Planning with temporally-extended actions

Palash Chatterjee, Roni Khardon

Indiana University, Bloomington
{palchatt, rkhardon}@iu.edu

Abstract

Continuous time systems are often modeled using discrete time dynamics but this requires a small simulation step to maintain accuracy. In turn, this requires a large planning horizon which leads to computationally demanding planning problems and reduced performance. Previous work, using action repeats, has partially addressed this issue in the context of model free reinforcement learning where a policy is learned to determine a discrete action duration. Instead we propose to control the continuous decision timescale directly by letting the planner treat the duration of the action as an additional optimization variable along with the standard action variables. This additional structure has multiple advantages. It speeds up simulation time of trajectories and, importantly, it allows for deep horizon search in terms of primitive steps while using a shallow search depth in the planner. In addition, in the model based reinforcement learning (MBRL) setting, it reduces compounding errors from model learning and improves training time for models. We provide an experimental evaluation both in planning and in MBRL, showing that our approach yields faster planning, better solutions, and that it enables solutions to problems that are not solved in the standard formulation.

1 Introduction

Many interesting real life systems evolve continuously with time, but the dynamics of such systems are often modeled using a discrete-time approximation. In these models, time evolves in discrete steps of δ_t called the *timescale* of the system. Simulators used in reinforcement learning or robotics often use such models to capture physical systems. As the discrete-time models require a good local approximation, the value of δ_t is set to a small value. To obtain a trajectory of the system using such models, the dynamics function needs to be evaluated at fixed intervals of δ_t. Because δ_t is small, the number of decisions required to solve even a simple task can be quite large. From the perspective of planning or MBRL, this translates to longer planning horizons (or rollouts) which can limit their effectiveness.

Shooting based planners like Cross Entropy Method (CEM) and Model Predictive Path Integral (MPPI) (Kobilarov 2012; Botev et al. 2013; Williams, Aldrich, and Theodorou 2017; Chua et al. 2018) rely on sampling to estimate the highly rewarding regions of the state space. They are known to perform poorly as the planning horizon increases, especially in environments with noisy dynamics or sparse rewards. This is because in such environments the variance of their estimates increases and they need a large number of samples in order to act reliably (Chatterjee et al. 2023). In addition, when planning with learned models as in MBRL, longer rollouts can lead to compounding errors (Janner et al. 2019).

Simulators like MuJoCo (Todorov, Erez, and Tassa 2012) or Arcade Learning Environment (Bellemare et al. 2013) make use of a fixed *frame-skip* in addition to timescale. A frame-skip of n means that the same action is repeated for n times before the agent is allowed to act again. So the *decision timescale* becomes $n \times \delta_t$. Frame-skip values are normally set heuristically, but as shown by Braylan et al. (2015), the ideal values of frame-skip vary depending on the environment and can heavily influence the performance of the agent.

This has led to many efforts in trying to control the decision timescale $(n \times \delta_t)$. Previous works have mostly controlled the frame-skip or *action-repeats* (n) by learning a policy (Durugkar et al. 2016; Lakshminarayanan, Sharma, and Ravindran 2017; Sharma, Srinivas, and Ravindran 2017). Ni and Jang (2022) propose to learn a policy to control the timescale (δ_t) rather than the frame-skip. To the best of our knowledge, none of the previous approaches consider the planning problem or the use of planning in MBRL.

In this paper, we propose to control the decision timescale directly by treating the duration of the action as an additional optimization variable. At each *decision point*, the planner optimizes for the standard *primitive actions*, as well as the duration of the action, resulting in a *temporally-extended action*. The duration of each temporally-extended action can be different and it is possible for the planner to choose an action with a long duration at one decision point, followed by an action with a small duration. This provides the planner with an added degree of flexibility and, at the same time, it reduces the search space by constraining the structure of the trajectories that the planner searches over. Further, as a small increase in the temporally-extended planning horizon can translate to a large increase in the primitive planning horizon, this allows the agent to search deeper, enabling it to

solve environments which are otherwise too difficult.

Finally, unlike previous works, we do not learn a policy. Rather, we use the framework of MBRL and learn a transition and reward function that works with temporally-extended actions, and use the learned model to plan. This leads to better performance and faster training.

To summarize, our main contributions are as follows:

1. We propose to control the decision timescale directly by letting the planner optimize for actions as well as the duration of the actions, and evaluate this idea both in planning and in MBRL.
2. We show that the new approach decreases the search space and the planning horizon for the planner and helps the planner solve previously unsolvable problems. In addition, using temporally-extended actions improves the stability of the search allowing the planner to search deeper and solve problems with sparse rewards.
3. We show that the use of planning with learned temporally extended transition functions in MBRL leads to better performance in terms of cumulative rewards and better run time.

2 Related Work

Macro-actions (Hauskrecht et al. 2013) and *options* (Sutton, Precup, and Singh 1999) are two of the most common methods to introduce temporal abstraction in planning and reinforcement learning (RL). A macro-action is a usually defined to be a sequence of primitive actions that the agent will take. On the other hand, an option consists of a policy, an initiation set and a terminating condition. The initiation set determines when the option can be taken, while its duration depends on when its terminating condition is satisfied. Both macro-actions and options can either be predefined or can be learned from data (Durugkar et al. 2016; Machado, Bellemare, and Bowling 2017; Machado et al. 2017; Ramesh, Tomar, and Ravindran 2019).

Frame-skips or action-repeats are simpler forms of macro-actions where each macro-action is essentially a single primitive action repeated multiple times. While frame-skipping has been used as a heuristic in many deep RL solutions (Bellemare, Veness, and Bowling 2012; Mnih et al. 2015), Braylan et al. (2015) showed that using a static value of frame-skips across environments can lead to sub-optimal performance. Following this, there have been multiple efforts using the model-free RL framework to make the frame-skip dynamic. Lakshminarayanan, Sharma, and Ravindran (2017) propose to learn a joint policy on an inflated action space of size $n|\mathcal{A}|$ where each action is tied to n corresponding action-repeats. The value of n is usually small to limit the size of the inflated action space. A more practical approach is to learn a separate network alongside the standard policy to predict the number of action repeats (Sharma, Srinivas, and Ravindran 2017; Biedenkapp et al. 2021).

Frame-skips or action-repeats are discrete values and, while they introduce a temporal abstraction for discrete time systems, a continuous representation is both more realistic and more flexible. Some prior work in model-free RL has explored this problem. Ni and Jang (2022) use a modified Soft Actor Critic (Haarnoja et al. 2018) to learn a policy and control the timescale rather than the action duration, with an explicit constraint on the average timescale of the policy. Their optimization objective is similar to ours, except that they use a linear approximation of the reward function to compute the reward due to a macro-action (see discussion in Appendix A). Wang and Beltrame (2024) introduce Soft-Elastic Actor Critic to output the duration of the action along with the action itself. However, they modify the reward structure by introducing penalties for the energy and the time taken by each action. Our work is in the planning and MBRL setting. Rather than learn a policy, we use either exact or learned transition and reward functions to plan. Further, our objective arises naturally from the formulation without the need to add additional constraints or engineer rewards.

Finally, in our work, action durations are chosen by the planner, which is different from the problem of planning with durative actions (Mausam and Weld 2008) where the durations are given by the model.

3 Background

The standard discrete time Markov Decision Process (MDP) is specified by $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma\}$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively and $\gamma \in (0, 1)$ is the discount factor. $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ represents the state and *primitive action* at timestep t. The one-step transition distribution is given by $\mathcal{T}(s_{t+1}|s_t, a_t)$ and the one-step reward distribution is given by $\mathcal{R}(s_t, a_t)$. Discrete time simulation of continuous systems assumes that $\mathcal{T}$ and $\mathcal{R}$ capture the transitions and the corresponding reward due to exactly δ_t duration, which is the timescale of the MDP.

The expected discounted returns due to a planning horizon of D is given by

$$J_t = \mathbb{E}\left[\sum_{i=0}^{D-1} \gamma^i \mathcal{R}(s_{t+i}, a_{t+i})\right]. \tag{1}$$

In cases when the dynamics $\mathcal{T}$ and the reward $\mathcal{R}$ are unknown to the agent, their empirical estimates, $\hat{\mathcal{T}}$ and $\hat{\mathcal{R}}$, can be learned using data collected by interacting with the MDP.

4 Modeling Temporally-Extended Actions

Although we eventually care about what *primitive actions* to take in the environment, a planner can work at an abstract level by using *temporally-extended actions*. We use the terms *decision steps* and *execution steps* to distinguish between the number of times the agent outputs an action and the number of *primitive actions* that are eventually executed in the environment.

Let us assume that the agent has access to the *primitive transition and reward function* f, and that f is accurate for all $0 \leq t \leq \delta_t^{\text{env}}$, where δ_t^{env} is the timescale of the environment.

In the standard setup, the agent uses f at each decision step and outputs an action whose duration is implicitly δ_t^{env}.

The returns due to a trajectory τ is given by

$$J_1 = \sum_{t=1}^{L(\tau)} \gamma^{t-1} \mathcal{R}(s_t, a_t) \tag{2}$$

where $L(\tau)$ is the number of decision points in τ and $\mathcal{R}(s_t, a_t)$ is the reward due to primitive step t. Here, the number of *execution steps* is exactly equal to the number of *decision steps*.

In our proposed framework, the planner explicitly outputs the duration of the action along with the action itself. At decision step k, let the planner output an action a_k and its corresponding duration $\delta t_k \in [\delta t_{\min}, \delta t_{\max}]$. Now, the number of execution steps need not necessarily be equal to the number of decision steps. Let $e_k = \lfloor \delta t_k / \delta_t^{\text{env}} \rfloor + \mathbb{1}(\delta t_k \bmod \delta_t^{\text{env}})$ be the number of *execution steps* associated with decision step k. Further, let $e_{<k} = \sum_{j=1}^{k-1} e_j$ be the total number of execution steps taken prior to decision step k. The returns due to a trajectory τ will be

$$J_2 = \sum_{k=1}^{L(\tau)} \gamma^{e_{<k}} \sum_{t=1}^{e_k} \gamma^{t-1} \mathcal{R}(s_{(e_{<k}+t)}, a_{(e_{<k}+t)}) \tag{3}$$

where $e_{<1} = 0$. This formulation avoids the need for a precise continuous time model of discounting. Note that when $\delta t_k = \delta_t^{\text{env}}$ for all k, then $J_2 = J_1$.

One way to view J_2 is to consider the inner summation as the return due to a temporally-extended action, but discounted based on the number of primitive actions taken prior to the current timestep. This view can allow us to have different discount factors, γ_1 and γ_2, giving us more fine-grained control over the behavior of the agent.

$$J_3 = \sum_{k=1}^{L(\tau)} \gamma_1^{e_{<k}} R_k^{\text{TE}} \tag{4}$$

$$\text{where } R_k^{\text{TE}} = \sum_{t=1}^{e_k} \gamma_2^{t-1} \mathcal{R}(s_{(e_{<k}+t)}, a_{(e_{<k}+t)}).$$

5 Method

We want the planner to have access to a *temporally-extended transition and reward function* (F) that can work with temporally-extended actions. If F is available, then using it for planning is straight-forward. For example, one can use a shooting-based planner with F to select an action from state s_k as shown in Algorithm 1. Note that using F allows the planner to treat the duration of the action similar to the other action variables except while computing $\gamma^{e_{<k}}$ in the objective. Hence for succinctness, we assume that action a_t at step t contains information about the duration of the action as well.

However, F is usually not readily available. Even in the standard planning setup, the agent has access to the primitive transition and reward function (f) which can either be deterministic or stochastic. If f is stochastic, it samples the next state.

As a simple solution, we can wrap f in a loop as shown in Algorithm 2 to obtain F. This holds as we assumed f is accurate for all $0 \le t \le \delta_t^{\text{env}}$. We call this the *iterative primitive transition and reward function* and represent it using F_{IP}. The reward is simply aggregated in the loop and is not shown explicitly here. Note that for a given environment, f is fixed, while s_k, a_k and δt_k are dependent on the timestep.

Algorithm 1 Action selection using a temporally-extended transition and reward function (F) with a shooting-based planner

Require: F : temporally-extended transition and reward function
s_k : current state
μ_a, var_a : initial action distribution
N, D_{TE} : number of rollouts, planning horizon

1: **for** $i = 1$ to optimization steps **do**
2: sample N action seq. of len. D_{TE} using μ_a, var_a
3: **for** all N action seq. $(a_t, \ldots, a_{t+D_{\text{TE}}})$ **do**
4: **for** step $t = 0$ to $D_{\text{TE}} - 1$ **do**
5: simulate transition using F and a_t and compute agg. reward
6: **end for**
7: **end for**
8: Update μ_a, var_a using action seq. and agg. reward
9: **end for**
10: **return** sample a_k using μ_a, var_a

For the computation to be exact, f should be explicitly dependent on time. If it is implicit, performing the computation at line 5 of Algorithm 2 will not be possible and F_{IP} computes the number of action-repeats instead.

Although F_{IP} accurately captures F, it suffers from being an iterative solution. The time required by F_{IP} to simulate the outcome due to a temporally-extended action is dependent on the precise duration of the action itself. However, if we had access to F, this would have been a constant time evaluation. So while F_{IP} captures F, it fails to facilitate the speed of execution.

To fix this, we use neural networks and approximate F using $\hat{F}_{\text{TE}}$. We can then use $\hat{F}_{\text{TE}}$ to predict the next state and reward due to a temporally-extended action from a given state. For learning $\hat{F}_{\text{TE}}$, we use the framework of MBRL. We collect data by interacting with the environment and use the data to train a neural network to predict a distribution over the next states and a point estimate for the reward. Once $\hat{F}_{\text{TE}}$ is learned, it can be used for planning. In this work, we use shooting-based planners and use Algorithm 1 with $F \approx \hat{F}_{\text{TE}}$ in order to select actions.

Discussion

Consider two agents - A_{standard} that uses primitive actions and A_{TE} that uses temporally-extended actions and let D_{standard} and D_{TE} be their corresponding planning horizons. Further, for A_{TE}, let $\delta t_{\max} = m \times \delta_t^{\text{env}}$ for some positive integer m. As the planning horizon for the agents is not directly comparable, we introduce the term *maximal primitive horizon* (H). For A_{standard}, $H = D_{\text{standard}}$, while for A_{TE}, $H = m \times D_{\text{TE}}$.

Algorithm 2 Iterative Primitive Transition and Reward Function

Require: f : primitive transition and reward function
s_k : current state
a_k : temporally-extended action
δt_k : duration of action

repeats $= \lfloor \delta t_k / \delta_t^{\text{env}} \rfloor$
for $i = 1$ to repeats **do**
  $s_k = f(s_k, a_k, \delta_t^{\text{env}})$ ▷ Control n
end for
$s_{k+1} = f(s_k, a_k, \delta t_k \bmod \delta_t^{\text{env}})$ ▷ Control δ_t
return s_{k+1}

Note that using maximal primitive horizon is not a fair comparison. This only ensures that the primitive planning horizon for A_{TE} is not greater than A_{standard} at any instant. For most practical purposes, the primitive planning horizon for A_{TE} will in-fact be smaller than A_{standard}. We argue that even in this unfair setting, using temporally-extended actions can have several advantages.

1. Using temporally-extended actions helps to reduce the space of possible trajectories that the agent searches over. For simplicity, let us consider an environment with binary actions. The search space for A_{standard} is 2^H, while for A_{TE}, the search space reduces to $2^{H/m}$. However, this reduction comes at a cost of the flexibility of trajectories. The larger the value of m, the "stiffer" the generated trajectories are.
2. A_{TE} has less variables to optimize for than A_{standard}. In general, if the action space has dimensions $|\mathcal{A}|$, then at each decision step, A_{standard} has to optimize for $H|\mathcal{A}|$ action variables, while A_{TE} will have to optimize for $(H/m)(|\mathcal{A}| + 1)$ action variables.
3. $\hat{F}_{\text{TE}}$ evaluates the outcome of temporally-extended actions in constant time. The evaluation time is similar to that taken by f to evaluate the outcome of a primitive action. As $D_{\text{TE}} < D_{\text{standard}}$, using $\hat{F}_{\text{TE}}$ with A_{TE} helps it make a decision faster than A_{standard}.
4. Manipulating the value of m allows us to scale up the maximal planning horizon of A_{TE} without adding any extra variables. This can be useful in environments where rewards are uninformative and a deeper search is required.

6 Experiments

We evaluate the use of temporally extended models both in planning and in MBRL. Full details of the experimental setup are given in Appendix B.

We compare the planning performance of A_{standard} and A_{TE} using CEM (Botev et al. 2013; Chua et al. 2018), which is a shooting-based planner. CEM maintains a sequence of sampling distributions from which it generates multiple action sequences. For each action sequence, it instantiates multiple particles, computes the trajectory and reward due to each particle. Then, it computes the mean reward per action sequence and uses the top k action sequences to bias its sampling distributions.

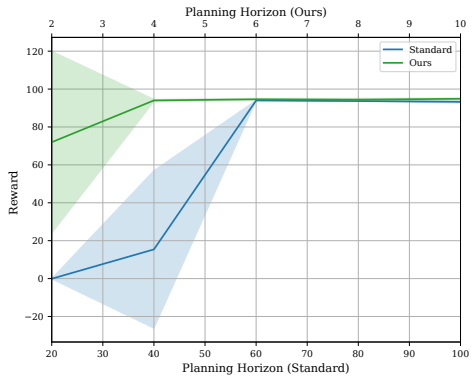
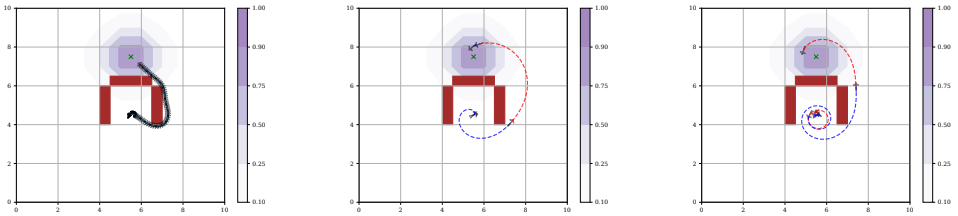


Figure 1: A_{standard} requires a large planning horizon to succeed in Mountain Car, but A_{TE} using $\delta t_{\max} = 100$ can work with an extremely small planning horizon.

For experiments with RL, following Chua et al. (2018), rather than learning a single neural network to approximate the transition and reward function, the agents learn an ensemble of neural networks. Specifically, A_{standard} learns to approximate f while A_{TE} learns to approximate F. During each training iteration, both the agents train on the all the data collected so far by randomly drawing mini-batches from the replay buffer. The model ensembles are then used with CEM to choose an action. Both the agents use the same network architecture. The only difference is that the A_{TE} has an extra input variable corresponding to the duration of the action, in addition to the state and action variables.

For evaluation, we wrap the primitive transition and reward function of the simulation environment in an iterative loop as shown in Algorithm 2. In addition, we need to specify the range of the action duration using $\delta t_{\min}$ and $\delta t_{\max}$. In our experiments, $\delta t_{\min} = \delta_t^{\text{env}}$ and $\delta t_{\max}$ is a hyperparameter that is defined per environment as specified below.

We first experiment in a planning regime where the agent has access to the exact transition and reward function. We use Algorithm 2 to generate the iterative primitive transition and reward function which is used by our framework. For this, we use the Mountain Car environment from Gymnasium (Kwiatkowski et al. 2024), a multi-hill Mountain Car environment from the Probabilistic and Reinforcement Learning Track of the International Planning Competition (IPC) 2023 (Taitler et al. 2024), and the Dubins car environment of Chatterjee et al. (2023) where an agent controls a car using linear and angular "acceleration". We run each of the planning experiments across 5 different seeds and average the results. We then experiment in the MBRL setting where $\mathcal{T}$ and $\mathcal{R}$ are not known. In this case, we learn a temporally-extended representation as discussed above, by interacting with the environment. We use Cart Pole from Gymnasium and Half Cheetah, Ant, Reacher and Pusher from MuJoCo (Todorov, Erez, and Tassa 2012). The experiments are organized so as to answer a set of questions as outlined below.



(a) A_{standard}, $|\mathcal{A}|$=2 (b) A_{TE}, $|\mathcal{A}|$=2 (c) A_{TE}, $|\mathcal{A}|$=102

Figure 2: When the number of actions is small, both the agents are able to solve the problem. But when the number of actions increases, A_{TE} is still able to solve the problem while A_{standard} fails due to large memory requirements. The shape of the curves is an artifact of the action space in this environment. Note that a constant action in acceleration space yields curved paths. The path in (c) is composed of 4 actions of different durations, as marked by the colors.

Instance	Reward ↑		Decision steps ↓		Time for an episode ↓		Success Probability ↑	
	Standard	Ours	Standard	Ours	Standard	Ours	Standard	Ours
1	90.98 ± 0.26	91.74 ± 1.55	108.8 ± 0.84	3.0 ± 1.41	4.86 ± 0.19	5.7 ± 1.13	1.00	1.00
2	-0.1 ± 0.01	88.67 ± 0.65	300.0 ± 0.0	2.8 ± 1.1	7.98 ± 0.28	5.59 ± 0.91	0.00	1.00
3	-0.1 ± 0.01	86.15 ± 1.11	300.0 ± 0.0	2.6 ± 0.89	7.94 ± 0.15	5.44 ± 0.76	0.00	1.00
4	-0.1 ± 0.01	66.5 ± 43.54	300.0 ± 0.0	3.2 ± 1.3	7.73 ± 0.28	6.01 ± 1.05	0.00	0.80
5	-0.1 ± 0.01	83.41 ± 0.54	300.0 ± 0.0	3.0 ± 0.0	8.0 ± 0.28	5.82 ± 0.2	0.00	1.00

Table 1: Results on Multi-hill Mountain Car from IPC 23 across 5 seeds where A_{TE} ($D_{\text{TE}} = 12$) solve all instances while A_{standard} ($D_{\text{TE}} = 175$) can only solve 1/5.

Does planning with temporally-extended actions help? Mountain Car has a sparse reward and it requires a large planning horizon to succeed. We compare the performance of A_{standard} and A_{TE} by varying the planning horizon and show that our proposed framework enables the agent to solve the environment using an extremely small number of planning steps (Figure 1).

We next experiment with the multi-hill version of Mountain Car from IPC 2023 (see Figure 5b in Appendix B). Each instance of the environment increases the difficulty by either adding more hills or altering the surface of the hills. The agent is allowed to take a maximum of 300 primitive steps in the environment. As shown in Table 1, while A_{standard} is able to solve only the first instance, A_{TE} solves all the instances.

We observe from the table that, even though A_{TE} takes a small number of decision steps, the time required to finish the episode is comparable to A_{standard}. This is because it uses an iterative version of the dynamics for simulation given by F_{IP}. Another observation is the number of decision steps is less than D_{TE}. This results because the number of decision steps is measured during evaluation and in Mountain Car, the episode terminates as soon as the agent reaches the goal.

Can temporally-extended actions help in reducing infeasible problems to feasible ones? We experiment with the custom Dubins Car environment ($\delta_t^{\text{env}} = 0.2$) and u-shaped map as shown in Figure 2. This is a challenging configuration because the car is initially facing the obstacles and a naive forward search hits the obstacles and does not yield useful information. In addition, as the reward is sparse, this map requires the planning horizon to be large. To solve the environment, A_{standard} requires 10,000 samples with a planning horizon of 1000, while A_{TE} requires a planning horizon of 75 and $\delta t_{\max} = 20$ (see detailed discussion in Appendix D).

To make the search problem more difficult, we augment the action space of the environment with 100 dummy action variables. Although these actions do not contribute to the dynamics or rewards, the agent is unaware of this and has to account for all the action variables. Even in this setting, A_{TE} is able to solve the environment while A_{standard} fails due to large memory requirements. A simple computation shows that A_{standard} needs around 4GB of memory to keep track of the sampled actions, whereas A_{TE} requires just 103MB. On the other hand, in other configurations (Appendix D) that reduce the memory requirements, A_{standard} fails to find the goal. This shows that using temporally-extended actions can often be helpful in turning infeasible search problems into feasible ones.

Do the benefits transfer if we learn the dynamics and reward model? Having access to the transition and reward function is not realistic. We would like to be able to learn these as we interact with the environment. For this experiment, we use Half Cheetah, Ant, Reacher and Pusher from MuJoCo along with Cart Pole. By default, the MuJoCo simulators use a preset value of frameskip which varies depending on the environment. This results in the effective timescale being greater than the original timescale. We modify the environments so that they all have a frameskip of 1, and use that for our experiments.

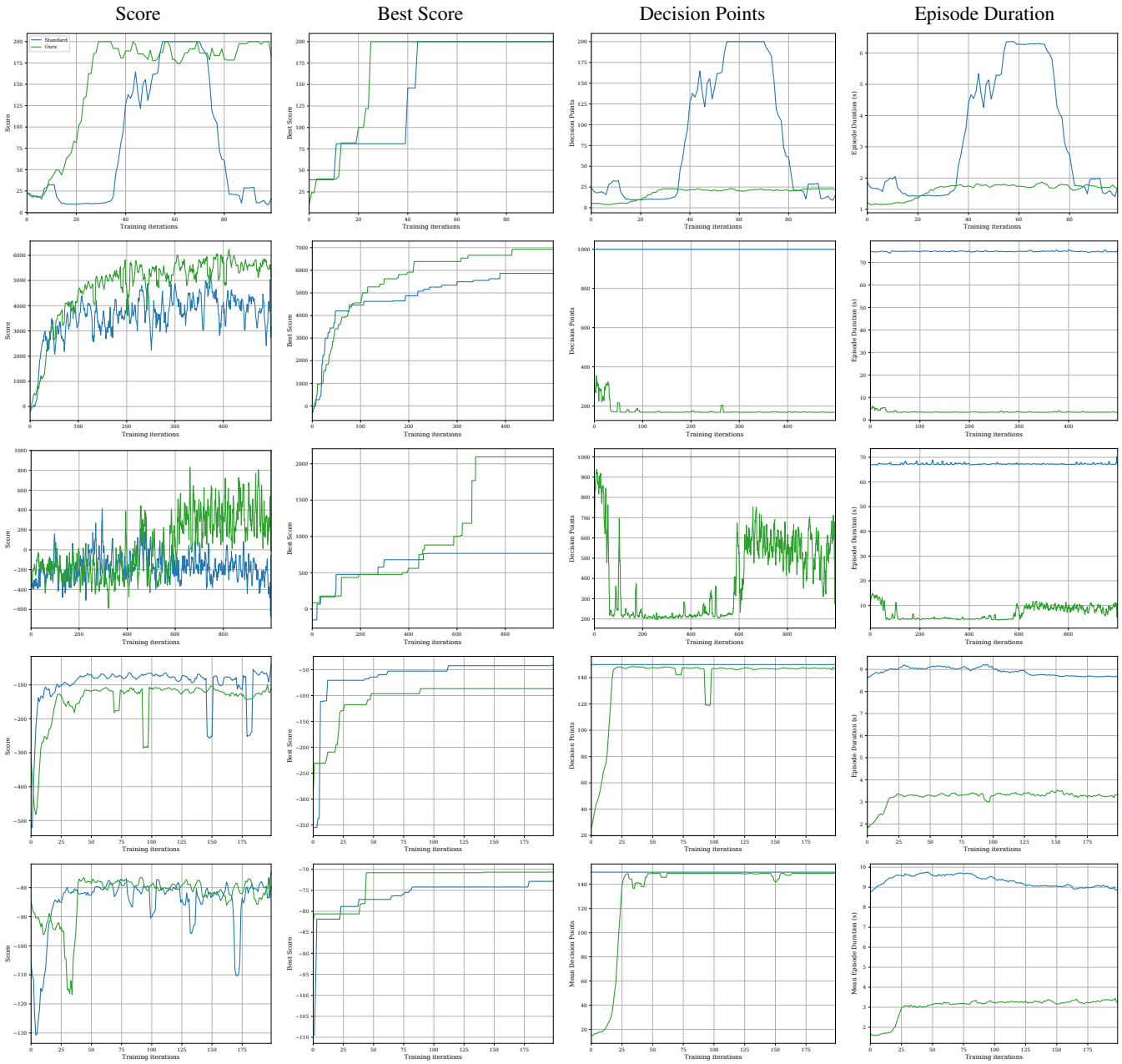


Figure 3: Running average of scores, highest scores achieved, rollout lengths and episode duration for Cart Pole, Half Cheetah, Ant, Reacher and Pusher (from top to bottom) respectively. Using temporally-extended actions helps the agent perform better in Half Cheetah and Ant, while the performance in Cart Pole, Reacher and Pusher is at par with the standard framework.

A_{TE} uses a learned temporally-extended model for planning, while for evaluation we wrap the primitive transition and reward functions in an iterative loop similar to F_{IP}.

Each experiment is run for a specific number of iterations, where in each iteration, we train the model using multiple mini-batches drawn from the replay buffer, and then evaluate the model for one episode. We plot the running average of scores obtained by the agents across the iterations, as well as the highest score achieved by the agent. Additionally, we track the number of decisions made by the agents as well as time required to complete an iteration. The results from our experiments are in Figure 3.

We observe that the proposed framework leads to higher scores while being faster. This speed-up comes from three sources. First, using temporally-extended actions leads to fewer decision points. Second, the reduction in search space further reduces planning time. Third, a decrease in decision points leads to a decrease in number of training examples for the model. As the number of mini-batches depends on the number of training examples in the agent's replay buffer, this also leads to less time spent by the model in training.

In Cart Pole, an episode lasts for 200 primitive actions. However, if the pole falls at any moment, the episode terminates. The sudden increase in decision points and episode duration for A_{standard} ($D = 30$) is because the agent learns to keep the pole from falling for the entire duration of the episode. On the other hand, for A_{TE} ($D_{\text{TE}} = 3$, $\delta t_{\max} = 10\delta_t^{\text{env}}$), there is no sudden spike in the number of decision points and episode duration. When A_{TE} learns to control the pole around episode 30, we simply observe a slight increase in the number of decision points and episode duration.

An episode in Half Cheetah and Ant lasts for 1000 primitive actions. In Half Cheetah, A_{TE} ($D_{\text{TE}} = 15$, $\delta t_{\max} = 7\delta_t^{\text{env}}$) outperforms A_{standard} ($D = 70$) while being 7x faster. In case of Ant, we have an interesting observation. For around 600 training iterations, the average performance of the both the agents hovers around a mean of 0 with no clear trend. During this period, A_{TE} ($D_{\text{TE}} = 15$, $\delta t_{\max} = 7\delta_t^{\text{env}}$) has about 200-250 decision points per episode. However, after this point we see that the performance of A_{TE} increases drastically showing a clear trend, while the performance of A_{standard} ($D = 90$) does not show any improvement. The number of decision points for A_{TE} also increases to 500-700 indicating that the agent is taking actions of small duration. In Reacher and Pusher, the performance of A_{TE} ($D_{\text{TE}} = 5$, $\delta t_{\max} = 20\delta_t^{\text{env}}$) is close to A_{standard} ($D = 70$). The rise in the number of decision points for both these environments indicate that A_{TE} identifies that a small decision timescale works better. This illustrates that in cases when a long duration is not suitable, A_{TE}'s perform is still competitive with the standard solution.

How does varying the two discount factors impact the agent's behaviour? The proposed formulation in Equation (4) has two discount factors. As we discuss next, while some expectations on the effect of γ_1 and γ_2 are intuitive, a complete characterization is not obvious.

We use the cave-mini map (see Figure 5a in Appendix B) in the Dubins Car environment (Chatterjee et al. 2023).

	γ_1	γ_2	Decision Steps	Primitive Steps
Case 1: Fixed γ_1	0.99	1.0	20.2 ± 0.84	121.8 ± 2.95
		0.99	20.2 ± 0.45	122.0 ± 2.00
		0.9	21.6 ± 1.34	121.8 ± 0.84
		0.8	22.2 ± 0.84	122.6 ± 1.52
		0.7	23.2 ± 0.45	122.8 ± 1.64
Case 2: Fixed γ_2	1.0	1.0	19.0 ± 0.71	128.6 ± 1.52
	0.99		20.2 ± 0.84	121.8 ± 2.95
	0.95		18.6 ± 0.55	117.6 ± 0.89
	0.9		16.4 ± 0.55	115.4 ± 0.55

Table 2: When γ_1 is fixed, decreasing γ_2 leads to an increase in number of decision steps. When γ_2 is fixed, decreasing γ_1 leads to a decrease in the number of primitive steps.

For each experiment corresponding to a particular combination of γ_1 and γ_2, we perform 5 runs using random seeds and average the results which are in Table 2. A_{TE} ($D_{\text{TE}} = 50, \delta t_{\max} = 10\delta_t^{\text{env}}$) is able to reach the goal before the maximum number of primitive steps in the environment is reached and so we can focus on the effect of the parameters.

Note that in the proposed setting, even though D_{TE} is fixed, the primitive planning horizon is not fixed and it is dependent on the duration of the actions chosen by the planner. This can result in different levels of discounting in different trajectories. In contrast, in the standard framework, the discounting due to a planning horizon D_{standard} is fixed. This makes predicting the behavior of A_{TE}, upon varying γ_1 and γ_2, difficult.

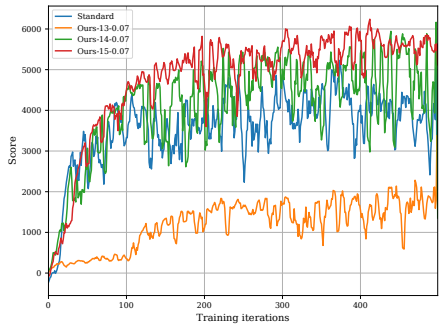
Another observation is that for all decision steps $k > 1$, γ_1 will always be the dominating component in the objective. This is because the exponent term for γ_1 is the number of primitive steps before the decision step k, while the exponent term for γ_2 is proportional to the duration of the temporally-extended action.

To explore the parameters we consider two cases. First, we keep γ_1 fixed and vary γ_2. We hypothesize that smaller values of γ_2 will yield a preference for shorter durations and hence larger number of decision points. Table 2 indeed shows a modest increase in number of decision points, whereas the total number of primitive steps remains the same.

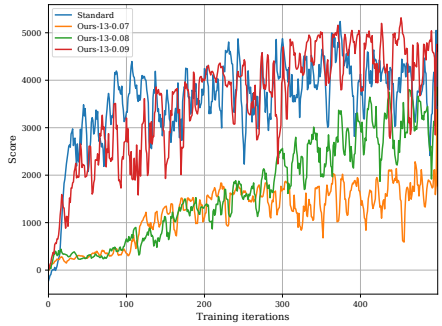
Next, we vary γ_1 and keep γ_2 fixed. In this setting, our hypothesis is that decreasing γ_1 should yield a preference for longer duration and a smaller total number of steps. The intuition is that if we reduce γ_1 and the planner doesn't reduce the number of primitive steps, then the impact of discounting on the overall objective will be higher. Here too, Table 2 confirms these trends.

What is the impact of $\delta t_{\max}$ and D_{TE} on performance? Both $\delta t_{\max}$ and D_{TE} can be used to increase the effective search depth of A_{TE}. While the optimal choice is domain specific, we explore their impact in Half Cheetah to gain insight into their relative strengths.

First, we set $\delta t_{\max}$ and D_{TE} such that the maximal primitive horizon (H) is equivalent for the two frameworks. As shown in Figure 4, this leads to a poor performance in Half



(a) Increase D_{TE}, $\delta t_{\max}$=0.07



(b) Increase $\delta t_{\max}$, D_{TE}=13

Figure 4: Planning horizon for standard framework is 90 with $\delta t = 0.01$. When maximal planning horizon is $13 \times 7 = 91$, the standard framework performs better than us in Half Cheetah. Increasing either D_{TE} or $\delta t_{\max}$ helps, but increasing D_{TE} is more beneficial as it lets the agent choose more flexible trajectories.

Cheetah. This is not unexpected because the effective number of steps for A_{TE} is not its maximal depth. In practice, the average duration will determine the primitive search depth and this may be importance for performance.

As both $\delta t_{\max}$ and D_{TE} impact H, varying one while keeping the other fixed should help segregate the effects. We observe that increasing D_{TE} has more effect than increasing $\delta t_{\max}$ (Figure 4). A potential reason could be the fact that even though both increase H, increasing D_{TE} also lets the planner choose more flexible trajectories and the range of D_{TE} is sufficiently small to be tight.

7 Conclusion and Future Work

Using the standard framework of planning with primitive actions, provides more granular control, but can be computationally expensive. In environments where δ_t^{env} is small, the agent needs to search over a large planning horizon which increases the search space as well as the number of variables it needs to optimize. We propose to use temporally-extended actions where the planner treats the duration of the action as an additional optimization variable. This helps to reduce the complexity of the search by restricting the search space and reducing the planning horizon, which in turn reduces the overall number of variables that the planner needs to optimize. We further show that temporally-extended transition and reward models can be learned using MBRL. An additional advantage in this case is that using our proposed framework with learned models leads to smaller compounding errors. It also leads to less examples in the replay buffer which leads to faster training. However, since trajectories generated using temporally-extended actions are not as flexible as those due to the standard framework, improvement in performance is not guaranteed in every environment.

For the framework to succeed, it is crucial to choose the right value of $\delta t_{\max}$. If it is too small relative to the optimal value, one loses the opportunity to obtain as much improvement as is potentially possible. On the other hand, if it is too large, the agent will require longer to optimize. Developing an algorithm that can identify a correct range for $\delta t_{\max}$ is an important challenge for future work.

Acknowledgements

This work was partly supported by NSF under grant 2246261. Some of the experiments in this paper were run on the Big Red computing system at Indiana University, supported in part by Lilly Endowment, Inc., through its support for the Indiana University Pervasive Technology Institute.

References

Bellemare, M.; Veness, J.; and Bowling, M. 2012. Investigating contingency awareness using Atari 2600 games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 864–871.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47: 253–279.

Biedenkapp, A.; Rajan, R.; Hutter, F.; and Lindauer, M. 2021. TempoRL: Learning when to act. In *International Conference on Machine Learning*, 914–924. PMLR.

Botev, Z. I.; Kroese, D. P.; Rubinstein, R. Y.; and L'Ecuyer, P. 2013. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, 35–59. Elsevier.

Braylan, A.; Hollenbeck, M.; Meyerson, E.; and Miikkulainen, R. 2015. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the twenty-ninth AAAI conference on artificial intelligence*.

Chatterjee, P.; Chapagain, A.; Chen, W.; and Khardon, R. 2023. DiSProD: differentiable symbolic propagation of distributions for planning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 5324–5332.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

Durugkar, I. P.; Rosenbaum, C.; Dernbach, S.; and Mahadevan, S. 2016. Deep reinforcement learning with macro-actions. *arXiv preprint arXiv:1606.04615*.

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

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T. L.; and Boutilier, C. 2013. Hierarchical solution of Markov decision processes using macro-actions. *arXiv preprint arXiv:1301.7381*.

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems*.

Kobilarov, M. 2012. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7): 855–871.

Kwiatkowski, A.; Towers, M.; Terry, J.; Balis, J. U.; Cola, G. D.; Deleu, T.; Goulão, M.; Kallinteris, A.; Krimmel, M.; KG, A.; Perez-Vicente, R.; Pierré, A.; Schulhoff, S.; Tai, J. J.; Tan, H.; and Younis, O. G. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *CoRR*.

Lakshminarayanan, A.; Sharma, S.; and Ravindran, B. 2017. Dynamic action repetition for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2017. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, 2295–2304. PMLR.

Machado, M. C.; Rosenbaum, C.; Guo, X.; Liu, M.; Tesauro, G.; and Campbell, M. 2017. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*.

Mausam; and Weld, D. S. 2008. Planning with Durative Actions in Stochastic Domains. *Journal of Artificial Intelligence Research*, 31: 33–82.

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

Ni, T.; and Jang, E. 2022. Continuous Control on Time. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*.

Ramesh, R.; Tomar, M.; and Ravindran, B. 2019. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*.

Sharma, S.; Srinivas, A.; and Ravindran, B. 2017. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*.

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

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; et al. 2024. The 2023 International Planning Competition.

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

Wang, D.; and Beltrame, G. 2024. Deployable reinforcement learning with variable control rate. *arXiv preprint arXiv:2401.09286*.

Williams, G.; Aldrich, A.; and Theodorou, E. A. 2017. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2): 344–357.

A Connection to (Ni and Jang 2022)

Ni and Jang (2022) propose to learn a policy that outputs the action variables as well as the time scale. In this section, we compare our objective to theirs. First, we repeat Equation (3) which computes the returns of a trajectory τ using our proposed framework:

$$J(\tau) = \sum_{k=1}^{L(\tau)} \gamma^{e_{<k}} \sum_{t=1}^{e_k} \gamma^{t-1} \mathcal{R}_{(e_{<k}+t)}$$

A similar objective is used by (Ni and Jang 2022). However, it is important to note that they control the timescale while we keep the timescale fixed and control the action duration. Let $\delta_k \in [\delta_{\min}, \delta_{\max}]$ be the timescale associated with the action at decision point k. For simplicity, they assume that the timescale is in integers (in physical unit of seconds), which means that δ_k is also the number of execution steps associated with the action. Using this, their objective reduces to

$$J(\tau) = \sum_{k=1}^{L(\tau)} \gamma^{e_{<k}} \mathcal{R}(s_{e_{<k}+1}, a_k)\delta_k \tag{5}$$

where $\mathcal{R}(s_{e_{<k}+1}, a_k)\delta_k$ is a linear approximation of the reward function. So, the objective used by Ni and Jang (2022) can be viewed as an approximation of Equation (3) where the reward function due to a temporally-extended action has been replaced by a linear approximation.

B Experimental Details

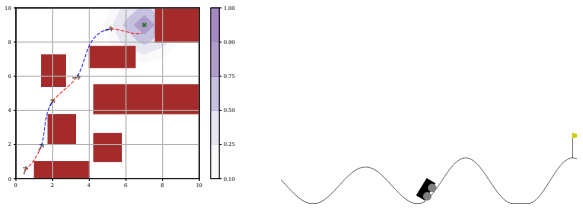
Our framework has two main hyperparameters - range of action duration, which is specified by $\delta t_{\min}$ and $\delta t_{\max}$, and planning horizon. Rather than controlling both these values, $\delta t_{\min}$ is always set to δ_t^{env}, and modifying $\delta t_{\max}$ controls the range.

For environments like Cart Pole and Mountain Car, we use values of planning horizon from prior work for the standard framework, and adjust the depth for A_{TE} by exploring related values. We cannot do this for MuJoCo-based environments since we set the frameskip parameter to 1. Rather we perform a search over some potential values for the planning horizon for the standard framework and $\delta t_{\max}$ and planning horizon for our framework and choose the configuration with the best performance. The other hyperparameters related to online training have been borrowed from (Chua et al. 2018).

Details for Dubins Car environment We use the Dubins Car environment of Chatterjee et al. (2023) where the agent controls the change in linear velocity (Δv) and angular velocity $(\Delta\omega)$. An episode terminates if the agent reaches the goal or if it takes 300 primitive steps. The default timescale for the environment is 0.2 while $\delta t_{\max}$ for our formulation is 20. Setting $\delta t_{\max}$ to such a large value allows us to have the same value across maps and just allow more time for optimization. The planning horizon needs to be tuned for different maps. Table 3 contains the planning horizon for the maps used in the experiments.

Domain	Range of action duration		Planning horizon	
	Standard	Ours	Standard	Ours
Dubins Car [u-shaped map]	0.2	[0.2-20]	1000	75
Dubins Car [cave-mini map]	0.2	[0.2-2]	120	50
Cartpole	1	[1-10]	30	3
Mountain Car	1	[1-100]	100	10
IPC Mountain Car	1	[1-125]	175	12
Reacher	0.01	[0.01, 0.2]	25	5
Pusher	0.01	[0.01, 0.2]	25	5
Ant	0.01	[0.01, 0.07]	70	15
Half Cheetah	0.01	[0.01, 0.07]	90	15

Table 3: Environment parameters for different environments.



(a) (b)

Figure 5: (a) An example of A_{TE} solving the cave-mini map ($\gamma_1 = 0.99, \gamma_2 = 1.0, \delta t_{\max} = 20$). (b) An instance of the IPC Multi-hill Mountain Car.

Note that for our experiment with the cave-mini map, we reduce $\delta t_{\max}$ to prevent the agent from completing the map in a few decision steps.

C Additional Experiments with IPC MC

How does the performance change if we increase the planning horizon? We vary the planning horizon of the agents and compare the performances across instances of IPC MC in Table 4. For the easier problems, a small planning horizon is sufficient, but for the difficult instances, a deeper search is required. The performance of the planner remains relatively stable as the planning horizon increases.

D Details of Experiments with u-shaped maps

We experiment with varying depths using u-shaped maps in the Dubins Car environment. The problem is not trivial as the car faces the obstacle and it needs to first turn around and then find a path the goal, which gives the agent a reward of 100. Every collision with an obstacle gets a penalty of 10. An episode terminates when the agent reaches the goal or when the agent takes 300 primitive steps.

To be successful in this setting, we observe that A_{standard} requires a planning horizon of 1000 and A_{TE} requires a planning horizon of 75. Both the agents use 10,000 samples and 50 optimization steps for each decision. A_{TE} succeeds but not in all runs. We look at the failure cases for A_{TE} and have an interesting observation. For planning horizon of 75 and 100, the failure is due to the fact that the agent runs out of

	D	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5
A_{standard}	60	37.49 ± 51.36	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	80	74.54 ± 41.7	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	100	91.93 ± 0.41	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	125	90.92 ± 0.22	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	150	91.07 ± 0.18	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	175	91.12 ± 0.3	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	200	91.11 ± 0.27	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	225	91.2 ± 0.32	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	250	91.01 ± 0.12	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	275	90.98 ± 0.11	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	300	90.91 ± 0.1	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
A_{TE}	2	-0.01 ± 0.01	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	5	73.52 ± 43.45	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	8	92.58 ± 1.01	90.99 ± 0.7	-0.0 ± 0.0	-0.0 ± 0.0	-0.0 ± 0.0
	10	71.43 ± 46.05	89.79 ± 1.28	87.27 ± 1.86	68.11 ± 42.49	85.02 ± 0.26
	12	92.41 ± 1.82	90.11 ± 1.19	87.44 ± 1.52	86.91 ± 1.53	85.45 ± 0.25
	15	92.33 ± 2.0	90.52 ± 0.6	87.91 ± 0.8	87.51 ± 0.98	85.19 ± 1.15
	20	92.52 ± 1.46	90.12 ± 1.36	87.74 ± 1.99	87.08 ± 2.57	86.43 ± 0.31
	25	92.39 ± 1.85	90.54 ± 1.06	88.39 ± 1.36	87.71 ± 1.61	87.06 ± 0.46
	30	72.07 ± 45.41	91.08 ± 0.37	88.94 ± 0.49	68.0 ± 45.24	86.08 ± 1.81
	35	51.48 ± 57.79	90.91 ± 1.15	89.28 ± 0.2	89.06 ± 0.81	67.07 ± 45.5
	40	71.21 ± 48.64	90.12 ± 1.42	89.75 ± 2.19	87.38 ± 2.38	67.86 ± 45.2

Table 4: Performance of A_{standard} and A_{TE} in various instances of IPC Multi-hill Mountain Car as the planning horizon is increased.

primitive actions, while for planning horizon 50, the failing scenario corresponds to the agent not being able to find a path out of the obstacles region. This means that if we allow the episodes to run for longer, the former failure can be mitigated but the latter cannot. The detailed results are in Table 5 and the failure cases for A_{TE} are shown in Figure 6.

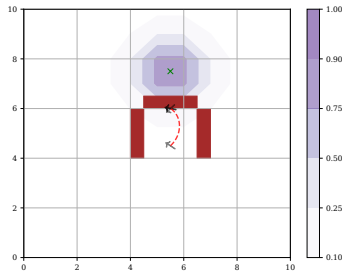
Next, we make the problem difficult by augmenting the action space with 100 dummy action variables. As the agents are unaware that these variables don't contribute to the reward or the dynamics, they still need to search over these variables. We set up the experiment similar to the previous one by keeping the number of samples to 10,000 and varying the planning horizon. For A_{standard}, almost all the runs fail - either because it does not find a path around the obstacle, or crashes due to huge memory requirements. In order to reduce the memory requirements, we perform a second experiment, fixing the planning horizon to 1000 and varying the number of samples. Reducing the number of samples fixes the issue of memory requirements, but does not help the agent identify a path. In contrast, the performance of A_{TE} is mostly unaffected by this addition of dummy action variables. The detailed results are in Table 6. A_{TE} uses the same configuration as the earlier experiment and achieves similar performance. As observed earlier, the failure case of A_{TE} for planning horizon of 50 is due to the fact that it cannot find a path around the obstacles, while the failure case for planning horizon of 100 is because it runs out of primitive steps.

	D	Reward ↑	Decision Steps ↓	Decision time ↓	Time for an episode ↓	Success Probability ↑
A_{standard}	250	-9.95 ± 0.0	300.0 ± 0.0	0.13 ± 0.0	413.46 ± 4.16	0
	500	-9.95 ± 0.0	300.0 ± 0.0	0.29 ± 0.0	806.97 ± 8.82	0
	750	32.04 ± 62.17	213.8 ± 118.03	0.42 ± 0.02	851.53 ± 466.61	0.4
	1000	100.0 ± 0.0	90.8 ± 20.17	0.56 ± 0.02	480.38 ± 102.06	1
A_{TE}	25	-497.53 ± 137.16	4.0 ± 0.0	2.31 ± 0.09	14.1 ± 0.43	0
	50	16.32 ± 187.12	4.2 ± 0.45	3.66 ± 0.1	20.03 ± 1.59	0.8
	75	80.0 ± 44.72	4.4 ± 1.14	5.08 ± 0.24	27.39 ± 5.84	0.8
	100	80.0 ± 44.72	4.4 ± 1.14	6.57 ± 0.21	34.56 ± 7.79	0.8

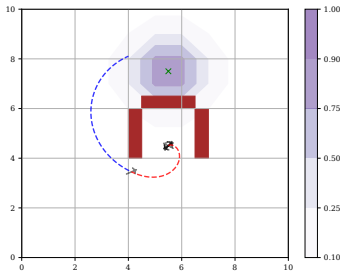
Table 5: Performance of A_{standard} and A_{TE} on the u-shaped map in Dubins Car as the planning horizon is varied. Both agents use 10,000 samples and 50 optimization steps.

	D	Number of samples	Reward ↑	Decision Steps ↓	Decision time ↓	Time for an episode ↓	Success Probability ↑
A_{standard}	250	10000	-9.95 ± 0.0	300.0 ± 0.0	0.66 ± 0.0	580.85 ± 2.91	0
	500		12.04 ± 49.17	255.2 ± 100.18	1.38 ± 0.01	970.11 ± 378.22	0.2
	750		-	-	-	-	0
	1000		-	-	-	-	0
A_{standard}	1000	100	-919.43 ± 238.36	300.0 ± 0.0	0.46 ± 0.03	1567.29 ± 25.31	0
		1000	-15.92 ± 8.9	300.0 ± 0.0	0.73 ± 0.0	1663.75 ± 8.56	0
		10000	-	-	-	-	0
A_{TE}	25	10000	-479.62 ± 145.46	4.0 ± 0.0	2.31 ± 0.09	14.05 ± 0.25	0
	50		18.31 ± 182.67	3.6 ± 0.55	4.0 ± 0.12	18.72 ± 2.41	0.8
	75		100.0 ± 0.0	4.6 ± 0.89	5.36 ± 0.16	30.01 ± 4.56	1.0
	100		80.0 ± 44.72	4.4 ± 0.89	6.87 ± 0.12	35.74 ± 6.44	0.8

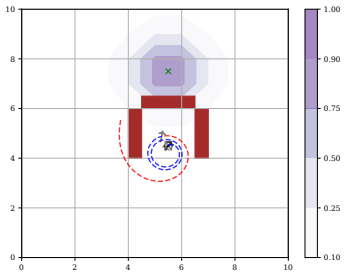
Table 6: Performance of A_{standard} and A_{TE} on the u-shaped map in Dubins Car when the action space is augmented with 100 dummy actions. Both agents use 50 optimization steps. Missing values indicate that the particular configuration was not feasible due to large memory requirements.



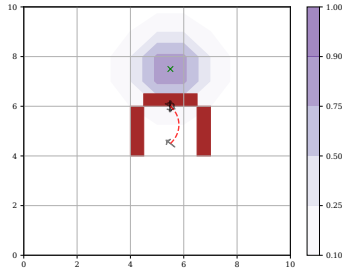
(a) $D_{\text{TE}} = 50, |\mathcal{A}| = 2$



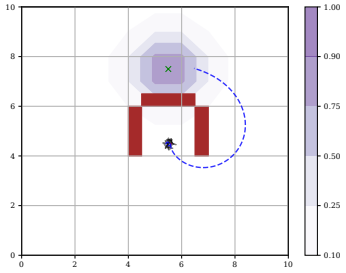
(b) $D_{\text{TE}} = 75, |\mathcal{A}| = 2$



(c) $D_{\text{TE}} = 100, |\mathcal{A}| = 2$



(d) $D_{\text{TE}} = 50, |\mathcal{A}| = 102$



(e) $D_{\text{TE}} = 100, |\mathcal{A}| = 102$

Figure 6: Failure cases for A_{TE} in the u-shaped map. When $|\mathcal{A}| = 2$, for $D_{\text{TE}} = 75$ and $D_{\text{TE}} = 100$, the agent identifies a path around the obstacles but runs out of the maximum number of primitive steps. On increasing the action space, a similar failure occurs for $D_{\text{TE}} = 100$. The failures due to $D_{\text{TE}} = 50$ happen because the agent does not find a path around the obstacles.