Skill Discovery for Exploration and Planning using Deep Skill Graphs Appendix

Akhil Bagaria   Jason Senthil   George Konidaris

A. Maintaining the Skill Graph

As options are executed, all three of their components $(\mathcal{I}_o, \pi_o, \beta_o)$ are updated. As a result, the relevant edges of the skill graph need to be modified.

A.1. Setting edge weights

Recall that the edge weight w_{ij} represents the cost corresponding to the edge e_{ij} in the graph. Now suppose that the planner chooses to execute option i to reach node j in the graph and that after executing option i, the agent lands in state s. If i is successful, then the cost related to edge e_{ij} is reduced, else it is increased:

$$f(s) = \begin{cases} 1, & \text{if } \mathcal{I}_o(s) = 1 \\ -1, & \text{otherwise,} \end{cases}$$
$$\mathcal{W}_{i,j} = \kappa^{f(s)} \times \mathcal{W}_{i,j}.$$

We used $\kappa = 0.95$ in all our experiments.

A.2. Deleting old edges

When the initiation set of one option (say o_2) no longer contains the effect set of another (say o_1) (either because $\mathcal{I}_{o_2}$ shrinks or the effect set $\mathcal{E}_{o_1}$ expands), the edge $e_{o_2 \to o_1}$ must be deleted from the graph; this is to ensure that all edges in the graph maintain the property that $\mathcal{E}_{o_i} \subseteq \mathcal{I}_{o_j}, \forall o_i, o_j \in \mathcal{O}$ (Konidaris et al., 2018)[1]. To implement this logic, we enumerate all the neighbors of an option o after it is executed in the environment and check if those edges still satisfy the aforementioned property.

A.3. Adding new edges

Of course, it is also possible that option o's initiation classifier $\mathcal{I}_o$ expands so that it can be connected to more nodes in the graph. To do this, we need to iterate over all nodes in the graph and check if $\mathcal{E}_{o_i} \subseteq \mathcal{I}_{o_j}, \forall o_i, o_j \in \mathcal{O}$. Given that this is a computationally expensive operation, we perform this operation once every 10 episodes.

[1]Recall that this property ensures that plans in the skill graph correspond to feasible paths in the ground MDP $\mathcal{M}$.

B. Discovering Goal Regions

Every N episodes, the DSG agent tries to expand the graph. The pseudocode for this is shown in Algorithm 1.

Algorithm 1 Discovering New Goal Regions

```
Require: Skill-graph $\mathcal{G}$
Require: Current episode number episode
Require: # permitted attempts max_tries
1: if episode % $N = 0$ then
2:   for num_tries $\in$ max_tries do
3:     Generate goal state $s_{rand} \sim \mathcal{S}$
4:     Find $v_{nn}$ according to Section 3.2.2
5:     Navigate to $v_{nn}$ according to Section 3.3.3
6:     MPC using $f_\xi$ and $\mathcal{R}(s_t, s_{rand})$ for $K$ steps
7:     if not-reject($s_t$) then
8:       Define region $\epsilon_{s_t}$ as $\mathbb{1}(s) : \{||s_t - s|| < \epsilon\}$
9:       return Goal region $\epsilon_{s_t}$
10:    end if
11:  end for
12: end if
```

B.1. Rejection Sampling

As described in Algorithm 1, ϵ_{s_t} is result of moving from v_{nn} in the direction of s_{rand}. Similar to RRT (LaValle, 1998), we ensure that Algorithm 1 results in graph expansion, by rejecting ϵ_{s_t} (`not-reject()` in line 7) if s_t is either inside any of the nodes in the current skill-graph (i.e, `reject(`s_t`)`: $\beta_o(s_t) = 1$ `or` $\mathcal{I}_o(s_t) = 1, \forall o \in \mathcal{V}$).

B.2. Multiple Attempts

If we reject ϵ_{s_t} more than `max_tries`$= 10$ times, we give up on expanding the graph, and choose to consolidate it instead (in our experiments, this only happened when the skill graph had largely covered the state-space).

C. Discussion about Optimality

Suppose at test-time, the agent starts at state s_0 and is required to reach state s_g (or more specifically, ϵ_{s_g}). DSG requires that the agent first travels to the s_g's nearest node in the graph v_{nn} and then use deep skill chaining outside the graph to reach s_g. The path to v_{nn} is obtained by using Dijkstra's algorithm (Dijkstra, 1959), which results in the shortest path on the skill-graph, but does not, in general, result in the shortest path in the ground MDP $\mathcal{M}$. Furthermore, it is possible that there is a shorter path from s_0 to s_g that does not involve going through v_{nn}, but ther agent foregoes that path to prioritize staying inside the graph over finding optimal paths in $\mathcal{M}$. The question of how the skill-graph could be used to derive hierarchically optimal (Barto & Mahadevan, 2003) solutions (or boundedly sub-optimal solutions (Ames & Konidaris, 2019)) is an interesting avenue for future work.

D. Optimistic vs Pessimistic Classifiers

For simplicity, we have discussed option initiation regions to be parameterized by a single binary classifier (Konidaris & Barto, 2009; Bagaria & Konidaris, 2020). However, Bagaria et al. (2021) showed that representing initiation sets using *two* classifiers, one optimistic $\mathcal{I}_o^\theta$ and the other pessimistic $\mathcal{I}_o^\phi$, results in more stable option learning. The optimistic classifier $\mathcal{I}_o^\theta$ determines states from which the option can be executed; the pessimistic classifier $\mathcal{I}_o^\phi$ forms the subgoal target for some other option targeting o.

To account for this dual parameterization of option initiation regions, we can re-write Equation 1 from the main paper as:

$$\mathcal{O}(s) = \{o | \mathcal{I}_o^\theta(s) = 1, o \in \mathcal{O}\}. \quad (1)$$

Using Equation 1, we can determine which options are available for execution at some state s.

When checking if two options in the graph should have an edge between them, we check if $\mathcal{E}_{o_1} \subseteq \mathcal{I}_o^\phi$. In other words, the effect set of option o_1 must be contained inside the *pessimistic* initiation classifier of option o_2.

E. Selecting Option Subgoal States

Given a goal vertex v_g, the planner computes a plan $(o_1, o_2, ..., o_L)$ that will take the agent from its current state s_t to v_g. Before executing option o_1 in the environment (by rolling out π_{o_1}), we must sample a specific goal state g_1 for π_{o_1} to target. Our choice of g_1 must ensure that reaching g_1 would permit the agent to execute the next option in the plan o_2, i.e, g_1 must be inside $\mathcal{I}_{o_2}$. To implement this condition, we first compute the subset of o_1's effect set $\mathcal{E}_{o_1}$ that is inside $\mathcal{I}_{o_2}^\phi$ and then randomly sample from it:

$$g_1 \sim \{s | \mathcal{I}_{o_2}^\phi(s) = 1, \forall s \in \mathcal{E}_{o_1}\}$$

F. Finding the Nearest Subgraph

As illustrated in Figure 3 in the main paper, DSG picks a node from the nearest unconnected subgraph as a target during the graph consolidation phase. To find the nearest unconnected subgraph, we first enumerate all unconnected subgraphs. Then, we find the closest descendant-ancestor pair (v_d, v_a) for all such subgraphs. Finally, we compare the distance between each (v_d, v_a) pair we found, and pick the one corresponding to the lowest distance between them. This procedure minimizes the region over which we rely on our distance metric (i.e, we only need the metric to be locally valid) while selecting targets that will increase the connectivity of the skill-graph.

Since enumerating *all* unconnected subgraphs can be an expensive operation, we only consider the ancestors of vertices that correspond to goal regions and not options.

G. Model-Based Policies

Dynamics Model To learn a dynamics model, we adopt the approach from Nagabandi et al. (2018). We parameterize our learned dynamics function $\hat{f}_\theta(s_t, a_t)$ as a deep neural network which take as input the current state s_t and action a_t, and predicts the change in state s_t over the time step duration of Δt. The next predicted state is thus $\hat{s}_{t+1} = s_t + \hat{f}_\theta(s_t, a_t)$. The neural network 2 dense layers with hidden sizes of 500, with LeakyRelu as the nonlinear activation function.

Data Collection and Preprocessing: To train the dynamics model, we use the transitions collected by the DSG agent. Each trajectory is sliced into inputs of (s_t, a_t) with corresponding output labels of $s_{t+1} - s_t$. We then normalize the data by subtracting the mean and dividing by the standard deviation.

Training the model: Following Nagabandi et al. (2018), we first collect 50 episodes of random transitions from the environment $\mathcal{D}_{rand}$. At this point, the dynamics model is trained for 50 epochs—meaning that we iterate over the dataset $\mathcal{D}_{rand}$ 50 times. Thereafter, the agent picks actions according to the RL algorithm and stores the resulting transitions in dataset $\mathcal{D}_{RL}$. At the end of every episode of training, we train the model for 5 epochs on the dataset $\mathcal{D} = \mathcal{D}_{rand} \cup \mathcal{D}_{RL}$.

Model Predictive Control (MPC): Following Nagabandi et al. (2018), we use the random-shooting method as our black-box optimizer to approximately solve Equation 6 in the main paper. We use $M = 14000$ randomly generated action sequences of length $K = 7$.

H. Experiment Details

H.1. Test environments

We evaluated our algorithm in four tasks that exhibit a hierarchical structure (Nachum et al., 2018; Fu et al., 2020;

Environment	# Training Episodes
Ant Reacher	1000
Ant U-Maze	1000
Ant Medium Maze	1500
Ant Large Maze	2000

Table 1. Number of training episodes per environment. Each episode comprises 1000 steps.

Brockman et al., 2016; Duan et al., 2016): (1) Ant Reacher, (2) Ant U-Maze, (3) Ant Medium Maze, (4) Ant Large Maze. In Ant Reacher, there is no maze, and the ant is required to navigate an open area spanned by $[-10, 10]^2$. The other three mazes (2)-(4) are taken from the D4RL repository[2]. In each task, the agent is reset back to its start state (a small distribution around $(0, 0)$) after 1000 steps per episode. All other environment specific configurations are unchanged from D4RL. The number of training episodes for each environment is given in Table 1.

Following D4RL (Fu et al., 2020) and other HRL algorithms (Nachum et al., 2018; Sharma et al., 2020), we used the negative distance from the goal as our reward function: $\mathcal{R}(s, g) = -||s.\text{CoM} - g||$, where $s.\text{CoM} \in \mathbb{R}^2$ refers to the x, y position of the Ant's center-of-mass and g refers to a target position. When s.CoM is sufficiently close to g, i.e, $||s.\text{CoM}-g|| < 0.6$, then $\mathcal{R}(s, g) = 0$.

H.2. Baseline Implementation Details

H.2.1. Model-Free Baseline: TD3+HER

Parameter	Value
Replay buffer size	$1e6$
Critic Learning rate	$3 \cdot 10^{-4}$
Actor Learning rate	$3 \cdot 10^{-4}$
Optimizer	Adam
Target Update Rate τ	$5 \cdot 10^{-3}$
Batch size	100
Iterations per time step	1
Discount Factor	0.99
Output Normalization	False

Table 2. TD3 + HER Hyperparameters

To compare against TD3 (Fujimoto et al., 2018), we used the TD3 author's open-source code base [3]. We used the default hyperparameters, which are listed in Table 2. The use of Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) requires that we sample a goal state g at the start of every episode; as is common, we sampled g uniformly at random from the set of positions that were not inside obstacles in the maze.

H.2.2. Hierarchical Baseline: HAC

We used the 3−layer HAC agent from the HAC author's open-source code base[4]. We used the same hyperparameters that they used in domains involving the Ant.

H.2.3. Hierarchical Baseline: DADS

We used the author's code base[5] without modification. Given that DSG's dynamics model f_ξ was trained inside the various mazes that we were testing on, we first tried to train the DADS skills on environments in which they were going to be tested. However, we found that the discovered skills lacked diversity and tended to collapse to the region around the start-state distribution—presumably because of the lack of space (in the x, y direction) for the agent to discover maximally diverse skills. As a result, we trained the skills on the Ant-Reacher domain (as Sharma et al. (2020) did in their paper) and used the resulting skills in the various mazes.

I. Hyperparameters

I.1. DSG

As shown in Table 3, DSG introduced two new hyperparameters; both their values were the same for all environments.

Parameter	Value
#−steps of model extrapolation (K)	100
Goal region discovery frequency (N)	50

Table 3. DSG specific hyperparameters

Skill chaining requires three hyperparameters, whose values are shown in Table 4; their values were also the same for all environments.

Parameter	Value
Gestation period	10
Option timeout	200
Buffer length	50

Table 4. DSC specific hyperparameters

I.2. Model-Based Baseline

The hyperparameters used to implement the model-based algorithm from Nagabandi et al. (2018) is shown in Table 5.

[2]github.com/rail-berkeley/d4rl
[3]github.com/sfujim/TD3
[4]github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-
[5]github.com/google-research/dads

Parameter	Value
Batch size	1024
Optimizer	Adam
Controller horizon H	7
Number actions sampled K	14000

Table 5. Hyperparameters for learning dynamics model f_ξ

J. Computational Details

J.1. Initiation Classifiers

Following related work (Bagaria & Konidaris, 2020; Eysenbach et al., 2019; Sharma et al., 2020; Levy et al., 2019), option initiation classifiers were defined over the x, y position of the agent. Following Bagaria et al. (2021), we used a one-class SVM (Tax & Duin, 1999) to represent the option's pessimistic classifier and a two-class SVM (Cortes & Vapnik, 1995) to represent its optimistic classifier.

J.2. Computational Resources

All experiments in this paper were performed on 2 NVIDIA 2080-Ti GPUs.

References

Ames, B. and Konidaris, G. Bounded-error lqr-trees. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 144–150. IEEE, 2019.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Bagaria, A. and Konidaris, G. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1gqipNYwH`.

Bagaria, A., Senthil, J., Slivinski, M., and Konidaris, G. Robustly learning composable options in deep reinforcement learning. In *30th International Joint Conference on Artificial Intelligence*, 2021.

Barto, A. G. and Mahadevan, S. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

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

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SJx63jRqFm`.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1582–1591, 2018.

Konidaris, G. and Barto, A. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, pp. 1015–1023, 2009.

Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

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

Levy, A., Konidaris, G., Platt, R., and Saenko, K. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryzECoAcY7`.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

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

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations (ICLR)*, 2020.

Tax, D. M. and Duin, R. P. Support vector domain description. *Pattern recognition letters*, 1999.