Skill Discovery for Exploration and Planning using Deep Skill Graphs

Akhil Bagaria [1] **Jason Senthil** [1] **George Konidaris** [1]

Abstract

We introduce a new skill-discovery algorithm that builds a discrete graph representation of large continuous MDPs, where nodes correspond to skill subgoals and the edges to skill policies. The agent constructs this graph during an unsupervised training phase where it interleaves discovering skills and planning using them to gain coverage over ever-increasing portions of the state-space. Given a novel goal at test time, the agent plans with the acquired skill graph to reach a nearby state, then switches to learning to reach the goal. We show that the resulting algorithm, *Deep Skill Graphs*, outperforms both flat and existing hierarchical reinforcement learning methods on four difficult continuous control tasks.

1. Introduction

General-purpose agents operating in challenging domains like robotics must cope with the reality of action and perception in high-dimensional spaces (Konidaris, 2019). Hierarchical Reinforcement Learning (HRL) (Barto & Mahadevan, 2003), with its emphasis on building procedural action abstractions, is a promising starting point for scaling RL to such problems.

The benefits of well-designed hierarchies have always been clear (Sutton et al., 1999; Dietterich, 2000), but the task of autonomously discovering them (known as *skill discovery*) is a challenging open problem (Precup, 2001; Mcgovern, 2002; Ravindran, 2004; Konidaris, 2011; Bacon, 2018). Furthermore, previous works have studied skill discovery in several different contexts—multi-task learning (Levy et al., 2019), exploration (Machado et al., 2017; Jinnai et al., 2019) and planning (Sharma et al., 2020)—but how these settings culminate in a single algorithm remains unclear.

We propose an algorithm that discovers skills that aid exploration, and which are subsequently useful for planning in a multi-task context. Our algorithm learns a collection of skills that form a graph, where the nodes are subgoals and the edges are the policies that navigate between them. To construct this graph, we first discover subgoal regions that optimize for coverage of the state-space, using techniques inspired by the robot motion-planning literature (LaValle, 1998) that causes the graph to grow towards large unexplored regions. These regions are subsequently connected using Deep Skill Chaining (Bagaria & Konidaris, 2020; Bagaria et al., 2021), resulting in a collection of skills that enable the agent to reliably move between subgoals. The chainability of the discovered skills, i.e, the property that successful execution of one permits the execution of an-

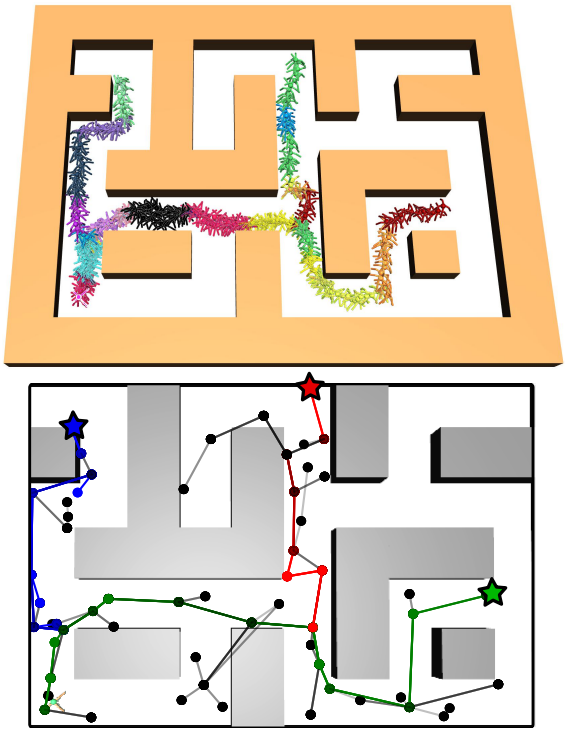

Figure 1. Deep Skill Graphs constructs a discrete graph-based abstraction *(bottom)* of a high-dimensional continuous control problem *(top)* during an unsupervised training phase. When given test-time goals (stars), the agent finds the high-level solution trajectories on the graph, which are shown in red, green and blue *(bottom)*. Then the high-level solutions are translated to joint torques using discovered skills, which are shown as different colors *(top)*.

[1]Department of Computer Science, Brown University, Providence, RI, USA. Correspondence to: Akhil Bagaria <akhil_bagaria@brown.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

other, allows the agent to build a graph that is suitable for deterministic planning (Konidaris et al., 2018). Finally, our algorithm operates in the multi-task setting (Taylor & Stone, 2009) because it discovers a multi-query graph (Kavraki et al., 1996) that it can use to plan between arbitrary start-goal pairs at test time. At test time, if a goal state lies inside the skill-graph, the agent can reach it without any further learning; if not, the agent plans to the nearest subgoal and then switches to learning a new skill chain that extends the graph to reach its new goal.

We test our algorithm on four maze-navigation tasks in MuJoCo (Todorov et al., 2012), where it outperforms flat model-free RL (Andrychowicz et al., 2017), model-based RL (Nagabandi et al., 2018) and state-of-the-art skill-discovery algorithms (Levy et al., 2019; Sharma et al., 2020).

2. Background and Related Work

We consider episodic, goal-oriented MDPs $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma, \mathcal{G}, \rho)$ (Sutton & Barto, 2018), where $\mathcal{G}$ is a distribution over goals. Like most goal-conditioned RL algorithms (Schaul et al., 2015; Andrychowicz et al., 2017), we assume access to a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$. ρ denotes the start-state distribution which is a small, fixed region at training time, but can change arbitrarily at test-time. This ensures that the agent cannot "teleport" to states that might otherwise be difficult to reach (Ecoffet et al., 2019) and thus must confront the exploration problem in its entirety (Kakade & Langford, 2002).

2.1. The Options Framework

We model abstract actions (or skills) as *options* (Sutton et al., 1999). Each option o in the agent's option repertoire $\mathcal{O}$ is defined as a three element tuple $(\mathcal{I}_o, \pi_o, \beta_o)$. The initiation set $\mathcal{I}_o : \mathcal{S} \to \{0, 1\}$ describes the set of states from which option o can be executed. The termination set $\beta_o : \mathcal{S} \to \{0, 1\}$ describes the set of states in which option execution terminates and is deemed successful. Finally, the option policy $\pi_o : \mathcal{S} \to \mathcal{A}$ is a controller that drives the agent from $\mathcal{I}_o$ to β_o. Augmenting the set of available actions with options results in a Semi-Markov Decision Process (SMDP) where the next state depends on the current state, action, and *time* (Bradtke & Duff, 1995).

In addition to the three elements described above, Konidaris et al. (2018) define the *effect set* $\mathcal{E}_o$ of option o as the set of states in which the option policy π_o actually triggers its termination condition β_o. Note that $\mathcal{E}_o \subseteq \beta_o$ since there may be states in β_o not reachable (or not reached in practice) from states in $\mathcal{I}_o$.

2.2. Rapidly Exploring Random Trees (RRT)

RRT is a robot motion planning algorithm used to find collision-free paths in a robot's configuration space (LaValle, 1998). It does so by incrementally constructing a space-covering tree rooted at a given start state using a random sampling procedure. Concretely, RRT proceeds in three steps: (a) randomly sample a state s_{rand} from the state-space, (b) identify its nearest neighbor node v_{nn} in the tree, and finally (c) use the given model of the robot's dynamics to move K steps from v_{nn} in the direction of s_{rand}, and add the resulting state as a new node in the tree. In Section 3, we generalize each of these steps to the RL setting. Despite its simplicity, RRTs have been highly successful in solving motion planning problems involving high-dimensional dynamical systems, largely due to the *Voronoi bias* of the tree construction procedure (Lindemann & LaValle, 2004), which refers to property that the direction in which the tree expands at each iteration is proportional to the volume of the Voronoi region of each edge node. So, the volume of the Voronoi region of a state can be seen as a measure of "how unexplored" the region around that state is; randomly selecting points to expand the tree therefore causes it to naturally grow in the direction of the unexplored frontiers of the state-space (LaValle & Kuffner Jr, 2001).

Although this work is inspired by RRT, we do not learn a single-query tree (LaValle, 2006), but instead learn a multiple-query graph, much like the related Probabilistic Road Map (PRM) algorithm (Kavraki et al., 1996), meaning that the resulting skill graph can be used to find solution paths between multiple start-goal pairs.

2.3. Skill Chaining

We use deep skill-chaining (DSC) (Bagaria & Konidaris, 2020; Bagaria et al., 2021) to construct chains of options that navigate between the regions discovered by our RRT-like graph expansion procedure.

In skill-chaining (Konidaris & Barto, 2009), each discovered option o learns its initiation region $\mathcal{I}_o$ using a binary classifier that describes the region from which the π_o reaches β_o with high probability. Simultaneously, π_o is learned to reach the subgoal region β_o. The algorithm is recursive: it first learns an option that initiates near the goal and reliably takes the agent to the goal; then it learns another option whose termination region is the initiation region of the first option; then it repeats the procedure targeting the new option. Options are chained together in this fashion until the agent discovers an option whose initiation region contains the start state. Deep skill chaining (DSC) (Bagaria & Konidaris, 2020) extended skill-chaining with deep RL, outperforming other skill-discovery algorithms (Levy et al., 2019; Bacon et al., 2017) in the single-task setting. The Robust DSC algorithm (Bagaria et al., 2021) improved the

vanilla algorithm, so that is the version of DSC that we use.

Given a single start state and a goal, skill chaining connects them using a chain of options. When there are multiple start states, skill chaining organizes options in the form of a tree rooted at the goal state. This topology is insufficient in the multi-task setting where the agent could start anywhere and be asked to go anywhere. Consequently, we propose to organize skills in the form of a graph to handle the multi-task setting. Furthermore, while skill chaining required that a goal region be specified during training, DSG proposes its own goals during training and learns options to reliably reach them.

2.4. Other Skill-Discovery Algorithms

Recent skill-discovery algorithms (see Abel (2020), chapter 2.3 for a survey) can be broadly categorized into (a) option-critic, (b) feudal, and (c) empowerment maximization methods.

Option-Critic describes an end-to-end framework for learning options (Bacon et al., 2017; Harutyunyan et al., 2019; Tiwari & Thomas, 2019; Khetarpal & Precup, 2019). Learned options are tied to the reward function during training, so they cannot be easily repurposed for the multi-task setting.

Feudal methods construct a hierarchy in which a higher-level manager outputs goals for the lower-level workers to achieve (Dayan & Hinton, 1993; Vezhnevets et al., 2017; Li et al., 2019). HIRO (Nachum et al., 2018) demonstrated impressive performance on continuous control problems with dense-rewards, but also has not been used in the multi-task setting. Hierarchical Actor Critic (HAC) (Levy et al., 2019) uses hindsight experience replay (Andrychowicz et al., 2017) to stably learn goal-conditioned policies (Schaul et al., 2015) and so can generalize to unseen goals at test-time. Furthermore, it outperformed other feudal algorithms on continuous control problems. As a result, we compare against HAC as a representative feudal method.

Empowerment driven methods discover maximally diverse skills (Gregor et al., 2016; Eysenbach et al., 2019a; Baumli et al., 2021; Campos Camúñez et al., 2020; Florensa et al., 2016). Recently, the DADS algorithm (Sharma et al., 2020) introduced a model-based objective to this literature and showed state-of-the-art performance on goal-reaching tasks in MuJoCo. These methods are instances of **unsupervised skill-discovery** algorithms since they learn task-agnostic skills in a pre-training phase; the learned skills are composed to reach unseen goals at test-time in a zero-shot fashion. Given the similarity to our problem setting, we compare against DADS as the best performing empowerment-driven skill-discovery method.

2.5. Graph-based methods

The intuition of constructing a graph in RL has received some attention recently. SoRB (Eysenbach et al., 2019b) uses graph search on the replay buffer (Lin, 1993) to find efficient paths between observations made by a model-free policy. PRM-RL (Faust et al., 2018) replaces the local planner used in motion planning with an RL policy. In LEAP (Nasiriany et al., 2019), a planner generates sub-goals for a low level model-free policy to meet. SoRB, PRM-RL and LEAP all assume access either to a well trained value function or an RL policy that can be meaningfully queried in arbitrary parts of the state-space. By contrast, our algorithm learns from scratch and incrementally extends the graph towards unexplored regions (Jinnai et al., 2020). SPTM (Savinov et al., 2018; Liu et al., 2020) is a memory augmented RL agent that plans over raw landmark observations in navigation problems. Our method bypasses the difficulty of planning in high-dimensional spaces (Laskin et al., 2020) by instead planning over abstract states.

3. Deep Skill Graphs

To solve goal-reaching tasks, options must be sequentially composable: the agent must be able to successfully execute one option after another until it reaches the goal. Sequential composition is possible if and only if successful execution of one option leads the agent to a state that permits the execution of another.

When the agent need only ever navigate from a single given start state to a single given goal state, it must therefore learn options that form a *chain* connecting those two states. In the more general case where the agent is given a single goal that must be reachable from *any* start state, it must instead build a *tree* of options, enabling it to follow a chain from any state to the goal. In the most general case, the agent may be asked to reach *any* goal state from *any* start state; since that requires a chain of options connecting *any* possible start-goal pair, the agent's options must necessarily be organized into a *graph*. We therefore propose to explicitly construct options so that they form such a graph.

3.1. Definitions

Before describing how the skill graph is constructed and used, we first define its key components:

Discovered goal regions. The DSG agent proposes its own goals during an unsupervised training phase. For each proposed goal state g, we define a goal region $\epsilon_g : \mathcal{S} \to \{0, 1\}$ as an ϵ-ball centered around g. $\mathcal{B} = \{\epsilon_{g_1}, .., \epsilon_{g_n}\}$ is the set of all such goal regions. To unify notation with options, we additionally define $\mathcal{I}_{\epsilon_g} = \epsilon_g$, and $\mathcal{E}_{\epsilon_g}$ as the set of states in which goal region ϵ_g was actually reached.

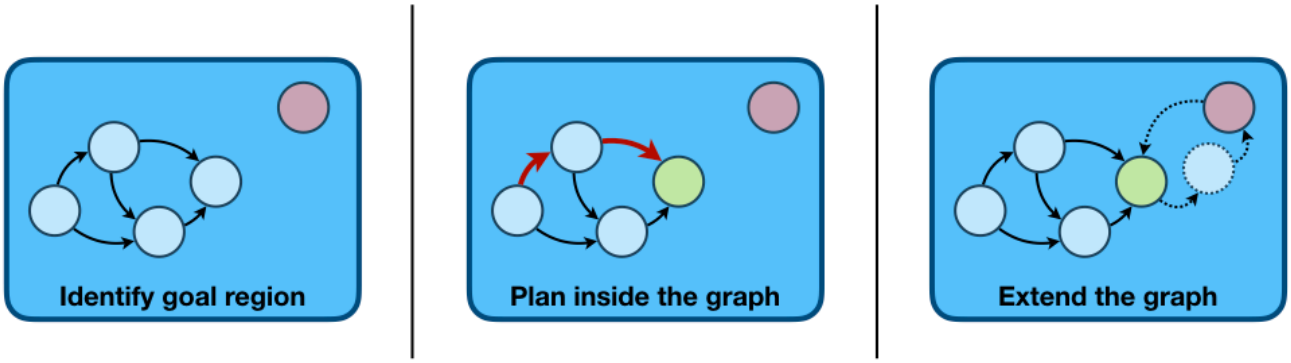

Figure 2. Given a goal (red) outside the graph, the agent uses the graph to plan to reach the node closest to that goal (green). It then expands the graph by constructing a skill chain connecting the goal to the graph.

Skill graph. A skill graph is a weighted, directed graph $\mathcal{G} = (\mathcal{V}, E, \mathcal{W})$. Each vertex $i \in \mathcal{V}$ either corresponds to an option or a goal region. Edge $e_{i \to j}$ exists between vertices i and j if and only if the effect set of i is inside the initiation set of j, i.e, $\mathcal{E}_i \subseteq \mathcal{I}_j$ (Konidaris et al., 2018). The edge weight $w_{i,j} \in \mathcal{W}$ is the reward accumulated by going from vertex i to j.

Mapping states to vertices. Given agent state s, $\mathcal{V}(s)$ denotes the set of vertices in the skill graph that s maps to. First, we find the set of options in the graph that the agent can execute at s:

$$\mathcal{O}(s) = \{o \mid \mathcal{I}_o(s) = 1, \forall o \in \mathcal{O}\}. \quad (1)$$

Then, we enumerate the set of goal regions that s satisfies:

$$\mathcal{B}(s) = \{\epsilon_g \mid \epsilon_g(s) = 1, \forall \epsilon_g \in \mathcal{B}\}. \quad (2)$$

Finally, the set of vertices that s maps to is given by:

$$\mathcal{V}(s) = \mathcal{O}(s) \cup \mathcal{B}(s). \quad (3)$$

The **descendants** $\mathcal{D}(v)$ of vertex v is the set of vertices reachable from v. The **ancestors** $\mathcal{A}(v)$ of v are the vertices from which v is reachable via $\mathcal{G}$:

$$\mathcal{D}(v) = \{v' \mid \mathcal{G}.\texttt{has-path}(v, v'),\ \forall v' \in \mathcal{V}\}, \quad (4)$$
$$\mathcal{A}(v) = \{v' \mid \mathcal{G}.\texttt{has-path}(v', v),\ \forall v' \in \mathcal{V}\}. \quad (5)$$

3.2. Overview of the Graph Construction Algorithm

During training, the agent alternates between (a) expanding the graph into unexplored regions and (b) increasing the connectivity of the existing graph. The high-level algorithm for both cases is similar; its three steps are illustrated in Figure 2.

The details of *how* each step is performed differs based on whether the agent is trying to expand the graph or consolidate it. During graph expansion, the agent generates new goals outside the graph (Section 3.3.1), navigates to its nearest neighbor in the graph (Section 3.3.2) and then moves K steps in the direction of the goal (Section 3.3.4).

During graph consolidation, the agent instead picks a node from the closest unconnected sub-graph as the goal (Section 3.4.1), navigates to its nearest neighbor in the current sub-graph (Section 3.4.2) and then learns an option chain that connects the sub-graphs (Section 3.4.3).

3.3. Graph Expansion

Every N episodes, the agent expands the skill-graph to new regions of the state-space.

3.3.1. Generate New Goals

Regions covered by the graph represent places where the agent has already achieved mastery (Kaelbling, 1993; Veeriah et al., 2018) because the agent can plan using learned skills to reliably reach arbitrary states inside the graph. To increase the portion of the world over which it has achieved mastery, the skill-graph should expand into largely unexplored regions of the state-space.

In other words, given the current graph $\mathcal{G}$, the agent should generate a goal state g such that $\mathcal{G}$ grows towards the largest region it has not yet covered. As Lindemann & LaValle (2004) (somewhat surprisingly) showed in the context of motion planning, randomly sampling g from the state-space does exactly that. So, we set $g = s_{rand} \sim \mathcal{S}$.[1]

3.3.2. Identify the Nearest Neighbor

Given a randomly sampled goal state s_{rand} that lies outside the graph, the agent identifies its nearest neighbor in the graph v_{nn}. To find v_{nn}, it first uses Equation 4 to instantiate the set of vertices reachable from the current state: $\mathcal{D}(s_t) = \{\mathcal{D}(v) | v \in \mathcal{V}(s_t)\}$. Then, it finds the vertex in $\mathcal{D}(s_t)$ closest

[1]This assumes that we can sample states from the environment. In MuJoCo, where the observation space is factored and bounded, this is straightforward. Sampling from pixel-based observation spaces is more involved, and is the subject of active research (Nair et al., 2018; Pong et al., 2019).

to s_{rand} and sets v_{nn} to it:

$$v_{nn} = \arg\min_{v \in \mathcal{D}(s_t)} ||s_{rand} - v||^2$$
$$= \arg\min_{v \in \mathcal{D}(s_t)} \left[\max_{s \in \mathcal{E}_v} ||s_{rand} - s||^2 \right].$$

Note that, in the equation above: (a) we define the distance between vertex v and state s_{rand} as the maximum distance between any of the states in the effect set of v and s_{rand} and (b) we use the Euclidean metric and leave discovering more sophisticated metrics (Mahadevan & Maggioni, 2007; Hartikainen et al., 2020; Pitis et al., 2020) for future work.

3.3.3. Navigate to the Nearest Neighbor

Given that the agent is at state s_t and wants to reach vertex v_{nn}, it must decide which option to execute. There are two possible cases: (a) if there is a path from $\mathcal{V}(s_t)$ to v_{nn} in $\mathcal{G}$, the agent uses Equation 1 to find $\mathcal{O}(s_t)$, the set of options available in s_t. Then, it uses graph search (Dijkstra, 1959) to find a plan from each $o \in \mathcal{O}(s_t)$ to v_{nn}, selects the least-cost plan, and executes its first option. Having landed in a new state $s_{t+\tau}$, it re-plans to go from $s_{t+\tau}$ to v_{nn}. This process continues until it reaches v_{nn}. (b) If there is no such path on the graph, the agent uses DSC's policy over options $\pi_{\mathcal{O}}$ to select options (or primitive actions). Since DSC trains this policy using SMDP Q-learning, it is not as effective as using the planner for option selection.

3.3.4. Extend the Graph

Having reached v_{nn}, the agent attempts to extend the graph towards s_{rand}. However, s_{rand} may be unreachable from v_{nn} either because it is inside an obstacle or because of the limits of the agent's current policy. As a result, s_{rand} itself cannot be used as a goal state. In RRT, this issue is resolved by using the given dynamics model to move K steps in the *direction* of s_{rand}. Since a dynamics model is not typically known a priori in the RL setting, we learn one from the data collected during training and use the learned model in conjunction with Model Predictive Control (MPC) (Garcia et al., 1989) to move in the direction of s_{rand}. The state that the model-based controller ends up in, s_{MPC}, becomes a new goal state and $\epsilon_{s_{MPC}}$ is added to the list of goal regions $\mathcal{B}$. An example of this process is shown in Figure 4c.

3.4. Graph Consolidation

For training episodes in which the agent is not expanding the graph, it improves the graph's connectivity by selecting an existing node and attempting to reach it.

3.4.1. Goal Identification

To inform the decision about which node to target, the agent identifies parts of the graph it currently has difficulty reaching. To do this, it first enumerates the vertices to which there is *no path* from its current state s_t. To pick a specific one to target, it uses a simple greedy heuristic: target the nearest unconnected sub-graph to greedily maximize the connectivity of the overall graph (Kuffner & LaValle, 2000). This process is illustrated in Figure 3.

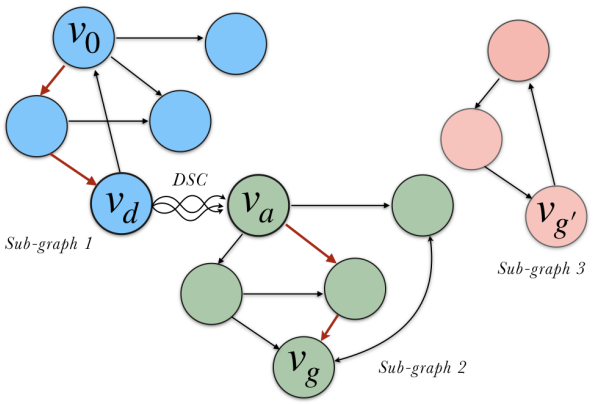

Figure 3. When an agent aims to consolidate the graph at training time and is in graph vertex v_0, it targets sub-graph 2 because it is closer than sub-graph 3. Then it identifies the closest descendant-ancestor pair (v_d, v_a). It uses its planner to traverse $v_0 \rightarrow v_d$ and $v_a \rightarrow v_g$. Since it does not have skills between v_d and v_a, it uses DSC to learn a chain of skills connecting sub-graphs 1 and 2.

3.4.2. Identify the Nearest Neighbor

Having selected a goal vertex v_g, DSG identifies its current sub-graph (the set of descendants $\mathcal{D}(s_t)$), and the sub-graph from which it can reach the goal vertex v_g (the set of ancestors $\mathcal{A}(v_g)$). Finally, it identifies the pair of nodes from $\mathcal{D}(s_t)$ and $\mathcal{A}(v_g)$ closest to each other:

$$v_d, v_a = \arg\min_{\substack{v_d \in \mathcal{D}(s_t), \\ v_a \in \mathcal{A}(v_g)}} ||v_d - v_a||^2$$
$$= \arg\min_{v_d, v_a} \left[\max_{\substack{s_d \in \mathcal{E}_{v_d}, \\ s_a \in \mathcal{E}_{v_a}}} ||s_d - s_a||^2 \right].$$

3.4.3. Extend the Graph

The agent uses its planner to reach v_d (using the procedure described in Section 3.3.3), DSC to reach v_a from v_d, and the planner again to finally reach v_g. This procedure minimizes the number of new skills required to connect the two sub-graphs. Once these skills have been learned, there is a path in $\mathcal{G}$ from s_0 to v_g, and the agent can henceforth plan to target v_g.

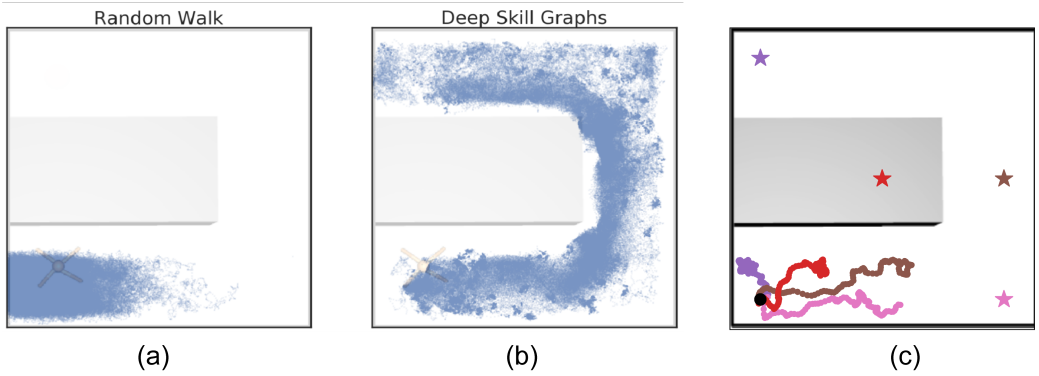

Figure 4. (a) States visited under a random walk and (b) using DSG. (c) model-based extrapolation for discovering goals during unsupervised training: the stars represent randomly sampled states; trajectories of the same color target the respective goals using MPC. All data is collected in Ant U-Maze, where the agent starts at the bottom-left of the maze.

Adding Edges to the Graph. For deterministic plans in the skill-graph to correspond to feasible solutions in the ground MDP, Konidaris et al. (2018) showed that any two options o_1, o_2 can have an edge $e_{1,2}$ between them if and only if there is a guarantee that successful execution of o_1 will allow the agent to execute o_2, i.e, $e_{1,2}$ exists iff $\mathcal{E}_{o_1} \subseteq \mathcal{I}_{o_2}$. To implement this rule, we store all the states in which o_1 successfully terminated and check if all of them lie inside $\mathcal{I}_{o_2}$. Similar logic can be applied to creating edges between options and goal regions in the graph.

3.5. Querying the Skill Graph at Test Time

Given a task expressed as a start state s_0 and a goal region ϵ_g at test time, the agent can use the graph as follows:

- if ϵ_g completely contains the effect set of an option, reaching that effect set implies reaching ϵ_g, so the agent plans to reach g without any additional learning.
- if the goal outside the graph, DSG extends the graph just as it did during training: it plans to the nearest node and then uses DSC to build a chain of options from the graph to reach g.
- If the start state s_0 itself lies outside the graph, DSG additionally creates a skill chain from s_0 to its nearest node in the graph.

4. Experiments

We tested DSG on the continuous control tasks shown in Figure 6. These tasks are adapted from the "Datasets for RL" benchmark (Fu et al., 2020)[2] and are challenging for non-hierarchical methods, which make little-to-no learning progress (Duan et al., 2016). Since we already assume a Euclidean distance metric in Section 3.3.1, we use the dense reward version of these tasks, i.e, $\mathcal{R}(s, g) = -||s - g||$. Video and code can be found on our website.

[2]We use the mazes from this suite, not the demonstration data.

4.1. Implementation Details

We follow Nagabandi et al. (2018) to learn a model-based controller to use as option policies $\pi_o, \forall o \in \mathcal{O}$. Specifically, transitions seen by all options are used to train a neural dynamics model f_ξ, which is then used to select actions by approximately solving the following H-horizon optimization problem:

$$\pi_o(s|g) = \underset{a \in \mathcal{U}^{|\mathcal{A}|}(-1,1)}{\arg\max} \sum_{t=1}^{H} \gamma^{t-1} \mathcal{R}(s_t, g), \qquad (6)$$

such that $s_{t+1} = f_\xi(s_t, a)$. The solution to Equation 6 is a sequence of actions; rather than executing this sequence open-loop, we use MPC to execute the first action and re-plan at the next time-step (Nagabandi et al., 2018). Before executing option policy π_o, we randomly sample from the option's subgoal region, i.e, $g \sim \beta_o$. For more details on how this sampling is done, please refer to Appendix E.

4.2. Qualitative Evaluation

Exploration Property of DSG. We compare the states visited by the DSG agent and those visited under a random walk. Figure 4 shows that in the Ant U-Maze environment, skills can lead to temporally extended exploration as opposed to the dithering behavior of random walks, even in the absence of an extrinsic reward function.

Incremental Graph Expansion. Figure 5 provides some intuition on why DSG can effectively explore large regions of the state-space—the skill-graph begins at the start state and incrementally expands into unexplored regions of the state-space. By planning and executing learned skills inside the graph, the agent can reliably get back to the frontier of its knowledge (as represented by the outermost vertices in

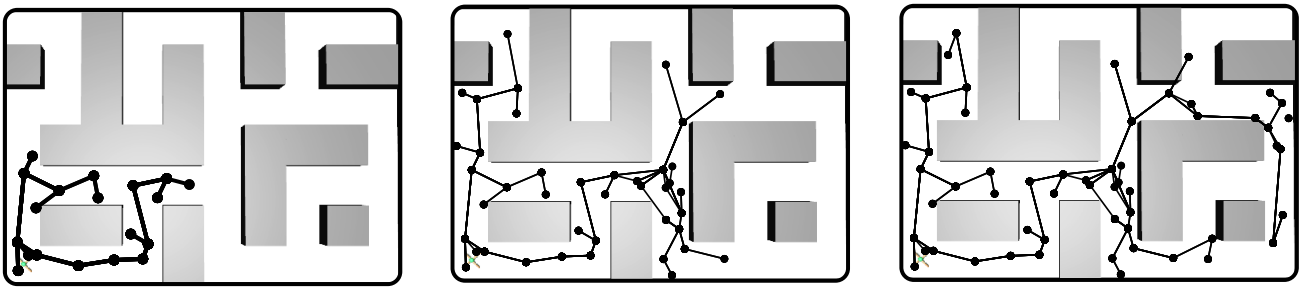

Figure 5. Incremental expansion of the skill graph towards unexplored regions in the large ant-maze. Although the skill-graph is directed, we visualize it as un-directed for simplicity. To visualize option nodes, we plot the median state of its effect set.

the graph). Exploration from the frontier in turn allows it to amass the experiences it needs to further expand the graph. By interleaving planning and chaining in this way, the DSG agent incrementally achieves coverage of ever increasing portions of the state-space

4.3. Quantitative Evaluation

Experimental Setup. The agent discovers skills during an unsupervised training phase (which lasts for 1000 episodes in Reacher and U-Maze, 1500 episodes in Medium-Maze and 2000 episodes in the Large-Maze). During this period, its start-state is fixed to be at the bottom-left of every maze. At test time, we generate 20 random start-goal pairs from the maze and record the average success rate (Andrychowicz et al., 2017) of the agent over 50 trials per start-goal pair (Sharma et al., 2020). All competing methods are tested on the same set of states.

Comparisons. We compare DSG to the following

1. Flat model-free baselines: **HER** (Andrychowicz et al., 2017) learns policies that generalize across goals by taking the goal state as an additional input during training. Because of their widespread use, we use them as a representative non-hierarchical baseline. Since HER does not have an explicit mechanism for exploration, we additionally favorably expand its start-state distribution during training to include all valid locations in the maze—we call this baseline **HER***. HER policies are learned using TD3 (Fujimoto et al., 2018).

2. Flat model-based baselines: Since we use the model-based **(MB)** controller from Nagabandi et al. (2018) as option policies, we include it as our model-based baseline. Similar to HER*, we include a version **MB*** which also gets favorable access to a wide start-state distribution.

3. Feudal HRL baseline: **HAC** (Levy et al., 2019) is a hierarchical extension of HER, and has achieved state-of-the-art performance on similar tasks. HER, HER*, MB, MB* and HAC sample goals uniformly from the free-space in the maze during training.

4. Empowerment HRL baseline: To compare against **DADS**, we trained the skill-primitives in the Ant-Reacher environment and used them at test-time in the different mazes. This strategy outperformed that of discovering skills in each specific maze environment, since there was no space for the DADS agent to learn diverse skills. For more details, refer to Appendix H.

5. Results

Figure 6 shows that DSG comfortably outperforms all baselines. Our results confirm that while in principle flat goal-conditioned RL can solve the kind of goal-reaching tasks we consider, they require orders of magnitude more data and struggle to generalize to distant goals.

Both HAC and DADS were able to achieve comparable (and good) results in Ant-Reacher (which was the task that they were evaluated on in their respective papers). However, unlike DSG, they struggled to maintain that level of performance as the problem got harder with larger, more complicated mazes. HAC implicitly assumes that the space of subgoals is smooth—which is violated in our mazes because several subgoal proposals made by the manager correspond to states that are inside obstacles, and hence unreachable by the workers. DADS discovers maximally diverse skills, but solving mazes requires sequencing a very specific sequence of actions (e.g., "go right until you can go left")—a property not captured by their diversity objective. Compared to DSG, an advantage of DADS is that their skills are *portable* (Konidaris & Barto, 2007), meaning that the same skill can be transferred to different regions of the state-space.

Upon examining the goal states that DSG *failed* to reach at test-time, we found that these failures could be attributed to our reliance on dense rewards. Specifically, if the goal state lay on the other side of an obstacle from its nearest node in the graph, DSG would be unable to extend the graph towards this goal. It is well known that dense reward functions can

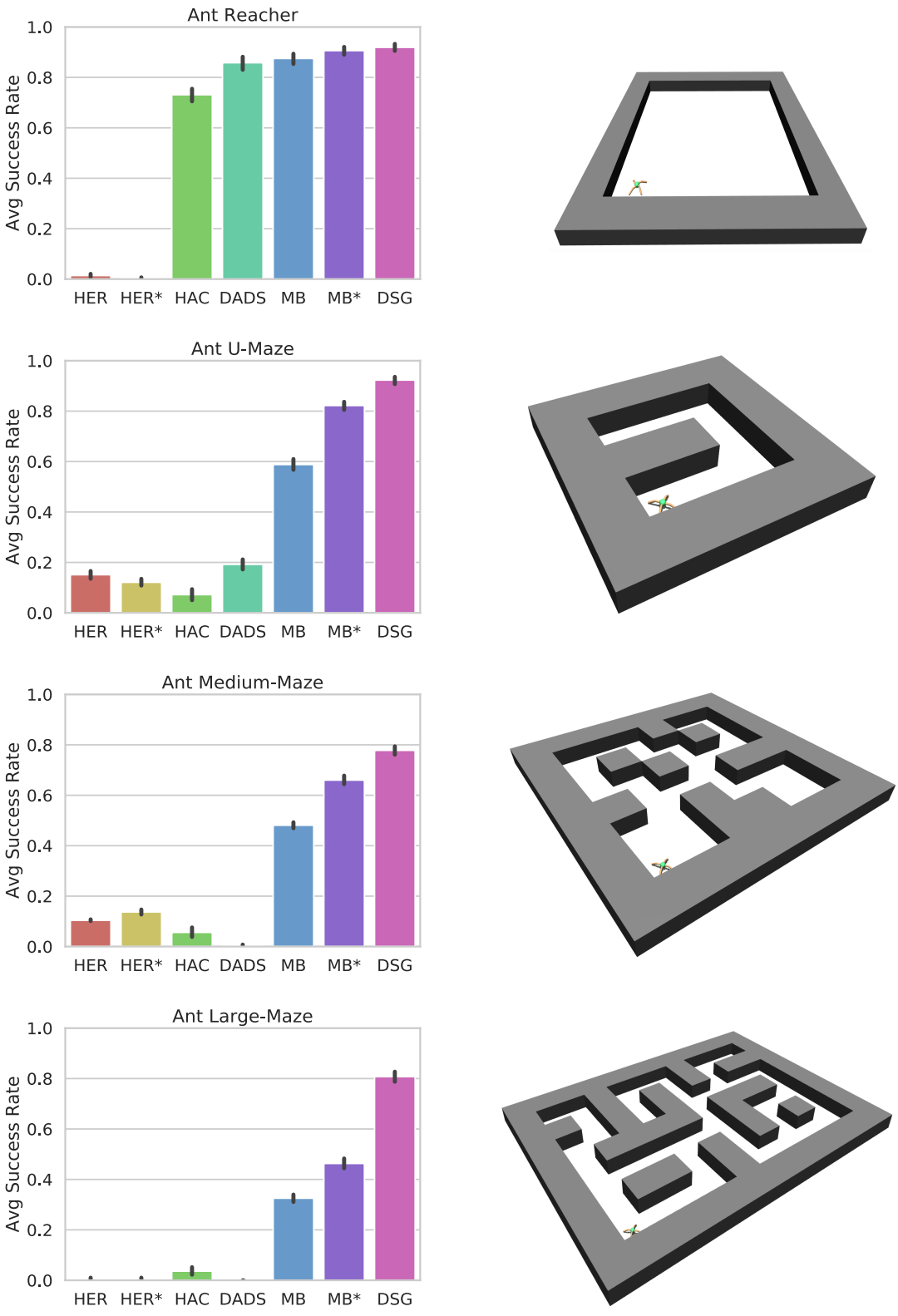

Figure 6. Test-time success rates averaged over 20 randomly sampled start-goal pairs and 50 trials; error bars denote standard deviation. All algorithms were trained for the same number of episodes prior to being tested—their training performance is not shown here.

lead to sub-optimal policies (Randløv & Alstrøm, 1998; Mataric, 1994), and extending our algorithm to the sparse-reward setting should mitigate these failures.

6. Conclusion

We introduced an algorithm that builds a graph abstraction of large continuous MDPs. Like all RL agents, DSG starts by exploring the environment since it does not know enough about it to plan. Then, it proposes its own goals and learns skills to reliably reach those goals. Together, proposed goals and skills enable high-level planning.

We showed that skill graphs grow outward from the start-state towards large unexplored regions, reflecting mastery over ever increasing portions of the state-space. Finally, we tested our algorithm on a series of maze navigation tasks and showed that DSG can be used to reach novel goals at test-time in a small number of trials. We compared the zero-shot generalization capability of our algorithm to that of popular flat and state-of-the-art hierarchical alternatives and showed that DSG significantly outperforms them.

Acknowledgements

We thank Stefanie Tellex, Michael Littman and other members of the Brown BigAI group for their suggestions. This research was supported by NSF grants 1955361, 1717569 and the DARPA Lifelong Learning Machines program under grant FA8750-18-2-0117.

References

Abel, D. *A Theory of Abstraction in Reinforcement Learning*. PhD thesis, Brown University, 2020.

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.

Bacon, P.-L. *Temporal Representation Learning*. PhD thesis, McGill University Libraries, 2018.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 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.

Baumli, K., Warde-Farley, D., Hansen, S., and Mnih, V. Relative variational intrinsic control. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pp. 6732–6740, 2021.

Bradtke, S. J. and Duff, M. O. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in neural information processing systems*, pp. 393–400, 1995.

Campos Camúñez, V., Trott, A., Xiong, C., Socher, R., Giró Nieto, X., and Torres Viñals, J. Explore, discover and learn: unsupervised discovery of state-covering skills. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, volume 119, pp. 1317–1327, 2020.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

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

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.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019. URL http://arxiv.org/abs/1901.10995.

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*, 2019a. URL https://openreview.net/forum?id=SJx63jRqFm.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems 32*, pp. 15246–15257. 2019b.

Faust, A., Oslund, K., Ramirez, O., Francis, A., Tapia, L., Fiser, M., and Davidson, J. PRM-RL: Long-range robotic

navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5113–5120. IEEE, 2018.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2016.

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 *Proceedings of the 35th International Conference on Machine Learning*, pp. 1582–1591, 2018.

Garcia, C. E., Prett, D. M., and Morari, M. Model predictive control: theory and practice—a survey. *Automatica*, 25 (3):335–348, 1989.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *ArXiv*, abs/1611.07507, 2016.

Hartikainen, K., Geng, X., Haarnoja, T., and Levine, S. Dynamical distance learning for semi-supervised and unsupervised skill discovery. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1lmhaVtvr.

Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., and Precup, D. The termination critic. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2231–2240, 2019.

Jinnai, Y., Park, J. W., Abel, D., and Konidaris, G. Discovering options for exploration by minimizing cover time. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2019. URL http://proceedings.mlr.press/v97/jinnai19b.html.

Jinnai, Y., Park, J. W., Machado, M. C., and Konidaris, G. Exploration in reinforcement learning with deep covering options. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SkeIyaVtwB.

Kaelbling, L. P. Learning to achieve goals. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1094–1099. Citeseer, 1993.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, volume 2, pp. 267–274, 2002.

Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

Khetarpal, K. and Precup, D. Learning options with interest functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 9955–9956, 2019.

Konidaris, G. *Autonomous Robot Skill Acquisition*. PhD thesis, University of Massachusetts at Amherst, 2011.

Konidaris, G. On the necessity of abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7, 2019.

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.

Konidaris, G. D. and Barto, A. G. Building portable options: Skill transfer in reinforcement learning. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pp. 895–900, 2007.

Kuffner, J. J. and LaValle, S. M. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 2, pp. 995–1001. IEEE, 2000.

Laskin, M., Emmons, S., Jain, A., Kurutach, T., Abbeel, P., and Pathak, D. Sparse graphical memory for robust planning. *arXiv preprint arXiv:2003.06417*, 2020.

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

LaValle, S. M. *Planning algorithms*. Cambridge university press, 2006.

LaValle, S. M. and Kuffner Jr, J. J. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

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.

Li, S., Wang, R., Tang, M., and Zhang, C. Hierarchical reinforcement learning with advantage-based auxiliary rewards. In *Advances in Neural Information Processing Systems*, pp. 1409–1419. 2019.

Lin, L.-J. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

Lindemann, S. R. and LaValle, S. M. Incrementally reducing dispersion by increasing voronoi bias in RRTs. In *IEEE International Conference on Robotics and Automation*, volume 4, pp. 3251–3257, 2004.

Liu, K., Kurutach, T., Tung, C., Abbeel, P., and Tamar, A. Hallucinative topological memory for zero-shot visual planning. *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Machado, M. C., Bellemare, M. G., and Bowling, M. A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pp. 2295–2304, 2017.

Mahadevan, S. and Maggioni, M. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231, 2007.

Mataric, M. J. Reward functions for accelerated learning. In *Machine learning proceedings 1994*, pp. 181–189. Elsevier, 1994.

Mcgovern, E. A. *Autonomous discovery of temporal abstractions from interaction with an environment*. PhD thesis, University of Massachusetts at Amherst, 2002.

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.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*, 31: 9191–9200, 2018.

Nasiriany, S., Pong, V., Lin, S., and Levine, S. Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Pitis, S., Chan, H., Jamali, K., and Ba, J. An inductive bias for distances: Neural nets that respect the triangle inequality. In *8th International Conference on Learning Representations, ICLR*, 2020.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. Skew-fit: State-covering self-supervised reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2019.

Precup, D. *Temporal abstraction in reinforcement learning.* PhD thesis, 2001.

Randløv, J. and Alstrøm, P. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, volume 98, pp. 463–471, 1998.

Ravindran, B. *An algebraic approach to abstraction in reinforcement learning*. PhD thesis, University of Massachusetts at Amherst, 2004.

Savinov, N., Dosovitskiy, A., and Koltun, V. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*, 2018.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

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.

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

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

Tiwari, S. and Thomas, P. S. Natural option critic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5175–5182, 2019.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Veeriah, V., Oh, J., and Singh, S. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3540–3549, 2017.

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_ξ and $\mathcal{R}(s_t, s_{rand})$ for K steps
7: **if** `not-reject(`s_t`)` **then**
8: Define region ϵ_{s_t} as $\mathbb{1}(s) : \{||s_t - s|| < \epsilon\}$
9: **return** Goal region ϵ_{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:

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

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.