

# Skill Discovery for Exploration and Planning using Deep Skill Graphs

By Jet Situ

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the bottom half of the slide.

# Deep Skill Graphs

- Traditionally, skill discovery and path planning within the continuous movement space were treated separately.
  - *Is there a combined, efficient method of solving for new skills and creating feasible plans in the same training session?*
- 
- **Deep Skill Graphs**, building on the prior work of **Deep Skill Chaining**, provides a mechanism for skill learning in conjunction with planning optimization.
  - With RL, we can build hierarchical task solving paradigms that efficiently unify both planning and learning.

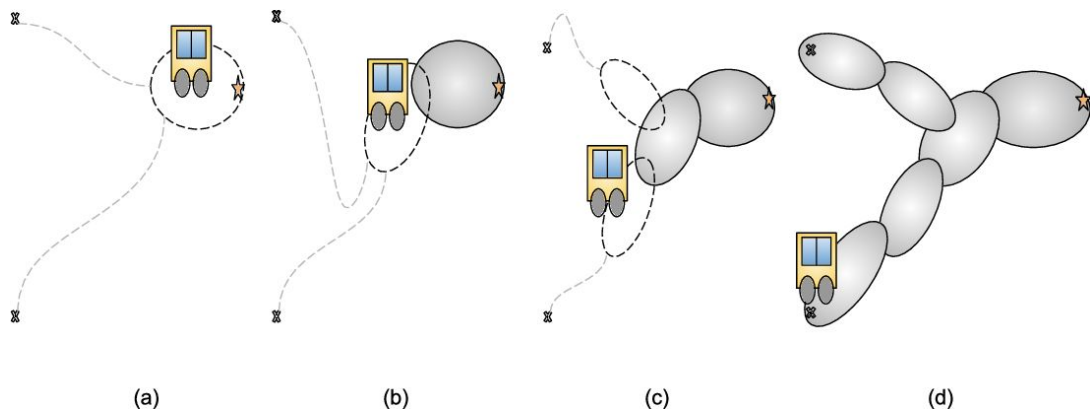
# Why DSG?

- **Deep Skill Chaining**, the prior work, has shown that in an explored region, skill chaining provides extremely high performance in constructing and solving for efficient skill paths between start and goal states.
- Learning value/reward states via MPC or any other RL protocol that can learn values allows you to **robustly construct MDPs** that represent your environment in terms of individual reachable actions.
- However, many goal states exist outside the known region - we need some way of expanding toward those regions and solving them without prior knowledge.
  - These goal states can be incredibly complex, **requiring new skill chains** to be made to approach the goal region of interest.
- *Thus, we take a graph-based approach to solving this problem.*

# Related Work

**Deep Skill Chaining** (Bagaria & Konidaris 2020, Bagaria et al., 2021)

- Within a known environment, we initiate the set of known options that encompass a local region.
  - This is known as the *initiation region*.
- We then look for another option that is toward the goal state with an *initiation region* of the previous option.
- Then, we recursively chain these until we connect the start and goal states.



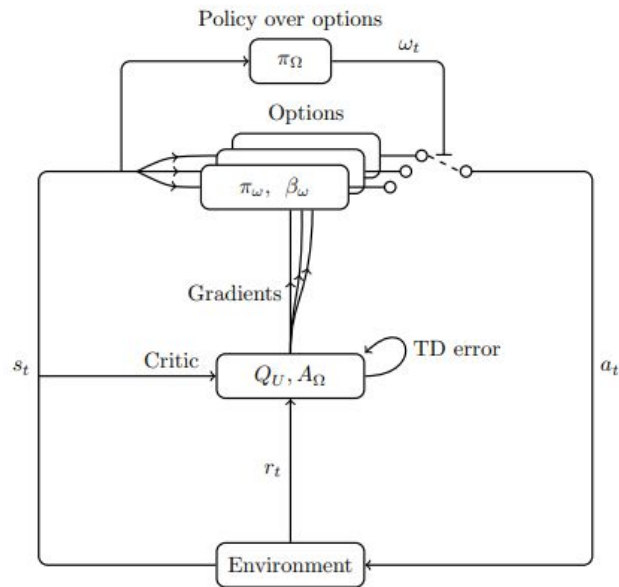
# Related Work

## Option-Critic (Bacon et al 2016):

- Hierarchical task planner, using a high level goal planner and solver.
- Then, lower level workers create the control framework to achieve these higher level goals.
- Can be combined with Actor-Critic and RL to learn *goal-conditioned* policies.
- *This is considered a feudal method.*

## Rapidly exploring random trees (RRT):

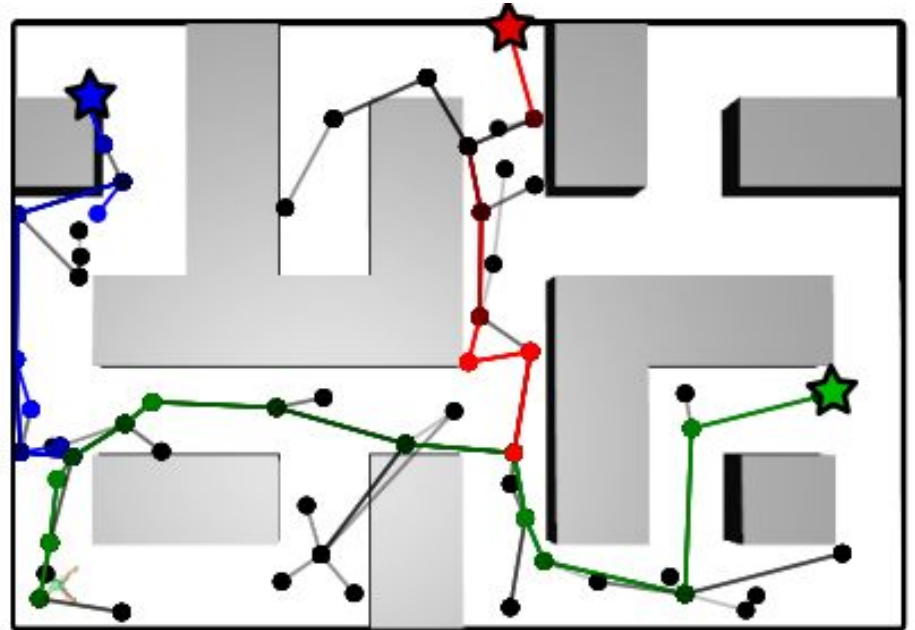
- Forms the crux of how we go from DSC to DSG.



# Defining the Problem

We define an  $n$ -dimensional continuous environment that:

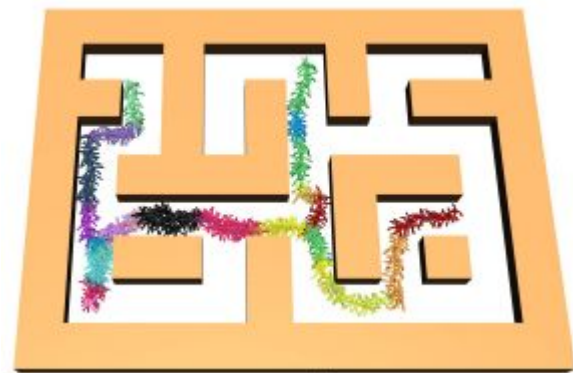
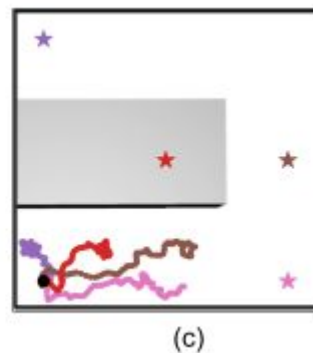
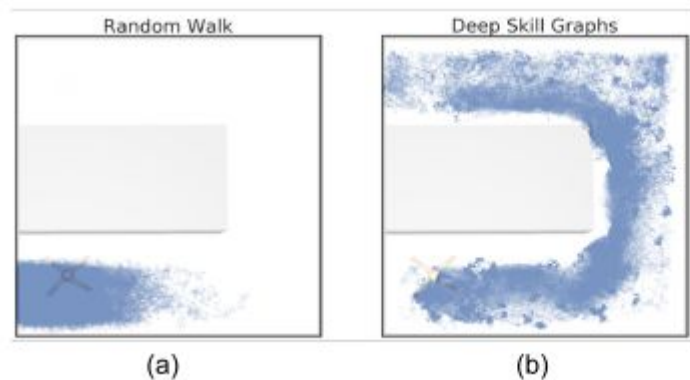
- Has a definable goal and start state, with a composable obstacle-free path between them.
- We can construct a Markov Decision Process (MDP) that can model the actions our robot can take.
- Thus, we can construct *skills*, which are directed graphs of *options*, defining our already solved region.



# Required Assumptions

- That we can define the problem as an **episodic, goal-oriented MDP**.
- That the probability of action is sufficiently consistent:
  - DSCs construct a region of high probability of success, which allows chaining.
  - If there is no high probability regions, we cannot create our deterministic planner.
- That the goal state and start state is clearly defined within the set, and there exists an obstacle-free path toward the goal state.
- That options (primitives) exist for the robot or planner in question to move and expand.

# Examples of Environments

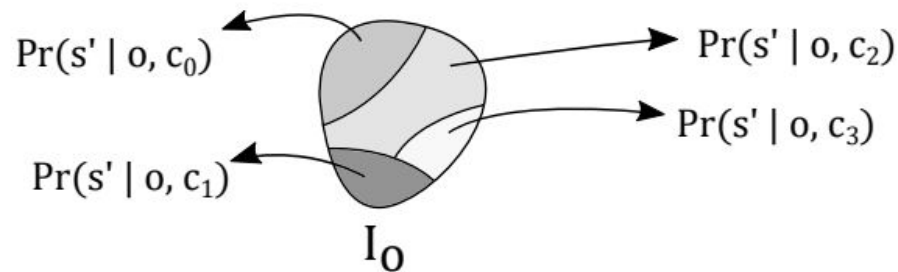
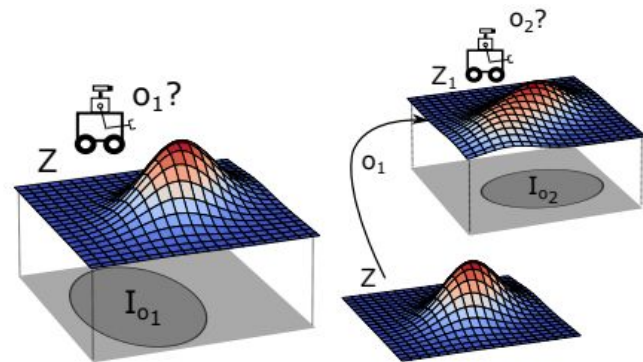


Ant U-Maze, standard two-dimensional graph-based maze.

# How does DSG Work? (Definitions)

## Options Framework:

- We can model abstraction actions as *options*, which map from an *initiation set* to a *termination set*.
  - These sets are the states in which the option can be initiated, and the states in which they end up after executing the option, respectively.
- We also define a controller that is able to execute the option at the low level with sufficiently high probability.
- With this, we can construct an SMDP, where we have our known primitives combined with our new options.



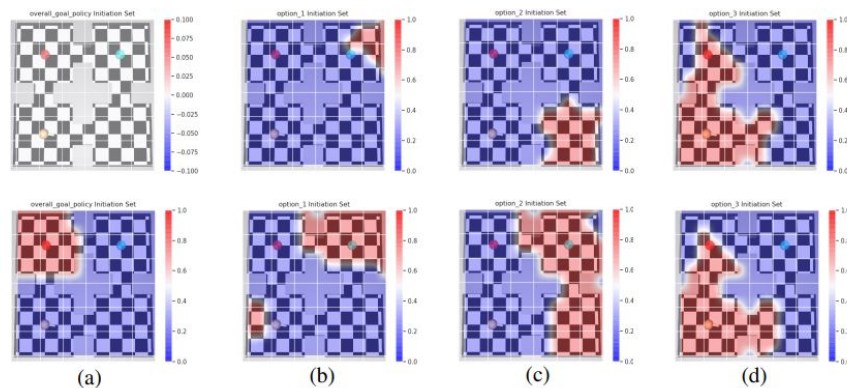
# How does DSG Work? (Definitions)

## Goal Region:

- The n-dimensional ball centered around goal vertex  $g$
- DSG proposes several subgoals, allowing us to have numerous goal regions that meet the criteria.

## Skill Graph:

- This is a directed graph, where each vertex  $V$  is either a goal region, or an option region.
- We then define edges as traversals between options, with weights and rewards.



# How does DSG Work? (More Definitions)

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

**Options Region(s):** The set of states reachable from the initiation region  $s$ .

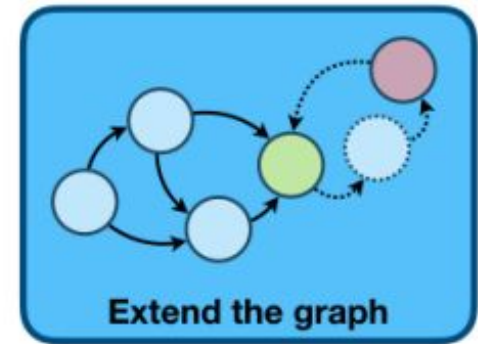
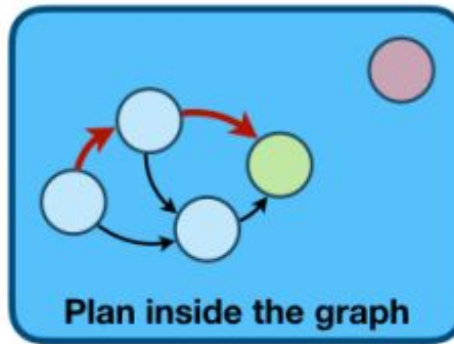
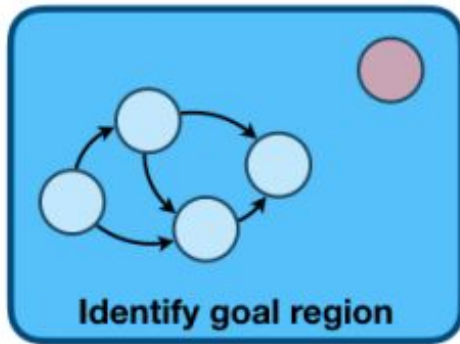
$$\mathcal{B}(s) = \{\epsilon_g \mid \epsilon_g(s) = 1, \forall \epsilon_g \in \mathcal{B}\}.$$

**Goal Region(s):** The set of states where the goal is satisfied by state  $s$ .

$$\mathcal{D}(v) = \{v' \mid \mathcal{G}.\text{has-path}(v, v'), \forall v' \in \mathcal{V}\}, \quad \text{Descendants}$$

$$\mathcal{A}(v) = \{v' \mid \mathcal{G}.\text{has-path}(v', v), \forall v' \in \mathcal{V}\}. \quad \text{Ancestors}$$

# How does DSG Work? (Overview)



We proceed similarly to RRT, but we alternate between two distinct stages: *exploration* and *consolidation*. Similar to RRT, we initiate several subgoal regions randomly (or biased) around the graph to aid in connectivity. At test time, we recursively backtrack and solve if it exists in the goal region, or just learn toward the goal.

# How does DSG Work? (Graph Expansion)

**We proceed analogously to RRT:**

1. Generate a subgoal  $s_{rand}$  that can be biased toward the goal region.
2. Identify the nearest neighbor, then navigate to it via Dijkstra.
3. Attempt to expand toward  $s_{rand}$ 
  - a. We limit this to  $K$  steps, a tunable parameter.
  - b. *While RRT assumes direct dynamics control, DSG assumes we use a proxy, such as MPC or learned RL policy to move  $K$  steps in the extension direction.*

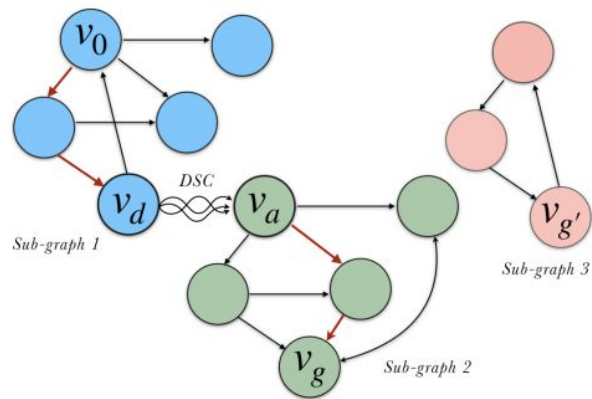
$$\begin{aligned} v_{nn} &= \arg \min_{v \in \mathcal{D}(s_t)} \|s_{rand} - v\|^2 \\ &= \arg \min_{v \in \mathcal{D}(s_t)} \left[ \max_{s \in \mathcal{E}_v} \|s_{rand} - s\|^2 \right] \end{aligned}$$

# How does DSG Work? (Graph Consolidation)

Proceed similarly to RRT-Connect/RRT\*:

1. Identify subgoals/regions of disconnect. A greedy algorithm targets the *nearest* unconnected graph.
2. Then, attempt to connect the two nearest neighbors. This is done using DSC in this case to minimize new skill generation.
3. Perform *smoothing*: add edges to connect options if and only if you can show that executing option  $o_1$  guarantees executability of option  $o_2$

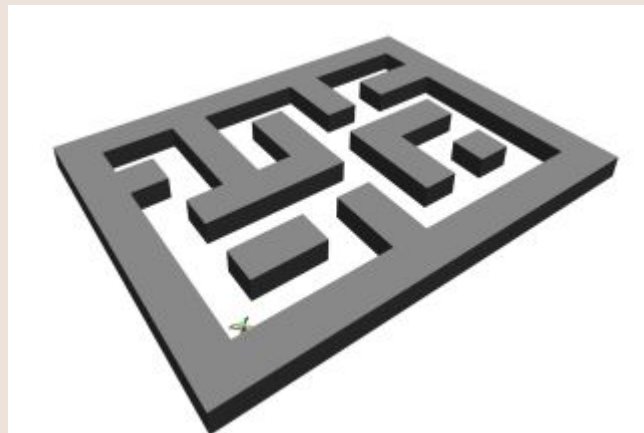
$$v_d, v_a = \arg \min_{\substack{v_d \in \mathcal{D}(s_t), \\ v_a \in \mathcal{A}(v_g)}} \|v_d - v_a\|^2$$
$$= \arg \min_{v_d, v_a} \left[ \max_{\substack{s_d \in \mathcal{E}_{v_d}, \\ s_a \in \mathcal{E}_{v_a}}} \|s_d - s_a\|^2 \right]$$



# Experiments and Design

Setup with mazes to represent arbitrary n-dimensional continuous skill problems:

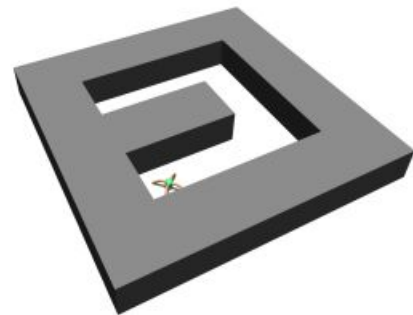
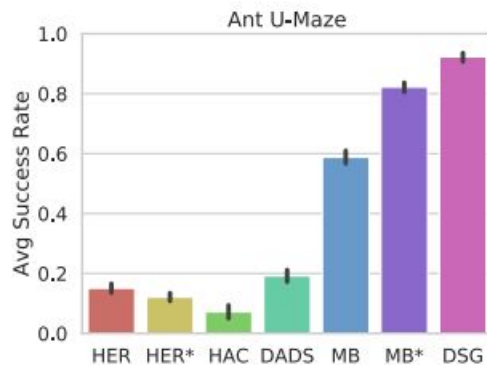
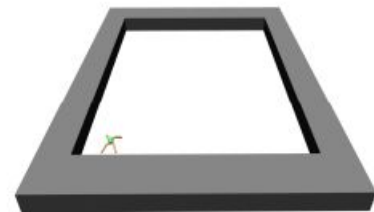
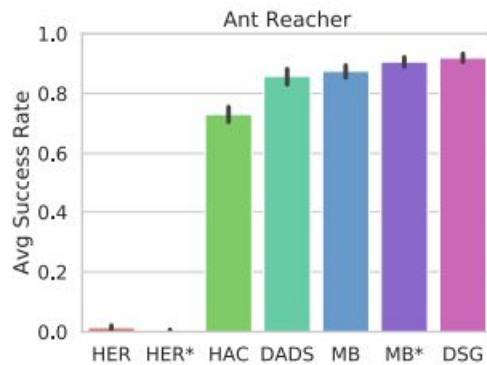
- MPC from (Nagabandi et al. 2018), a neural network approach they used to learn option policies.
- Played out an unsupervised training stage as follows:
  - 1000 episodes in Reacher and U-Maze
  - 1500 episodes of Medium-Maze
  - 2000 episodes of of Large-Maze
- At test time, they generated 20 random start-goal pairs, and evaluated their performance over 50 trials.



# Results

Compared against several baselines:

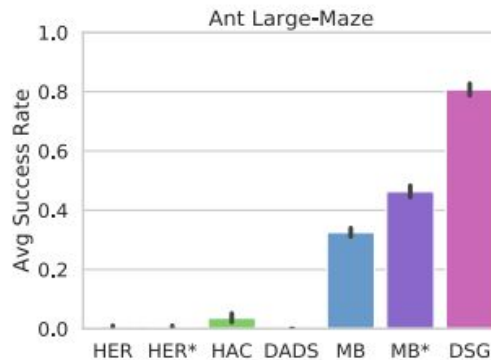
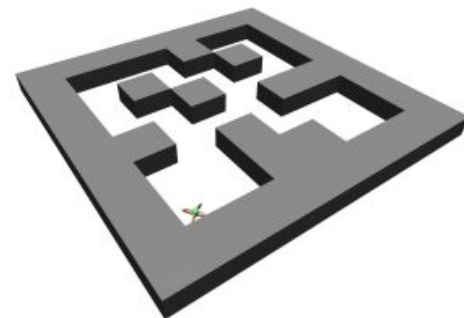
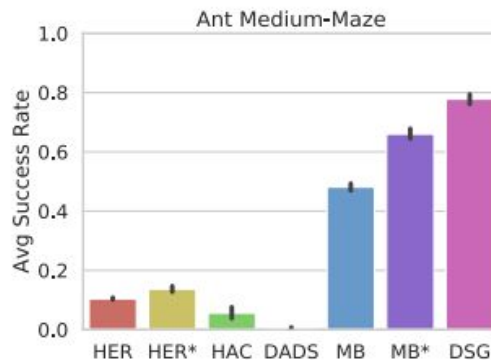
- **Flat Model-Free Baseline:** *Hindsight Experience Replay\**, Andrychowicz et al, 2017, which is a hierarchical policy learner, storing episodes to develop skills and rewards.
- **Flat Model-Based Baseline:** (Nagabandi et al. 2018)
- **Feudal HRL Baseline:** HAC, an extension of HER, (Levy et al. 2019)
- **Empowerment HRL Baseline:** DADS, (Sharma et al. 2020)



# Results

**DSG comfortably outperforms all other baselines:**

- HAC and DADS achieved similar performance on the simple mazes, but failed as the task increased in complexity.
  - This is because of assumptions made inside HAC and DADS about the problem space.
  - DADS creates the largest number of skills, but the skill fidelity is too low for complex problems.



# Limitations

- DSG tends to struggle with dense rewards - that is, it tends to get stuck if the nearest neighbor is on the other side of an obstacle, which they intend to correct with increased sparsity of rewards.
- DSG relies on deterministic outcomes in order to build out its skill graph - if the outcome is highly variable/probabilistic, DSG can easily get stuck/be unable to properly plan/relearn.
- States must be defined as episodic, goal-oriented MDPs and be solvable in this region - this makes it possibly difficult for problems of extreme variance.

Questions?