In Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI 2024)
Vancouver, BC, Canada

Hierarchical Planning and Learning for Robots in Stochastic Settings Using Zero-Shot Option Invention

Naman Shah, Siddharth Srivastava

School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA
shah.naman, siddharths@asu.edu

Abstract

This paper addresses the problem of inventing and using hierarchical representations for stochastic robot-planning problems. Rather than using hand-coded state or action representations as input, it presents new methods for learning how to create a high-level action representation for long-horizon, sparse reward robot planning problems in stochastic settings with unknown dynamics. After training, this system yields a robot-specific but environment independent planning system. Given new problem instances in unseen stochastic environments, it first creates zero-shot options (without any experience on the new environment) with dense pseudo-rewards and then uses them to solve the input problem in a hierarchical planning and refinement process. Theoretical results identify sufficient conditions for completeness of the presented approach. Extensive empirical analysis shows that even in settings that go beyond these sufficient conditions, this approach convincingly outperforms baselines by $2\times$ in terms of solution time with orders of magnitude improvement in solution quality.

1 Introduction

Recent work on robot planning and learning has led to strong progress on problems with short horizons, dense rewards, and/or deterministic dynamics encoded in simulators such as MuJoCo. While state-of-the-art methods perform well in such settings this progress has been difficult to translate to pervasive robotics problems that feature long-horizons, sparse rewards, and stochastic dynamics. In these settings, their performance falls rapidly. E.g., Fig. 1 shows that recent advances fail to scale for robot motion planning in a large office space with stochastic dynamics. In fact, their performance drops below that of naïve approaches such as continual re-planning. Such settings constitute a massive departure from the short, dense and/or deterministic settings that has been the focus of much recent work on the topic.

This paper addresses the problem of robot planning in the relatively under-studied long horizon, sparse reward and stochastic setting with unknown system dynamics. Solving such problems is challenging: stochasticity implies that deterministic motion planning is not sufficient: the robot can reach unexpected states and thus we need solutions in the form of policies rather than motion plans. Furthermore, the absence of well-defined dynamics models typically requires reinforcement learning (RL) based approaches, but RL algorithms are difficult to scale in long-horizon, sparse-reward settings (as also evidenced in Fig. 1).

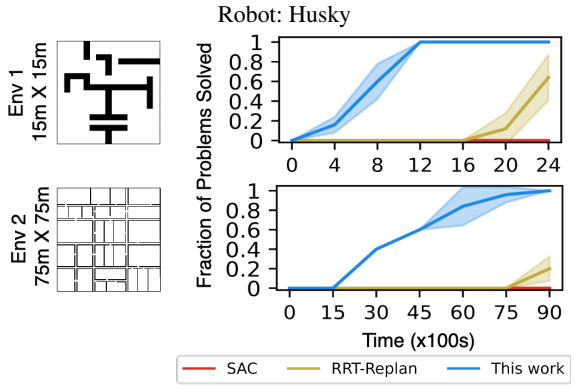

Figure 1: Performance of the two state-of-the-art approaches as a measure of fraction of problems solved (y-axis) in the given time (x-axis). The blue line presents the lower performing variant of the approach developed in this paper. Existing approaches for robot planning show limited scalability in stochastic environments. As the environment size increases, performance falls below that of naïve RRT-Replan.

We address these technical problems using a novel approach for hierarchical planning and learning that *learns how to invent neuro-symbolic abstractions of stochastic robot planning problems* in terms of useful high-level actions represented as options (Sutton, Precup, and Singh 1999) or composable sub-policies. A training phase adapts our general approach to a given robot-class, and yields a robot-attuned, environment-independent planning system (Sec. 3). In other words, this approach learns how to transform an input continuous problem into a discrete symbolic search problem over the identified options, which is then solved and refined using a hierarchical planning and refinement algorithm (Sec. 4).

After training, when given a new robot planning problem in an unseen, stochastic environment, this approach invents zero-shot options (without any new experience) in the form of desirable pairs of initiation and termination sets, and zero-shot, dense pseudo-reward that can be used to carry

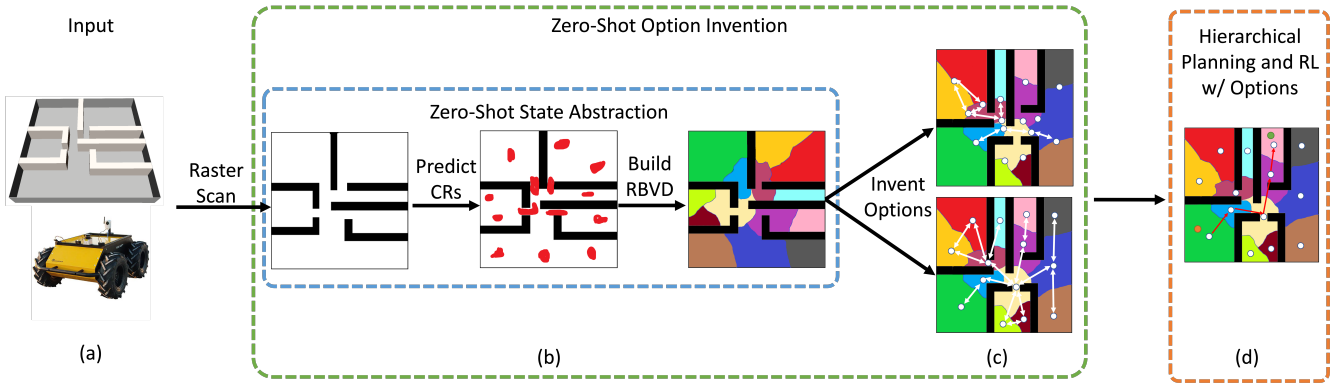

Figure 2: Our overall approach for automatically inventing high-level options. (a) shows the input to our system. (b) shows the zero shot abstraction process along with the raster scan of the input environment (left), critical regions predicted by the learned network (center) and the zero-shot state abstraction (right). The top image in (c) shows a subset of automatically invented interface options and the bottom image shows a subset of automatically invented centroid options. Lastly, these learned options are used for hierarchical planning and learning. Red arrows in (d) shows an example of a high-level plan over centroid options given an initial configuration (orange area) and a goal configuration (green area). Policies for these options are learned using deep reinforcement learning and the auto-generated dense pseudo-reward function.

out RL for learning policies for the invented options (Fig 2). Short horizons and dense pseudo-rewards computed for zero-shot options make option-policy learning significantly more sample-efficient than end-to-end policy learning.

Our main contributions are (a) a unified hierarchical learning, planning and refinement approach for learning how to solve new long-horizon, stochastic problems in sparse reward settings by abstracting them into a space with discrete states and actions; (b) algorithms for learning how to zero-shot invent high-level actions for a given robot for stochastic planning problems in test environments not encountered during training; and (c) zero-shot invention of dense pseudo-reward functions for the invented options.

To our knowledge, this paper presents the first approach for *learning how to zero-shot invent* a hierarchical representation for stochastic robot planning problems not seen during training. Unlike prior work on the topic, our approach does not require input state or action abstractions, and requires only a kinematic robot specification and a problem generator. This results in robust transferability of learning. If multiple test problems come from the same environment, options can be re-used across instances. Furthermore, robots used during testing or deployment need not be the same as the robots used while training the option invention pipeline, as long as they have similar kinematic constraints.

Theoretical results characterize the formal properties of this approach such as sufficient conditions for completeness (Sec. 4.1), and show that this approach ensures downward-refinability of the invented options for a class of robots. Extensive empirical analysis (Sec. 5) shows that even in situations that go beyond our currently defined sufficient conditions, this approach provides $\sim 2\times$ improvement over existing approaches in terms of computational efficiency and order-of-magnitude improvements in solution quality.

2 Background

Let $\mathcal{X} \subseteq \mathbb{R}^d = \mathcal{X}_{\text{free}} \cup \mathcal{X}_{\text{obs}}$ be the configuration space (C-space) of a robot R and let O be a set of obstacles in a given environment. Given a collision function $f : \mathcal{X} \to \{0, 1\}$, $\mathcal{X}_{\text{free}}$ represents the set of configurations that are not in collision with any obstacle $o \in O$ such that $f(x) = 0$ and let $\mathcal{X}_{\text{obs}} = \mathcal{X} \setminus \mathcal{X}_{\text{free}}$. Let $x_i \in \mathcal{X}_{\text{free}}$ be the initial configuration of the robot and $x_g \in \mathcal{X}_{\text{free}}$ be the goal configuration of the robot. The motion planning problem can be defined as:

Definition 1. *A **motion planning problem** $\mathcal{M}$ is defined as a 4-tuple $\langle \mathcal{X}, f, x_i, x_g \rangle$, where $\mathcal{X}$ is the C-space, f is the collision function, x_i and x_g are initial and goal configurations.*

A solution to a motion planning problem is a motion plan τ. A motion plan is a sequence of configurations $\langle x_0, \ldots, x_n \rangle$ such that $x_0 = x_i$, $x_n = x_g$, and $\forall x \in \tau, f(x) = 0$. Robots use controllers that accept sequenced configurations from the motion plan and generate controls that take the robot from one configuration to the next configuration. In practice, environment dynamics can be noisy, which introduces stochasticity in the problem.

We define stochastic motion planning (SMP) problems in a manner similar to stochastic shortest path problems (SSPs) (Bertsekas and Tsitsiklis 1991). Formally, a *stochastic motion planning problem* P is defined as $P = \langle \mathcal{X}, \mathcal{U}, T, x_0, x_g \rangle$ where $\mathcal{X} \subseteq \mathbb{R}^d$ is a d-dimensional configuration space. $\mathcal{U} \subseteq \mathbb{R}^d$ is the uncountably infinite set of stochastic control actions defined in terms of the intended change in each degree of freedom of the robot. Each $u \in \mathcal{U}$ follows a stochastic transition function $T_u : x \mapsto \mu(x + u)$ where $\mu(x + u)$ is a probability measure parameterized using the intended target $x + u$ of the control action. x_0 is the initial configuration and x_g is the goal configuration. A solution to a stochastic motion planning problem is a partial policy $\pi : \mathcal{X} \to \mathcal{U}$ that maps each reachable configuration in the configuration space (when starting with x_0 and following π)

to a control action from the set of controls (actions) $\mathcal{U}$.

Options Options provide temporal action abstractions over primitive robot actions. Sutton, Precup, and Singh (1999) define an option o as a 3-tuple $o = \langle \mathcal{I}_o, \beta_o, \pi_o \rangle$. Here, $\mathcal{I}_o$ defines an initiation set, i.e., the set of states where the option o can be executed; β_o defines a termination set, i.e., the set of states where execution of the option o ends; π_o defines a local option policy. We say two options o_i and o_j are composable iff $\mathcal{I}_{o_j} \subseteq \beta_{o_i}$, i.e., initiation set of an option is a subset of the termination set of another option.

2.1 State Abstractions

In this work we use the notion of state abstractions as a finite partitioning of a continuous state space. Formally, a state abstraction from a concrete state space $\mathcal{X}$ to a finite, discrete state space S is a function that maps each $x \in \mathcal{X}$ to an element of S. State abstractions created by domain experts have been used extensively to speed up planning and learning. Recent work also presents approaches for learning state abstractions (e.g., (Konidaris, Kaelbling, and Lozano-Perez 2018; Shah and Srivastava 2022)). In this work we utilize the critical-region based state abstraction technique developed by Shah and Srivastava (2022) since it allows zero-shot state abstractions for new problems. We present here a brief summary of this approach to highlight the properties and input requirements that are utilized in the current paper.

In this approach, critical regions of a configuration space characterize regions that tend to have a high solution density and thus are essential for solving problem instances using sampling based motion planners. This concept generalizes and unifies notions of hubs (e.g., the center of a room from which multiple locations are accessible) and bottlenecks (e.g., a doorway that forces the robot to follow a narrow path). Given the kinematic model of a robot, it is possible to train a deep neural network to predict critical regions for new environments. A critical region prediction can, in turn be used to define an abstraction of the configuration space:

Definition 2. *Given a configuration space $\mathcal{X}$, let d^c define the minimum distance between a configuration $x \in \mathcal{X}$ and a region $\phi \subseteq \mathcal{X}$. Given a set of critical regions Φ and a robot R, a **region-based Voronoi diagram (RBVD)** Ψ is a partitioning of $\mathcal{X}$ such that for every Voronoi cell $\psi_i \in \Psi$ there exists a region $\phi_i \in \Phi$ such that forall $x \in \psi_i$ and forall $\phi_j \neq \phi_i$, $d^c(x, \phi_i) < d^c(x, \phi_j)$ and each ψ_i is connected.*

In this framework, abstract states are defined using a bijective function $\ell : \Psi \rightarrow \mathcal{S}$ that maps each Voronoi cell $\psi \in \Psi$ to an abstract state $s \in \mathcal{S}$. The RBVD Ψ induces an abstraction function $\alpha : \mathcal{X} \rightarrow \mathcal{S}$ such that $\alpha(x) = s$ iff there exists a Voronoi cell ψ such that $x \in \psi$ and $\ell(\psi) = s$. A configuration $x \in \mathcal{X}$ is said to be in the *high-level abstract state* $s \in \mathcal{S}$ (denoted by $x \in s$) if $\alpha(x) = s$. A neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$ such that for a pair of states $s_1, s_2 \in \mathcal{S}$, $\mathcal{V}(s_1, s_2) = 1$ iff s_1 and s_2 are neighbors. We say a pair of abstract states s_1 and s_2 are neighbors iff there exists a pair of configuration $x_1 \in s_1$ and $x_2 \in s_2$ such that there exists a motion plan between x_1 and x_2.

Algorithm 1: OptionInventor

Input: robot R, training environments E_{train}, test environment E_{test}
Output: set of option $\mathcal{O}$, cost function C

1 $\Theta \leftarrow$ get_critical_region_predicter(R);
2 **if** Θ *is not trained* **then**
3 train Θ using E_{train}
4 $\Phi \leftarrow$ predict_critical_regions(E_{test},Θ);
5 $\Psi, \mathcal{S}, \mathcal{V} \leftarrow$ construct_RBVD(E_{test},Φ);
6 $\mathcal{O}, C \leftarrow$ construct_options(Ψ,$\mathcal{S}$,$\mathcal{V}$);
7 **foreach** $o \in \mathcal{O}$ **do**
8 $\text{mp}_o \leftarrow$ compute_motion_plan(o);
9 $\mathcal{G}_o \leftarrow$ compute_option_guide(o,mp_o);
10 return $\mathcal{O}, C$

Critical regions predictor We must first identify critical regions in an environment in order to synthesize state abstractions as discussed above. Shah and Srivastava (2022) provide an approach for learning generalizable critical region predictors in the form of a deep neural network. These predictors are environment independent and generalizable across robots to a large extent. They are only trained once per the kinematic charecteristics of a robot and reused for similar robots. E.g., the non-holonomic robots used to evaluate our approach (details in Sec. 5) are different from the robots used by Shah and Srivastava (2022), however, we used the critical regions predictor developed and made available by them for a rectangular holonomic robot without any additional training. We now briefly highlight the training process.

The training data is generated using a set of training environments and a random problem generator for these environments. The predictors used for this work are learned using 20 training environments. The random goal generator is used to generated 100 different initial and goal configurations and each problem was solved 50 times using a sampling-based motion planner to generate training labels. A UNet (Ronneberger, Fischer, and Brox 2015) was trained using these labels and occupancy matrix of the environment as input. The trained neural network can then be used to predict critical regions in any unseen test environment.

3 Zero-Shot Option Inventors

Our overall approach for solving long-horizon, stochastic robot planning problems is to zero-shot invent a set of options for the given problem (Alg. 1), and then to carry out hierarchical planning using these options (Alg. 2). In this section we outline our approach for automatically identifying options (OptionInventor, Alg. 1) for a given environment.

Given a stochastic motion planning problem, Alg. 1 creates a zero-shot state abstraction (lines 1-5) using the methods presented above (Sec. 2.1). Fig. 2(a) and (b) show this process in an example environment. Once abstract states are constructed, we define abstract actions as options (line 6) with their initiation set in one abstract state and the termination set in a different abstract state (discussed in Sec. 3.1). These options (action abstractions) are independent of problem instances, i.e., they are constructed once per environment

and robot and reused for different problems (pairs of initial and goal configurations). However, we still need to learn policies for executing such options. As defined, option termination sets turn out to be insufficient for efficiency: they result in a sparse-reward setting, which makes it difficult to scale RL algorithms for policy learning. To address this limitation, lines 7-9 also create in zero-shot fashion (without collecting additional experience from the environment), an *option guide*: a dense pseudo-reward function for the invented options (discussed in Sec. 3.2).

3.1 Zero-Shot Option Endpoints

Given a set of zero-shot abstract states $\mathcal{S}$ created using the predicted critical regions for a new environment (Def. 2), a neighborhood function $\mathcal{V}$, and an abstraction function α, we define two types of options: (1) *centroid options* that take the robot from the centroid of one critical region to another, and (2) *interface options* that take the robot across an abstract state, i.e., from the boundary between s_i and s_j to the boundary between s_i and s_k. Both forms of options can be composed to solve long-horizon problems (this process is discussed in the next section).

First, we discuss centroid options. Intuitively, these options define abstract actions that transition between a pair of critical regions. Formally, they are defined as follows:

Definition 3. *Let $s_i \in \mathcal{S}$ be an abstract state in the RBVD Ψ with the corresponding critical region $\phi_i \in \Phi$. Let d be the Euclidean distance measure and let t define a threshold distance. Let c_i be the centroid of the critical region r_i. A **centroid region** of the critical region r_i with the centroid c_i is defined as a set of configurations: $\{x | x \in s_i \wedge d(x, c_i) < t\}$.*

We use this definition to define the endpoints for the centroid options as follows:

Definition 4. *Let $s_i, s_j \in \mathcal{S}$ be neighboring abstract states such that $\mathcal{V}(s_i, s_j) = 1$ in an RBVD Ψ constructed using the set of critical regions Φ. Let $\phi_i, \phi_j \in \Phi$ be the critical regions for the abstract states s_i and s_j and let c_i and c_j be their centroids regions. The **endpoints for a centroid option** are defined as a pair $\langle \mathcal{I}_{ij}, \beta_{ij} \rangle$ such that $\mathcal{I}_{ij} = c_i$ and $\beta_{ij} = c_j$.*

Interface options serve as a dual to centroid options. Rather than defining high-level actions that move from the "center" of one abstract state to the "center" of another, they define high-level actions for going across an abstract state, from one boundary to another. To formally define interface options, we first need to define "interface" regions between a pair of neighboring abstract states:

Definition 5. *Let $s_i, s_j \in \mathcal{S}$ be a pair of neighboring states such that $\mathcal{V}(s_i, s_j) = 1$ and ϕ_i and ϕ_j be their corresponding critical regions. Let $d^c(x, \phi)$ define the minimum Euclidean distance between configuration $x \in \mathcal{X}$ and some point in a region $\phi \subset \mathcal{X}$. Let p be a configuration such that $d^c(p, \phi_i) = d^c(p, \phi_j)$ that is, p is on the border of the Voronoi cells that define s_i and s_j. Given the Euclidean distance measure d and a threshold distance t, an **interface region** for a pair of neighboring states (s_i, s_j) is defined as a set $\{x | (x \in s_i \vee x \in s_j) \wedge d(x, p) < t\}$.*

We use this definition of interface regions to define endpoints for the interface options as follows:

Definition 6. *Let $s_i, s_j, s_k \in \mathcal{S}$ be abstract states such that $\mathcal{V}(s_i, s_j) = 1$ and $\mathcal{V}(s_j, s_k) = 1$. Let $\hat{\phi}_{ij}$ and $\hat{\phi}_{jk}$ be the interface regions for pairs of high-level states (s_i, s_j) and (s_j, s_k). The **endpoints for an interface option** are defined as a pair $\langle \mathcal{I}_{o_{ijk}}, \beta_{o_{ijk}} \rangle$ such that $\mathcal{I}_{o_{ijk}} = \hat{\phi}_{ij}$ and $\beta_{o_{ijk}} = \hat{\phi}_{jk}$.*

We can now utilize these definitions to define, in zero-shot fashion, the complete set of centroid options and interface options for a new environment. Recall that the RBVD Ψ induces a neighborhood function $\mathcal{V} : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$. The set of centroid options is defined as $\mathcal{O}_c = \{o_{ij} | \forall s_i, s_j \in \mathcal{S},\ \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{I}_{ij} = c_i \wedge \beta_{ij} = c_j\}$, where c_i represents the centroid of the critical region r_i for the abstract state s_i.

Similarly, the set of interface options is defined as $\mathcal{O}_i = \{o_{ijk} | \forall s_i, s_j, s_k \in \mathcal{S},\ \mathcal{V}(s_i, s_j) = 1 \wedge \mathcal{V}(s_j, s_k) = 1 \wedge \mathcal{I}_{ij} = \hat{\phi}_{ij} \wedge \beta_{ij} = \hat{\phi}_{jk}\}$, where $\hat{\phi}_{ij}$ represents an interface region for a pair of neighboring abstract states s_i and s_j. Fig. 2(c) shows an example of automatically invented centroid and interface options for the environment shown in Fig. 2(a). This options can be used for hierarchical planning and learning as shown in Fig. 2(d) (explained in Sec. 4).

3.2 Zero-Shot Option Guides

Given an option defined using the methods discussed above, we define an option guide as a dense pseudo-reward function. We will use the option guide to improve sample efficiency while learning policy for an option in sparse reward settings.

Intuitively, option guides are defined using conceptual envelopes around *deterministic* motion plans that can be computed relatively easily using existing methods. Formally, we define an *ϵ-clear motion plan* as a motion plan in which every configuration has an ϵ-neighborhood that is collision free. With a slight abuse of notation we use the abstraction function with a set of low-level configurations rather than a single configuration such that for a set A, $\alpha(A) = \{\alpha(x) | \forall x \in A\}$.

Let o_i be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$, and centroids $c_{\mathcal{I}_i}$ and c_{β_i} for $\mathcal{I}_i$ and β_i respectively.

Given a threshold distance t, an arbitrary neighborhood radius ϵ, and a Euclidean distance measure d, an ϵ-clear motion plan $\mathcal{G}$ for an option o is defined as $\mathcal{G} = \langle p_0, \ldots, p_n \rangle$ such that $p_0 = c_{\mathcal{I}}$, $p_n = c_{\beta}$, every point in $p_i \in \mathcal{G}$ has an ϵ-clear neighborhood, and for every pair of points $p_i, p_{i+1} \in \mathcal{G}$, $d(p_i, p_{i+1}) < t (< 2\epsilon)$. In practice, we found that any sampling-based motion planner with ϵ-inflated obstacles can be used to construct such motion plans.

We define the option guide for o_i as a dense pseudo-reward defined using $\mathcal{G}_i$ as follows. Intuitively, the option guide is a dense pseudo-reward function that provides the robot with a large positive reward when it reaches the termination set of the goal, a penalty for drifting to a different abstract state, and a smoothened reward for making progress on the option guide. Formally, this is defined as follows:

Definition 7. *Let o_i be an option with endpoints $\langle \mathcal{I}_i, \beta_i \rangle$ and let $\mathcal{G}_i = \langle p_1, \ldots, p_m \rangle$ be an ϵ-clear motion plan for a given ϵ. Given a configuration $x \in \mathcal{X}$, let $n(x) = p_i$ define the*

Algorithm 2: Stochastic Hierarchical Abstraction-guided Robot Planner (SHARP)

Input: Training environments E_{train}, test environment E_{test}, initial and goal configurations x_i and x_g
Output: A policy Π composed of options

1 **if** *abstraction is* ***not*** *constructed* **then**
2 $\quad \mathcal{O}, C \leftarrow$ OptionInventor(R,E_{train},E_{test});
3 $s_i, s_g \leftarrow$ get_abstract_states(x_i,x_g);
4 **while** *not refined* **do**
5 $\quad p \leftarrow$ get_new_high_level_plan(s_i,s_g,$\mathcal{O}$,C);
6 $\quad$ **if** $p = \emptyset$ **then**
7 $\quad\quad$ break;
8 $\quad \Pi$ = empty_list;
9 $\quad \pi_0 \leftarrow$ lear_ll_policy(x_i,$\mathcal{I}_{o_1}$);
10 $\quad \Pi$.add(π_0);
11 $\quad$ **foreach** $o \in p$ **do**
12 $\quad\quad$ **if** π_o *does* ***not*** *exist* **then**
13 $\quad\quad\quad$ **if** $\mathcal{G}_o = \emptyset$ **then**
14 $\quad\quad\quad\quad$ flag o infeasible;
15 $\quad\quad\quad\quad$ break;
16 $\quad\quad\quad$ $\pi_o \leftarrow$ learn_ll_policy($\mathcal{I}_o$,β_o,$\mathcal{G}_o$);
17 $\quad\quad\quad$ adjust the option cost C_o;
18 $\quad\quad$ Π.add(π_o);
19 $\quad$ refined $\leftarrow$ True;
20 **if** *refined* **then**
21 $\quad \pi_{n+1} \leftarrow$ learn_ll_policy(β_{o_n},x_g);
22 $\quad \Pi$.add(π_{n+1});
23 $\quad$ **return** Π;
24 **else**
25 $\quad$ **return** failure;

closest point on $\mathcal{G}_i$. The ***option guide*** *$R_i(x)$ is defined as:*

$$R_i(x) = \begin{cases} r_t & \text{if } x \in \beta_i \\ r_p & \text{if } \alpha(x) \in \mathcal{S}/\{\alpha(\mathcal{I}_i), \alpha(\beta_i)\} \\ -(d(x, n(x)) & \\ \quad +d(n(x), p_m)) & \text{otherwise} \end{cases}$$

The next section uses these concepts to present our approach for solving a stochastic motion planning problem.

4 Hierarchical Stochastic Motion Planning Using Zero-Shot Options

The SHARP algorithm (Alg. 2) presents our overall approach for using the zero-shot options defined above for hierarchical motion planning under uncertainty. It takes as input an SMP problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$, a simulator, and an occupancy matrix of the environment, and produces a partial policy $\Pi : \mathcal{X} \rightarrow \mathcal{U}$ that maps each reachable robot configuration to a control action. The algorithm starts by invoking the OptionInventor in line 2 to construct zero-shot state and action abstractions (in the form of options) if they have not been constructed for the given robot R and the environment E_{test} pair (Sec. 3).

Lines 4-19 use these options as high-level actions for computing high-level plans. Line 5 uses an incremental plan generator that takes the set of invented options along with the abstract initial and goal states as input and generates a high-level plan using A* search. This module considers the initiation and termination sets of the invented options as preconditions and effects. It uses the Euclidean distance between the termination set of the option and the goal configuration as the heuristic and the Euclidean distance between the initiation and termination sets as an initial approximation to the cost of the option.

Once a plan in the form of a sequence of options is obtained in line 5, SHARP starts refining each option in the plan by computing option policies. However, before computing the policy for the first option in the plan, it generates an additional option o_0 such that $\mathcal{I}_{o_0} = x_i$ and $\beta_{o_0} = \mathcal{I}_{o_1}$ and learns its policy (line 9). If a policy exists for the option from the previous invocation of the algorithm, then our approach uses the same policy. Before computing a policy for an option, Alg. 2 checks for its option guide. If an option guide does not exist, the option is marked as infeasible and a new high-level plan is computed from the initial abstract state (line 14). Once an option guide is computed for an option, line 16 uses an off-the-shelf low-level policy learner to compute a policy for it. After computing (or reusing) policies for all the options in the plan, line 21 generates an additional option o_{n+1} such that $\mathcal{I}_{o_{n+1}} = \beta_{o_n}$ and $\beta_{o_{n+1}} = x_g$ and learns its policy.

Finally, we compute a *composed policy* by composing policies for every option in this high-level plan (lines 18 and 22). A **composed policy** Π for a high-level plan is an finite state controller with one state for each option in the plan. For a controller state q_i, $\Pi(x) = \pi_i(x)$ where π_i represents the policy for option $o_i \in \mathcal{O}$. The controller makes a transition $q_i \rightarrow q_{i+1}$ when the robot reaches a configuration $x \in \mathcal{I}_{o_{i+1}}$.

In order to aid transferability, SHARP only synthesizes options once per each environment and robot. It efficiently transfers the learned option policies by updating the option costs (C) using the average number of steps from initiation sets to the termination sets of the options in multiple rollouts of the learned option policies (line 17).

4.1 Theoretical Results

We now present theoretical properties of Alg. 2. Let $B_\delta(x)$ for $\delta > 0$ define the δ-neighborhood of $x \in \mathcal{X}$ under the Euclidean metric. Recall that each controller implicitly defines a transition function with a probability distribution $\mu(x+u)$ for the control action u (see Sec. 2). A *δ-compliant controller* is defined as one whose set of support for $\mu(x+u)$ is $B_\delta(x+u)$. Our formal guarantees do not require knowledge of μ other than an upper bound on the support radius. Here, we refer to δ as the *support radius* for the given controller.

The following theoretical results characterize formal properties of the presented approach. We present the results below; proofs are included in extended version of our paper (Shah and Srivastava 2024).

Thm. 4.1 shows that the construction process of the options ensures that the zero-shot options are indeed composable and can be used for high-level deterministic planning.

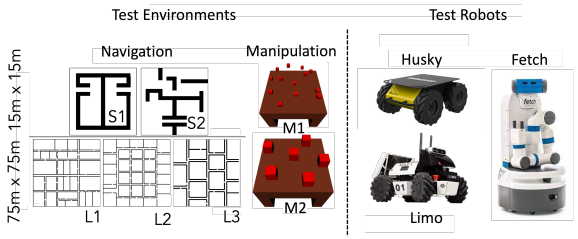

Figure 3: Our test environments and robots.

Theorem 4.1. *For a given stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_1, x_n \rangle$, let Φ be the set of identified critical regions and Ψ be the RBVD that induces the set of abstract state $\mathcal{S}$ and a neighborhood function $\mathcal{V}$. If there exists a sequence of distinct abstract states $\langle s_1, \ldots, s_n \rangle$ such that $\mathcal{V}(s_i, s_{i+1}) = 1$ then there exists a composed policy Π such that the resulting configuration after the termination of every option in Π would be the goal configuration x_n.*

Thm. 4.2 asserts that when used with an optimal low-level policy learner, SHARP is probabilistically complete for holonomic robots.

Theorem 4.2. *Given a stochastic motion planning problem $P = \langle \mathcal{X}, \mathcal{U}, x_i, x_g \rangle$ for a holonomic robot R using a controller with a support radius $\delta_c < \delta$, a motion planner that can compute δ-clear motion plans, and an optimal low-level policy learner, if there exists a δ-clear motion plan for the robot R from x_1 to x_n that forms a sequence of distinct abstract states, then Alg. 2 will find a proper policy for the given stochastic motion planning problem.*

These results provide the foundations for analyzing such approaches and show a completeness result for the presented approach. However, our approach generalizes beyond the sufficient (and not necessary) conditions used in the theorems above. In fact our empirical evaluation (Sec. 5) is conducted on non-holonomic robots that violate the premises of these results. Furthermore, we use default controllers with unknown support radii.

5 Empirical Results

We present the salient aspects of our implementation, setup, and observations here; additional results, code, and data are available in the supplementary material.

Our evaluation is organized to address the following key questions: (1) Does the presented approach of zero-shot option invention followed by hierarchical planning and refinement improve performance in terms of computational efficiency and solution quality?; and (2) Can zero-shot options be transferred to new problems in the same environment?

Results across an extensive evaluation suite indicate that the presented approach creates and uses zero-shot options effectively. In larger environments (L1-L3), ours is the only approach that shows significant learning, and it achieves a significantly higher solution quality than all baselines. We now present our evaluation framework and results in detail.

Evaluation framework and metrics We organized the overall evaluation of the presented approach as follows. Given a previously unseen environment E_{test} and a problem instance $\langle x_i, x_g \rangle$, SHARP (Alg. 2) zero-shot invents options for E_{test} and uses them to compute a policy for the test problem instance. The total solution time recorded for SHARP includes the time taken to run OptionInventor (which includes predicting critical regions, creating state abstractions, inventing option signatures, and computing option guides), and to execute hierarchical planning and refinement process listed under the SHARP algorithm (Alg. 2).

We evaluated the *computational efficiency* of all considered approaches in terms of the number of problems solved in a given amount of time. For learning-based approaches, a problem is considered to be solved in these experiments when the current learned policy yields an average reward of +500 over 10 rollouts. For RRT-replan, a problem is considered to be solved when the robot reaches the 0.2m neighborhood of the goal configuration. All approaches were assigned a uniform timeout per problem of 2400 or 9000 seconds.

In addition, we use two metrics to evaluate solution quality since it is often easy to compute meaningless policies in a short time frame: The *average solution cost* is defined as the average number of steps taken while executing a computed solution; *solution reliability* is defined as the likelihood of solving the given problem by executing the computed solution. Both metrics are computed over 20 independent trials of the computed solution on the input problem instance.

Figs. 4, 5, and 6 summarize the results of our evaluation in terms of these metrics across a wide range of robots, environments and test problems. We discuss the details of this evaluation including notes on our implementation, environment and baseline selection, and our main observations below.

Our implementation We implemented two variants of our approach: SHARP-centroids and SHARP-interfaces, which invent and use centroid options and interface options, respectively. Both implementations use PyBullet and PyTorch (Paszke et al. 2019). PyBullet does not feature stochasticity robot movements. We introduced stochasticity by adding random perturbations (unknown to Alg. 2) in control targets of actions during training and execution. We used default robot controllers to evaluate the learned policies. We used HARP (Shah and Srivastava 2022) with $\epsilon = 0$ for computing zero-shot option guides.

We used $2-$layered neural networks with 256 neurons in each layer for representing local policies for the learned options. Inputs to these networks were the current configuration of the robot and a vector to the nearest point on the option guide for the current option. We used +1000 as a pseudo reward for reaching the termination set of each option and -100 as a penalty for drifting to a different abstract state. We use SAC (Haarnoja et al. 2018) as a low-level policy learner in lines 9, 16, and 21 of Alg. 2.

Test environments and robots We evaluated our approach across 7 test environments (Fig. 3) (not included in training the critical region predictor), 3 non-holonomic robots (Fig. 3) and a total of 60 navigation and manipulation problems. Dimensions of the environments are as follows: S1, S2: $15m \times 15m$; L1, L2, L3: $75m \times 75m$. Problem specific timeouts were set at 2400s for small environments and manipulation problems and 9000s for larger environments. For each

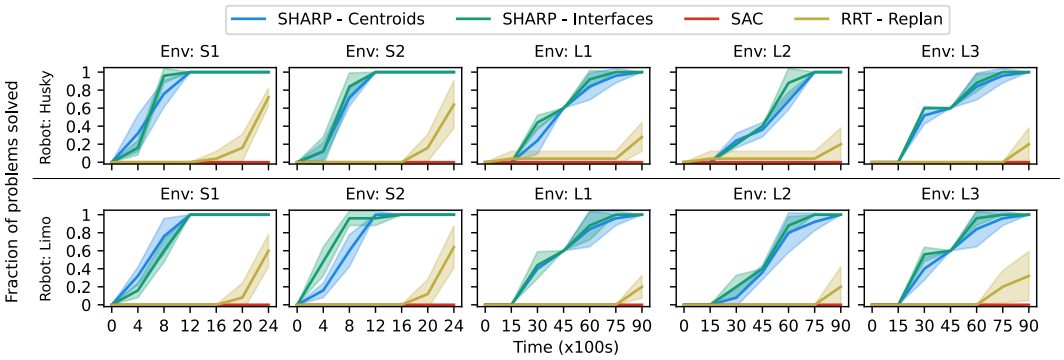

Figure 4: (Higher values are better) Times taken (averaged over 5 trials) by our approach (SHARP) and baselines to compute solutions in the test environments. X-axis shows the time and y-axis shows the fraction of the problems solved in the given time.

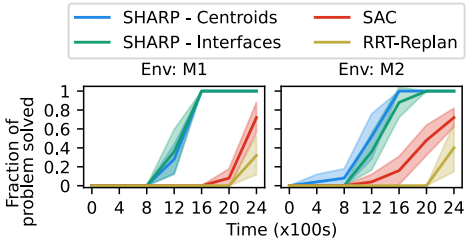

Figure 5: (Contd. from Fig. 4 with same setup) Results for manipulation problems with the Fetch robot.

environment, we generated 5 problem instances by randomly sampling different initial and goal configuration pairs. We used the following robots: the ClearPath Husky (3-DOF), the AgileX Limo (3-DOF), and the Fetch manipulator robot (7-DOF). Details of these robots are presented in the extended version of our paper (Shah and Srivastava 2024).

Baseline selection We considered and evaluated several learning and planning approaches (LaValle 1998; Haarnoja et al. 2018; Kulkarni et al. 2016; Lillicrap et al. 2016; Levy et al. 2019; Bagaria and Konidaris 2020) as potential baselines for this work. Of these, only RRT-Replan (LaValle 1998) and SAC (Haarnoja et al. 2018) solved any problem instances within the timeouts discussed above. Therefore, we compared our approach against SAC and RRT-Replan. SAC is an off-policy deep reinforcement learning approach that learns a single policy for the overall stochastic motion planning problem. We used the same network architecture as ours for SAC's neural policy. We used a terminal reward of $+1000$ and a step reward of -1 to train the SAC agent. RRT-Replan is a version of the popular RRT algorithm that recomputes a plan from the robot's current configuration if the robot fails to successfully reach the goal after executing the initial plan. All approaches considered used the same input robot models, simulators, and low-level controllers as our approach.

5.1 Analysis of Results

Computational Efficiency Figs. 4 and 5 show the fraction of problem instances solved in a given amount of time by both variants of SHARP and the baselines. In our case, this includes the time taken to create the abstract states and actions as well as to compute the solutions. Each subsequent problem uses learned high-level actions (policies and options) from the previous problem instances when available. Results show SHARP shows significantly greater scalability and computational efficiency. In most cases, baselines take $2\times$ the time taken by SHARP to compute a solution. These differences increase for larger environments, where baselines were able to solve less than 50% of the environments that SHARP solved within the same timeouts.

These results illustrate the impact of learning to zero-shot invent and utilize options: even when the time for predicting critical regions, building abstractions, computing high-level plans, and learning low-level policies is included, SHARP significantly outperforms the baselines. Manipulation environments show a relatively smaller difference between performance of all the approaches owing to smaller horizons. This reinforces the key contribution of our approach of creating problems with smaller horizons using options in order to solve problems with significantly large horizons.

Solution quality Fig. 6 shows solution cost and solution reliability (as defined above) for solutions computed by all considered approaches. These results show that SHARP's planning over zero-shot options results in lower cost solutions: they require significantly fewer steps during execution compared to baselines, with the differences frequently spanning *orders of magnitude*. We acknowledge that RRT-Replan is not an optimal planning approach. However, the solution quality also represents the amount of time RRT-Replan had to re-compute and re-execute the solution.

Computing policies that account for stochasticity makes SHARP's solution reliability uniformly above 90%, nearly $3\times$ that of RRT-Replan (the best performing baseline) on the larger test environments. RRT-Replan's solutions had an execution success rate of $\sim 50\%$ in the smaller navigation (S1, S2) and manipulation (M1, M2) environments, and a success rate of less than 33% in the larger environments (L1-L3). SAC's solution reliability was lower, indicating limited scalability of end-to-end learning in long-horizon problems.

Zero-shot option invention and reuse The extended ver-

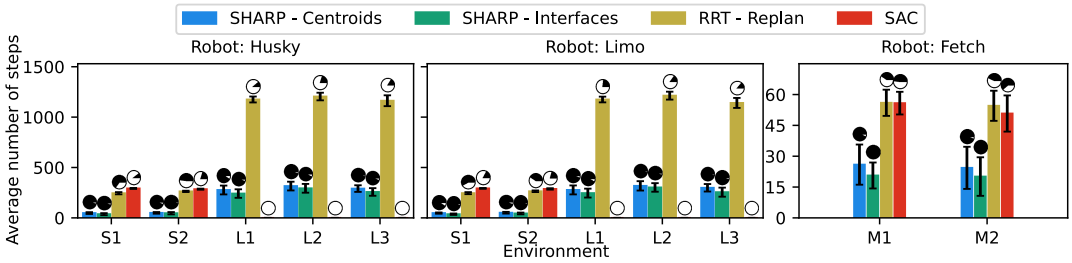

Figure 6: (Smaller bars and darker circles are better) Average number of steps taken in **successful** executions of the learned policies and success rates for our approach and the baselines. The pie chart over each bar represents the success rate (shaded black area) while executing the learned policy.

sion of the paper (Shah and Srivastava 2024) shows the predicted critical regions, 2D projections of the RBVDs, and synthesized option endpoints for our test environments. These results show that our approach is able to zero-short invent options for new, unseen test environments. When new problem instances come from a common environment, our approach is able to transfer these zero-shot options and their policies to new problem instances. Centroid options showed greater reuse rates on average across all environments (52%) than interface options (45%). Details can be found in the extended version of the paper (Shah and Srivastava 2024).

6 Related Work

We discuss the relationship of our work with the most closely related approaches here and present a broader discussion in the extended version of the paper (Shah and Srivastava 2024). To the best of our knowledge, this is the first approach for zero-shot option invention and hierarchical planning and refinement for stochastic robot planning problems that does not require hand-coded abstractions or options as input. In addition, it can be applied to problems and environments not seen during training.

Approaches for stochastic motion planning (Du et al. 2010; Kurniawati, Bandyopadhyay, and Patrikalakis 2012; Vitus, Zhang, and Tomlin 2012; Huynh, Karaman, and Frazzoli 2016; Berg, Patil, and Alterovitz 2017; Hibbard et al. 2022) utilize analytical dynamical models of the robot while this paper addresses problems where such models may not be available.

Another direction of research aims to learn task-specific subgoals in the given test environment (Kulkarni et al. 2016; Bacon, Harb, and Precup 2017; Nachum et al. 2018, 2019; Czechowski et al. 2021). These approaches utilize interactions with the test environments to learn useful subgoals which can then be utilized for learning options and other forms of high-level actions. A related direction of research focuses on learning task-specific options while interacting in the target environment (Stolle and Precup 2002; Şimşek, Wolfe, and Barto 2005; Brunskill and Li 2014; Eysenbach, Salakhutdinov, and Levine 2019; Bagaria and Konidaris 2020; Bagaria, Senthil, and Konidaris 2021). In contrast, this paper focuses on zero-shot options that are created without interacting with the test environments or tasks to improve efficiency and scalability.

Finally, there has been a lot of progress on short-horizon ($\sim$ 5 seconds) dense-reward problems where the robot receives frequent feedback for its actions from the environment. These approaches include conventional control approaches as well as DRL approaches for visual model predictive control (MPC) (Watter et al. 2015; Levine et al. 2016; Finn et al. 2016; Gal, McAllister, and Rasmussen 2016; Henaff, Whitney, and LeCun 2017; Tamar et al. 2017; Kurutach et al. 2018; Ebert et al. 2018; Amos et al. 2018; Hafner et al. 2019). While this paper's focus is on long-horizon sparse-reward planning problems with unknown stochastic dynamics, (visual) MPC techniques can be used for learning low-level policies in conjunction with our approach (Alg. 2, line 22).

7 Conclusion

This paper presents the first approach that uses a data-driven process to learn to create state and action abstractions for unseen environments and problem instances. We provide theoretical results as well as a thorough empirical evaluation for the presented methods. These results show that the presented approach effectively learns to create abstractions that provide strong performance and quality advantages on a broad set of problems that are currently beyond the scope of known methods.

Acknowledgments

We thank Kiran Prasad for helping to implement the initial version of the approach. The work is funded by NSF under the grant IIS 1942856.

References

Amos, B.; Jimenez, I.; Sacks, J.; Boots, B.; and Kolter, J. Z. 2018. Differentiable MPC for end-to-end planning and control. In *Proc. NeurIPS*.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *Proc. AAAI*.

Bagaria, A.; and Konidaris, G. 2020. Option discovery using deep skill chaining. In *Proc. ICLR*.

Bagaria, A.; Senthil, J. K.; and Konidaris, G. 2021. Skill discovery for exploration and planning using deep skill graphs. In *Proc. ICML*.

Berg, J. v. d.; Patil, S.; and Alterovitz, R. 2017. Motion planning under uncertainty using differential dynamic programming in belief space. *Intertational Journal of Robotics Research*, 473–490.

Bertsekas, D. P.; and Tsitsiklis, J. N. 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3): 580–595.

Brunskill, E.; and Li, L. 2014. Pac-inspired option discovery in lifelong reinforcement learning. In *Proc. ICML*.

Czechowski, K.; Odrzygóźdź, T.; Zbysiński, M.; Zawalski, M.; Olejnik, K.; Wu, Y.; Kuciński, Ł.; and Miłoś, P. 2021. Subgoal search for complex reasoning tasks. In *Proc. NeurIPS*.

Du, Y.; Hsu, D.; Kurniawati, H.; Sun, W.; Sylvie, L.; Ong, C.; and Png, S. W. 2010. A POMDP approach to robot motion planning under uncertainty. In *Proc. ICAPS, Workshop on Solving Real-World POMDP Problems*. Citeseer.

Ebert, F.; Finn, C.; Dasari, S.; Xie, A.; Lee, A.; and Levine, S. 2018. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*.

Eysenbach, B.; Salakhutdinov, R. R.; and Levine, S. 2019. Search on the replay buffer: Bridging planning and reinforcement learning. In *Proc. NeurIPS*.

Finn, C.; Tan, X. Y.; Duan, Y.; Darrell, T.; Levine, S.; and Abbeel, P. 2016. Deep spatial autoencoders for visuomotor learning. In *Proc. ICRA*.

Gal, Y.; McAllister, R.; and Rasmussen, C. E. 2016. Improving PILCO with Bayesian neural network dynamics models. In *Data-efficient machine learning workshop, ICML*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. ICML*.

Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; and Davidson, J. 2019. Learning latent dynamics for planning from pixels. In *Proc. ICML*.

Henaff, M.; Whitney, W. F.; and LeCun, Y. 2017. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*.

Hibbard, M.; Vinod, A. P.; Quattrociocchi, J.; and Topcu, U. 2022. Safely: safe stochastic motion planning under constrained sensing via Duality. *arXiv preprint arXiv:2203.02816*.

Huynh, V. A.; Karaman, S.; and Frazzoli, E. 2016. An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research*, 35(4): 305–333.

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

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proc. NeurIPS*.

Kurniawati, H.; Bandyopadhyay, T.; and Patrikalakis, N. M. 2012. Global motion planning under uncertain motion, sensing, and environment map. *Autonomous Robots*, 33(3): 255–272.

Kurutach, T.; Tamar, A.; Yang, G.; Russell, S. J.; and Abbeel, P. 2018. Learning plannable representations with causal InfoGANs. In *Proc. NeruIPS*.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Technical Report 9811, Iowa State University.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1): 1334–1373.

Levy, A.; Konidaris, G.; Platt, R.; and Saenko, K. 2019. Learning multi-level hierarchies with hindsight. In *Proc. ICLR*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *Proc. ICLR*.

Nachum, O.; Gu, S.; Lee, H.; and Levine, S. 2019. Near-optimal representation learning for hierarchical reinforcement learning. In *Proc. ICLR*.

Nachum, O.; Gu, S. S.; Lee, H.; and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. In *Proc. NeurIPS*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proc. NeurIPS*.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: convolutional networks for biomedical image segmentation. In *Proc. MICCAI*.

Shah, N.; and Srivastava, S. 2022. Using deep learning to bootstrap abstractions for hierarchical robot planning. In *Proc. AAMAS*.

Shah, N.; and Srivastava, S. 2024. Hierarchical Planning and Learning for Robots in Stochastic Settings Using Zero-Shot Option Invention (Extended Version). In *Proc. AAAI*.

Şimşek, Ö.; Wolfe, A. P.; and Barto, A. G. 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proc. ICML*.

Stolle, M.; and Precup, D. 2002. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, 212–223. Springer.

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

Tamar, A.; Thomas, G.; Zhang, T.; Levine, S.; and Abbeel, P. 2017. Learning from the hindsight plan—episodic MPC improvement. In *Proc. ICRA*.

Vitus, M. P.; Zhang, W.; and Tomlin, C. J. 2012. A hierarchical method for stochastic motion planning in uncertain environments. In *Proc. IROS*.

Watter, M.; Springenberg, J.; Boedecker, J.; and Riedmiller, M. 2015. Embed to control: A locally linear latent dynamics model for control from raw images. In *Proc. NeurIPS*.