforcement Learning with Generalized Policy Updates. *Proceedings of the National Academy of Sciences*, 117(48).

Brunskill, E.; and Li, L. 2014. PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *Proceedings of the 31st International Conference on Machine Learning*. PMLR.

Chelu, V.; Precup, D.; and van Hasselt, H. P. 2020. Forethought and Hindsight in Credit Assignment. In *Advances in Neural Information Processing Systems*.

Chitnis, R.; Silver, T.; Tenenbaum, J. B.; Lozano-Pérez, T.; and Kaelbling, L. P. 2022. Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning. In *International Conference on Intelligent Robots and Systems*. IEEE.

Dayan, P.; and Hinton, G. E. 1992. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303.

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

Dubey, R. K.; Sohn, S. S.; Abualdenien, J.; Thrash, T.; Hoelscher, C.; Borrmann, A.; and Kapadia, M. 2021. SNAP:Successor Entropy Based Incremental Subgoal Discovery for Adaptive Navigation. In *Motion, Interaction and Games*.

Emmons, S.; Jain, A.; Laskin, M.; Kurutach, T.; Abbeel, P.; and Pathak, D. 2020. Sparse Graphical Memory for Robust Planning. In *Advances in Neural Information Processing Systems*.

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

Farahmand, A.-m.; Barreto, A. M. S.; and Nikovski, D. N. 2017. Value-Aware Loss Function for Model-based Reinforcement Learning. In *International Conference on Artificial Intelligence and Statistics*.

Farquhar, G.; Rocktäschel, T.; Igl, M.; and Whiteson, S. 2018. TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning. In *International Conference on Learning Representations*.

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

Gopalan, N.; Littman, M.; MacGlashan, J.; Squire, S.; Tellex, S.; Winder, J.; Wong, L.; et al. 2017. Planning with abstract Markov decision processes. In *International Conference on Automated Planning and Scheduling*.

Hafner, D.; Pasukonis, J.; Ba, J.; and Lillicrap, T. 2023. Mastering Diverse Domains through World Models. *arXiv:2301.04104*.

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T. L.; and Boutilier, C. 2013. Hierarchical solution of Markov decision processes using macro-actions. In *Uncertainty in Artificial Intelligence*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing human-level performance on ImageNet Classification. In *IEEE International Conference on Computer Vision*.

Hoang, C.; Sohn, S.; Choi, J.; Carvalho, W.; and Lee, H. 2021. Successor Feature Landmarks for Long-Horizon Goal-Conditioned Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

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

Huang, Z.; Liu, F.; and Su, H. 2019. Mapping State Space using Landmarks for Universal Goal Reaching. In *Advances in Neural Information Processing Systems*.

Jafferjee, T.; Imani, E.; Talvitie, E.; White, M.; and Bowling, M. 2020. Hallucinating Value: A Pitfall of Dyna-style Planning with Imperfect Environment Models. *arXiv:2006.04363*.

Jinnai, Y.; Abel, D.; Hershkowitz, D.; Littman, M.; and Konidaris, G. 2019. Finding Options That Minimize Planning Time. In *International Conference on Machine Learning*.

Kim, J.; Seo, Y.; and Shin, J. 2021. Landmark-Guided Subgoal Generation in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

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

Konidaris, G.; Kaelbling, L.; and Lozano-Perez, T. 2014. Constructing symbolic representations for high-level planning. In *AAAI Conference on Artificial Intelligence*.

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

Lambert, N.; Pister, K.; and Calandra, R. 2022. Investigating Compounding Prediction Errors in Learned Dynamics Models. *arXiv:2203.09637*.

Lo, C.; Roice, K.; Panahi, P. M.; Jordan, S.; White, A.; Mihucz, G.; Aminmansour, F.; and White, M. 2024. Goal-Space Planning with Subgoal Models. arXiv:2206.02902.

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

Mann, T. A.; Mannor, S.; and Precup, D. 2015. Approximate Value Iteration with Temporally Extended Actions. *Journal of Artificial Intelligence Research*, 53.

Nasiriany, S.; Pong, V.; Lin, S.; and Levine, S. 2019. Planning with Goal-Conditioned Policies. In *Advances in Neural Information Processing Systems*.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In *International Conference on Machine Learning*.

Oh, J.; Singh, S.; and Lee, H. 2017. Value Prediction Network. In *Advances in Neural Information Processing Systems*.

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

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

Rummery, G. A. 1995. *Problem solving with reinforcement learning*. Ph.D. thesis, Citeseer.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal Value Function Approximators. In *International Conference on Machine Learning*.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; Lillicrap, T.; and Silver, D. 2020. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839).

Silver, D.; Hasselt, H.; Hessel, M.; Schaul, T.; Guez, A.; Harley, T.; Dulac-Arnold, G.; Reichert, D.; Rabinowitz, N.; Barreto, A.; and Degris, T. 2017. The Predictron: End-To-End Learning and Planning. In *International Conference on Machine Learning*.

Silver, D.; Sutton, R. S.; and Müller, M. 2008. Sample-Based Learning and Search with Permanent and Transient Memories. In *International Conference on Machine Learning*.

Singh, S.; Barto, A.; and Chentanez, N. 2004. Intrinsically Motivated Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

Singh, S. P. 1992. Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Machine Learning Proceedings 1992*, 406–415. Elsevier.

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

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S.; Machado, M. C.; Holland, G. Z.; Szepesvári, D.; Timbers, F.; Tanner, B.; and White, A. 2022. Reward-Respecting Subtasks for Model-Based Reinforcement Learning. *Artificial Intelligence*, 324.

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *International Conference on Autonomous Agents and Multiagent Systems*.

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

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

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

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value Iteration Networks. In *Advances in Neural Information Processing Systems*.

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

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-learning. In *AAAI Conference on Artificial Intelligence*.

van Hasselt, H.; Hessel, M.; and Aslanides, J. 2019. When to use Parametric Models in Reinforcement Learning? In *Advances in Neural Information Processing Systems*.

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

Wan, Y.; Naik, A.; and Sutton, R. S. 2021. Average-Reward Learning and Planning with Options. In *Advances in Neural Information Processing Systems*.

Wan, Y.; Zaheer, M.; White, A.; White, M.; and Sutton, R. S. 2019. Planning with Expectation Models. In *International Joint Conference on Artificial Intelligence*.

Weber, T.; Racanière, S.; Reichert, D. P.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; Pascanu, R.; Battaglia, P.; Silver, D.; and Wierstra, D. 2017. Imagination-Augmented Agents for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

Wolfe, J.; Marthi, B.; and Russell, S. 2010. Combined Task and Motion Planning for Mobile Manipulation. *International Conference on Automated Planning and Scheduling*.

Zahavy, T.; Hasidim, A.; Kaplan, H.; and Mansour, Y. 2020. Planning in Hierarchical Reinforcement Learning: Guarantees for Using Local Policies. In *Proceedings of the 31st International Conference on Algorithmic Learning Theory*. PMLR.

Zhang, L.; Yang, G.; and Stadie, B. C. 2021. World Model as a Graph: Learning Latent Landmarks for Planning. In *International Conference on Machine Learning*.

Zhang, T.; Guo, S.; Tan, T.; Hu, X.; and Chen, F. 2020. Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*.

A Environments

PinBall is a continuous state domain where the agent navigates a ball through a set of obstacles to reach the main goal. The environment uses a four-dimensional state representation of positions and velocities, $(x, y, \dot{x}, \dot{y}) \in [0,1] \times [0,1] \times [-2,2] \times [-2,2]$. The agent chooses from one of five actions at each timestep. $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}, \texttt{nothing}\}$, where the `nothing` action adds no change to the ball's velocity, and the other actions each add an impulse force in one of the four cardinal directions. All collisions are elastic and we use a drag coefficient of 0.995. This is an episodic task with a fixed starting state and main goal. An episode ends when the agent reaches the main goal or after 1,000 time steps.

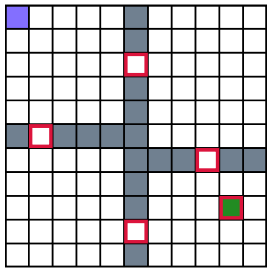

Figure 8: The FourRooms domain. The blue square is the initial state, green square the goal state, and red boxes the subgoals. A subgoal's initiation set contains the states in any room connected to that subgoal.

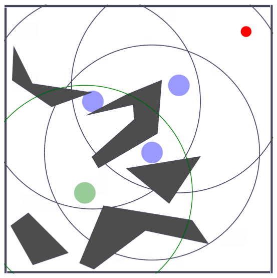

Figure 9: Obstacles and subgoals for GridBall and PinBall. The larger circles show the initiation set boundaries. Subgoals are defined in position space.

B Learning the Option Policies

In the simplest case, it is enough to learn π_g that makes $r_\gamma(s, g)$ maximal for every relevant s (i.e., $\forall s \in \mathcal{S}\, \text{s.t.}\, d(s, g) > 0$). 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 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.

Hyperparameters In FourRooms, we use Sarsa(0) and Sarsa(0.9) base learners with learning rate $\alpha = 0.01$, discount factor $\gamma_c = 0.99$ and an $\epsilon = 0.02$ for its ϵ-greedy policy. In GridBall, we used Sarsa(0) and Sarsa(0.9) base learners with $\alpha = 0.05$, $\gamma_c = 0.99$ and $\epsilon = 0.1$. ϵ is decayed by 0.5% each timestep. In the linear function approximation setting, these learners use a tilecoder with 16 tiles and 4 tilings across each of the both the GridBall dimensions. In PinBall, the Sarsa(0.9) learner was tuned to $\alpha = 0.1$, $\gamma_c = 0.99$, $\epsilon = 0.1$, decayed in the same manner as in GridBall. The same tile coder was used on on the 4-dimensional state space of PinBall. 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 could have also learned the action-value vari-

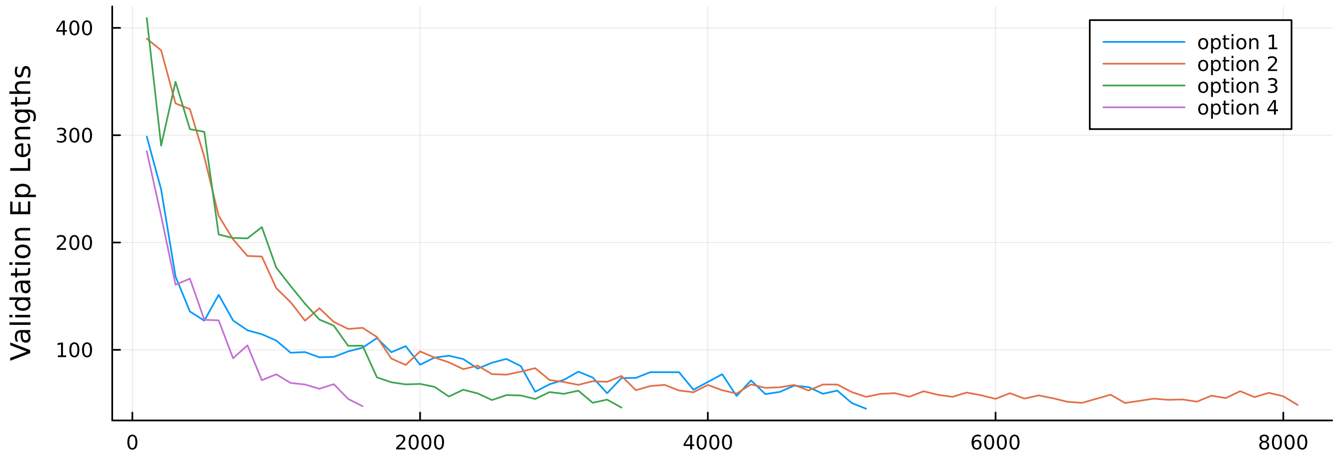

Figure 10: Evaluation of PinBall option policies by average trajectory length. Policies were saved once they were able to reach their respective subgoal in under 50 steps, averaged across 100 trajectories. Subgoal 2 was the hardest to learn an option policy for, due to its proximity to obstacles.

ant $r_\gamma(s,a,g)$ using a Sarsa update, and set $\pi_g(s) = \text{argmax}_{a\in\mathcal{A}} r_\gamma(s,a,g)$, where we overloaded the definition of r_γ. We can then extract $r_\gamma(s,g) = \max_{a\in\mathcal{A}} r_\gamma(s,a,g)$, to use in all the above updates and in planning. In our experiments, this strategy is sufficient for learning π_g.

C Learning the Subgoal Models

In our experiments, the data is generated offline according to each π_g. We then use this episode 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-room, and by stochastic gradient descent with neural network models in GridBall and PinBall.

We first collect a dataset of n episodes leading to a subgoal 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 $\mathbf{g}_\gamma \in \mathbb{R}^l$,

$$\mathbf{X} = \begin{pmatrix} S_{i,1} \\ S_{i,2} \\ \vdots \\ S_{n,T_n} \end{pmatrix}, \mathbf{g}_r = \begin{pmatrix} R_{i,2} + \gamma r_\gamma(S_{i,3}, g) \\ R_{i,3} + \gamma r_\gamma(S_{i,4}, g) \\ \vdots \\ R_{n,T_n} \end{pmatrix},$$

$$\mathbf{g}_\gamma = \begin{pmatrix} \gamma^{T_1 - 0} \\ \gamma^{T_1 - 1} \\ \vdots \\ \gamma^{T_n - T_n} \end{pmatrix},$$

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 j^{th} row of $\mathbf{X}$. $\mathbf{g}_{r,j}$ and $\mathbf{g}_{\gamma,j}$ are the j^{th} 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.

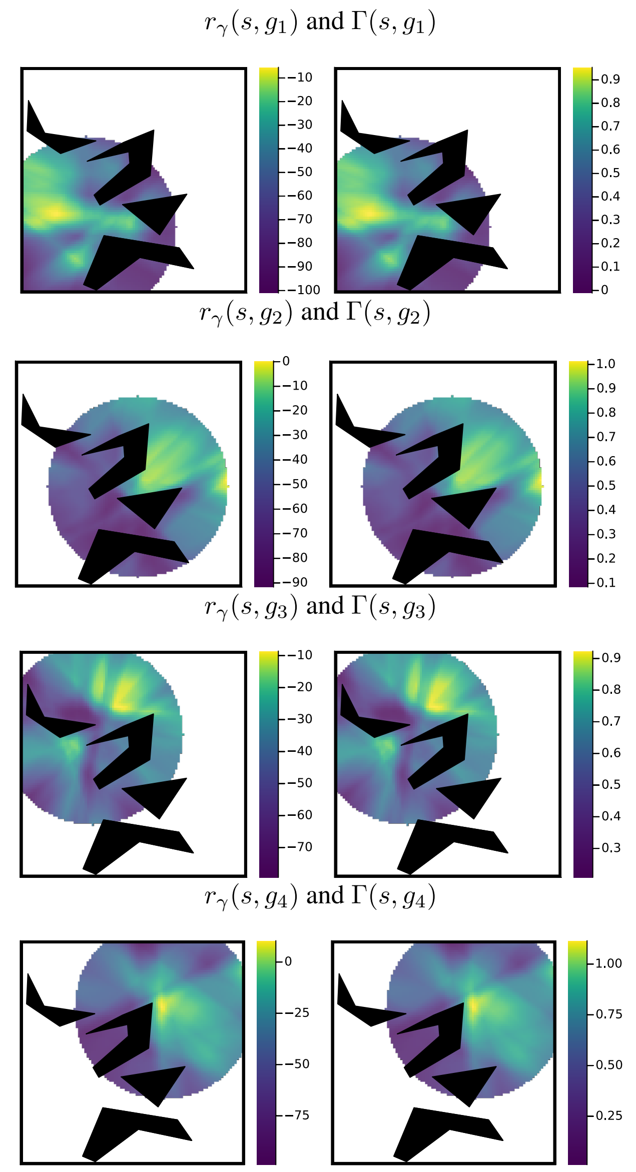

Figure 11: State-to-Subgoal models learnt by neural models after 100 epochs.

D Pseudocode

Algorithm 2: `MainPolicyUpdate`$(s, a, r, s', \gamma, a')$

```
// For a Sarsa($\lambda$) 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 3: `Planning()`

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

Algorithm 4: `ModelUpdate`(s, a, r, s', γ)

```
Add new transition $(s, a, s', r, \gamma)$ to buffer $B$
for $g' \in \bar{\mathcal{G}}$, for multiple transitions $(s, a, r, s', \gamma)$ sampled from $B$ do
    $\gamma_{g'} \leftarrow \gamma(1 - m(s', g'))$
    // Update option policy - e.g. by Sarsa
    $a' \leftarrow \pi_{g'}(s'; \boldsymbol{\theta}^\pi)$
    $\delta^\pi \leftarrow \frac{1}{2}(r - 1) + \gamma_{g'} \tilde{q}(s', a', g'; \boldsymbol{\theta}^\pi) - \tilde{q}(s, a, g'; \boldsymbol{\theta}^\pi)$
    $\boldsymbol{\theta}^\pi \leftarrow \boldsymbol{\theta}^\pi + \alpha^\pi \delta^\pi \nabla_{\boldsymbol{\theta}^\pi} q(s, a, g'; \boldsymbol{\theta}^\pi)$
    // Update reward model and discount model
    $\delta^r \leftarrow r + \gamma_{g'} r_\gamma(s', a', g'; \boldsymbol{\theta}^r) - r_\gamma(s, a, g'; \boldsymbol{\theta}^r)$
    $\delta^\Gamma \leftarrow m(s', g)\gamma + \gamma_{g'} \Gamma(s', a', g'; \boldsymbol{\theta}^\Gamma) - \Gamma(s, a, g'; \boldsymbol{\theta}^\Gamma)$
    $\boldsymbol{\theta}^r \leftarrow \boldsymbol{\theta}^r + \alpha^r \delta^r \nabla_{\boldsymbol{\theta}^r} r_\gamma(s, a, g'; \boldsymbol{\theta}^r)$
    $\boldsymbol{\theta}^\Gamma \leftarrow \boldsymbol{\theta}^\Gamma + \alpha^\Gamma \delta^\Gamma \nabla_{\boldsymbol{\theta}^\Gamma} \Gamma(s, a, g'; \boldsymbol{\theta}^\Gamma)$
    // Update goal-to-goal models using state-to-goal models
    for each $g$ such that $m(s, g) > 0$ do
        $\tilde{\boldsymbol{\theta}}^r \leftarrow \tilde{\boldsymbol{\theta}}^r + \tilde{\alpha}^r (r_\gamma(s, g'; \boldsymbol{\theta}) - \tilde{r}_\gamma(g, g'; \tilde{\boldsymbol{\theta}}^r)) \nabla_{\boldsymbol{\theta}^r} \tilde{r}_\gamma(g, g'; \tilde{\boldsymbol{\theta}}^r)$
        $\tilde{\boldsymbol{\theta}}^\Gamma \leftarrow \tilde{\boldsymbol{\theta}}^\Gamma + \tilde{\alpha}^\Gamma (\Gamma(s, g'; \boldsymbol{\theta}) - \tilde{\Gamma}(g, g'; \tilde{\boldsymbol{\theta}}^r)) \nabla_{\boldsymbol{\theta}^\Gamma} \tilde{\Gamma}(g, g'; \tilde{\boldsymbol{\theta}}^\Gamma)$
    end for
end for
```

It is simple to extend the above pseudocode for the main policy update and the option policy update to use Double DQN (van Hasselt, Guez, and Silver 2016) updates with neural networks. The changes from the above pseudocode are 1) the use of a target network to stabilize learning with neural networks, 2) changing the one-step bootstrap target to the DDQN equivalent, 3) adding a replay buffer for learning the main policy, and 4) changing the update from using a single sample to using a batch update. Because the number of subgoals is discrete, the equations for learning $\tilde{\boldsymbol{\theta}}^r$ and $\tilde{\boldsymbol{\theta}}^\Gamma$ does not change. We previously summarized these changes for learning the main policy in Algorithm 5 and now detail the subgoal model learning in Algorithm 6.

Algorithm 5: `GSP (built on DDQN)`

```
Initialize base learner parameters $\mathbf{w}, \mathbf{w}_{\text{targ}} = \mathbf{w}_0$, set of subgoals $\mathcal{G}$, relevance function $d$
Sample initial state $s_0$ from the environment
for $t \in 0, 1, 2, ...$ do
    Take action $a_t$ using $q$ (e.g., $\epsilon$-greedy),
    Observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$
    Add $(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$ to replay buffer $D$
    DDQNModelUpdate() (see Algorithm 6)
    Planning() (see Algorithm 3)
    for $n$ mini-batches do
        Sample batch $B = \{(s, a, r, s', \gamma)\}$ from $D$
        if $d(s, \cdot), d(s', \cdot) > 0$ then
            $v_{g^\star}(s) = \max_{g \in \bar{\mathcal{G}}: d(s,g)>0} r_\gamma(s, g) + \Gamma(s, g)\tilde{v}(g)$
            $v_{g^\star}(s') = \max_{g \in \bar{\mathcal{G}}: d(s',g)>0} r_\gamma(s', g) + \Gamma(s', g)\tilde{v}(g)$
            $\tilde{r} = r + \gamma v_{g^\star}(s') - v_{g^\star}(s)$
        else
            $\tilde{r} = r$
        end if
        $Y = \tilde{r} + \gamma q(s', \text{argmax}_{a'} q(s', a'; \mathbf{w}); \mathbf{w}_{\text{targ}})$
        $L = \frac{1}{|B|} \sum_{(s,a,r,s',\gamma) \in B} (Y(s, a, r, s', \gamma) - q(s, a; \mathbf{w}))^2$
        $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} L$
        if $n_{\text{updates}} \% \tau == 0$ then
            $\mathbf{w}_{\text{targ}} \leftarrow \mathbf{w}$
        end if
        $n_{\text{updates}} = n_{\text{updates}} + 1$
    end for
end for
```

Algorithm 6: `DDQNModelUpdate`(s, a, r, s', γ)

```
Add new transition $(s, a, s', r, \gamma)$ to buffer $D_{\text{model}}$
for $g' \in \bar{\mathcal{G}}$ do
    for $n_{\text{model}}$ mini-batches do
        Sample batch $B_{\text{model}} = \{(s, a, r, s', \gamma)\}$ from $D_{\text{model}}$
        $\gamma_{g'} \leftarrow \gamma(1 - m(s', g'))$
        // Update option policy
        $a' \leftarrow \text{argmax}_{a' \in \mathcal{A}} \tilde{q}(s', a', g'; \boldsymbol{\theta}^\pi)$
        $\delta^\pi(s, a, s', r, \gamma) \leftarrow \frac{1}{2}(r-1) + \gamma_{g'} \tilde{q}(s', a', g'; \boldsymbol{\theta}^\pi_{\text{targ}}) - q(s, a, g'; \boldsymbol{\theta}^\pi)$
        $\boldsymbol{\theta}^\pi \leftarrow \boldsymbol{\theta}^\pi - \alpha^\pi \nabla_{\boldsymbol{\theta}^\pi} \frac{1}{|B_{\text{model}}|} \sum_{(s,a,r,s',\gamma) \in B_{\text{model}}} (\delta^\pi)^2$
        $\boldsymbol{\theta}^\pi{}_{\text{targ}} \leftarrow \rho_{\text{model}} \boldsymbol{\theta}^\pi + (1 - \rho_{\text{model}}) \boldsymbol{\theta}^\pi{}_{\text{targ}}$
        // Update reward model and discount model
        $\delta^r \leftarrow r + \gamma_{g'}(\gamma, s') r_\gamma(s', a', g'; \boldsymbol{\theta}^r{}_{\text{targ}}) - r_\gamma(s, a, g'; \boldsymbol{\theta}^r)$
        $\delta^\Gamma \leftarrow m(s', g)\gamma + \gamma_{g'}(\gamma, s') \Gamma(s', a', g'; \boldsymbol{\theta}^\Gamma{}_{\text{targ}}) - \Gamma(s, a, g'; \boldsymbol{\theta}^\Gamma)$
        $\boldsymbol{\theta}^r \leftarrow \boldsymbol{\theta}^r - \alpha^r \nabla_{\boldsymbol{\theta}^r} \frac{1}{|B_{\text{model}}|} \sum_{(s,a,r,s',\gamma) \in B} (\delta^r)^2$
        $\boldsymbol{\theta}^\Gamma \leftarrow \boldsymbol{\theta}^\Gamma - \alpha^\Gamma \nabla_{\boldsymbol{\theta}^\Gamma} \frac{1}{|B_{\text{model}}|} \sum_{(s,a,r,s',\gamma) \in B} (\delta^\Gamma)^2$
        if $n_{\text{updates}} \% \tau == 0$ then
            $\boldsymbol{\theta}^r{}_{\text{targ}} \leftarrow \boldsymbol{\theta}^r$
            $\boldsymbol{\theta}^\Gamma{}_{\text{targ}} \leftarrow \boldsymbol{\theta}^\Gamma$
        end if
        $n_{\text{updates}} = n_{\text{updates}} + 1$
    end for
    // Update goal-to-goal models using state-to-goal models
    ... same as in prior pseudocode.
end for
```