# MDP Abstraction with Successor Features

**Dongge Han,**[1] **Michael Wooldridge,** [1] **Sebastian Tschiatschek** [2]

[1] Department of Computer Science, University of Oxford, [2] Department of Computer Science, University of Vienna,
dongge.han@cs.ox.ac.uk, michael.wooldridge@cs.ox.ac.uk, sebastian.tschiatschek@univie.ac.at

**Abstract**

Abstraction plays an important role for generalisation of knowledge and skills, and is key to sample efficient learning and planning. For many complex, long-horizon planning problems such as the game Minecraft, abstraction can be particular useful. To solve these tasks, an abstract plan can be first formed, then instantiated by filling in the necessary low-level details, and such abstract plans can often generalize well to related new problem settings. In this work, we study temporal and state abstraction in reinforcement learning, where temporal abstractions represent temporally-extended actions in the form of options, while state abstraction induces abstract MDPs by aggregating similar states as abstract states. Many existing abstraction schemes overlook the relation between state and temporal abstraction, consequently, acquired option policies often cannot be directly transferred to new environments due to changes in the state space and transition dynamics. To address these issues, we propose successor abstraction, a novel abstraction scheme building on successor features. This includes an algorithm for encoding and instantiation of abstract options across different environments, and a state abstraction mechanism based on the abstract options. Our abstraction scheme allows us to create abstract environment models with semantics that are transferable across different environments through encoding and instantiation of abstract options. Empirically, we achieve better transfer and improved performance on a set of benchmark tasks compared to state of the art baselines.

## 1 Introduction

While reinforcement learning (RL) has recently shown many remarkable successes, e.g., in playing Atari and Go (Mnih et al. 2015; Silver et al. 2017), its large sample complexity is still a key problem limiting its applications. Allowing RL agents to learn transferable and reusable options (Sutton, Precup, and Singh 1999) (i.e., skills) is a promising approach to alleviate the issue of sample complexity. As such, an important problem is to characterise option policies by abstract options that can be transferred and instantiated across different environments. Moreover, planning with abstract options can be particularly useful for many complex, long-horizon planning problems such as the game of Minecraft.

Figure 1 illustrates an example scene in Minecraft. In order to obtain a baked potato, the agent can first formulate a

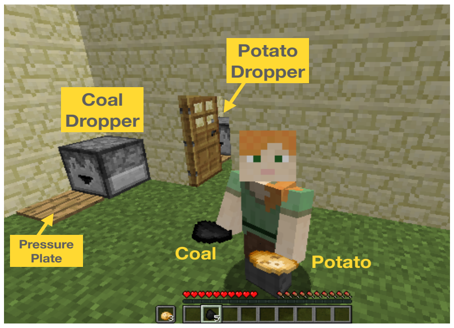


Figure 1: Example Scene in the Game Minecraft

simple, abstract plan to collect coal, then collect a potato, and finally perform crafting. Then the agent can instantiate the options by navigating through different rooms to collect the required items. Having acquired the abstract options in a first environment, the agent can also transfer the abstract options and grounds them in an previously unseen environment, then form an abstract semi-Markov Decision Process (SMDP) for planning using the transferred options, c.f. Figure 2 for an example.

In this work, we propose abstract option representations that can be: (1) shared across different environments, (2) grounded in unseen environments with a certain precision, and (3) used for planning with near-optimal performance. In the context of RL, however, options are described by policies which are typically not transferable due to new state spaces and transition dynamics. This issue is underlined by the fact that abstract options such as *"open a door"* can often correspond to different policies in different MDPs.

To enable transferable abstract options, we define shared *features* across different environments (e.g., doors), and abstract options as successor features (Dayan 1993; Barreto et al. 2016), which can effectively capture the feature expectations of the option trajectories.

One way of grounding such an abstract option could be through finding an option policy which maximises a preference function over the successor features (Barreto et al. 2019). However, such rewards may yield imprecise option behaviours due to the interference between the different components of the feature, hence the preference function needs to be carefully hand-crafted in unseen environments in order to produce an intended option behaviour (e.g., to acquire all required feature components before terminating). To address these issues, we formulate option grounding as a feature-matching problem where the feature expectations of the learned option match the abstract option. More specifically, we adapt techniques from inverse reinforcement learning (IRL), and present an algorithm which finds ground options efficiently with a specified precision. Finally, we demonstrate that the abstract options can be used for SMDP planning with near-optimal performance.

This paper is structured as follows: in Section 2 we introduce the background, we define the abstract option representations in Section 3.1 and the grounding algorithms in Section 3.2. In Section 4, we introduce our SMDP abstraction framework which produces an abstract SMDP model for planning. Our empirical results are presented in Section 5, a discussion with related work in Section 6, and we conclude our paper in Section 7.

## 2 Background

**Markov Decision Processes.** We model an agent's decision making process using a Markov decision process. An MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$ is a tuple, where $\mathcal{S}$ is a set of states the environment the agent is interacting with can be in, $\mathcal{A}$ is a set of actions the agent can take, $P\colon \mathcal{S} \times \mathcal{A} \to [0,1]^{|\mathcal{S}|}$ is a transition kernel describing transitions between states of the MDP upon taking actions, $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma$ is a discount factor. In the remainder of the paper, we use $S_t$ and $A_t$ for random variables denoting the state of the environment and the action chosen by the agent at time $t$, and $s_t$, $a_t$ and $r_t$ to refer to specific instantiations of the state, action and reward at time $t$.

**Successor Features.** The *successor representation* (Dayan 1993) of a state $s_0$ is defined as the expected discounted future occupancy of all state-action pairs when starting from state $s_0$ and following policy $\pi$, i.e., $\forall s \in \mathcal{S}, a \in \mathcal{A}$, $\mu^{\pi}_{s_0}(s,a) = \mathbb{E}_{\mathcal{M},\pi}[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{S_t=s,A_t=a} | S_0 = s_0]$, where $\mathbb{1}_x$ is the indicator function which is 1 if $x$ is true and 0 otherwise. To generalize the successor representation and allow for better transferrability, Barreto et al. (2016) defined the *successor feature* $\boldsymbol{\psi}^{\pi}_{s_0}$ of a state $s_0$ as the expected discounted sum of features of state-action pairs encountered when following policy $\pi$ starting from $s_0$, i.e., $\boldsymbol{\psi}^{\pi}_{s_0} = \mathbb{E}_{\mathcal{M},\pi}[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\theta}(S_t, A_t) \mid S_0 = s_0]$,where $\boldsymbol{\theta}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ denotes the state-action features.

**Options Framework.** The options framework (Sutton, Precup, and Singh 1999) is one of the most common hierarchical RL frameworks. An option $o$ is defined as a tuple $\langle I^o, \pi^o, \beta^o \rangle$, where $I^o \subseteq \mathcal{S}$ is the initiation set, i.e., the set of states an option can be started in, $\pi^o\colon \mathcal{S} \to [0,1]^{|\mathcal{A}|}$ is the option policy, and $\beta^o\colon \mathcal{S} \to [0,1]$ defines the termination condition, i.e., the probability that the option terminates in state $s \in \mathcal{S}$. The transition dynamics $P^o_{s,s'}$ and reward $r^o_s$ of an option $o$ are

$$P^o_{s,s'} = \sum_{k=1}^{\infty} P(s,o,s',k)\gamma^k,$$
$$r^o_s = \mathbb{E}[r_{t+1} + \ldots + \gamma^k r_{t+k} \mid \mathcal{E}(o,s,t)],$$

where $P(s,o,s',k)$ is the probability of transiting from $s$ to $s'$ in $k$ steps and terminating in $s'$, and $\mathcal{E}(o,s,t)$ is the event that option $o$ is initiated at time $t$ in state $s$, and $k$ is the duration of the option.

A semi-Markov decision process (SMDP) is an MDP with a set of options, i.e., $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, P, r, \gamma \rangle$, where $P\colon \mathcal{S} \times \mathcal{O} \to [0,1]^{|S|}$ are the option's transition dynamics, and $r\colon \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ is the option's reward function. A family of variable-reward SMDPs (Mehta et al. 2008) (which we will denote as $\boldsymbol{\psi}$-SMDP) is defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, P, \boldsymbol{\psi}, \gamma \rangle$, where $\boldsymbol{\psi}^o_s$ defines the feature (successor feature) of option $o$ starting at state $s$. A $\boldsymbol{\psi}$-SMDP induces an SMDP if the reward is linear in the features, i.e., there is a reward vector $w_r$ such that $r^o_s = w_r^T \boldsymbol{\psi}^o_s$.

**Inverse Reinforcement Learning (IRL).** IRL is an approach for learning from demonstrations (Arora and Doshi 2018). In IRL, the reward function is assumed to be unknown and the goal of the learner is to infer the expert's objective from its demonstrated behaviour (Ng, Russell et al. 2000; Abbeel and Ng 2004). A common assumption is that the rewards are linear in some features, i.e., $r^a_s = w_r^T \boldsymbol{\theta}(s,a)$, where $w_r \in \mathbb{R}^d$ is a real-valued weight vector specifying the reward of observing the different features. Based on this assumption, Abbeel and Ng (2004) observed that for a policy to perform as well as the expert's policy, it suffices that their feature expectations match. Therefore, IRL has been widely posed as a feature-matching problem (Abbeel and Ng 2004; Syed, Bowling, and Schapire 2008; Ziebart et al. 2008; Ho and Ermon 2016), where the learner tries to match the feature expectation of the expert.

## 3 Options as Successor Features

Our goal is to define abstract options which can be transferred and reused among different MDPs $\mathcal{M}_1, \ldots, \mathcal{M}_k$. To this end we need a common representation for shared features among the MDPs. Therefore, we define a *feature function* which maps state-action pairs for each MDP to a shared feature space, i.e., $\boldsymbol{\theta}_{\mathcal{M}_i}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$. Such features are commonly adopted by prior works (Barreto et al. 2016; Syed, Bowling, and Schapire 2008; Abbeel and Ng 2004), and can be often obtained through feature extractors such as an object detector (Girshick 2015). Based on the shared feature space, we introduce abstract options which are transferable among different MDPs.

### 3.1 Abstract Successor Options

We propose the use of *abstract successor options* represented by successor features. The underlying idea is to describe an option by a sketch characterized by a cumulative feature expectation that it should realise. We now define abstract successor options and their corresponding ground options:

**Definition 3.1** (Successor Feature of Options). *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$ be an MDP and $\boldsymbol{\theta}_{\mathcal{M}}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ be the feature function. The successor feature of an option $o$ when starting from state $s \in \mathcal{S}$ is given by $\boldsymbol{\psi}^o_s = \mathbb{E}[\sum_{\kappa=0}^{k} \gamma^{\kappa} \boldsymbol{\theta}_{\mathcal{M}}(S_{t+\kappa}, A_{t+\kappa}) | \mathcal{E}(o,s,t)]$, where $\mathcal{E}(o,s,t)$ is the event that option $o$ is initiated at time $t$ in state $s$, $k$ is the duration of the option.*

Algorithm 1: Option Grounding (IRL-naive)

```
Input: $\mathcal{M}$=$\langle\mathcal{S},\mathcal{A},P,r,\gamma\rangle$, abstract option $\boldsymbol{\psi}^{\bar{o}}$, $\epsilon_{\text{thresh.}}$
Output: initiation set $I^{\bar{o}}$, dict. of ground option policies $\Pi^{\bar{o}}$, dict.
    of termination probabilities $\Xi^{\bar{o}}$
 1: // Construct augmented MDP
 2: $\mathcal{S}' \leftarrow \mathcal{S} \cup \{s_{\text{null}}\}, \mathcal{A}' \leftarrow \mathcal{A} \cup \{a_{\text{T}}\}, P' \leftarrow P$
 3: $\forall s \in \mathcal{S}'$: $P'(s_{\text{null}}|s, a_{\text{T}}) = 1$
 4: $\forall a \in \mathcal{A}'$: $P'(s_{\text{null}} \mid s_{\text{null}}, a) = 1$
 5: // Find ground options (for IRL see Algo. 3)
 6: for all $s_{\text{start}} \in \mathcal{S}$, do
 7:     $\epsilon, \pi^o_{s_{\text{start}}} \leftarrow \text{IRL}(\mathcal{S}', \mathcal{A}', P', \gamma, s_{\text{start}}, \boldsymbol{\psi}^{\bar{o}})$
 8:     if $\epsilon \leq \epsilon_{\text{thresh.}}$ then
 9:         $I^{\bar{o}} \leftarrow I^{\bar{o}} \cup \{s_{\text{start}}\}$
10:         $\Pi^{\bar{o}}(s_{\text{start}}) = \pi^o_{s_{\text{start}}}$
11:         $\Xi^{\bar{o}}(s_{\text{start}}) = \beta^o$, where $\beta^o(s) = \pi^o_{s_{\text{start}}}(a_{\text{T}}|s)$
12:     end if
13: end for
14: return $I^{\bar{o}}, \Pi^{\bar{o}}, \Xi^{\bar{o}}$
```

**Definition 3.2** (Abstract Successor Options)**.** *An abstract successor option $\bar{o}$ is defined by a feature vector $\boldsymbol{\psi}^{\bar{o}} \in \mathbb{R}^d$. Let $g_s : \mathcal{O} \to \mathbb{R}^d$ denote a state-dependent mapping from ground options to abstract options s.t. a ground option starting from state $s$ maps to an abstract successor option $\bar{o}$ iff its induced successor feature matches $\boldsymbol{\psi}^{\bar{o}}$, i.e., $g_s(o) = \bar{o} \iff \boldsymbol{\psi}^o_s = \boldsymbol{\psi}^{\bar{o}}$.*

Let $\mathcal{M} = \langle\mathcal{S},\mathcal{O},P,r,\gamma\rangle$ be an SMDP, we denote $g_s^{-1}(\bar{o})$ as the ground options induced by $\bar{o}$, and denote the initiation set as the set of states from which the abstract option can be realised, i.e., $I^{\bar{o}} := \{s \in \mathcal{S} \mid \exists o \colon g_s(o) = \boldsymbol{\psi}^{\bar{o}}\}$.

## 3.2 Grounding Abstract Successor Options

We model abstract option grounding as a feature-matching problem, i.e., finding a policy which induces a feature expectation equal to the abstract option's feature vector. This problem, although with a different aim, has been extensively studied in the inverse reinforcement learning (IRL) literature, where a learner aims to match the expected discounted cumulative feature values of an expert (Syed, Bowling, and Schapire 2008; Abbeel and Ng 2004).

**IRL-naive (Algorithm 1)** In Algorithm 1 we present a naive algorithm for grounding an abstract option $\bar{o}$. The algorithm uses feature matching based IRL to find for each possible starting state $s_{\text{start}}$ an option realizing the feature vector $\boldsymbol{\psi}^{\bar{o}}$, if possible. To use IRL algorithms, we need to create an augmented MDP which enables termination of options: the termination condition can be modeled by augmenting the action space with a terminate action $a_{\text{T}}$, and appending a null state $s_{\text{null}}$ to the state space. Specifically, $a_{\text{T}}$ will take the agent from any "regular" state to the null state. Taking any action in the null state will lead to itself, and yield a zero feature vector, i.e., $\boldsymbol{\theta}(s_{\text{null}}, \cdot) = \mathbf{0}$.

With this augmented MDP, an IRL algorithm can compute an option policy. In particular, for the experiments in this paper, we adapt the linear programming IRL approach by (Syed, Bowling, and Schapire 2008) ( Algorithm 3). It first finds the discounted state-action visitation frequencies for all state-action pairs that together match the abstract successor option,

Algorithm 2: IRL (used by Algorithm 1 IRL-naive)

**Input:** Augmented MDP $\mathcal{M} = \langle\mathcal{S}',\mathcal{A}',P',r,\gamma\rangle$, starting state $s_{\text{start}}$, state-action features $\boldsymbol{\theta} \colon \mathcal{S}' \times \mathcal{A}' \to \mathbb{R}^d$, abstract option $\boldsymbol{\psi}^{\bar{o}} \in \mathbb{R}^d$.

**Output:** ground and abstract option feature difference $\epsilon = \sum_{k=1}^d \epsilon_k$, option policy $\pi^o_{s_{\text{start}}}$

Solve LP for visitation frequencies $\mu_{s,a}$ and feature matching errors $\epsilon_k$

$$\begin{aligned}
\min_{\mu,\epsilon_k} \quad & \sum_k \epsilon_k - \lambda_1 \sum_a \mu_{s_{\text{null}},a} + \lambda_2 \sum_s \mathbb{1}_{\mu_{s,a_{\text{T}}}>0} \\
\text{s.t.} \quad & \sum_a \mu_{s,a} = \mathbb{1}_{s=s_{\text{start}}} + \gamma \sum_{s',a} P'(s|s',a)\mu_{s',a} \qquad \forall s \in \mathcal{S}' \\
& \sum_{s,a} \mu_{s,a}\boldsymbol{\theta}_k(s,a) - \boldsymbol{\psi}^{\bar{o}}_k \leq \epsilon_k \qquad \forall k \in 1,\ldots,d \\
& \sum_{s,a} \mu_{s,a}\boldsymbol{\theta}_k(s,a) - \boldsymbol{\psi}^{\bar{o}}_k \geq -\epsilon_k \qquad \forall k \in 1,\ldots,d \\
& \mu_{s,a} \geq 0 \quad \forall s \in \mathcal{S}', a \in \mathcal{A}'
\end{aligned}$$

Compute option policy $\pi^o_{s_{\text{start}}}(a|s) = \frac{\mu_{s,a}}{\sum_a \mu_{s,a}}$

then the option policy $\pi^o$ (including termination of the option) can be deduced from the visitation frequencies. Finally, if the discrepancy $\epsilon$ of the successor feature of the learned option and the abstract option is below a certain threshold $\epsilon_{\text{thresh.}}$, the start state $s_{\text{start}}$ will be added to the initiation set.

**IRL-batch** IRL-naive needs to solve an IRL problem for each state in $\mathcal{S}$ which is computationally demanding. To improve the efficiency of grounding, we propose a batch learning algorithm which performs IRL for starting states in batches. The main challenge is that the state-action visitation frequencies found by the LP for matching the option feature may be a mixture of different option policies from different starting states. To solve this issue, we introduce IRL-batch, a recursive procedure with a batched IRL component which effectively regularises the learned option policy. This algorithm can significantly reduce the number of IRL problems that need to be solved while preserving near-optimal performance, cf. Table 1. The pseudocode is omitted due to space constraints.

**Transfer using abstract successor options.** Since features are shared across MDPs, an existing option from a source environment can be transferred to a target environment through first finding the abstract successor option $\bar{o} = g_s(o)$ by computing the ground option's successor feature, then grounding $\bar{o}$ in the target environment $o' = g_s^{-1}(\bar{o})$ using the IRL-naive or IRL-batch algorithm.

**Option grounding in unknown environments.** When the transition dynamics are unknown, exploration is needed before option grounding to construct the (approximate) MDP transition graph, e.g., through random walk. However, random walks can often get trapped in local communities (Pons and Latapy 2005) and thus can be inefficient for exploration. On the other hand, unlike solving a long-horizon task with sparse rewards, options are often relatively localised, and hence can be readily found with partially constructed transition graphs. This enables simultaneous exploration and option grounding: given a set of abstract options $\bar{\mathcal{O}}$, we start with an episode of random walk which constructs an approximate MDP $\tilde{\mathcal{M}}_0$. In each subsequent episode $k$, we ground the abstract options using $\tilde{\mathcal{M}}_{k-1}$, and update the MDP $\tilde{\mathcal{M}}_k$ by exploring with a random walk using both primitive actions and the computed ground options. We show empirically that, using our approach, ground options can be learned quickly and significantly improve the exploration efficiency, cf. Section 5.2.

## 4 SMDP Abstraction and Planning with Abstract Successor Options

MDP abstraction aims to induce an abstract MDP by grouping similar states as an abstract state, with the goal of reducing the complexity for planning and exploration while ensuring close to optimal performance w.r.t. to the original MDP. The framework was first introduced by Dean and Givan (1997) through stochastic bisimulation. Li, Walsh, and Littman (2006) provided a unifying theory of abstraction, and Abel, Hershkowitz, and Littman (2016) formulated their approximate counterparts.

### 4.1 Successor Homomorphism for SMDP Abstraction

Most prior state-abstraction methods do not carry transferable temporal semantics, and are not reuseable across tasks (i.e., for different reward functions). To address these issues, we propose successor homomorphisms, which induce abstract SMDPs with near-optimal performance for planning with abstract successor options, which are reuseable across tasks. For this purpose, we adopt the formulation of variable-reward SMDPs (cf. Section 2). A $\boldsymbol{\psi}$-SMDP defines a family of SMDPs with shared transition dynamics and features, where each reward weight $w_r$ on the features induces an SMDP.

**Definition 4.1** ($\epsilon$-Approximate Successor Homomorphism)**.** *A mapping $h = (f(s), g_s(o), w_s)$ from a $\boldsymbol{\psi}$-SMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, P, \boldsymbol{\psi}, \gamma \rangle$ to $\boldsymbol{\psi}$-SMDP $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{O}}, \bar{P}, \bar{\boldsymbol{\psi}}, \gamma \rangle$, where (1) $f\colon \mathcal{S} \to \bar{\mathcal{S}}$ is a state mapping function, (2) $g_s\colon \mathcal{O} \to \bar{\mathcal{O}}$ is a state-dependent option mapping, and (3) a weight function $w\colon \mathcal{S} \to [0,1]$ over the ground states such that $\forall \bar{s} \in \bar{\mathcal{S}}, \sum_{s \in f^{-1}(\bar{s})} w_s = 1$, is an* $\epsilon$-approximate successor homomorphism *if for $\epsilon_P, \epsilon_\psi > 0$, $\forall s_1, s_2 \in \mathcal{S}, o_1, o_2 \in \mathcal{O}$, $h(s_1, o_1) = h(s_2, o_2) \implies \forall s' \in \mathcal{S}, \sum_{s'' \in f^{-1}(f(s'))} [P^{o_1}_{s_1,s''} - P^{o_2}_{s_2,s''}]| \leq \epsilon_P$,* and *$|\boldsymbol{\psi}^{o_1}_{s_1} - \boldsymbol{\psi}^{o_2}_{s_2}| \leq \epsilon_\psi$. The transition dynamics and features of $\bar{\mathcal{M}}$ are:*
$$\bar{P}^{\bar{o}}_{\bar{s},\bar{s}'} = \sum_{s \in f^{-1}(\bar{s})} w_s \sum_{s' \in f^{-1}(\bar{s}')} P^{g_s^{-1}(\bar{o})}_{s,s'}, \text{ and } \bar{\boldsymbol{\psi}}^{\bar{o}}_{\bar{s}} = \sum_{s \in f^{-1}(\bar{s})} w_s \boldsymbol{\psi}^{g_s^{-1}(\bar{o})}_{s}.$$

Note that $\sum_{s'' \in f^{-1}(f(s'))} P^{o}_{s,s''}$ refers to the transition probability from ground state $s$ to an abstract state $\bar{s}' = f(s')$ following option $o$. Intuitively, two states mapping to the same abstract state have approximately equal option transition dynamics towards all *abstract* states, and the corresponding ground options induce similar successor features. For efficient computation of the abstract model, the transition dynamics condition can be alternatively defined on the ground states s.t. $h(s_1, o_1) = h(s_2, o_2) \implies \forall s' \in S$,

$$|P^{o_1}_{s_1,s'} - P^{o_2}_{s_2,s'}| \leq \epsilon_P, \text{ and } \quad |\boldsymbol{\psi}^{o_1}_{s_1} - \boldsymbol{\psi}^{o_2}_{s_2}| \leq \epsilon_\psi, \qquad (1)$$

which states that two states mapping to the same abstract state have approximately equal option transition dynamics towards all *ground* states, and the corresponding ground options induce similar successor features. In cases in which multiple ground options map to the same abstract option, $g_s^{-1}(\bar{o})$ picks one of the ground options, e.g., with shortest duration, maximum entropy (Ziebart et al. 2008), etc.

Figure 2 shows an abstract $\boldsymbol{\psi}$-SMDP which is induced by our approximate successor homomorphism in an Object-Rooms environment.

### 4.2 Properties of Successor Homomorphisms

Our successor homomorphism has the following appealing properties:

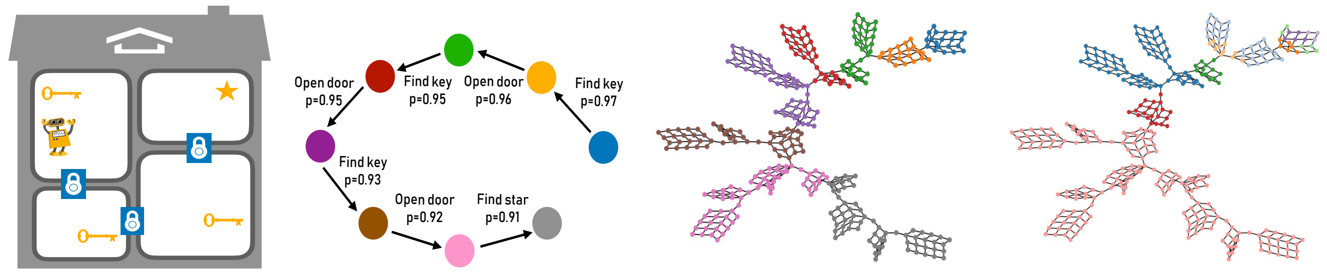
1. *Semantically-meaningful abstraction:* In our formulation, given a set of temporal semantics (i.e., abstract successor options, e.g., from encoding existing options), ground options which satisfy the successor homomorphism can be directly computed through grounding the abstract options. Therefore, the induced abstract $\boldsymbol{\psi}$-SMDP carry meaningful temporal semantics, and can be interpreted based on the features. Moreover, as the abstract options are transferable across MDPs, they can be used to construct abstract $\boldsymbol{\psi}$-SMDPs with shared semantics in different environments. Different from prior abstraction formulations (Def. A.2) which are reward-based, our feature-based successor homomorphism produces abstract models with meaningful temporal semantics, and is robust under task changes. Furthermore, we include a generic formulation of feature-based abstraction (cf. Def. A.1) as a basis for potential feature-based abstractions other than successor homomorphism.

2. *Performance guarantees across tasks:* Our induced abstract $\boldsymbol{\psi}$-SMDPs define a family of abstract SMDPs with variable reward weights. Given any arbitrary task defined by a reward weight $w_r$ on the features, the induced abstract SMDP can be classified as a model-irrelevance abstraction (Li, Walsh, and Littman 2006). Extending results from (Abel, Hershkowitz, and Littman 2016; Abel et al. 2020) to our setting, we can guarantee performance of the abstract model.

**Theorem 4.1.** *Let $w_r\colon \mathbb{R}^d \to \mathbb{R}$ be a linear reward vector such that $r^a_s = w_r^T \boldsymbol{\theta}(s,a)$. Under this reward function, the value of an optimal abstract policy obtained through the $\epsilon$-approximate successor homomorphism is close to optimal ground SMDP policy, where the difference is bounded by $\frac{2\kappa}{(1-\gamma)^2}$, where $\kappa = |w_r|(2\epsilon_\psi + \frac{\epsilon_P |\bar{\mathcal{S}}| \max_{s,a} |\boldsymbol{\theta}(s,a)|}{1-\gamma})$.*

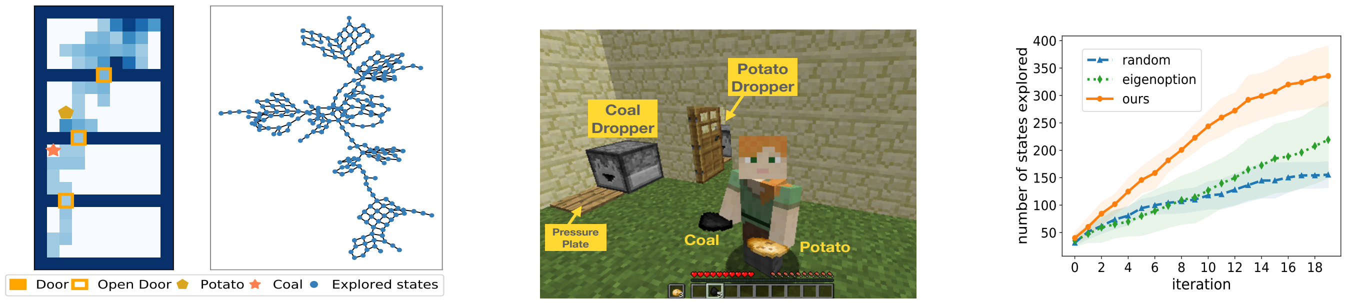
## 5 Experiments

We empirically evaluate our proposed abstraction scheme on three tasks: 1. Transferring and grounding abstract options in unseen environments. 2. Grounding abstract options in new environments with unknown transition dynamics, and

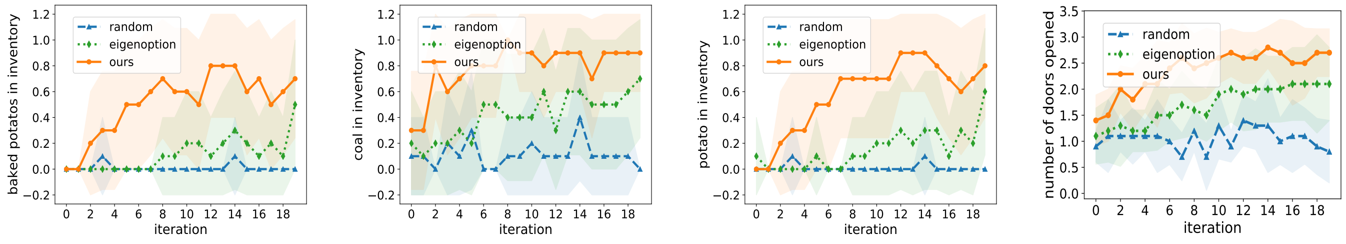


(a) Object-Rooms (b) Abstract $\psi$-SMDP (c) Successor Homomorphism (d) $Q$ (all)-Abstraction

Figure 2: MDP Abstraction in the Object-Rooms domain. (b) Abstract $\psi$-SMDP induced by our successor homomorphism from the ground MDP as shown in (c), the abstract states in (b) correspond to aggregated ground states of the same colour in (c). (d) Abstraction induced by approximate $Q^*$-irrelevance abstraction (cf. Sec. A.4) for the task *find key*; the abstraction does not carry temporal semantics, and is not reusable for other tasks e.g., *find star*. More details can be found in the experiments section.



(a) State visitation frequencies (b) Setting (c) States explored



(d) Baked potato obtained (e) Coal obtained (f) Potato obtained (g) Doors opened

Figure 3: Exploring and grounding options in unknown environments in the Minecraft setting.

3. SMDP abstraction through successor homomorphism. Because of space limits, some details of the experiments are omitted below.

## 5.1 Options Transfer

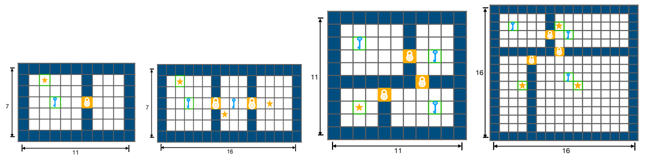
We evaluate the performance and efficiency of our option grounding algorithm for transferring options given by expert demonstrations to new environments.

**Object-Rooms Setting.** $N$ rooms are connected by doors with keys and stars inside. There are 6 actions: $\mathcal{A} = \{$Up, Down, Left, Right, Pick up, Open$\}$. The agent can pick up the keys and stars, and use the keys to open doors. See Figure 6 for an illustration.

**Training and Results:** • *ours: IRL-naive (Algorithm 1) and IRL-batch (Algorithm 4)*. Both of our algorithms (IRL-naive and IRL-batch) *transfer* an expert demonstration from a 2-Room source environment to a target environment of 2-4 Rooms, by first encoding the expert demonstration as an abstract option, then grounding the abstract option in the target environment.

• *Option Keyboard (OK)* (Barreto et al. 2019): OK first trains primitive options (find-key, open-door, find-star) to maximise a cumulant (pseudo-reward on the history). The cumulants are hand-crafted in the target environment, where the agent is rewarded when terminating the option after achieving the goal. Then by putting preference weights over the cumulants, the composite options (1. find-key, open-door, 2. find-key, open-door, find star) are synthesized via generalized policy improvement (GPI) to maximise the weighted sum of cumulants.

Table 1 shows the success rate of the computed options for achieving all required goals across all starting states in the initiation sets. Both IRL-naive and IRL-batch successfully transfer the options and achieve close to optimal feature-matching. The ok composed options have low success rate for achieving the required goals due to the learned imprecise



(a) 2 Rooms, 157 states (b) 3 Rooms, 487 states (c) 4 Rooms (small), 348 states (d) 4 Rooms (large), 1463 states

Figure 4: Object-room layouts used in our experiments. In each object room instance, blue grids are the walls, orange grids are doors, and there are two types of objects: stars and keys, which can be picked-up by the agent. The agent can pick up a key and use it to open a door.

| | | 2 Rooms | | 3 Rooms | | 4 Rooms | |
|---|---|---|---|---|---|---|---|
| | | success | *LP* | success | *LP* | success | *LP* |
| Find Key | *IRL-naive* | 1.0 | 157 | 1.0 | 487 | 1.0 | 1463 |
| | *IRL-batch* | 1.0 | **2** | 1.0 | **2** | 1.0 | **2** |
| | *ok* (Barreto et al. 2019) | 1.0 | - | 1.0 | - | 1.0 | - |
| Find Star | *IRL-naive* | 1.0 | 157 | 1.0 | 487 | 1.0 | 1463 |
| | *IRL-batch* | 1.0 | **2** | 1.0 | **2** | 1.0 | **2** |
| | *ok* (Barreto et al. 2019) | 1.0 | - | 1.0 | - | 1.0 | - |
| Find Key, Open Door | *IRL-naive* | 1.0 | 157 | 1.0 | 487 | **1.0** | 1463 |
| | *IRL-batch* | **1.0** | **3** | **1.0** | **15** | 0.95 | **29** |
| | *ok* (Barreto et al. 2019) | 0.08 | - | 0.56 | - | 0.71 | - |
| Find Key, Open Door, Find Star | *IRL-naive* | 1.0 | 157 | 1.0 | 487 | 1.0 | 1463 |
| | *IRL-batch* | **1.0** | **4** | **1.0** | **16** | **1.0** | **7** |
| | *ok* (Barreto et al. 2019) | 0.0 | - | 0.51 | - | 0.66 | - |

Table 1: Performance and efficiency of option grounding in the Object Rooms domain. We show the success rate of the learned options for achieving all specified goals across all starting states in the initiation set, and the number of LP's required to find the option policy.

behaviours, e.g., for grounding "find key, open door, find star" in the 3-Room setting, the learned options often terminates after achieving 1-2 subgoals. Compared with IRL-naive, IRL-batch significantly reduces the number of LP's solved by learning shared solutions among the start states.

## 5.2 Grounding Abstract Options in Unknown Environments

We test our abstract option grounding algorithm in environments with unknown transition dynamics, where the agent simultaneously explores and computes ground options.

**Minecraft environment:** we use the standard AI research platform Malmo (Johnson et al. 2016), mimicking the case of a human-like robotic agent navigating in a room and interacting with domestic objects.

• *Bake-Rooms* (Figure 3): Agents can collect coal, potato, and craft baked potatoes. 4 rooms (R1-R4 from bottom to top) are connected by doors. The agent starts in R1. To obtain a baked potato, the agent needs to open doors, collect coal from a coal dropper in R2, collect potato from a dropper in R3, and issue a *craft* command to bake the potato. Agents observe their current location, objects in nearby $3 \times 3$ grids (e.g. door), and the items in inventory (e.g. potato, coal).

**Training and results.** We compare our algorithm *IRL-batch* with the following two baselines: (i) *random walk* and (ii) *eigenoptions* (Machado et al. 2017). Each agent runs for 20 iterations, with 200 steps per iteration as follows: In the first iteration, all agents execute randomly chosen actions. After each iteration, the agents construct an MDP graph based on collected transitions from all prior iterations. The eigenoption agent computes $k = 3$ eigenoptions of smallest eigenvalues using the normalized graph Laplacian, while *IRL-batch* grounds the $k = 3$ abstract options: 1. open and go to door 2. collect coal and 3. collect potato. In the next iteration, agents perform a random walk with both primitive actions and the acquired options, update the MDP graph, compute new options, and so on.

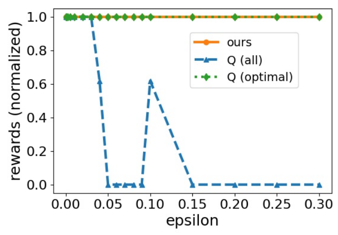
Figure 3 (a) shows the state visitation frequency of our algorithm in the 14th iteration and a constructed transition graph. The trajectory shows that from R1 (bottom room), the agent learns to open door, collect coal in R2, open door, collect potato in R3, and navigate around the rooms. (c) shows that our algorithm explores on average more than 50 percent states than both baselines in 20 iterations. (d) - (g) shows the objects collected by the agents (max count is 1) and the number of doors opened. The agent using our algorithm learns to open door and collect coal within around 2 iterations, and it learns to bake a potato 50 percent of the time within 5 iterations. The results are averaged over 10 seeds and shaded regions represent standard deviations.

## 5.3 SMDP Abstraction through Abstract Successor Options

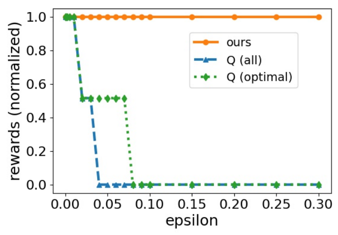
We present results on abstract SMDPs found using successor homomorphism, and the performance of planning using the abstract SMDP. We use 3-room and 4-room variants of our *Object-Rooms* environment, and compare the abstract SMDP models produced by our successor homomorphism with two MDP abstraction methods which induces near-optimal abstract policies (Li, Walsh, and Littman 2006; Abel, Hershkowitz, and Littman 2016):

*1. Q(all)*: ($Q^*$-irrelevance abstraction), if $f(s_1) = f(s_2)$, then $\forall a$, $|Q^*(s_1, a) - Q^*(s_2, a)| \leq \epsilon$, where $Q^*(s, a)$ is the optimal Q-function for the considered MDP.
*2. Q(optimal)*: ($a^*$-irrelevant abstraction), if $f(s_1) = f(s_2)$ then $s_1$ and $s_2$ share the same optimal action $a^*$ and $|Q^*(s_1, a^*) - Q^*(s_2, a^*)| \leq \epsilon$.

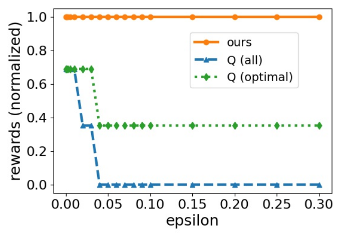
**Training and results.** 3 abstract options are given: 1. open door, 2. find key, and 3. find star. To find the abstract model induced by our successor homomorphism, we first compute the ground options corresponding to each abstract option, hence the obtained ground options yield successor features within $\epsilon$ distance to the abstract option. Then we cluster the states $s$ according to the pairwise distance of their option termination state distributions $\max_{s', \bar{o}} |P^{g_{s_1}^{-1}(\bar{o})}_{s_1, s'} - P^{g_{s_2}^{-1}(\bar{o})}_{s_2, s'}|$ through agglomerative clustering (Pedregosa et al. 2011) with distance threshold $\epsilon$. After forming the abstract states, the abstract transition dynamics and feature function can be computed. For Q (all) and Q (optimal), we first perform Q-value iteration on a source task to obtain the optimal Q-values, then cluster the states by their pairwise differences, e.g., $|Q^*(s_1, \cdot) - Q^*(s_2, \cdot)|_\infty$ for the Q (all) approach, then



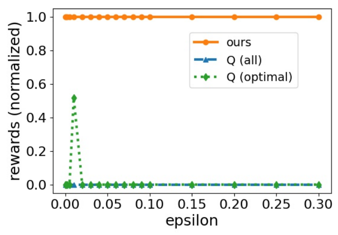
(a) dense reward



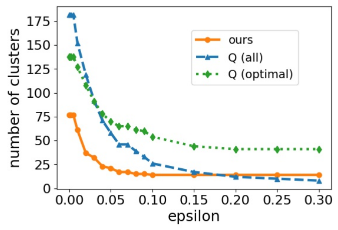
(b) sparse reward



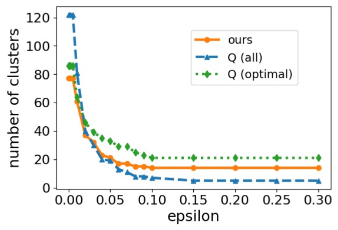
(c) transfer (w. overlap)



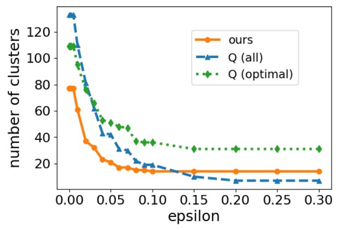
(d) transfer (w.o. overlap)



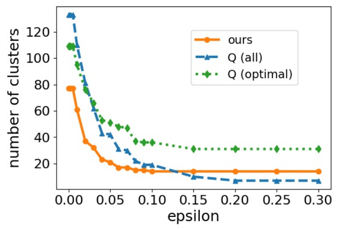
(e) dense reward



(f) sparse reward



(g) transfer (w. overlap)



(h) transfer (w.o. overlap)

Figure 5: Performance of planning with the abstract MDPs. The upper row shows the total rewards (normalized by the maximum possible total rewards) obtained, and the lower row shows the corresponding number of abstract states of the abstract MDP. The $x$-axes are the distance thresholds $\epsilon$. *transfer* refers to task transfer (i.e., different reward function)

compute the abstract transition dynamics.

Figure 5 shows the results of using the induced abstract model for planning. Our successor homomorphism model performs well across all tested settings with few abstract states (number of clusters). Since successor homomorphism does not depend on rewards, the abstract model can transfer across tasks (with varying reward functions), and is robust under sparse rewards settings. Whereas abstraction schemes based on the reward function perform worse when the source task for performing abstraction is different from the target task where the abstract MDP is used for planning.

## 6 Related Work

Agent-space options (Konidaris and Barto 2007) is one of our conceptual inspirations. In our work, we tie the agent-space to the problem-space through features, and our option grounding algorithms allows the agents to transfer agent-space options across different problem spaces.

**Successor features and options derived therefrom.** Successor representations (SR) (Stachenfeld, Botvinick, and Gershman 2017) were first introduced by Dayan (1993). Barreto et al. (2016) generalised SR and proposed successor features (SF). Machado et al. (2017) discovered eigenoptions from successor features and showed their equivalence to options derived from the graph Laplacian. Ramesh, Tomar, and Ravindran (2019) discovered successor options via clustering over the SR. The Option Keyboard (Barreto et al. 2019) is a pioneering work for option combinations: primitive options are first trained to maximise cumulants (i.e., pseudo-rewards on histories), then by putting preference weights over the cumulants, new options are synthesized to maximise the weighted sum of cumulants. For the purpose of option transfer and grounding, however, this may yield imprecise option behaviours due to the interference between cumulants. In contrast, our successor feature-matching formulation generates precise option behaviours as demonstrated in Table 1.

**MDP Abstraction.** The framework of MDP abstraction through stochastic bisimulation was first introduced by Dean and Givan (1997). Ravindran and Barto (2002) introduced MDP homomorphisms, which account for action equivalence with a state-dependent action mapping. Li, Walsh, and Littman (2006) proposed a unified theory of abstraction, and approximate abstraction mechanisms were studied by Ferns, Panangaden, and Precup (2004) and Abel, Hershkowitz, and Littman (2016). Most relevant to our work is the SMDP homomorphism (Ravindran 2003) and (Abel et al. 2020), which define theoretical formulations for reward-based abstractions in SMDPs. In comparison, our successor homomorphism can be efficiently computed through our option grounding algorithms. Different from prior abstraction formulations which are reward-based, our feature-based successor homomorphism produces abstract models which are reuseable under changing task rewards.

## 7 Conclusion

In this work, we studied abstraction in the context of robotic RL agents. Specifically, we developed an abstract option representation which can be transferred from existing ground options and grounded in unseen environments. Based on the abstract options, we developed an SMDP abstraction scheme which produces abstract SMDPs for planning with abstract options with near-optimal performance across different tasks. We demonstrated empirically that the abstract options can be transferred and grounded effectively and efficiently in both known and unknown environments. We also showed in our experiments that our abstract SMDP models exhibit meaningful temporal semantics and is reuseable and robust under task changes.

## References

Abbeel, P.; and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1.

Abel, D.; Hershkowitz, D.; and Littman, M. 2016. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, 2915–2923. PMLR.

Abel, D.; Umbanhowar, N.; Khetarpal, K.; Arumugam, D.; Precup, D.; and Littman, M. 2020. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, 1639–1650. PMLR.

Arora, S.; and Doshi, P. 2018. A survey of inverse reinforcement learning: Challenges, methods and progress. *arXiv preprint arXiv:1806.06877*.

Barreto, A.; Borsa, D.; Hou, S.; Comanici, G.; Aygün, E.; Hamel, P.; Toyama, D. K.; Hunt, J. J.; Mourad, S.; Silver, D.; et al. 2019. The option keyboard: Combining skills in reinforcement learning.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; Van Hasselt, H.; and Silver, D. 2016. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*.

Dayan, P. 1993. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4): 613–624.

Dean, T.; and Givan, R. 1997. Model minimization in Markov decision processes. In *AAAI/IAAI*, 106–111.

Ferns, N.; Panangaden, P.; and Precup, D. 2004. Metrics for Finite Markov Decision Processes. In *UAI*, volume 4, 162–169.

Girshick, R. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, 1440–1448.

Gurobi Optimization, L. 2021. Gurobi Optimizer Reference Manual.

Ho, J.; and Ermon, S. 2016. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*.

Johnson, M.; Hofmann, K.; Hutton, T.; and Bignell, D. 2016. The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, 4246–4247. Citeseer.

Konidaris, G. D.; and Barto, A. G. 2007. Building Portable Options: Skill Transfer in Reinforcement Learning. In *IJCAI*, volume 7, 895–900.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a Unified Theory of State Abstraction for MDPs. In *ISAIM*.

Machado, M. C.; Rosenbaum, C.; Guo, X.; Liu, M.; Tesauro, G.; and Campbell, M. 2017. Eigenoption discovery through the deep successor representation. *arXiv preprint arXiv:1710.11089*.

Malek, A.; Abbasi-Yadkori, Y.; and Bartlett, P. 2014. Linear programming for large-scale Markov decision problems. In *International Conference on Machine Learning*, 496–504. PMLR.

Manne, A. S. 1960. Linear programming and sequential decisions. *Management Science*, 6(3): 259–267.

Mehta, N.; Natarajan, S.; Tadepalli, P.; and Fern, A. 2008. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, 73(3): 289.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, 2.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Pons, P.; and Latapy, M. 2005. Computing communities in large networks using random walks. In *International symposium on computer and information sciences*, 284–293. Springer.

Poupart, P. 2013. Module 8: Linear Programming; Lectures: Sequential Decision Making and Reinforcement Learning.

Ramesh, R.; Tomar, M.; and Ravindran, B. 2019. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*.

Ravindran, B. 2003. SMDP homomorphisms: An algebraic approach to abstraction in semi markov decision processes.(2003).

Ravindran, B.; and Barto, A. G. 2002. Model minimization in hierarchical reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, 196–211. Springer.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Stachenfeld, K. L.; Botvinick, M. M.; and Gershman, S. J. 2017. The hippocampus as a predictive map. *Nature neuroscience*, 20(11): 1643.

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.

Syed, U.; Bowling, M.; and Schapire, R. E. 2008. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, 1032–1039.

Ziebart, B. D.; Maas, A. L.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, 1433–1438. Chicago, IL, USA.

## A Appendix

### A.1 Algorithm

In the main text, we have discussed a naive option grounding algorithm:

- **Algorithm 1 (IRL-naive)**: naive option grounding algorithm which performs IRL over all starting states independently.

In this section, we present additional algorithms useful for option grounding:

- **Algorithm 3 (IRL module for IRL-naive)**: the IRL algorithm adapted from (Syed, Bowling, and Schapire 2008), and used by the IRL-naive option grounding algorithm (Algorithm 1) to find the option policies starting from each starting state.
- **Algorithm 4 (IRL-batch):** an efficient option grounding algorithm which improves IRL-naive and performs batched learning using IRL.
- **Algorithm 5: (IRL module for IRL-batch)** the IRL module used by the IRL-batch option grounding algorithm (Algorithm 4) to find option policies starting from a batch of starting states.

**IRL algorithm for the naive option grounding (Algorithm 3)** First, we introduce the IRL module adapted from (Syed, Bowling, and Schapire 2008) and used by the IRL-naive option grounding algorithm:

```
Algorithm 3: IRL (used by Algorithm 1 IRL-naive)
    Input: Augmented MDP $\mathcal{M} = \langle \mathcal{S}', \mathcal{A}', P', r, \gamma \rangle$, starting state $s_{\text{start}}$, state-action features $\boldsymbol{\theta}\colon \mathcal{S}' \times \mathcal{A}' \to \mathbb{R}^d$, abstract option $\boldsymbol{\psi}^{\bar{o}} \in \mathbb{R}^d$.
    Output: ground and abstract option feature difference $\epsilon = \sum_{k=1}^{d} \epsilon_k$, option policy $\pi^o_{s_{\text{start}}}$
  Solve LP for visitation frequencies $\mu_{s,a}$ and feature matching errors $\epsilon_k$
```

$$
\begin{aligned}
\min_{\mu, \epsilon_k} \quad & \sum_k \epsilon_k - \lambda_1 \sum_a \mu_{s_{\text{null}},a} + \lambda_2 \sum_s \mathbb{1}_{\mu_{s,a_{\text{T}}}>0} \\
\text{s.t.} \quad & \sum_a \mu_{s,a} = \mathbb{1}_{s=s_{\text{start}}} + \gamma \sum_{s',a} P'(s|s',a)\mu_{s',a} \qquad \forall s \in \mathcal{S}' \\
& \sum_{s,a} \mu_{s,a} \boldsymbol{\theta}_k(s,a) - \boldsymbol{\psi}^{\bar{o}}_k \le \epsilon_k \qquad \forall k \in 1, \dots, d \\
& \sum_{s,a} \mu_{s,a} \boldsymbol{\theta}_k(s,a) - \boldsymbol{\psi}^{\bar{o}}_k \ge -\epsilon_k \qquad \forall k \in 1, \dots, d \\
& \mu_{s,a} \ge 0 \quad \forall s \in \mathcal{S}', a \in \mathcal{A}'
\end{aligned}
$$

```
  Compute option policy $\pi^o_{s_{\text{start}}}(a|s) = \frac{\mu_{s,a}}{\sum_a \mu_{s,a}}$
```

The algorithm finds the policy and termination condition of the option $\bar{o}$ in the following two steps:

1. Compute the state-action visitation frequencies $\mu_{s,a}$ such that the corresponding expected feature vector (approximately) matches the abstract option feature $\boldsymbol{\psi}^{\bar{o}}$.
2. Compute the option policy (which includes the termination condition) from $\mu_{s,a}$.

**Linear program.** Adapting from prior work which uses linear programming approaches for solving MDPs and IRL (Syed, Bowling, and Schapire 2008; Malek, Abbasi-Yadkori, and Bartlett 2014; Manne 1960), our LP aims to find the state-action visitation frequencies $\mu_{s,a}$ for all states and actions, which together (approximately) match the abstract option feature, i.e., $\boldsymbol{\psi}^o_{\text{start}} = \sum_{s,a} \mu_{s,a} \boldsymbol{\theta}(s,a) \approx \boldsymbol{\psi}^{\bar{o}}$. In particular, $\|\boldsymbol{\psi}^o_{\text{start}} - \boldsymbol{\psi}^{\bar{o}}\|_1 \le \sum_{k=1}^{d} \epsilon_k$.

***Inputs:*** Recall that to enable the modelling of options and their termination conditions, the input augmented MDP $\mathcal{M} = \langle \mathcal{S}', \mathcal{A}', P', r, \gamma \rangle$ was constructed in Algorithm 1 by adding a null state $s_{\text{null}}$ and termination action $a_{\text{T}}$ such that $a_{\text{T}}$ leads from any regular state to $s_{\text{null}}$.

***Variables:*** 1. $\epsilon_k$: Upper bounds for the absolute difference between the (learner) learned ground option successor feature and the abstract option (expert) feature in the $k$-th dimension.

2. $\mu_{s,a}$: Expected cumulative state-action visitation of state-action pair $s, a$. Additionally, denote $\mu_s$ as the expected cumulative state visitation of $s$, i.e., $\forall s \in \mathcal{S}, a \in \mathcal{A}$,

$$\mu_{s,a} = \mathbb{E}_{\mathcal{M},\pi}[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{S_t=s, A_t=a} | s_0 \sim P_{\text{start}}], \quad \text{and} \quad \mu_s = \mathbb{E}_{\mathcal{M},\pi}[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{S_t=s} | s_0 \sim P_{\text{start}}] \tag{2}$$

where $P_{\text{start}}$ is the distribution over starting states. In this naive option grounding algorithm the start state is a single state $s_{\text{start}}$. Observe that $\mu_s$ and $\mu_{s,a}$ are related by the policy $\pi\colon \mathcal{S} \to \mathcal{A}$ as

$$\mu_{s,a} = \pi(a|s)\mu_s, \quad \text{and} \quad \mu_s = \sum_a \pi(a|s)\mu_s = \sum_a \mu_{s,a}. \tag{3}$$

And the Bellman flow constraint is given by either $\mu_s$ or $\mu_{s,a}$:

$$\mu_s = \mathbb{1}_{s=s_{\text{start}}} + \gamma \sum_{s',a} P(s|s',a)\pi(a|s')\mu_{s'} \iff \sum_a \mu_{s,a} = \mathbb{1}_{s=s_{\text{start}}} + \gamma \sum_{s',a} P(s|s',a)\mu_{s',a} \tag{4}$$

By Equation (3), the policy $\pi$ corresponding to the state-action visitation frequencies is computed as

$$\pi^o_{s_{\text{start}}} = \frac{\mu_{s,a}}{\mu_s} = \frac{\mu_{s,a}}{\sum_a \mu_{s,a}}. \tag{5}$$

***Objective function:*** $\sum_k \epsilon_k - \lambda_1 \sum_a \mu_{s_{\text{null, a}}} + \lambda_2 \sum_s \mathbb{1}_{\mu_{s,a_{\text{T}}}>0}$

1. $\sum_k \epsilon_k$: the first term is the feature difference $\epsilon$ between the abstract and computed ground option.

2. $-\lambda_1 \sum_a \mu_{s_{\text{null, a}}}$: The second term is a small penalty on the option length to encourage short options. This is achieved by putting a small bonus (e.g., $\lambda_1 = 0.01$) on the expected cumulative visitation of the null state $s_{\text{null}}$, which the agent reaches after using the terminate action.

3. $\lambda_2 \sum_s \mathbb{1}_{\mu_{s,a_{\text{T}}}>0}$: The last term is a regularisation on the number of terminating states. This helps guide the LP to avoid finding mixtures of option policies which together match the feature expectation.

In this way, the linear program can find an option policy which terminates automatically.

***Constraints:*** Equation (**??**) is the Bellman flow constraint (Syed, Bowling, and Schapire 2008; Malek, Abbasi-Yadkori, and Bartlett 2014) which specifies how the state-action visitation frequencies are related by the transition dynamics, see Equation (4) and (Poupart 2013) for a detailed introduction. Equations (**??**) and (**??**) define the feature difference $\epsilon$ of the ground option $\boldsymbol{\psi}^o_{s_{\text{start}}} = \sum_{s,a} \mu_{s,a}\boldsymbol{\theta}(s,a)$ and abstract option $\boldsymbol{\psi}^{\bar{o}}$ for the $k$-th dimension.

*Step 2* derives the policy from the state-action visitation frequencies $\mu_{s,a}$, see Equation (5) and (Poupart 2013).

Algorithm 4: Grounding Abstract Options (IRL-Batch)

```
    Input: MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$, abstract option $\boldsymbol{\psi}^{\bar{o}}$, $\epsilon_{\text{threshold}}$
    Output: initiation set $I^{\bar{o}}$, dictionary of ground option policies $\Pi^{\bar{o}}$, termination probabilities $\Xi^{\bar{o}}$
 1: // Construct augmented MDP
 2: $\mathcal{S}' \leftarrow \mathcal{S} \cup \{s_{\text{null}}\}, \mathcal{A}' \leftarrow \mathcal{A} \cup \{a_{\text{T}}\}, P' \leftarrow P$
 3: $\forall s \in \mathcal{S}' \colon P'(s_{\text{null}} \mid s, a_{\text{T}}) = 1; \forall a \in \mathcal{A}' \colon P'(s_{\text{null}} \mid s_{\text{null}}, a) = 1$

 4: // Find ground options for all starting states
 5: $I^{\bar{o}}, \Pi^{\bar{o}}, \Xi^{\bar{o}}$ = MATCH-AND-DIVIDE($\mathcal{S}' \setminus \{s_{\text{null}}\}$)

 6: // Recursive Function
 7: function MATCH-AND-DIVIDE($c_{\text{start}}$)
 8:     $\epsilon_{c_{\text{start}}}, \Pi^{o}_{c_{\text{start}}} \leftarrow$ IRL($\mathcal{S}', \mathcal{A}', P', \gamma, c_{\text{start}}, \boldsymbol{\psi}^{\bar{o}}$),        ▷ for IRL see Algorithm 5
 9:     $c_{\text{match}}, c_{\text{no-match}}, C_{\text{ambiguous}}$ = CLASSIFY($c_{\text{start}}, \Pi^{o}_{c_{\text{start}}}, \epsilon_{c_{\text{start}}}, \epsilon_{\text{threshold}}$)
10:     for all $s_{\text{start}} \in c_{\text{match}}$ do
11:         $\pi^{o}_{s_{\text{start}}} \leftarrow \Pi^{o}_{c_{\text{start}}}(s_{\text{start}})$
12:         $I^{\bar{o}} \leftarrow I^{\bar{o}} \cup \{s_{\text{start}}\}, \Pi^{\bar{o}}(s_{\text{start}}) = \pi^{o}_{s_{\text{start}}}, \Xi^{\bar{o}}(s_{\text{start}}) = \beta^{o},$ where $\beta^{o}(s) = \pi^{o}_{s_{\text{start}}}$
13:     end for
14:     if $C_{\text{ambiguous}} \neq \emptyset$ then
15:         for all $c_i \in C_{\text{ambiguous}}$ do
16:             MATCH-AND-DIVIDE($c_i$)        ▷ Note: $C_{\text{ambiguous}}$ is a set of clusters
17:         end for
18:     end if
19:     return $I^{\bar{o}}, \Pi^{\bar{o}}, \Xi^{\bar{o}}$
20: end function

21: // Classify start states according to the policy found by IRL
22: function CLASSIFY($c_{\text{start}}, \Pi^{o}_{c_{\text{start}}}, \epsilon_{c_{\text{start}}}, \epsilon_{\text{threshold}}$)
23:     $c_{\text{match}}, c_{\text{no-match}}, c_{\text{ambiguous}} \leftarrow \emptyset$
24:     for all $s_{\text{start}} \in c_{\text{start}}$ do
25:         Execute $o$ from $s_{\text{start}}$ to get successor feature $\boldsymbol{\psi}^{o}_{s_{\text{start}}}$ and terminating distribution $P^{o}_{s_{\text{start}}, s'}$
26:         Compute feature difference $\epsilon^{o}_{s_{\text{start}}} = |\boldsymbol{\psi}^{\bar{o}} - \boldsymbol{\psi}^{o}_{s_{\text{start}}}|$
27:     end for
28:     $c_{\text{no-match}} \leftarrow c_{\text{start}}$ if $\min_{s_{\text{start}} \in c_{\text{start}}} \epsilon^{o}_{s_{\text{start}}} > \epsilon_{\text{threshold}}$ and $\epsilon_{c_{\text{start}}} > \epsilon_{\text{threshold}}$
29:     $c_{\text{match}} \leftarrow \{s_{\text{start}} \in c_{\text{start}} | \epsilon^{o}_{s_{\text{start}}} \leq \epsilon_{\text{threshold}}\}$
30:     $c_{\text{ambiguous}} \leftarrow c_{\text{start}} \setminus c_{\text{match}}, c_{\text{no-match}}$
31:     if $c_{\text{ambiguous}} \neq \emptyset$ then        ▷ Cluster by termination distributions or successor features
32:         $C_{\text{ambiguous}} \leftarrow$ CLUSTER($c_{\text{ambiguous}}, P^{o}_{s_{\text{start}}, s'}$)
33:     end if
34:     return $c_{\text{match}}, c_{\text{no-match}}, C_{\text{ambiguous}}$
35: end function
```

**Grounding Abstract Options for Starting States in Batches (Algorithm 4)** Based on the naive option grounding algorithm (Algorithm 1), we now introduce an efficient algorithm for grounding abstract options, which performs IRL over the start states in batches (Algorithm 4).

**Challenges.** The main challenges regarding performing batched IRL for grounding the options are: 1. By naively putting a uniform distribution over all possible starting states, the IRL LP cannot typically find the ground options which match the abstract option. Moreover, a closely related problem as well as one of the reasons for the first problem is 2. Since there are many different starting states, the state-action visitation frequencies found by the LP for matching the option feature may be a mixture of different option policies from the different starting states, and the induced ground option policies individually cannot achieve the successor feature of the abstract option.

**Solutions.** For the first challenge, we introduce the batched IRL module (Algorithm 5) to be used by IRL-batch. It flexibly learns a starting state distribution with entropy regularisation.

For the second challenge, Algorithm 4 is a recursive approach where each recursion performs batched-IRL on a set of starting states. Then, by executing the options from each starting state using the transition dynamics, we prune the starting states which successfully match the abstract successor option's feature, and those where matching the abstract option is impossible. And cluster the remaining ambiguous states based on their option termination distribution (or their achieved successor features) and go to the next recursion. Intuitively, we form clusters of similar states (e.g., which are nearby and belong to a same community according to the option termination distribution). And running batched-IRL over a cluster of similar starting states typically returns a single ground option policy which applies to all these starting states.

**Algorithm 4 (IRL-batch: Grounding Abstract Options in Batches).** The algorithm first constructs the augmented MDP with the null states and terminate actions in the same way as the naive algorithm. Then it uses a recursive function *Match-And-Divide($c_{\text{start}}$)*, which first computes the ground option policies corresponding to the set of starting states $c_{\text{start}}$ through batched IRL (Algorithm 5) over $c_{\text{start}}$, then *Classify* the starting states by their corresponding ground options' termination distributions or successor feature, into the following 3 categories: 1. $c_{\text{match}}$: the start states where an abstract option can be initiated (i.e., there exists a ground option whose successor feature matches the abstract option); 2. $c_{\text{no-match}}$: the start states where the abstract option cannot be initiated; and 3. $C_{\text{ambiguous}}$: a set of clusters dividing the remaining ambiguous states. If $C_{\text{ambiguous}}$ is not empty, then each cluster goes through the next recursion of *Match-And-Divide*. Otherwise the algorithm terminates and outputs the initiation set, option policy and termination conditions.

```
Algorithm 5: IRL (used by Algorithm 4 IRL-Batch)
    Input: Augmented MDP $\mathcal{M} = \langle \mathcal{S}', \mathcal{A}', P', r, \gamma \rangle$,, state-action features $\boldsymbol{\theta}: \mathcal{S}' \times \mathcal{A}' \to \mathbb{R}^d$, abstract option $\boldsymbol{\psi}^{\bar{o}} \in \mathbb{R}^d$, a set of starting states $c_{\text{start}} \subseteq \mathcal{S}' \setminus \{s_{\text{null}}\}$.
    Output: (joint over $c_{\text{start}}$) ground and abstract option feature difference $\epsilon = \sum_{k=1}^{d} \epsilon_k$, dictionary of ground option policy $\Pi^o(s_{\text{start}})$ for all start states in $c_{\text{start}}$
     // Step 1: Solve LP for state-action visitation frequencies $\mu_{s,a}$
```

$$
\begin{aligned}
\min_{\mu,\epsilon_k,p^{\text{start}}} \quad & \sum_k \epsilon_k - \lambda_1 \sum_a \mu_{s_{\text{null}},a} + \lambda_2 \sum_s p_s^{\text{start}} \log p_s^{\text{start}} + \lambda_3 \sum_s \mathbb{1}_{\mu_{s,a_{\text{T}}}>0} & \\
\text{s.t.} \quad & \sum_a \mu_{s,a} = p_s^{\text{start}} + \gamma \sum_{s',a} P'(s|s',a)\mu_{s',a} & \forall s \in \mathcal{S}' \\
& \sum_{s,a} \mu_{s,a}\boldsymbol{\theta}_k(s,a) - \boldsymbol{\psi}_k^{\bar{o}} \leq \epsilon_k & \forall k \in 1,\ldots,d \\
& \sum_{s,a} \mu_{s,a}\boldsymbol{\theta}_k(s,a) - \boldsymbol{\psi}_k^{\bar{o}} \geq -\epsilon_k & \forall k \in 1,\ldots,d \\
& \mu_{s,a} \geq 0 & \forall s \in P', a \in \mathcal{A}' \\
& \sum_{s \in c_{\text{start}}} p_s^{\text{start}} = 1 & \\
& p_s^{\text{start}} \geq 0 & \forall s \in c_{\text{start}} \\
& p_s^{\text{start}} = 0 & \forall s \notin c_{\text{start}}
\end{aligned}
$$

```
     // Step 2: Compute dictionary of option policies $\Pi^o$, $\forall s_{\text{start}} \in c_{\text{start}}$:
```

$$
\forall s \in \mathcal{S}, a \in \mathcal{A}', \pi^o_{s_{\text{start}}}(a|s) = \frac{\mu_{s,a}}{\sum_a \mu_{s,a}}, \qquad \text{then add to dictionary} \quad \Pi^o(s_{\text{start}}) \leftarrow \pi^o_{s_{\text{start}}}
$$

**Algorithm 5 (IRL module for IRL-batch).** Compared with the IRL algorithm presented in Algorithm 1, this batched algorithm performs IRL over a batch of starting states. As we discussed in the above challenges, naively putting a uniform distribution over all possible starting states would typically fail. Therefore, we enable the LP to *learn a starting state distribution* $p_s^{\text{start}}$ which best matches the abstract option feature. However, the LP would again reduce to a single starting state which best matches the abstract option. To counteract this effect, we add a small entropy regularisation to the objective to increase the entropy of the starting state distribution, i.e., we add $\lambda_2 \sum_s p_s^{\text{start}} \log p_s^{\text{start}}$. Clearly, entropy is not a linear function. In practise, the objective is implemented using a piecewise-linear approximation method provided by the Gurobi optimisation package (Gurobi Optimization 2021).

## A.2 Feature-based (Variable-reward) SMDP Abstraction

In this section, we provide a general framework for feature-based abstraction using the variable-reward SMDPs. The state mapping and action mapping functions $f(s), g_s(o)$ can be defined to instantiate a new abstraction method.

**Definition A.1** (Abstract $\boldsymbol{\psi}$-SMDP). *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, P, \boldsymbol{\psi}, \gamma \rangle$ be a ground $\boldsymbol{\psi}$-SMDP. We say that $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{O}}, \bar{P}, \bar{\boldsymbol{\psi}}, \gamma \rangle$ is an* abstract $\boldsymbol{\psi}$-SMDP of $\mathcal{M}$ *if there exists (1) a state abstraction mapping $f: \mathcal{S} \to \bar{\mathcal{S}}$ which maps each ground state to an abstract* *state, (2) a weight function $w \colon \mathcal{S} \to [0,1]$ over the ground states such that $\forall \bar{s} \in \bar{\mathcal{S}}, \sum_{s \in f^{-1}(\bar{s})} w_s = 1$, (3) a state-dependent option abstraction mapping $g_s \colon \mathcal{O} \to \bar{\mathcal{O}}$, and (4) the abstract transition dynamics and features are*

$$\bar{P}^{\bar{o}}_{\bar{s},\bar{s}'} = \sum_{s \in f^{-1}(\bar{s})} w_s \sum_{s' \in f^{-1}(\bar{s}')} P^{g_s^{-1}(\bar{o})}_{s,s'} \text{ , and } \quad \bar{\boldsymbol{\psi}}^{\bar{o}}_{\bar{s}} = \sum_{s \in f^{-1}(\bar{s})} w_s \boldsymbol{\psi}^{g_s^{-1}(\bar{o})}_{s}.$$

## A.3 Reward-based MDP and SMDP abstraction

In the following we define the abstract MDP and abstract SMDPs, following the conventional notations of (Li, Walsh, and Littman 2006; Abel, Hershkowitz, and Littman 2016; Ravindran 2003).

**Definition A.2** (Abstract MDP)**.** *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$ be a ground MDP. We say that $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \mathcal{A}, \bar{P}, \bar{r}, \gamma \rangle$ is an* abstract MDP of $\mathcal{M}$ *if there exists (1) a state abstraction mapping $f \colon \mathcal{S} \to \bar{\mathcal{S}}$, which maps each ground state to an abstract state, (2) a weight function $w \colon \mathcal{S} \to [0,1]$ for the ground states such that $\forall \bar{s} \in \bar{\mathcal{S}}$, $\sum_{s \in f^{-1}(\bar{s})} w_s = 1$, and (3) a state-dependent action mapping $g_s \colon \mathcal{A} \to \bar{\mathcal{A}}$, the abstract transition dynamics and rewards are defined as*

$$\bar{P}^{\bar{a}}_{\bar{s},\bar{s}'} = \sum_{s \in f^{-1}(\bar{s})} w_s \sum_{s' \in f^{-1}(\bar{s}')} P^{g_s^{-1}(\bar{a})}_{s,s'} \text{ , and } \quad \bar{r}^{\bar{a}}_{\bar{s}} = \sum_{s \in f^{-1}(\bar{s})} w_s r^{g_s^{-1}(\bar{a})}_{s}.$$

In cases in which multiple ground actions map to the same abstract action, $g_s^{-1}(\bar{a})$ picks one of the ground actions. $g_s$ is commonly defined as the identity mapping (Li, Walsh, and Littman 2006; Abel, Hershkowitz, and Littman 2016). We now generalize this definition to abstract SMDPs:

**Definition A.3** (Abstract SMDP)**.** *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, P, r, \gamma \rangle$ be a ground SMDP. We say that $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{O}}, \bar{P}, \bar{r}, \gamma \rangle$ is an* abstract SMDP of $\mathcal{M}$ *if there exists (1) a state abstraction mapping $f \colon \mathcal{S} \to \bar{\mathcal{S}}$ which maps each ground state to an abstract state, (2) a weight function $w \colon \mathcal{S} \to [0,1]$ over the ground states such that $\forall \bar{s} \in \bar{\mathcal{S}}, \sum_{s \in f^{-1}(\bar{s})} w_s = 1$, (3) a state-dependent option abstraction mapping $g_s \colon \times \mathcal{O} \to \bar{\mathcal{O}}$, and (4) the abstract transition dynamics and rewards are*

$$\bar{P}^{\bar{o}}_{\bar{s},\bar{s}'} = \sum_{s \in f^{-1}(\bar{s})} w_s \sum_{s' \in f^{-1}(\bar{s}')} P^{g_s^{-1}(\bar{o})}_{s,s'} \text{ , and } \quad \bar{r}^{\bar{o}}_{\bar{s}} = \sum_{s \in f^{-1}(\bar{s})} w_s r^{g_s^{-1}(\bar{o})}_{s}.$$

In cases in which multiple ground options map to the same abstract option, $g_s^{-1}(\bar{o})$ picks one of the ground options, e.g., the option of shortest duration, maximum entropy (Ziebart et al. 2008), etc.

## A.4 Relation to Other MDP Abstraction Methods

The framework of MDP abstraction was first introduced by Dean and Givan (1997) through stochastic bisimulation, and (Ravindran and Barto 2002) extended it to MDP homomorphisms. Later, Li, Walsh, and Littman (2006) classified exact MDP abstraction into 5 categories and Abel, Hershkowitz, and Littman (2016) formulated their approximate counterparts: model-irrelevance, $Q^\pi$-irrelevance, $Q^*$-irrelevance, $a^*$-irrelevance and $\pi^*$-irrelevance abstractions. Our successor homomorphism follows the formulation of MDP homomorphism (Ravindran 2003), which broadly fall into the category of model-irrelevance abstraction, where states are aggregated according to their one-step/multi-step transition dynamics and rewards. On the other hand, abstraction schemes which aggregate states according to their Q-values are $Q^*$-irrelevance abstraction and $a^*$ abstraction, where $Q^*$ aggregate states according to all actions, while $a^*$ aggregate states with the same optimal action and Q-value, cf. Figure 2 for an illustration of the induced abstract MDPs. Different from prior abstraction formulations (Definition A.2) which are reward-based, our feature-based successor homomorphism produces abstract models with meaningful temporal semantics, and is robust under task changes. Furthermore, we include a generic formulation of feature-based (variable-reward) abstraction (Definition A.1), which provides a basis for potential feature-based abstractions other than successor homomorphism.

## A.5 Proof for Theorem 4.1

**Theorem 4.1.** *Let $w_r \colon \mathbb{R}^d \to \mathbb{R}$ be a linear reward vector such that $r^a_s = w_r^T \boldsymbol{\theta}(s,a)$. Under this reward function, the value of an optimal abstract policy obtained through the $\epsilon$-approximate successor homomorphism is close to optimal ground SMDP policy, where the difference is bounded by $\frac{2\kappa}{(1-\gamma)^2}$, where $\kappa = |w_r|(2\epsilon_\psi + \frac{\epsilon_P |\bar{\mathcal{S}}| \max_{s,a} |\boldsymbol{\theta}(s,a)|}{1-\gamma})$.*

*Proof.* Given $\epsilon$-Approximate Successor Homomorphism: $h = (f(s), g_s(o), w_s)$ from ground $\boldsymbol{\psi}$-SMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, P, \boldsymbol{\psi}, \gamma \rangle$ to abstract $\boldsymbol{\psi}$-SMDP $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{O}}, \bar{P}, \boldsymbol{\psi}, \gamma \rangle$, such that $\forall s_1, s_2 \in \mathcal{S}, o_1, o_2 \in \mathcal{O}$, $h(s_1, o_1) = h(s_2, o_2) \implies$

$$|\sum_{s_j \in f^{-1}(\bar{s}')} P^{o_1}_{s_1,s_j} - \sum_{s_j \in f^{-1}(\bar{s}')} P^{o_2}_{s_2,s_j}| \le \epsilon_P \tag{6}$$

$$|\boldsymbol{\psi}^{o_1}_{s_1} - \boldsymbol{\psi}^{o_2}_{s_2}| \le \epsilon_{\boldsymbol{\psi}} \tag{7}$$

The abstract transition dynamics and features are

$$P^{\bar{o}}_{\bar{s},\bar{s}'} = \sum_{s \in f^{-1}(\bar{s})} w_s \sum_{s' \in f^{-1}(\bar{s}')} P^{g_s^{-1}(\bar{o})}_{s,s'}, \text{ and } \boldsymbol{\psi}^{\bar{o}}_{\bar{s}} = \sum_{s \in f^{-1}(\bar{s})} w_s \boldsymbol{\psi}^{g_s^{-1}(\bar{o})}_{s}. \tag{8}$$

We show that given features $\boldsymbol{\theta} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ in the underlying (feature-based) MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, \boldsymbol{\theta}, \gamma \rangle$, and linear reward function on the features $w_r \in \mathbb{R}^d$, the difference in value of the optimal policy in the induced abstract SMDP and the ground SMDP is bounded by $\frac{2\kappa}{(1-\gamma)^2}$, where $\kappa = |w_r|(2\epsilon_{\boldsymbol{\psi}} + \frac{\epsilon_P |\bar{\mathcal{S}}| \max_{s,a} |\boldsymbol{\theta}(s,a)|}{1-\gamma})$.

To show the above error bound, we extend the proof of (Abel, Hershkowitz, and Littman 2016) for the error bound induced by approximate MDP abstraction, which follows the following three steps:

*Step 1: Show that* $\forall s_1 \in \mathcal{S}, o_1 \in \mathcal{O}, \bar{s} = f(s_1), \bar{o} = g(o_1) \implies |Q(\bar{s}, \bar{o}) - Q(s_1, o_1)| \leq \frac{\epsilon}{1-\gamma}$.

$$r_{\bar{s}}^{\bar{o}} = w_r^T \boldsymbol{\psi}_{\bar{s}}^{\bar{o}} = \sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} w_r^T \boldsymbol{\psi}_{s_i}^{g_{s_i}^{-1}(\bar{o})} = \sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} r_{s_i}^{g_{s_i}^{-1}(\bar{o})} \text{ denote } r_{s_i}^{g_{s_i}^{-1}(\bar{o})} \text{ as } w_{s_i} r_{s_i}^{o_i} \tag{9}$$

$$Q(\bar{s}, \bar{o}) = \mathbb{E}[r_{\bar{s}}^{\bar{o}} + \gamma^k \max_{\bar{o}'} Q(\bar{s}', \bar{o}')] \overset{\text{Eq.(8),(9)}}{=} \sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} r_{s_i}^{o_i} + \sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} P_{s_i, s_j}^{g^{-1}(\bar{o})} \max_{\bar{o}'} Q(\bar{s}', \bar{o}') \tag{10}$$

$$Q(s_1, o_1) = \mathbb{E}[r_{s_1}^{o_1} + \gamma^k \max_{o_j'} Q(s_j', o_j')] = r_{s_1}^{o_1} + \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} P_{s_1, s_j}^{o_1} \max_{\bar{o}'} Q(s_j, o_j') \tag{11}$$

$$|Q(\bar{s}, \bar{o}) - Q(s_1, o_1)| \tag{12}$$

$$= |\sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} (r_{s_i}^{o_i} - r_{s_1}^{o_1}) + \sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} \left( P_{s_i, s_j}^{g_{s_i}^{-1}(\bar{o})} \max_{\bar{o}'} Q(\bar{s}', \bar{o}') - P_{s_1, s_j}^{o_1} \max_{o_j'} Q(s_j', o_j') \right) | \tag{13}$$

$$\leq \underbrace{|\sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} w_r^T (\boldsymbol{\psi}_{s_i}^{o_i} - \boldsymbol{\psi}_{s_1}^{o_1})| + |\sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} \left( P_{s_i, s_j}^{g_{s_i}^{-1}(\bar{o})} \max_{\bar{o}'} Q(\bar{s}', \bar{o}') - P_{s_1, s_j}^{o_1} \max_{\bar{o}'} Q(\bar{s}', \bar{o}') \right) |}_{(1)} \tag{14}$$

$$+ \underbrace{|\sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} P_{s_1, s_j}^{o_1} \left( \max_{\bar{o}'} Q(\bar{s}', \bar{o}') - \max_{o_j'} Q(s_j', o_j') \right) |}_{(2)} \tag{15}$$

$$(1) \leq 2|w_r|\epsilon_{\boldsymbol{\psi}} + |\sum_{s_i \in f^{-1}(\bar{s})} w_{s_i} \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} \left( P_{s_i, s_j}^{g_{s_i}^{-1}(\bar{o})} - P_{s_1, s_j}^{o_1} \right) \max_{\bar{o}'} Q(\bar{s}', \bar{o}')| \tag{16}$$

$$\leq 2|w_r|\epsilon_{\boldsymbol{\psi}} + \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \epsilon_P \max_{\bar{o}'} Q(\bar{s}', \bar{o}') \leq 2|w_r|\epsilon_{\boldsymbol{\psi}} + |\bar{\mathcal{S}}|\epsilon_P \frac{|w_r| \max_{s,a} |\boldsymbol{\theta}(s,a)|}{1-\gamma} \tag{17}$$

$$\text{Denote } \kappa = |w_r| \left( 2\epsilon_{\boldsymbol{\psi}} + \frac{|\bar{\mathcal{S}}|\epsilon_P \max_{s,a} |\boldsymbol{\theta}(s,a)|}{1-\gamma} \right) \tag{18}$$

$$(2) = |\underbrace{\sum_{s_i \in f^{-1}(\bar{s})} w_{s_i}}_{=1} \underbrace{\sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} P_{s_1, s_j}^{o_1}}_{(3) \leq \gamma} \left( \max_{\bar{o}'} Q(\bar{s}', \bar{o}') - \max_{o_j'} Q(s_j', o_j') \right) | \tag{19}$$

$$\leq \gamma \max_{s \in \mathcal{S}} | \max_{\bar{o}} Q(f(s), \bar{o}) - \max_{o} Q(s, o)| \quad \text{Let } o^* = \arg\max_{o} Q(s, o) \tag{20}$$

$$\leq \gamma \max_{s \in \mathcal{S}} |Q(f(s), g_s(o^*)) - Q(s, o^*)| \quad \text{which goes back to Equation (12).} \tag{21}$$

(3) is since the option lasts at least one step and terminates with probability 1:
$\sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} \sum_{k=1}^{\infty} P_{s_1, s_j, k}^{o_1} = 1$
Hence (3) $= \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} \sum_{k=1}^{\infty} \gamma^k P_{s_1, s_j, k}^{o_1} \leq \sum_{\bar{s}' \in \bar{\mathcal{S}}'} \sum_{s_j \in f^{-1}(\bar{s}')} \gamma^{k=1} P_{s_1, s_j, k}^{o_1} = \gamma$.

$$
\begin{aligned}
|Q(\bar{s},\bar{o}) - Q(s_1,o_1)| &\leq (1) + (2) && (22)\\
&\leq \kappa + \gamma \underbrace{\max_{s\in\mathcal{S}}[|Q(f(s),g_s(o^*)) - Q(s,o^*)|]}_{(4)} \quad \text{, expand (4) again using Equation (12)} && (23)\\
&\leq \kappa + \gamma \left(\kappa + \gamma \max_{s\in\mathcal{S}}[|Q(f(s),g_s(o^*)) - Q(s,o^*)|]\right) && (24)\\
&\leq \kappa + \gamma\kappa + \gamma^2\kappa + \ldots \leq \frac{\kappa}{1-\gamma} && (25)
\end{aligned}
$$

*Step 2: The optimal option in the abstract MDP has a Q-value in the ground MDP that is nearly optimal, i.e.:*

$$\forall s_1 \in \mathcal{S}, Q(s_1,o_1^*) - Q(s_1, g_{s_1}^{-1}(\bar{o}^*)) \leq \frac{2\kappa}{1-\gamma} \tag{26}$$

where $o_1^* = \arg\max_o Q(s_1,o)$ and $\bar{o}^* = \arg\max_{\bar{o}} Q(f(s_1),\bar{o})$.

From step 1, we have $|Q(\bar{s},\bar{o}) - Q(s_1,o_1)| \leq \frac{\kappa}{1-\gamma}$. Then, by definition of optimality,

$$Q(\bar{s}, g_{s_1}(o_1^*)) \leq Q(\bar{s},\bar{o}^*) \implies Q(\bar{s}, g_{s_1}(o_1^*)) + \frac{\kappa}{1-\gamma} \leq Q(\bar{s},\bar{o}^*) + \frac{\kappa}{1-\gamma} \tag{27}$$

$$\text{Therefore, by step 1 then by optimality: } Q(s_1,o_1^*) \leq Q(\bar{s}, g_{s_1}(o_1^*)) + \frac{\kappa}{1-\gamma} \leq Q(\bar{s},\bar{o}^*) + \frac{\kappa}{1-\gamma} \tag{28}$$

$$\text{Again by step 1: } Q(\bar{s},\bar{o}^*) \leq Q(s_1, g_{s_1}^{-1}(\bar{o}^*)) + \frac{\kappa}{1-\gamma} \tag{29}$$

$$\text{Therefore, } Q(s_1,o_1^*) \overset{Eq.(28)}{\leq} Q(\bar{s},\bar{o}^*) + \frac{\kappa}{1-\gamma} \overset{Eq.(29)}{\leq} Q(s_1, g^{-1}(\bar{o}^*)) + \frac{2\kappa}{1-\gamma} \tag{30}$$

*Step 3: The optimal abstract SMDP policy yields near optimal performance in the ground SMDP:*

Denote $g^{-1}(\bar{\pi})$ as the ground SMDP policy implementing the abstract SMDP policy $\bar{\pi}$, i.e., at state $s$, the ground option corresponding to the abstract option $\bar{o}$ chosen by the abstract policy is $g_s^{-1}(\bar{o})$.

$$
\begin{aligned}
&Q^{\pi^*}(s,o^*) - Q^{g^{-1}(\bar{\pi}^*)}(s, g_s^{-1}(\bar{o}^*)) && (31)\\
&\leq \frac{2\kappa}{1-\gamma} + Q^{\pi^*}(s, g_s^{-1}(\bar{o}^*)) - Q^{g_s^{-1}(\bar{\pi}^*)}(s, g^{-1}(\bar{o}^*)) \text{ , by step 2} && (32)\\
&= \frac{2\kappa}{1-\gamma} + [r_s^{g_s^{-1}(\bar{o}^*)} + \sum_{s'\in S} P_{s,s'}^{g_s^{-1}(\bar{o}^*)} Q^{\pi^*}(s',o_{s'}^*)] - [r_s^{g_s^{-1}(\bar{o}^*)} + \sum_{s'\in S} P_{s,s'}^{g_s^{-1}(\bar{o}^*)} Q^{g^{-1}(\bar{\pi}^*)}(s', g^{-1}(o_{\bar{s}'}^*))] && (33)\\
&= \frac{2\kappa}{1-\gamma} + \underbrace{\sum_{s'\in S} P_{s,s'}^{g_s^{-1}(\bar{o}^*)}}_{(1)\leq\gamma} [Q^{\pi^*}(s',o_{s'}^*) - Q^{g^{-1}(\bar{\pi}^*)}(s', g_{s'}^{-1}(o_{\bar{s}'}^*))] && (34)\\
&\leq \frac{2\kappa}{1-\gamma} + \gamma \max_{s'\in S}[|Q^{\pi^*}(s',o_{s'}^*) - Q^{g^{-1}(\bar{\pi}^*)}(s', g_{s'}^{-1}(o_{\bar{s}'}^*))|] && (35)\\
&\overset{\text{Eq.(31)}}{\leq} \frac{2\kappa}{1-\gamma} + \gamma\left(\frac{2\kappa}{1-\gamma} + \gamma \max_{s'\in S}[Q^{\pi^*}(s', g_{s'}^{-1}(o_{\bar{s}'}^*)) - Q^{g_{s'}^{-1}(\bar{\pi}^*)}(s', g^{-1}(o_{\bar{s}'}^*))]\right) && (36)\\
&\leq \frac{2\kappa}{1-\gamma} + \gamma\frac{2\kappa}{1-\gamma} + \gamma^2\frac{2\kappa}{1-\gamma}\ldots && (37)\\
&\leq \frac{2\kappa}{(1-\gamma)^2} && (38)
\end{aligned}
$$

where (1) is because the option terminates with probability 1 and takes at least 1 step. □

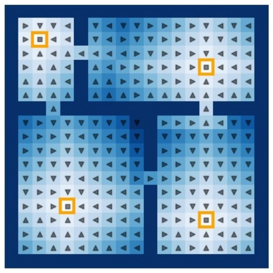
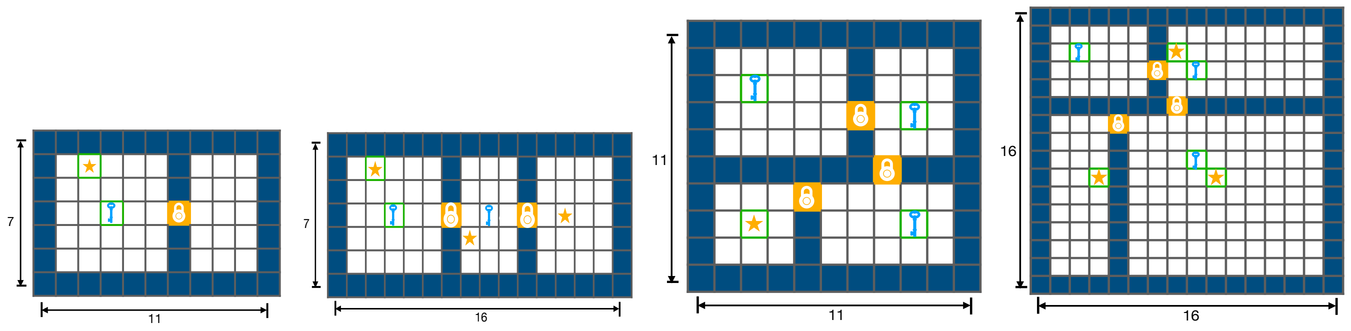


Figure 7: Ground options learned in the 4-Room domain via Algorithm 4 (presented in the Appendix). The features correspond to indicators of the room centers (marked in the orange square). Hence, the ground option should move to any one of the four room centres. The option policy accurately takes the agent from each starting state to a nearby room centre and then terminates.

## A.6 Additional Experiment Details



(a) 2 Rooms, 157 states (b) 3 Rooms, 487 states (c) 4 Rooms (small), 348 states (d) 4 Rooms (large), 1463 states

Figure 6: Object-room layouts used in our experiments. In each object room instance, blue grids are the walls, orange grids are doors, and there are two types of objects: stars and keys, which can be picked-up by the agent. The agent can pick up a key and use it to open a door.

**Additional details on the experimental settings** *Object-Rooms*: $N$ rooms are connected by doors with keys and stars inside. There are 6 actions, i.e., $\mathcal{A} = \{\text{Up, Down, Left, Right, Pick up, Open}\}$. The agent can pick up the keys and stars, and use a key to open a door next to it. The agent starts from the (upper) left room.

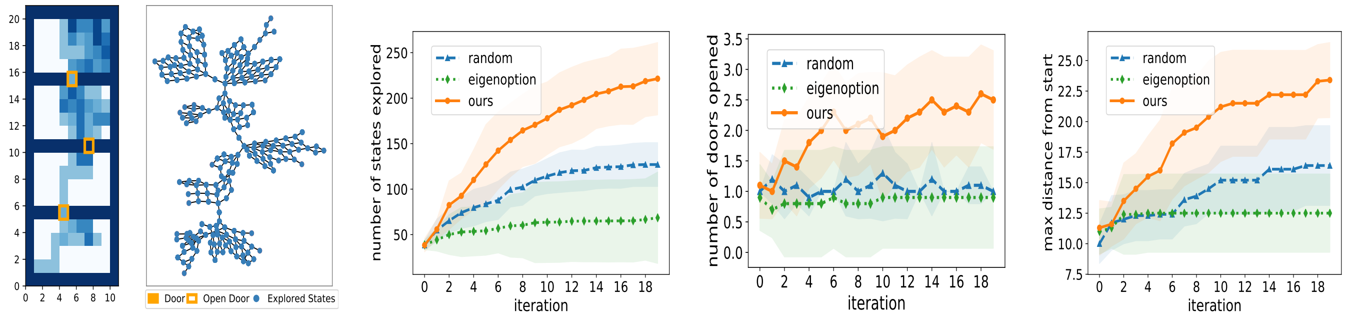
*Settings for Table 1:* The source room where the agent demonstrates and encodes the abstract options is the 2 Room variant. The 2-4 target rooms setting used for grounding the options are the 2 Rooms in Figure 6 (a), 3 Rooms in Figure 6 (b), and 4 Rooms (large) in Figure 6 (d) variants.

*Settings for Figure 2 and Figure 13:* The abstract $\psi$-SMDP is generated using the setting 4 Room (small) in Figure 6 (c), where the first 3 rooms each contain a key and a star is in the final room. For Figure 13, the task specifications are as follows: We refer to transfer as the task transfer, i.e., change of reward function. The total reward is discounted and normalized by the maximum reward achieved.

- dense reward (no transfer): the agent receives a reward for each key picked up, door opened, and star picked up, i.e., the reward vector $w_r = [1, 1, 1]$ over the features (key, open door, star).
- sparse reward (no transfer): the agent receives a reward for each door opened, i.e., the reward vector $w_r = [0, 1, 0]$.
- transfer (w. overlap): overlap refers to the overlap between reward function in the source and target task. In the source task, the agent receives a reward for each key picked up, and each door opened, i.e., the reward vector $w_r = [1, 1, 0]$. In the target task, the agent receives a reward for each door opened and star picked up, i.e., the reward vector $w_r = [0, 1, 1]$.
- transfer (w.o. overlap): In the source task, the agent receives a reward for each star picked up, i.e., the reward vector $w_r = [0, 0, 1]$. In the target task, the agent receives a reward for each key picked up, i.e., the reward vector $w_r = [1, 0, 0]$.

*Settings for Figure 5 and Figure 12:* The abstract $\psi$-SMDP is generated using the 3 Room setting in Figure 6 (b), where the first 2 rooms each contains a key and a star, and the final room contains a star. For figure 5, the task specifications is the same as the above description for Figure 13.

**Additional Experiment Results:** In this section, we present additional experimental results.



(a) State visitation (b) number of states explored (c) number of doors opened (d) distance from start

Figure 8: (Minecraft Door-Rooms) Exploring and grounding options in unknown environments. 4 rooms are connected by doors, the agent starts from the bottom, opens doors and enters the other rooms. The random agent takes random actions. Our agent starts with an iteration of random walk, then after each iteration, it computes the constructed MDP and fits the "go to and open door" option, (while the eigenoptions agent finds an eigenoption), and uses the options for exploration in the next iteration. (a) shows the state visitation frequency in iteration 20 of our agent (Algorithm 4), and the MDP constructed throughout the 20 iterations. (b)-(d) compares our agent with the baselines on the average number of the explored states, doors opened and max distance traversed. The results are averaged over 10 seeds and shaded regions show the standard deviation across seeds. Our agent using abstract successor options explores on average twice as many states as the baselines, as well as quickly learns to open doors and navigating to new rooms.

**Classic 4-Rooms.** Figure 7 shows the ground option policy learned by *IRL-batch* on the 4-Room domain. The option is learned by solving 1 linear program by *IRL-batch*, i.e., the state-action visitation frequencies returned by the IRL program corresponds to an optimal policy for all starting states. Please refer to the figure for details of the option.

**Minecraft Door-Rooms Experiments:** As introduced in Section 5, to test our batched option grounding algorithm (Algorithm 4) on new environments with unknown transition dynamics, we built two settings in the Malmo Minecraft environment: *Bake-Rooms* and *Door-Rooms*. The results on Bake-Rooms can be found in Figure 3 in the main text. Here, we present the results on the Door-Rooms setting shown in Figure 8.

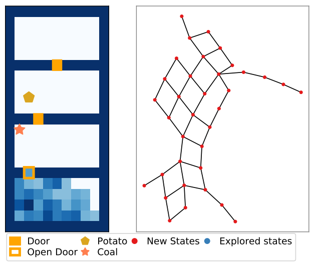
*Training and Results:* We compare our algorithm with the following two baselines: eigenoptions (Machado et al. 2017) and random walk. For this experiment, each agent runs for 20 iterations, with 200 steps per iteration as follows: In the first iteration, all agents execute randomly chosen actions. After each iteration, the agents construct an MDP graph based on collected transitions from all prior iterations. The eigenoption agent computes $k = 1$ eigenoption of the second smallest eigenvalue (Fiedler vector) using the normalized graph Laplacian, while our algorithm grounds the $k = 1$ abstract option: *open door and go to door*. In the next iteration, the agents perform random walks with both the primitive actions and the acquired options, update the MDP graphs, compute new options, . . .

Figure 8 shows our obtained results. Figure 8(a) shows the state visitation frequencies of our algorithm in the 20th iteration and the constructed MDP graph. The agent starts from the bottom room (R1), and learns to navigate towards the door, open the door and enter the next rooms. Figures 8(b)-(d) compare the agents in terms of the total number of states explored, number of doors opened and the maximum distance from the starting location. Note that the door layouts are different from the Bake-Rooms environment. Our agent explores on average more than twice as many states as the two baselines, quickly learns to open the doors and navigate to new rooms, while the baselines on average only learn to open the first door and mostly stay in the starting room.

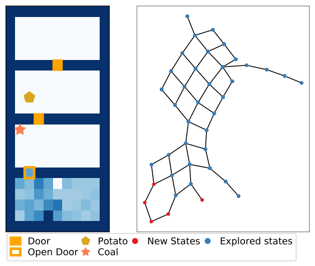
**More details on Minecraft Bake-Rooms:** Besides Figure 3, we now present more details of our option grounding algorithm in environments with unknown transition dynamics in the Minecraft Bake-Rooms experiment. Figures 9, 10 and 11 show the state visitation frequencies and MDP graph constructed over 20 iterations by our algorithm. The agent starts from the bottom room R1, a coal dropper is in R2, a potato dropper is in R3. The rooms are connected by doors which can be opened by the agent. For clarity of presentation, we show the undirected graph constructed. Blue nodes denote explored states and red nodes denote new states explored in the respective iteration.

The shown figures demonstrate that our agent learns to open the door, and open the door and enter R2 to collect coal in iteration 2, while the eigenoptions agent learns to collect coal in iteration 18, and the random agent collects a coal block in iteration 14. Our agent learns to collect potato in R3 in iteration 3, while the eigenoptions agent learns this in iteration 19, and the random agent has not reached R3 within 20 iterations.

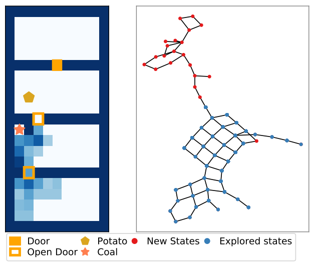
**Additional results on abstraction:** Figure 12 shows the abstract MDPs in the Object-Rooms with $N = 3$ rooms. Figure 12(a) is the abstract $\psi$-SMDP model induced by our approximate successor homomorphism, the colors of the nodes match their corresponding ground states in the ground MDP shown in Figure 12(b). The edges with temporal semantics correspond to abstract successor options and the option transition dynamics. To avoid disconnect graphs, we can augment the abstract successor options with shortest path options, which connect ground states of disconnected abstract states to their nearest abstract states. Figure 12(c) and (d) show the abstract MDPs induced by the $Q^*$-irrelevance (Q-all) and $a^*$-irrelevance (Q-optimal) abstraction



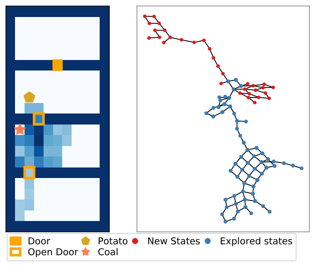
(a) iteration 0



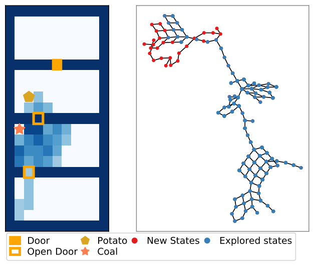
(b) iteration 1



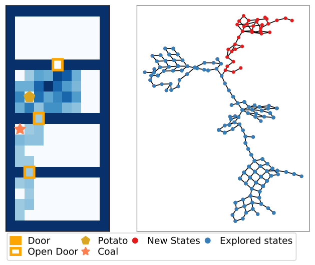
(c) iteration 2



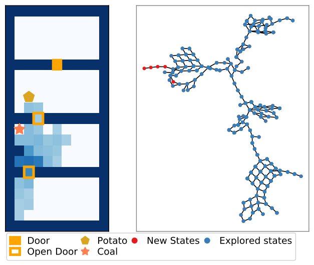
(d) iteration 3



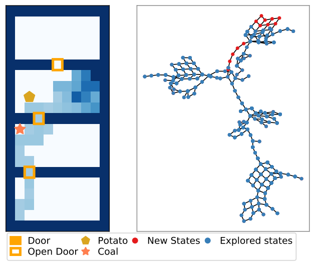
(e) iteration 4



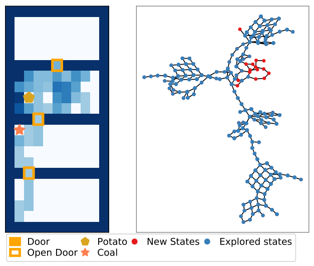
(f) iteration 5



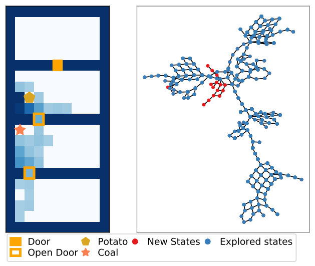
(g) iteration 6



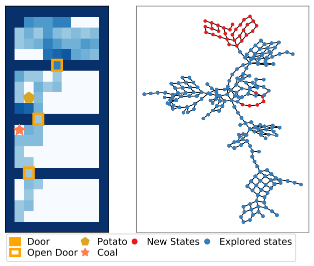
(h) iteration 7



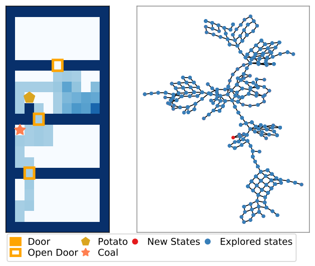
(i) iteration 8



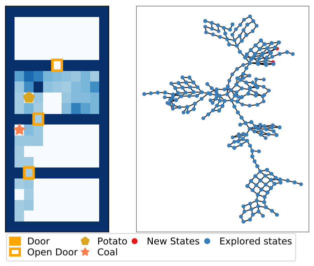
(j) iteration 9



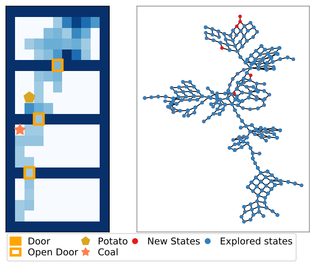
(k) iteration 10



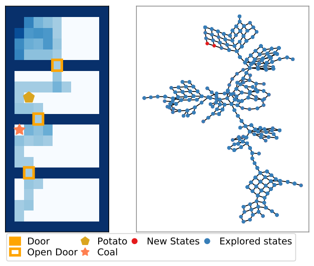
(l) iteration 11



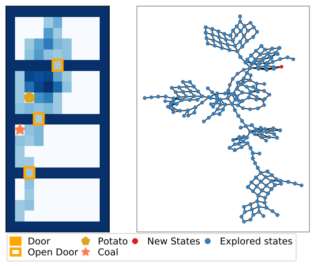
(m) iteration 12



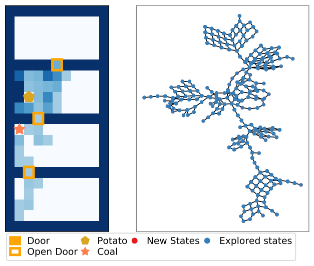
(n) iteration 13



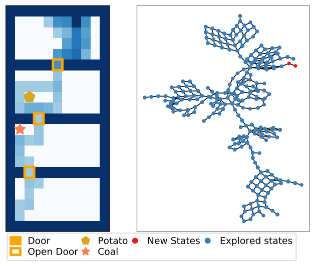
(o) iteration 14



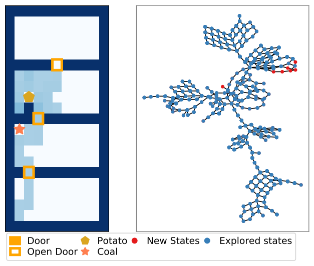
(p) iteration 15



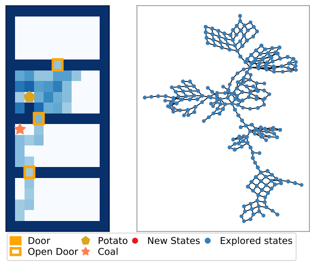
(q) iteration 16



(r) iteration 17

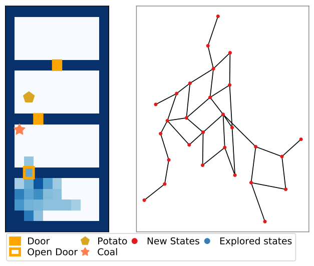


(s) iteration 18

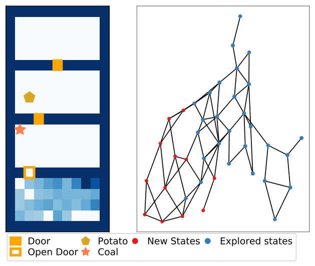


(t) iteration 19

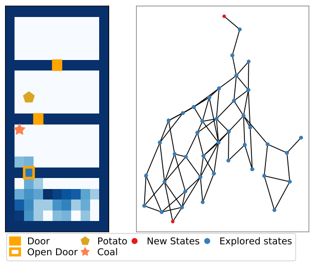
Figure 9: (Our Agent - Algorithm 4) State visitation frequencies and constructed graph per iteration in Minecraft Bake-Rooms



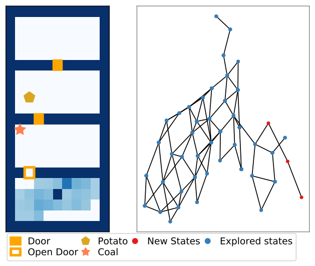
(a) iteration 0



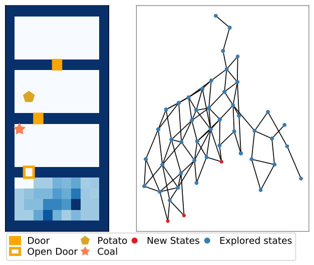
(b) iteration 1



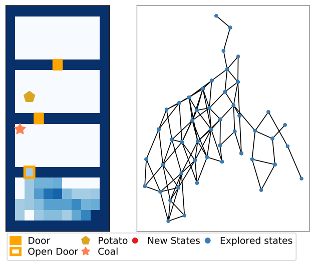
(c) iteration 2



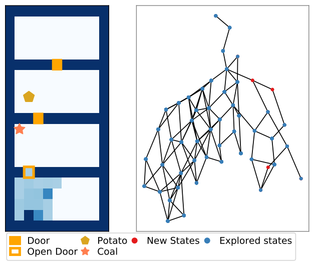
(d) iteration 3



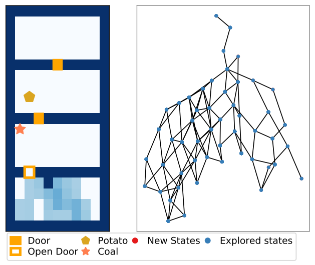
(e) iteration 4



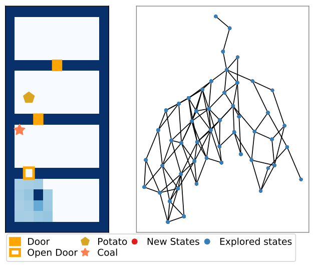
(f) iteration 5



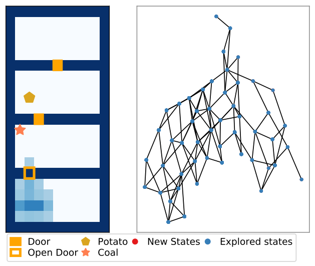
(g) iteration 6



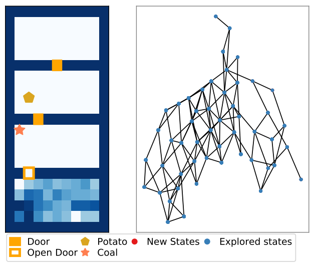
(h) iteration 7



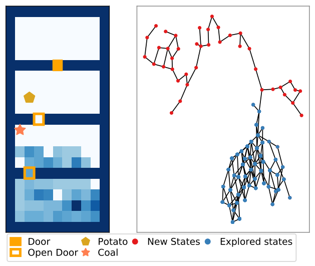
(i) iteration 8



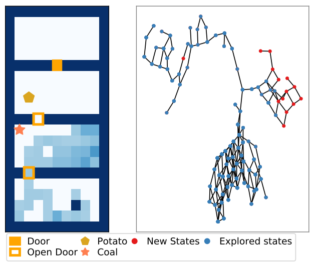
(j) iteration 9



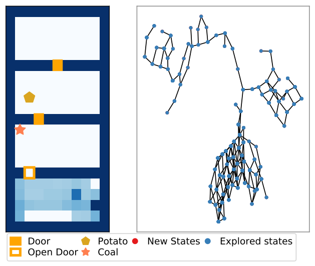
(k) iteration 10



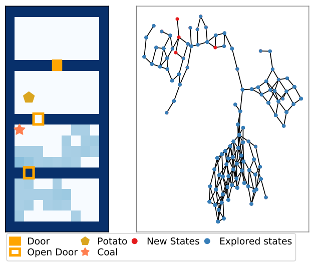
(l) iteration 11



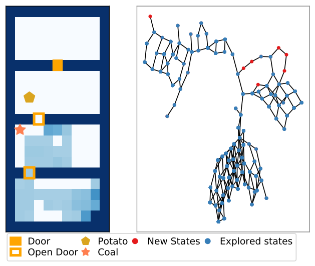
(m) iteration 12



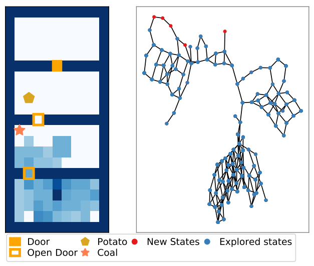
(n) iteration 13



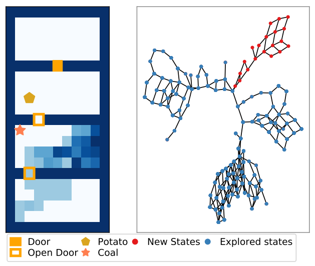
(o) iteration 14



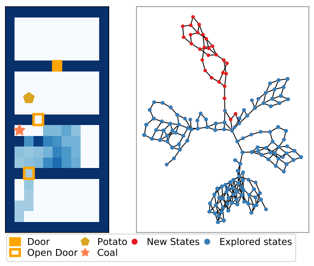
(p) iteration 15



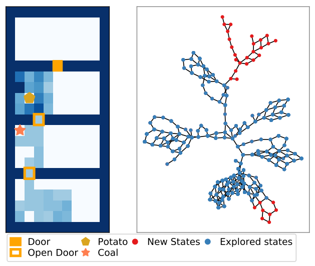
(q) iteration 16



(r) iteration 17

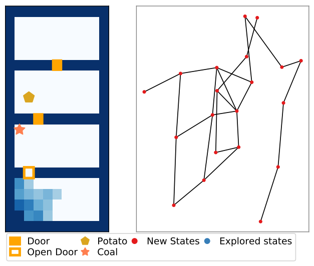


(s) iteration 18

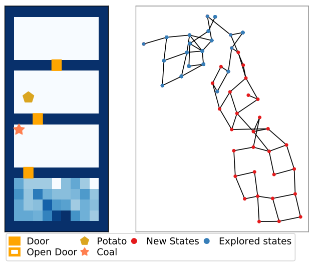


(t) iteration 19

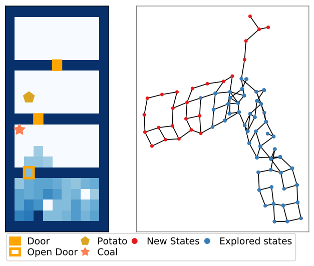
Figure 10: (Eigenoptions Agent) State visitation frequencies and constructed graph per iteration in Minecraft Bake-Rooms



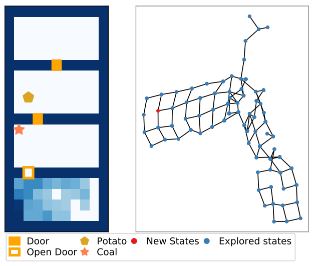
(a) iteration 0



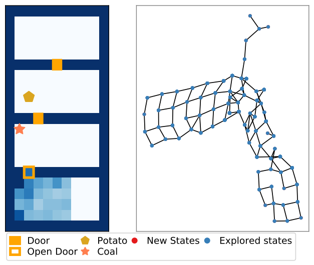
(b) iteration 1



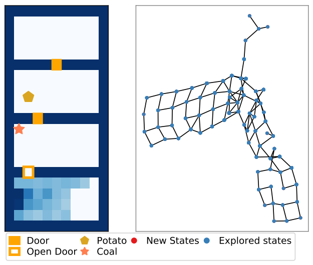
(c) iteration 2



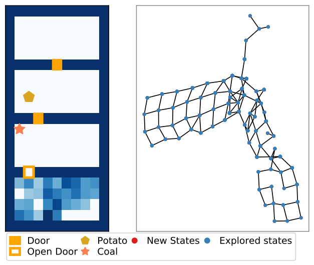
(d) iteration 3



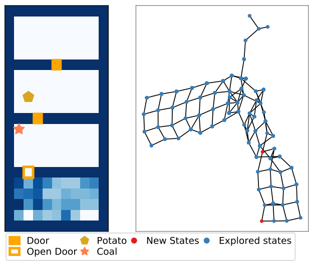
(e) iteration 4



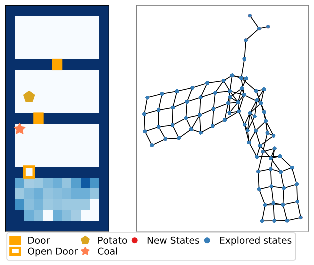
(f) iteration 5



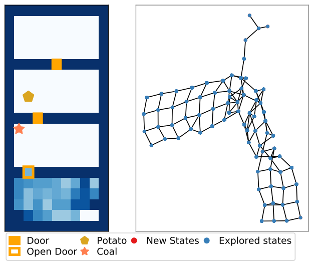
(g) iteration 6



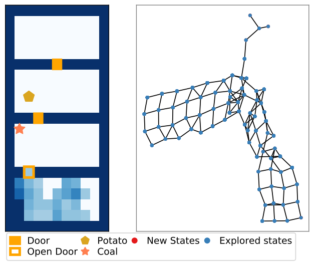
(h) iteration 7



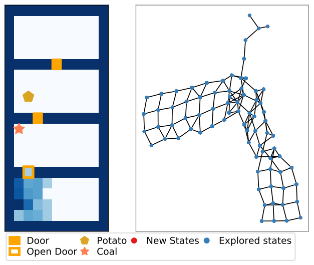
(i) iteration 8



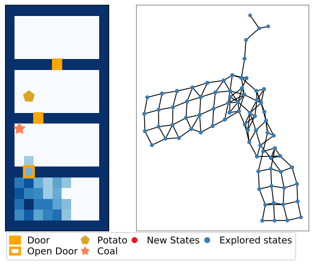
(j) iteration 9



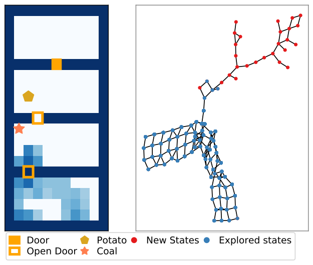
(k) iteration 10



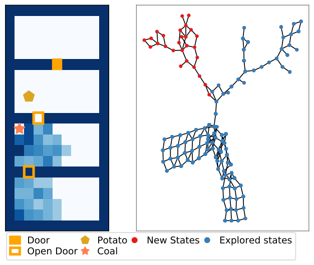
(l) iteration 11



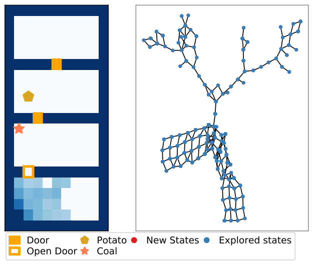
(m) iteration 12



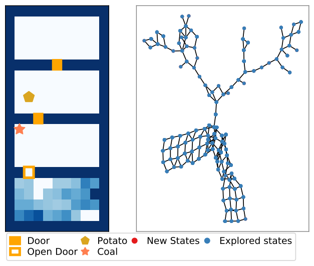
(n) iteration 13



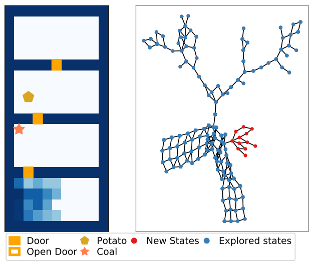
(o) iteration 14



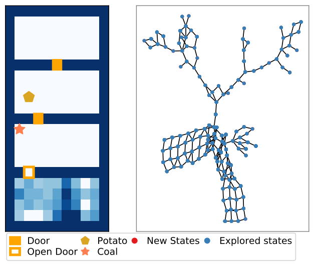
(p) iteration 15



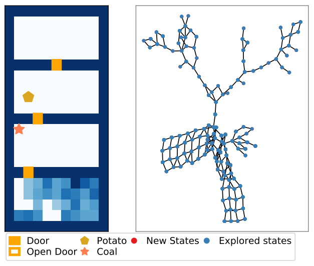
(q) iteration 16



(r) iteration 17

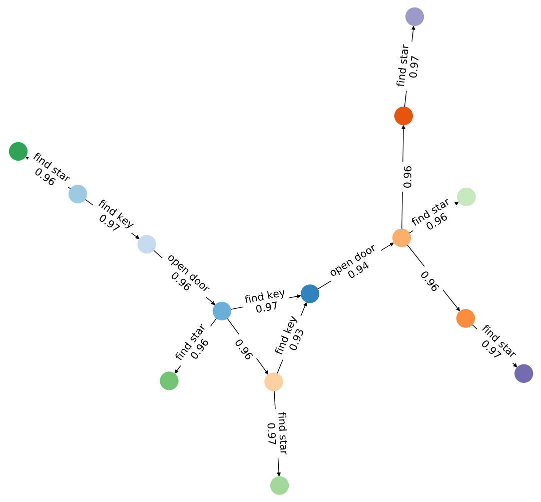


(s) iteration 18

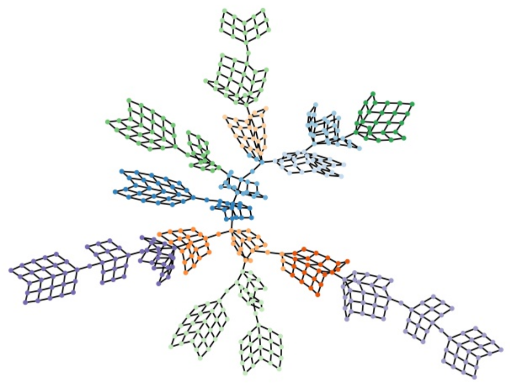


(t) iteration 19

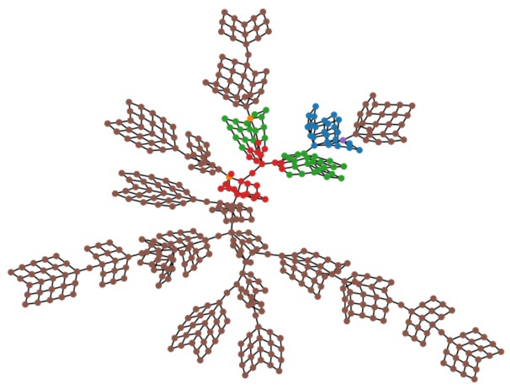
Figure 11: (Random Agent) State visitation frequencies and constructed graph per iteration in Minecraft Bake-Rooms



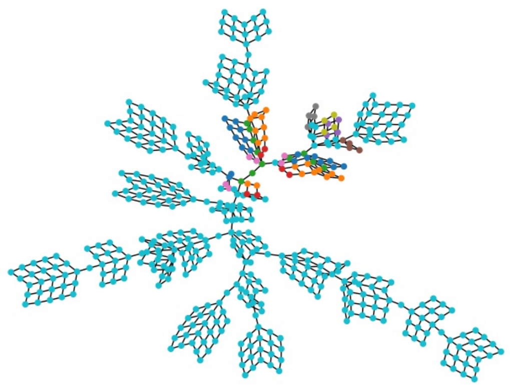
(a) Abstract $\psi$-SMDP



(b) Ground MDP



(c) Q-Abstraction w. All Actions

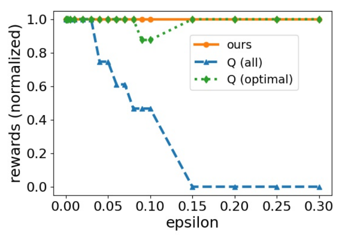


(d) Q-Abstraction w. Optimal Action

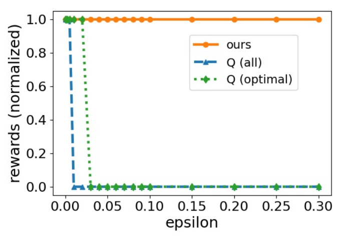
Figure 12: Abstract MDP with different abstraction schemes. (a) is the abstract MDP induced by our proposed successor homomorphism and (b) shows how the ground states are mapped to the abstract states. Node colors correspond to the abstract states in (a). (c) and (d) are abstraction induced by the $Q^*$-irrelevance and $a^*$-irrelevance abstraction schemes.

methods, for the task *find key*. The distance threshold $\epsilon = 0.1$.

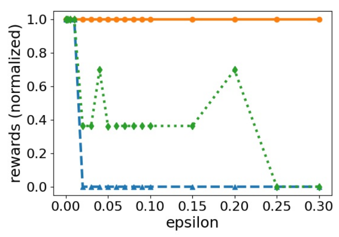
Figure 13 shows the results of using the abstract MDP for planning in the Object-Rooms with $N = 4$ rooms. Please refer to Section A.6 for a detailed description of the settings. Our successor homomorphism model performs well across all tested settings with few abstract states (number of clusters). Since successor homomorphism does not depend on rewards, the abstract model can transfer across tasks (with varying reward functions), and is robust under sparse rewards settings. Whereas abstraction schemes based on the reward function perform worse when the source task for performing abstraction is different from the target task where the abstract MDP is used for planning.



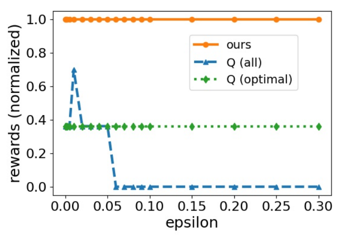
(a) no transfer (dense reward)



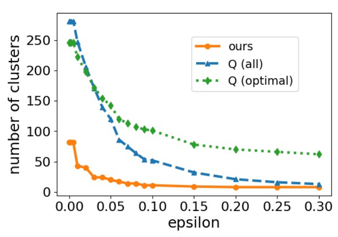
(b) no transfer (sparse reward)



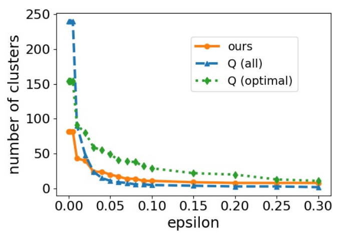
(c) transfer (w. overlap)



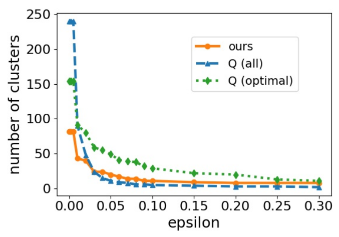
(d) transfer (w.o. overlap)



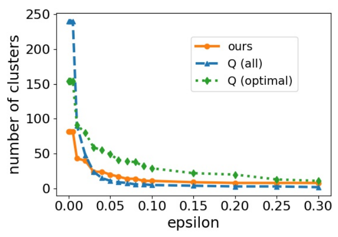
(e) no transfer (dense reward)



(f) no transfer (sparse reward)



(g) transfer (w. overlap)



(h) transfer (w.o. overlap)

Figure 13: Performance of planning with the abstract MDPs. The upper row shows the total rewards (normalized by the maximum possible total rewards) obtained, and the lower row shows the corresponding number of abstract states of the abstract MDP. The x-axes are the distance thresholds $\epsilon$. transfer refers to task transfer (i.e., different reward function)