A Notations and Proof

A.1 Basic Concepts and Notations

Markov Decision Process (MDP): The reinforcement learning problem can be described with an MDP, denoted by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to R^1$ is the reward function, and $\gamma \in (0, 1]$ is the discount factor.

State transition graph in an MDP: The state transitions in $\mathcal{M}$ can be modelled as a state transition graph $G = (V_G, E_G)$, where V_G is a set of vertices representing the states in $\mathcal{S}$, and E_G is a set of undirected edges representing state adjacency in $\mathcal{M}$. We note that:

Remark. *There is an edge between state s and s' (i.e., s and s' are adjacent) if and only if $\exists\, a \in \mathcal{A},\ s.t.\ \mathcal{P}(s, a, s') > 0\ \vee\ \mathcal{P}(s', a, s) > 0$.*

The adjacency matrix A of G is an $|\mathcal{S}| \times |\mathcal{S}|$ matrix whose (i, j) entry is 1 when s_i and s_j are adjacent, and 0 otherwise. The degree matrix D is a diagonal matrix whose entry (i, i) equals the number of edges incident to s_i. The Laplacian matrix of G is defined as $L = D - A$. Its second smallest eigenvalue $\lambda_2(L)$ is called the algebraic connectivity of the graph G, and the corresponding normalized eigenvector is called the Fiedler vector [36]. Last, the normalized Laplacian matrix is defined as $\mathcal{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$.

A.2 Derivation of the Variational Lower Bound in Section 3.1

To start with, we can find a lower bound of the second term in Eq. (4) as follows:

$$
\begin{aligned}
&-\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}[I(c,\tau|s_0)] = -\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[-\sum_{\tau} P(\tau|s_0)\log P(\tau|s_0) + \sum_{c,\tau} P(c,\tau|s_0)\log P_\theta(\tau|s_0,c)\right] \\
&= -\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[-\sum_{c,\tau} P(c,\tau|s_0)\log P(\tau|s_0) + \sum_{c,\tau} P(c,\tau|s_0)\log P_\theta(\tau|s_0,c)\right] \\
&= -\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,\tau} P_\omega(c|s_0)P_\theta(\tau|c,s_0)\log\frac{P_\theta(\tau|s_0,c)}{P(\tau|s_0)}\right] \\
&= -\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,\tau} P_\omega(c|s_0)P_\theta(\tau|c,s_0)\ \left[\log\frac{P_\theta(\tau|s_0,c)}{Unif(\tau|s_0)} - \log\frac{P(\tau|s_0)}{Unif(\tau|s_0)}\right]\right] \\
&= -\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\left[\sum_{c,\tau} P_\omega(c|s_0)P_\theta(\tau|c,s_0)\log\frac{P_\theta(\tau|s_0,c)}{Unif(\tau|s_0)}\right] - D_{KL}(P(\tau|s_0)||Unif(\tau|s_0))\right] \\
&\geq -\beta \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,\tau} P_\omega(c|s_0)P_\theta(\tau|c,s_0)\log\frac{P_\theta(\tau|s_0,c)}{Unif(\tau|s_0)}\right] \\
&= -\beta \mathop{\mathbb{E}}_{\substack{s_0\sim\mu(\cdot)\\ c\sim P_\omega(\cdot|s_0)}}[D_{KL}(P_\theta(\tau|s_0,c)||Unif(\tau|s_0))]
\end{aligned}
\tag{13}
$$

where $D_{KL}(\cdot)$ denotes the Kullback-Leibler (KL) Divergence which is non-negative, $P_\theta(\tau|s_0, c) = \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t, c)P(s_{t+1}|s_t, a_t)$ is the probability of the trajectory τ given s_0 and c under the option policy π_θ, and $Unif(\tau|s_0)$ is the probability of the same trajectory given s_0 under the random walk policy. Instead of explicitly calculating $P(\tau|s_0)$ which is impractical, we introduce $Unif(\tau|s_0)$ to convert the second term in Eq. (4) into a regularization term to encourage exploration and diversity.

As for the first term in Eq. (4), we can deal with it as Eq. (14), where we introduce $P_\phi(c|s_0, G)$ as the variational estimation of $P(c|s_0, G)$ which is hard to acquire. The first inequality in Eq. (14) is based on the fact that KL Divergence is non-negative. While, the second inequality holds because we only keep the trajectory τ from which G is sampled, so the trajectory and its corresponding landmark states form a bijection.

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}[I(c,G|s_0)] = \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[-\sum_c P_\omega(c|s_0)\log P_\omega(c|s_0) + \sum_{c,G} P_\omega(c|s_0)P_\theta(G|s_0,c)\log P(c|s_0,G)\right] \\
&= \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[-\sum_c P_\omega(c|s_0)\log P_\omega(c|s_0) + \sum_{c,G} P_\omega(c|s_0)P_\theta(G|s_0,c)\log\left[P_\phi(c|s_0,G)\frac{P(c|s_0,G)}{P_\phi(c|s_0,G)}\right]\right] \\
&= \mathcal{H}(\mathcal{C}|\mathcal{S}) + \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,G} P_\omega(c|s_0)P_\theta(G|s_0,c)\log P_\phi(c|s_0,G)\right] + \sum_G P(G|s_0)D_{KL}(P(c|s_0,G)||P_\phi(c|s_0,G)) \\
&\geq \mathcal{H}(\mathcal{C}|\mathcal{S}) + \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,G} P_\omega(c|s_0)P_\theta(G|s_0,c)\log P_\phi(c|s_0,G)\right] \\
&= \mathcal{H}(\mathcal{C}|\mathcal{S}) + \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,G} P_\omega(c|s_0)\left[\sum_{\tau'} P_\theta(\tau'|s_0,c)P^{DPP}(G|\tau')\right]\log P_\phi(c|s_0,G)\right] \\
&\geq \mathcal{H}(\mathcal{C}|\mathcal{S}) + \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}\left[\sum_{c,\tau} P_\omega(c|s_0)P_\theta(\tau|s_0,c)P^{DPP}(G|\tau)\log P_\phi(c|s_0,G)\right]
\end{aligned}
\tag{14}
$$

A.3 Derivation of the Gradients Shown in Section 3.3

First, we take the gradient with respect to ω and get the following result:

$$
\begin{aligned}
\nabla_\omega\mathcal{L} = \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)}[&-\sum_c[\nabla_\omega P_\omega(c|s_0)\log P_\omega(c|s_0) + \nabla_\omega P_\omega(c|s_0)] \\
&+ \sum_{c,\tau}\nabla_\omega P_\omega(c|s_0)P_\theta(\tau|s_0,c)P^{DPP}(G|\tau)\log P_\phi(c|s_0,G) \\
&- \beta\sum_c \nabla_\omega P_\omega(c|s_0)D_{KL}(P_\theta(\tau|s_0,c)||Unif(\tau|s_0)) \\
&+ \alpha_1\sum_{c,\tau}\nabla_\omega P_\omega(c|s_0)P_\theta(\tau|s_0,c)f(\tau) \\
&- \alpha_2\sum_c \nabla_\omega P_\omega(c|s_0)\sum_{\overrightarrow{\tau}(s_0,c)} P_\theta(\overrightarrow{\tau}|s_0,c)g(\overrightarrow{\tau}(s_0,c)) \\
&+ \alpha_3\sum_c \nabla_\omega P_\omega(c|s_0)\sum_{\overrightarrow{\tau}(s_0,c)} P_\theta(\overrightarrow{\tau}|s_0,c)h(\bigcup_{c'}\overrightarrow{\tau}(s_0,c'))]
\end{aligned}
\tag{15}
$$

Given that $\nabla_\omega P_\omega(c|s_0) = P_\omega(c|s_0)\nabla_\omega\log P_\omega(c|s_0)$ and the definition of KL Divergence, i.e., $D_{KL}(P_\theta(\tau|s_0,c)||Unif(\tau|s_0)) = \sum_\tau P_\theta(\tau|s_0,c)\sum_{t=0}^{T-1}[\log\pi_\theta(a_t|s_t,c) - \log\pi_{unif}(a_t|s_t)]$, we can simplify Eq. (15) as:

$$
\nabla_\omega\mathcal{L} = \mathop{\mathbb{E}}_{\substack{s_0\sim\mu(\cdot)\\ c\sim P_\omega(\cdot|s_0)}}\left[\nabla_\omega\log P_\omega(c|s_0)A^{P_\omega}(c,s_0)\right]
\tag{16}
$$

where the related advantage function $A^{P_\omega}(c,s_0)$ is defined as:

$$
\begin{aligned}
A^{P_\omega}(c,s_0) = &-\log P_\omega(c|s_0) + \mathop{\mathbb{E}}_{\overrightarrow{\tau}_{(s_0,c)}\sim P_\theta(\cdot|s_0,c)}\left[-\alpha_2 g(\overrightarrow{\tau}_{(s_0,c)}) + \alpha_3 h(\bigcup_{c'}\overrightarrow{\tau}_{(s_0,c')})\right] \\
&+ \mathop{\mathbb{E}}_{\tau\sim P_\theta(\cdot|s_0,c)}\left[P^{DPP}(G|\tau)\log P_\phi(c|s_0,G) - \beta\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t,c) + \alpha_1 f(\tau)\right]
\end{aligned}
\tag{17}
$$

Next, we calculate the gradient with respect to θ as follows:

$$\begin{aligned}\nabla_\theta \mathcal{L} = & \mathop{\mathbb{E}}_{s_0\sim\mu(\cdot)} \sum_{c,\tau} P_\omega(c|s_0)\nabla_\theta P_\theta(\tau|s_0,c) P^{DPP}(G|\tau)\log P_\phi(c|s_0,G) \\ & - \beta \sum_{c,\tau} P_\omega(c|s_0)\nabla_\theta P_\theta(\tau|s_0,c) \sum_{t=0}^{T-1} [\log \pi_\theta(a_t|s_t,c) - \log \pi_{unif}(a_t|s_t)] \\ & - \beta \sum_{c,\tau} P_\omega(c|s_0) P_\theta(\tau|s_0,c) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t,c) \\ & + \alpha_1 \sum_{c,\tau} P_\omega(c|s_0)\nabla_\theta P_\theta(\tau|s_0,c) f(\tau) \\ & + \sum_{c} P_\omega(c|s_0) \sum_{\overrightarrow{\tau}(s_0,c)} \nabla_\theta P_\theta(\overrightarrow{\tau}|s_0,c) \left[-\alpha_2 g(\overrightarrow{\tau}(s_0,c)) + \alpha_3 h(\bigcup_{c'} \overrightarrow{\tau}(s_0,c'))\right]\end{aligned} \tag{18}$$

With $\nabla_\theta P_\theta(\tau|s_0,c) = P_\theta(\tau|s_0,c)\nabla_\theta \log P_\theta(\tau|s_0,c) = P_\theta(\tau|s_0,c)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t,c)$, and $\nabla_\theta P_\theta(\overrightarrow{\tau}|s_0,c) = P_\theta(\overrightarrow{\tau}|s_0,c)\sum_{m=1}^{M}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^m|s_t^m,c)$ where s_t^m (a_t^m) is the state (action) at step t in trajectory m, Eq. (18) can be written as follows:

$$\begin{aligned}\nabla_\theta \mathcal{L} = & \mathop{\mathbb{E}}_{s_0,c,\tau}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t,c)\left[P^{DPP}(G|\tau)\log P_\phi(c|s_0,G) - \beta\sum_{t=0}^{T-1}\log \pi_\theta(a_t|s_t,c) + \alpha_1 f(\tau)\right]\right] \\ & + \mathop{\mathbb{E}}_{s_0,c,\overrightarrow{\tau}}\left[\sum_{m=1}^{M}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^m|s_t^m,c)\left[-\alpha_2 g(\overrightarrow{\tau}_{(s_0,c)}) + \alpha_3 h(\bigcup_{c'}\overrightarrow{\tau}_{(s_0,c')})\right]\right] \\ = & \mathop{\mathbb{E}}_{s_0,c,\overrightarrow{\tau}}\left[\sum_{m=1}^{M}\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t^m|s_t^m,c) A_m^{\pi_\theta}(\overrightarrow{\tau},s_0,c)\right]\end{aligned} \tag{19}$$

where the advantage term is as Eq. (20), $\overrightarrow{\tau} = \{\tau_1,\cdots,\tau_M\}$, $\tau_m = (s_0^m, a_0^m, \cdots, s_{T-1}^m, a_{T-1}^m, s_T^m)$:

$$\begin{aligned}A_m^{\pi_\theta}(\overrightarrow{\tau},s_0,c) = & \frac{P^{DPP}(G_m|\tau_m)\log P_\phi(c|s_0,G_m)}{M} - \frac{\beta}{M}\sum_{t=0}^{T-1}\log \pi_\theta(a_t^m|s_t^m,c) \\ & + \frac{\alpha_1}{M} f(\tau_m) - \alpha_2 g(\overrightarrow{\tau}_{(s_0,c)}) + \alpha_3 h(\bigcup_{c'}\overrightarrow{\tau}_{(s_0,c')})\end{aligned} \tag{20}$$

Then, it's not hard to see the relationship between A^{P_ω} and $A_m^{\pi_\theta}$ as:

$$A^{P_\omega}(c,s_0) = -\log P_\omega(c|s_0) + \mathop{\mathbb{E}}_{\overrightarrow{\tau}}\left[\sum_{m=1}^{M} A_m^{\pi_\theta}(\overrightarrow{\tau},s_0,c)\right] \tag{21}$$

B Pseudo Code of ODPP

In this section, we present the pseudo code of our proposed algorithm as Algorithm 1.

Algorithm 1 Unsupervised Option Discovery based on DPP (ODPP)

Initialize the prior network P_ω, policy network π_θ and decoder P_ϕ
for $each\ training\ episode$ **do**
  $\overrightarrow{\tau} \longleftarrow \{\}$
  for $i = 1, 2, \ldots, N$ **do**
    Sample an initial state $s_0^i \sim \mu(\cdot)$ and an option $c \sim P_\omega(\cdot|s_0^i)$
    Collect a trajectory $\tau_i = (s_0^i, a_0^i, \cdots, s_{T-1}^i, a_{T-1}^i, s_T^i)$, where $a_t^i \sim \pi_\theta(\cdot|s_t^i, c)$
    $\overrightarrow{\tau} \longleftarrow \overrightarrow{\tau} \cup \{\tau_i\}$
  end for
  Update P_ω, π_θ with PPO and P_ϕ with SGD, based on $\overrightarrow{\tau}$ and Eq. (10)-(12)
end for

C Implementation Details and Analysis of ODPP

C.1 The Choice of Diversity Measure

The expected cardinality is a better choice for the diversity measure than the likelihood shown as Eq. (1). Using the log-likelihood based on the Determinant of the DPP kernel matrix directly would heavily penalize repeated items in the sampled set $\mathcal{W}$ in Eq. (1). For example, if there are very similar points in $\mathcal{W}$, the corresponding rows in the kernel matrix will be almost identical and lead to a zero determinant, which will cause numerical issues for the logarithm function. Take our work as an example: at the beginning of the training stage, the moving range of the Mujoco agent is very limited, then we always include very close states in a trajectory, which will always lead to a zero determinant and thus cannot provide training signals. However, the expected cardinality only counts the number of diverse states in a trajectory that will not be heavily influenced by the repeated items. Therefore, we select the expected cardinality as the diversity measure in this paper.

C.2 Fast Greedy MAP Inference for DPP

The maximum a posteriori (MAP) inference of DPP aims at finding the subset of items with the highest possibility under the DPP measure, which is NP-hard [25]. The log-probability function in DPP, i.e., $l(W) = \log\det(L_W)$, is submodular, which means:

$$\forall\, i \in \mathcal{W},\ W_1 \subseteq W_2 \subseteq \mathcal{W}\backslash\{i\},\ l(W_1 \cup \{i\}) - l(W_1) \geq l(W_2 \cup \{i\}) - l(W_2) \tag{22}$$

Thus, the MAP inference for DPP can be converted to a submodular maximization problem, where greedy algorithms have shown promising empirical success. Recently, the authors of [26] propose a fast greedy method for MAP inference in DPP with time complexity $\mathcal{O}(S^2N)$ to return S items out of a sample space of size N. The key step of their algorithm is that for each iteration, the item which maximizes the marginal gain:

$$j = \underset{i \in \mathcal{W}\backslash W_{map}}{\arg\max}\ l(W_{map} \cup \{i\}) - l(W_{map}) \tag{23}$$

is added to W_{map} starting from an initial set $\emptyset$, until the maximal marginal gain becomes negative or the target sample number is reached (i.e., $stopping\ criteria$). This part is not our contribution. We provide its detailed pseudo code as Algorithm 2. For the derivation, please refer to the original paper [26]. We also provide its implementation code as a part of the complete code of ODPP.

C.3 Computation of the Laplacian Spectrum for the Infinite-scale State Spaces

As mentioned in Section 3.2, the feature vector of each state $\overrightarrow{b_i}$ is defined with the eigenvectors corresponding to the D-smallest eigenvalues of the Laplacian matrix of the state transition graph. However, for the infinite-scale state spaces, we cannot obtain this Laplacian spectrum through matrix-based methods, so we adopt the NN-based method proposed in [28] for estimating the Laplacian

Algorithm 2 Fast Greedy MAP Inference for DPP

Input: The set of items $\mathcal{W}$ and its kernel matrix $\mathbf{L}$, $stopping\ criteria$
Initialize: For $i \in \mathcal{W}$, $\mathbf{c}_i = []$, $d_i^2 = \mathbf{L}_{ii}$; $W_{map} = \{j\}$, where $j = \arg\max_{i \in \mathcal{W}} \log(d_i^2)$
while $stopping\ criteria$ not satisfied **do**
  for $i \in \mathcal{W} \backslash W_{map}$ **do**
    $e_i = (\mathbf{L}_{ji} - < \mathbf{c}_j, \mathbf{c}_i >)/d_j$
    $\mathbf{c}_i = [\mathbf{c}_i\ e_i]$, $d_i^2 = d_i^2 - e_i^2$
  end for
  $j = \arg\max_{i \in \mathcal{W} \backslash W_{map}} \log(d_i^2)$, $W_{map} = W_{map} \cup \{j\}$
end while
Return W_{map}

spectrum, which has been proved to be scalable for infinite-scale state spaces and sufficiently accurate compared with the groundtruth. Since this algorithm is not our contribution, we only provide the take-away messages here for implementation.

According to [28], the k smallest eigenvalues $\lambda_{1:k}$ and corresponding eigenvectors $v_{1:k}$ of the Laplacian L can be estimated by: ($k = D$ for our case)

$$\min_{v_1, \cdots, v_k} \sum_{i=1}^{k} (k - i + 1) v_i^T L v_i,\ s.t.\ v_i^T v_j = \delta_{ij}, \forall\, i, j = 1, \cdots, k \tag{24}$$

For the large-scale state space, the eigenvectors can be represented as a neural network that takes a state s as input and outputs a k-dimension vector $[f_1(s), \cdots, f_k(s)]$ as an estimation of $[v_1(s), \cdots, v_k(s)]$. Accordingly, the objective in Equation (24) can be expressed as: (please refer to [28] for details)

$$G(f_1, \cdots, f_k) = \frac{1}{2} \mathbb{E}_{(s,s') \sim \mathcal{T}} \left[\sum_{l=1}^{k} \sum_{i=1}^{l} (f_i(s) - f_i(s'))^2 \right] \tag{25}$$

where $\mathcal{T}$ is a set of state-transitions collected by interacting with the environment through a random policy. Further, the orthonormal constraints in Equation (24) are implemented as a penalty term:

$$P(f_1, \cdots, f_k) = \alpha \mathbb{E}_{s \sim \rho, s' \sim \rho} \left[\sum_{l=1}^{k} \sum_{i=1}^{l} \sum_{j=1}^{l} (f_i(s) f_j(s) - \delta_{ij})(f_i(s') f_j(s') - \delta_{ij}) \right] \tag{26}$$

where α is the weight term and ρ is the distribution of states in $\mathcal{T}$. To sum up, the eigenfunctions f can be trained as an NN by minimizing the loss function:

$$L(f_1, \cdots, f_k) = G(f_1, \cdots, f_k) + P(f_1, \cdots, f_k) \tag{27}$$

C.4 Computation Complexity Analysis

To solve the MAP inference shown as Eq. (3), we adopt a fast greedy algorithm, of which the implementation details are in Appendix C.2. The time complexity for this greedy algorithm is $\mathcal{O}(S^2 N)$ to return S items out of a sample space of size N, which can easily scale to N = 1000 or 10000. In our setting, we need to sample 10 landmarks out of 50 states in a trajectory which can be solved in real-time. Note that trajectory length for an option does not have to be large and can be fine-tuned as a hyperparameter. For example, in DIAYN [7], they use 100 or 10 for different tasks.

As for the spectral feature vector, i.e., $\overrightarrow{b_i}$ in Section 3.2, we do not need to explicitly computing the Laplacian matrix and it eigenvectors. Instead, we can get the eigenvectors corresponding to the D smallest eigenvalues of the Laplacian matrix as the output of a neural network trained with the representation learning algorithm introduced in Section C.3, so that our algorithm can scale to infinite state spaces.

Finally, as specified in Eq. (6)-(8), calculating the three DPP-based losses necessitates eigen decomposition of kernel matrices associated with sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$. This process exhibits a time complexity of $\mathcal{O}(N^3)$, where N represents the kernel matrix dimension. The N values for sets $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ correspond to the number of states in a trajectory (i.e., 50), the number of trajectories in a batch pertaining to the

same option starting from the same initial state (i.e., 20), and the total number of trajectories in a batch (i.e., 200), respectively. Thus, the objectives in Eq. (6)-(8) can all be computed in real-time on a regular PC. Deep reinforcement learning are typically performed in batches, which ensures real-time learning and efficient updates in our algorithm.

C.5 Important Hyperparameters

First, we introduce the structure of the networks used in our algorithm and the baselines as follows. We use s_dim, a_dim to represent the dimension of the state space and action space respectively, and use c_num to represent the number of options to learn at a time, which can be 10, 20, 40 or 60 in our experiments. Also, we use $tanh$ and $relu$ to denote the hyperbolic tangent function and rectified linear unit used as the activation functions, $FC(X, Y)$, $BiLSTM(X, Y)$ to denote the fully-connected and bidirectional LSTM layer with the input size X and output size Y.

- The prior network P_ω is used in all the algorithms other than DCO and its structure is $[FC(s\_dim, 64),\ tanh,\ FC(64, 64),\ tanh,\ FC(64, 64),\ tanh,\ FC(64, c\_num)]$. The value network corresponding to P_ω has the same structure as P_ω, except that the output is of size 1.
- The policy network π_θ is used in all the algorithms, with the structure $[FC(s\_dim + c\_num, 64),\ tanh,\ FC(64, 64),\ tanh,\ FC(64, 64),\ tanh,\ FC(64, a\_dim),\ tanh]$. Its corresponding value network has the same structure except that the output is of size 1 and there is not $tanh$ at the end.
- The decoder P_ϕ used in ODPP and VALOR takes a sequence of states in the trajectory as input, so it uses bidirectional LSTM as part of the network, i.e., $[BiLSTM(s\_dim, 64),\ FC(2 * 64, c\_num)]$. While, the decoder of VIC and DIAYN takes one state as input rather than sequential data, so it uses the fully-connected layer instead, i.e., $[FC(s\_dim, 180),\ tanh,\ FC(180, 180),\ tanh,\ FC(180, 180),\ tanh,\ FC(180, c\_num)]$.
- The option selector P_ψ has the same structure as P_ω and is used in all the algorithms.
- The eigenfunction network introduced in Section C.3 is used in ODPP and DCO to estimate the Laplacian spectrum. Its structure is $[FC(s\_dim, 256),\ relu,\ FC(256, 256),\ relu,\ FC(256, 256),\ relu,\ FC(256, 256),\ relu, FC(256, 30)]$, where 30 denotes the dimension of the feature vector $\overrightarrow{b_i}$ mentioned in Section 3.2.

Next, we introduce the weights for each term in the objective function of ODPP (Eq. (5) and (9)): $\beta = 10^{-3}, \alpha_1 = 10^{-4}, \alpha_2 = 10^{-2}, \alpha_3 = 10^{-2}$. For other parameters, please refer to the configuration file in the provided codes. The codes are run on a machine with 4 GeForce RTX 2080 GPUs.

In our approach, we fine-tune important hyperparameters using a sequential, greedy method based on options' visualization, such as in Figure 2, and following our ablation study process. For instance, in Figure 2(e), we retain only the $\mathcal{L}^{IB}$ objective and select its weight $\beta = 10^{-3}$ from five possible choices: $1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$, guided by the visualization results. Next, as Figure 2(f), we introduce $\mathcal{L}_1^{DPP}$ and fine-tune the corresponding weight α_1 while keeping β fixed at 10^{-3}. Ultimately, we incorporate $\mathcal{L}_2^{DPP}$ and $\mathcal{L}_3^{DPP}$, adjusting α_2 and α_3 accordingly. It is crucial to note that the final two terms must work in tandem to ensure that the discovered options exhibit diversity across different options and consistency for a specific option choice.

D Additional Evaluation Results

D.1 Complementary Results on the Effect of the Prior Network

As discussed in Section 3, we learn a prior network P_ω concurrently with the option policy network π_θ. Figure 6 demonstrates how initializing the option selector P_ψ with P_ω can lead to further performance improvement in the downstream task, using the Point Corridor goal-achieving task as an example. This is based on the fact that both networks share the same structure.

First, in (a), we sample 10,000 trajectories for each option and visualize the agent orientation distribution corresponding to different options. In (b), we present the coordinate system setup along

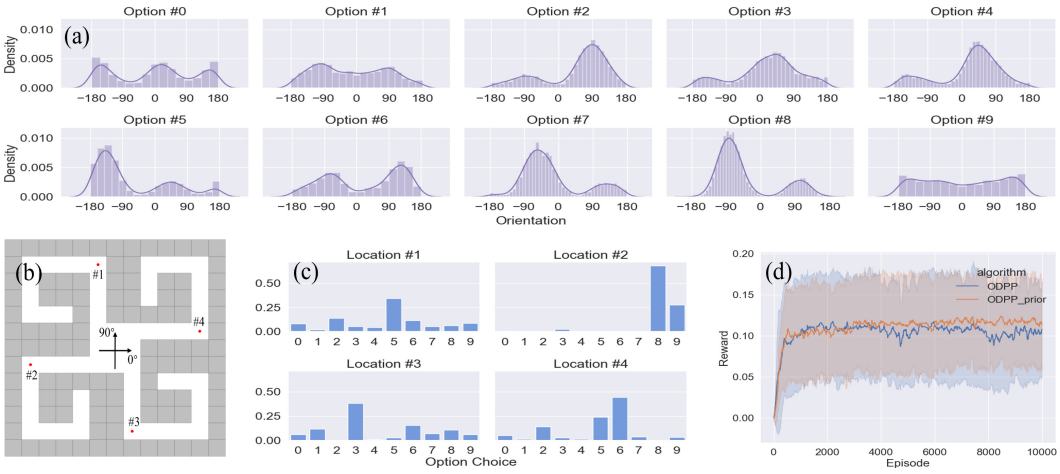
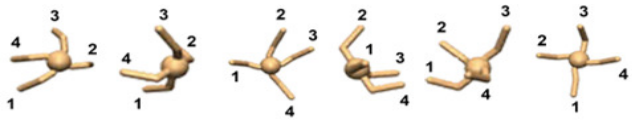


Figure 6: (a) Agent orientation distribution corresponding to different options. (b) Setup of the coordinate system and start points. (c) The output distribution of the prior network at different start points. (d) Performance improvement in the downstream task when applying the prior initialization. The trained prior gives preference to more useful options at corresponding states. At Location #1, Option #5, which tends to go left or down, is preferred; at Location #3, Option #3 is preferred which can lead the agent to go up or right. Moreover, Option #8 is preferred at Location #2 to lead the agent to go down, while Option #6 is preferred at Location #4 to lead the agent to go up.

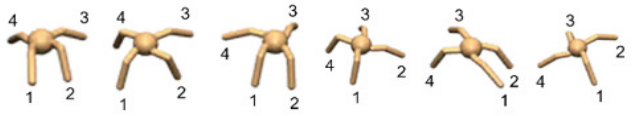
with the four turning points for evaluation. The option choices at these turning points provided by the prior network are displayed in (c). It can be observed that P_ω favors the most significant options at a given state. For instance, at Location #1, Option #5, which tends to go left or down as shown in (a), is preferred, while at Location #3, Option #3 is favored, guiding the agent to go up or right. Lastly, in (d), we demonstrate that initializing the option selector with the prior network can further enhance the agent's performance in the downstream goal-achieving task.

D.2 More Visualization of the Learned Ant Locomotion Behaviors

In Figure 7, we show more visualization results of the learned Ant locomotion behaviors without the supervision of task-specific rewards.

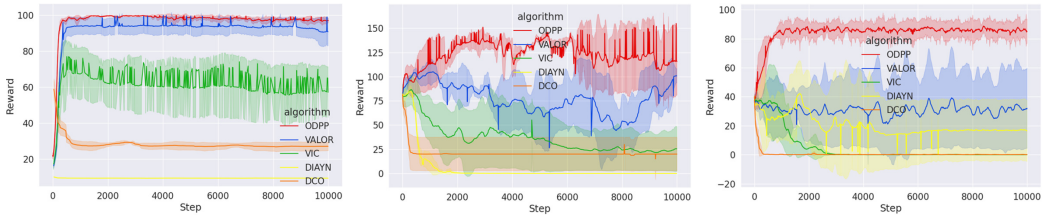


(a) Flip



(b) Rotation

Figure 7: (a) The Ant agent learns to flip over first, then tries to flip back, and finally stands on its Leg 1. (b) The Ant agent walks to the right while rotating. It uses Leg 2&3 as the front legs at first and Leg 1&2 as the front leags at last.



(a) CartPole (b) RiverRaid (c) AirRaid

Figure 8: Evaluation results on the Atari games (including CartPole, RiverRaid and AirRaid) to show the superiority of our algorithm on more general, non-maze tasks. Each experiment is repeated for five times with different random seeds. The solid lines represent the mean value across different random seeds of the trajectory reward of the learned options in the training process and the shadow areas represent the standard deviation. Our algorithm performs the best in all the three tasks, and the performance improvement becomes more significant as the task difficulty increases.

D.3 Evaluation in Atari Video Games

To further demonstrate the applicability and effectiveness of our algorithm (i.e., ODPP), as shown in Figure 8, we compare it with the SOTA baselines on the more general, challenging Atari tasks [3], including CartPole, RiverRaid, and AirRaid. For CartPole, a pole is attached by an unactuated joint to a cart, which moves along a frictionless track; the pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity. While, RiverRaid is a top-down shooting game with the aim of destroying enemy tankers, helicopters, and jets; the player gets a score for each entity destroyed; with only 4 lives and finite fuel (can be refilled in game), the player aims to maximize the total score. Last, in AirRaid, the player controls a ship that scrolls side-to-side directly above two buildings, with the objective of protecting the buildings from being destroyed by the bombs of enemy ships above. CartPole has a 4-dimensional continuous state space. While, for RiverRaid and AirRaid, they use gray-scale images as states which are flattened as 128-dimensional vectors for input.

We adopt ODPP and the baselines to learn skills (i.e., options) for these tasks in an unsupervised manner. Specifically, the skills are discovered/trained based on the mutual information or Laplacian-based objectives. Then, these skills are evaluated with the carefully-crafted reward functions designed for each task, which are provided by the designers of the Atari benchmark. For each algorithm, we learn 10 skills of which the average cumulative rewards (i.e., the sum of the rewards within the duration of a skill) in the training process are shown in Figure 8. The skill duration is set as 100 for CartPole and 50 for the other two. Note that the complete episode horizon is 200 for CartPole and 10000 for AirRaid and RiverRaid. Thus, it would be unfair to compare the cumulative reward of a skill with the one of a whole episode in a game. For each task, we repeat the evaluations for five times with different random seeds, and plot the mean value across the five experiments as the solid lines and the standard deviation as the shadow areas. It can be observed from Figure 8 that our algorithm performs the best in all the three tasks, and the performance improvement becomes more significant as the task difficulty increases. Similar to the results in Figure 5(b), it's reasonable that the performance of some algorithms drops in the middle, because the "Reward" used as evaluation metrics (i.e., the Y-axis in Figure 8), are the Atari environment/task rewards as mentioned above, while the unsupervised training of the skills is based on the variational or Laplacian objectives. Even without the supervision of reward signals, our algorithm can find effective skills for specific tasks with high return.