HIERARCHICAL REINFORCEMENT LEARNING WITH MULTI-LEVEL STATE-ACTION ABSTRACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

State and action abstractions can help reinforcement learning agents scale to complex environments. State abstraction groups states that can be treated equivalently, action abstraction allows agents to act with temporally extended behaviours, and combining the two at multiple levels of abstraction enables agents to learn over compact representations of an environment. Theoretical foundations exist for hierarchical state-action abstractions, but discovering such hierarchies and learning over them are open problems. We address both, proposing a temporal-difference learning architecture for multi-level state-action abstractions, with a restricted bootstrap rule that prevents value estimates at more abstract layers from biasing learning at less abstract ones. The architecture is agnostic to the discovery method, requiring only a hierarchy of nested state partitions. We construct hierarchies using graph partitioning methods. We find learning with the architecture is more sample-efficient than with the corresponding action-only abstraction in most cases, and is competitive with or better than established option discovery methods.

1 INTRODUCTION

Good abstractions allow an agent to focus on the core aspects of a problem and disregard everything else, and are widely seen as fundamental to intelligence. Two commonly explored types of abstraction are over states and actions. State abstraction compresses the state space by grouping states that can be treated equivalently, and action abstraction allows for temporally extended behaviours. Combining the two enables planning and learning over a smaller and simpler representation of the environment.

Abel et al. (2020) formalised ϕ-relative options as a structure for defining state-action abstractions and proved value-loss bounds for the resulting hierarchies. These results provide a strong foundation for creating a useful state-action abstraction hierarchy but leave two questions open: How should an autonomous agent discover useful state-action abstractions? And how should it learn with the hierarchy? Our work addresses both.

We propose a temporal-difference learning architecture that operates over a multi-level state-action abstraction hierarchy. The architecture has two key mechanisms. First, a restricted bootstrap rule allows value estimates to propagate from less abstract to more abstract layers but not the reverse, so learning at the ground layer reduces to standard Q-learning over the underlying problem. Second, primitive actions are still available at every state, so any behaviour possible in the underlying problem remains possible by construction, including an optimal one.

Our contributions are: a hierarchical reinforcement learning architecture for multi-level state-action abstraction, with a restricted bootstrap learning rule; and two instantiations of the architecture based on graph partitioning methods. We evaluate the proposed architecture in six environments. Learning with multi-level state-action abstraction is more sample-efficient than learning with the corresponding action-only abstraction in most environments, including the two largest. The benefit increases with the size of the state space, and the architecture is competitive with or better than established option discovery methods. We demonstrate that the learning architecture remains beneficial when the state-action abstraction is constructed incrementally during online interaction.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

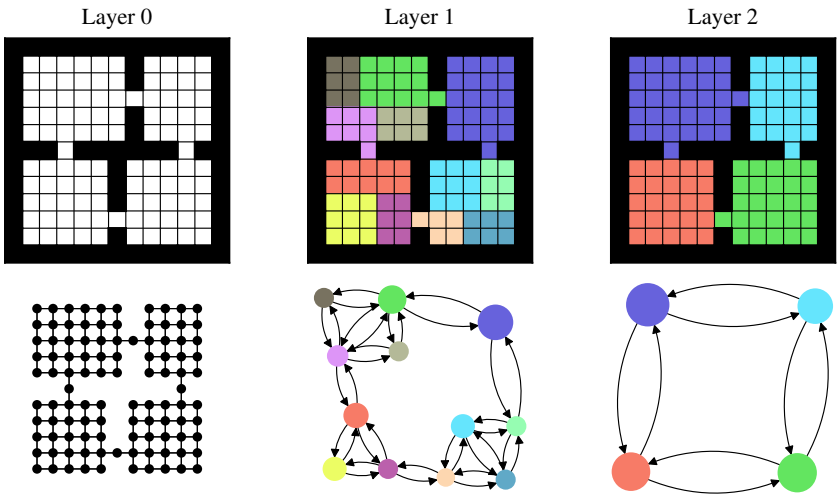


Figure 1: An illustrative state-action abstraction formed in the `Four Rooms` environment. The state abstraction at each layer is shown in the top row and as graph nodes on the bottom row. The corresponding action abstractions are represented by graph edges.

2 Preliminaries

We use $\Delta(\mathcal{X})$ to denote the set of probability distributions over a set $\mathcal{X}$. We consider sequential decision making in a Markov decision process (MDP; Puterman, 1994; Sutton and Barto, 2018), denoted by a tuple $M = \langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$, where $\mathcal{S}$ is a set of states; $\mathcal{A}$ is a set of actions; R is the reward function, where taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and transitioning to state $s' \in \mathcal{S}$ gives a scalar reward $r \in \mathbb{R}$ with expected value $R(s, a, s')$; the transition function $P(s'|s, a)$ gives the probability of transitioning to state s' when taking action a in state s; and $\gamma \in [0, 1]$ is the discount factor. At decision step $t \geq 0$, the agent observes state $S_t \in \mathcal{S}$ and executes action $A_t \in \mathcal{A}$ according to policy $\pi \colon \mathcal{S} \to \Delta(\mathcal{A})$. It then transitions into state $S_{t+1} \in \mathcal{S}$ and receives a scalar reward R_{t+1}. The objective is to learn a policy that maximises the expected discounted return $\mathbb{E}_\pi [G_t]$, where $G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.

In addition to the one-step actions $\mathcal{A}$, which we refer to as *primitive actions*, we define a set of temporally extended actions $\mathcal{O}$ in the form of options (Sutton, Precup and Singh, 1999; Precup, 2000). An *option* $o \in \mathcal{O}$ is a tuple $\langle \mathcal{I}_o, \pi_o, \beta_o \rangle$ where $\mathcal{I}_o \subseteq \mathcal{S}$ is the initiation set, specifying the set of states where the option can start execution; $\pi_o \colon \mathcal{S} \to \Delta(\mathcal{A})$ is the option policy; and $\beta_o \colon \mathcal{S} \to [0, 1]$ is the termination condition, specifying the probability of termination in a given state. Primitive actions can be represented as options that deterministically terminate after one step. This work builds on the ϕ-relative state-action abstractions of Abel et al. (2020). Definitions 1 to 4 recall, adapt, and extend their formalisms for our application.

Definition 1 (Action abstraction)**.** *An action abstraction of an MDP $\langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$ is a set of options $\mathcal{O}$ replacing the primitive action set $\mathcal{A}$, producing a semi-Markov decision process (SMDP; Sutton, Precup and Singh, 1999) $\langle \mathcal{S}, \mathcal{O}, R, P, \gamma \rangle$. In contrast to Abel et al. (2020), we include the primitive action set $\mathcal{A}$ in $\mathcal{O}$.*

The objective in the resulting SMDP is to learn a policy $\pi \colon \mathcal{S} \to \Delta(\mathcal{O})$ that maximises the expected discounted return. A state abstraction maps the ground state space of an MDP into a more compact representation, allowing agents to learn and make decisions more efficiently (Li, Walsh and Littman, 2006). We adopt the multi-level state abstraction formulation of Abel et al. (2020), with $\mathcal{S}_0$ denoting the *ground state space* and $\mathcal{S}_i$ denoting an *abstract state space* at layer $i \geq 1$.

Definition 2 (State abstraction)**.** *A state abstraction function $\phi_i \colon \mathcal{S}_{i-1} \to \mathcal{S}_i$, for $i \geq 1$, maps each state from layer $i - 1$ to a more abstract state at layer i. At the ground layer we take $\phi_0 \colon \mathcal{S}_0 \to \mathcal{S}_0$ to be the identity map, applying no abstraction. The composite abstraction $\Phi_i \colon \mathcal{S}_0 \to \mathcal{S}_i$ maps a ground state to its layer i abstraction by $\Phi_i(s) \doteq \phi_i(\phi_{i-1}(\dots \phi_0(s) \dots))$ for $i \geq 0$.*

We use $\Phi_i^{-1}(\bar{s}_i)$ for the set of ground states that map to $\bar{s}_i$, and bar notation to denote abstract states belonging to $\mathcal{S}_i$ (e.g. $\bar{s}_i, \bar{u}_i$). Defining state abstraction ϕ_i as a function on $\mathcal{S}_{i-1}$ yields an abstraction

hierarchy. Every state has an abstract representation at every layer of the hierarchy, and each abstract state at layer $(i-1)$ is wholly contained in a single abstract state at layer i. Abel et al. (2020) introduce ϕ-relative options as those that initiate in a single abstract state, terminate in all other abstract states, and are defined over the preceding layer's state space.

Definition 3 (ϕ-relative options)**.** *Given a state abstraction $\phi_i \colon \mathcal{S}_{i-1} \to \mathcal{S}_i$, a ϕ_i-relative option o at layer i acts over the layer $(i-1)$ state space $\mathcal{S}_{i-1}$ and option set $\mathcal{O}_{i-1}$, with option policy $\pi_o \colon \mathcal{S}_{i-1} \to \Delta(\mathcal{O}_{i-1})$. A set of ϕ_i-relative options $\mathcal{O}_i$ contains at least one option for every abstract state $\bar{s}_i \in \mathcal{S}_i$. Each option $o = \langle \mathcal{I}_o, \pi_o, \beta_o \rangle$ associated with the abstract state $\bar{s}_i$ is defined as:*

$$\mathcal{I}_o = \{\bar{s}_{i-1} \in \mathcal{S}_{i-1} \mid \phi_i(\bar{s}_{i-1}) = \bar{s}_i\}, \quad \beta_o(\bar{s}_{i-1}) = 1 \text{ if } \bar{s}_{i-1} \notin \mathcal{I}_o, \text{ and } 0 \text{ otherwise.}$$

We refer to i as the *layer* of an option, with primitive actions at layer 0. Individual option policies are left open, but we require each policy to reach an abstract state that satisfies its termination condition with probability 1. We apply a ϕ-relative option's termination function β_o and policy π_o to ground state $s \in \mathcal{S}_0$ with $\beta_o(s) \doteq \beta_o(\Phi_{i-1}(s))$ and $\pi_o(\cdot \mid s) \doteq \pi_o(\cdot \mid \Phi_{i-1}(s))$ respectively, for any option o at layer i. We use $\mathcal{I}_o^{\Downarrow} \subseteq \mathcal{S}$ to denote the ground initiation set of an option: for a layer i option that initiates in abstract state $\bar{s}_i \in \mathcal{S}_i$, $\mathcal{I}_o^{\Downarrow} = \Phi_i^{-1}(\bar{s}_i)$. Any deterministic ϕ-relative option policy induces a deterministic primitive action at every ground state in its initiation set (Abel et al., 2020, Remarks 1 and 2), which we denote $\pi_o^{\Downarrow}(s) \in \mathcal{A}$.

Definition 4 (ϕ-relative hierarchy)**.** *A ϕ-relative hierarchy is a multi-level state-action abstraction. A ϕ-relative hierarchy of depth L, $H_L = (\phi_{0:L}, \mathcal{O}_{0:L})$, consists of a list of state abstractions $\phi_{0:L} = (\phi_0, \phi_1, \ldots, \phi_L)$ and a list of ϕ-relative option sets $\mathcal{O}_{0:L} = (\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_L)$, with $\mathcal{O}_0 = \mathcal{A}$.*

This is the $(\phi, \mathcal{O}_\phi)$-Hierarchy of Abel et al. (2020, Definition 6) but extended to include the ground state and action spaces at layer 0. Each layer in the hierarchy pairs a state abstraction with options for navigating within and between its abstract states, so state and action abstractions are coupled by construction at every level. Next we address two gaps left open by these formalisms: discovering a ϕ-relative hierarchy (Section 3.1) and learning with it (Section 3.2).

3 Proposed Approach

3.1 Producing State and Action Abstractions

We first produce a hierarchical partitioning of the state space in a reward-free setting using some multi-level state abstraction $\phi_{0:L}$. This results in an abstract state space $\mathcal{S}_i$ at each layer of the L-level abstraction hierarchy, with $\mathcal{S}_0$ the ground state space and $\mathcal{S}_L$ as the most abstract representation. Next, we construct the ϕ_i-relative option sets $\mathcal{O}_i$, defining skills that navigate between neighbouring abstract states at each layer of the hierarchy. Option sets are built layer by layer, starting at layer 1 and finishing at layer L. At layer 1, we define an option from abstract state $\bar{u}_1 \in \mathcal{S}_1$ to abstract state $\bar{v}_1 \in \mathcal{S}_1$ if at least one transition (s, a, s') exists with state s inside $\bar{u}_1$ and state s' inside $\bar{v}_1$. The option's policy is defined over the ground states and primitive actions. At each layer $i > 1$, we apply the same construction one level up. We define a new layer i option from $\bar{u}_i$ to $\bar{v}_i$ if at least one option in $\mathcal{O}_{i-1}$ transitions from a layer $(i-1)$ state inside $\bar{u}_i$ to one inside $\bar{v}_i$, with a policy that selects among layer $(i-1)$ options. Formally, for each directed pair of neighbouring abstract states $(\bar{u}_i, \bar{v}_i)$, we define a new ϕ_i-relative option $o = \langle \mathcal{I}_o, \pi_o, \beta_o \rangle$ over $\mathcal{S}_{i-1}$ and $\mathcal{O}_{i-1}$ for moving from $\bar{u}_i$ to $\bar{v}_i$:

$$\mathcal{I}_o = \{\bar{s}_{i-1} \in \mathcal{S}_{i-1} \mid \phi_i(\bar{s}_{i-1}) = \bar{u}_i\}, \quad \beta_o(\bar{s}_{i-1}) = 1 \text{ if } \bar{s}_{i-1} \notin \mathcal{I}_o, \text{ and } 0 \text{ otherwise.}$$

This formulation naturally satisfies the definition of a ϕ-relative option (Abel et al., 2020). The option policy $\pi_o : \mathcal{S}_{i-1} \to \Delta(\mathcal{O}_{i-1})$ is defined so as to reach $\bar{v}_i$ from $\bar{u}_i$ using layer $(i-1)$ options. Each option belongs to a single layer of the hierarchy. A layer i option is a distinct, newly constructed option, not promoted or re-used from the layer below: it composes layer $(i-1)$ options into new behaviour, realised by no single layer $(i-1)$ option, in the same way that a layer 1 option composes primitive actions. The specific training objective is given in Appendix D.2.

We use the Louvain algorithm (Blondel et al., 2008), a modularity-maximising graph partitioning method, applied to the state-transition graph, as the basis for state-action abstraction. Three considerations motivate this choice. First, the Louvain algorithm produces a multi-level partitioning in which one or more clusters at layer $(i-1)$ are merged into a single cluster at layer i, directly producing

a hierarchical state abstraction (Definition 2). Second, prior work has already used the Louvain algorithm to construct effective multi-level *action* abstraction hierarchies (Evans and Şimşek, 2023). Third, Evans (2026) argues that the connection between modularity and Markov stability (Delvenne, Yaliraki and Barahona, 2010; Lambiotte, Delvenne and Barahona, 2014) makes the Louvain algorithm particularly well-suited to building useful action abstractions for reinforcement learning. Figure 1 shows an illustrative multi-level state-action abstraction produced from a Louvain partitioning. The proposed approach can also be used with other partitioning algorithms, and we present empirical results for an alternative based on spectral clustering in Section 5.

3.2 Learning a Policy over Options

We write $\mathcal{O}_i(\bar{s}_i)$ for the set of layer i options that initiate in $\bar{s}_i$. At every ground state s, the agent's *choice set* $\mathcal{O}(s) = \bigcup_{i=0}^{L} \mathcal{O}_i(\Phi_i(s))$ is the union of the options available at every layer of the depth L hierarchy. The behaviour policy μ samples an option from the choice set $\mathcal{O}(s)$. We require μ to be a soft policy, so that every option, including primitive actions, is selected with non-zero probability at every state where it is available. The selected option runs until its termination condition is satisfied. The agent learns a value function $Q_i \colon \mathcal{S}_i \times \mathcal{O}_i \to \mathbb{R}$ at each layer $i \in \{0, 1, \ldots, L\}$, using both SMDP updates and intra-option updates (Sutton, Precup and Singh, 1999) as detailed below. The greedy policy $\tilde{\pi}$ selects an available option $o \in \mathcal{O}(s)$ (over all layers in the hierarchy) with the highest value $Q_i(\Phi_i(s), o)$, where i is the layer of option o.

Bootstrapping with a hierarchy of abstractions. Temporal-difference learning uses bootstrapping when computing update targets. If we used the full choice set $\mathcal{O}(s)$ for bootstrapping, the averaging effect of state abstraction would bias the targets. An abstract option's value estimate is, in effect, a weighted average over its ground initiation set $\mathcal{I}_o^{\Downarrow}$ and so will overestimate the value at some of those states. Standard bootstrapping would take the maximum over the full choice set, making it a one-sided error that corrupts value estimates at every lower layer, including the ground layer. To prevent this, updates at layer i bootstrap only from values of options at layer i or below. This means that bootstrap estimates can propagate up the abstraction layers, from fine to coarse, but estimates at coarse levels of abstraction do not bias the lower layers. Specifically, we define the restricted bootstrap value $\overline{V}_i(s)$ as:

$$\overline{V}_i(s) = \max_{i' \le i,\ o \in \mathcal{O}_{i'}(\Phi_{i'}(s))} Q_{i'}(\Phi_{i'}(s), o). \tag{1}$$

SMDP updates. SMDP updates are applied whenever an option o at layer i terminates. Let $(s_\tau, \ldots, s_{\tau+K})$ be the sequence of ground states visited during its execution and $G_\tau = \sum_{t=\tau}^{\tau+K-1} \gamma^{t-\tau} r_{t+1}$ be its accumulated discounted return. The option value is updated with learning rate α_{smdp}, using the restricted bootstrap value at the option termination state $s_{\tau+K}$:

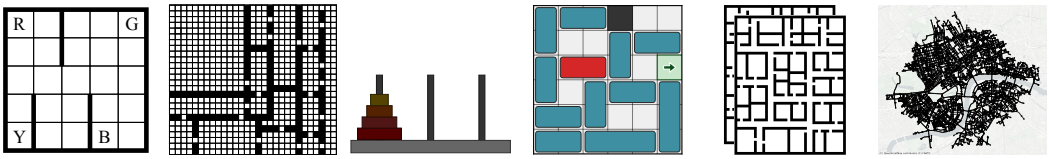
$$Q_i(\Phi_i(s_\tau), o) \leftarrow Q_i(\Phi_i(s_\tau), o) + \alpha_{\text{smdp}} \left[G_\tau + \gamma^K \overline{V}_i(s_{\tau+K}) - Q_i(\Phi_i(s_\tau), o) \right].$$

Off-policy intra-option updates. Intra-option updates are applied after every transition (s, a, r, s'). We perform an update for every option $o \in \mathcal{O}(s)$ whose induced ground policy would have executed primitive action a, that is, where $\pi_o^{\Downarrow}(s) = a$. The option value is then updated with learning rate α_{intra}:

$$Q_i(\Phi_i(s), o) \leftarrow Q_i(\Phi_i(s), o) + \alpha_{\text{intra}} \left[r + \gamma U_i(s', o) - Q_i(\Phi_i(s), o) \right], \text{ where}$$
$$U_i(s, o) = (1 - \beta_o(s)) \cdot Q_i(\Phi_i(s), o) + \beta_o(s) \cdot \overline{V}_i(s).$$

Remark 1 (Convergence of Q_0 to optimal values Q^*)**.** *At layer* 0*,* Φ_0 *is the identity mapping (i.e.* $\mathcal{S}_0 = \mathcal{S}$*), and* $\mathcal{O}_0 = \mathcal{A}$*, so the restricted bootstrap value* $\overline{V}_0(s) = \max_{a \in \mathcal{A}} Q_0(s, a)$ *depends only on* Q_0 *itself. The layer* 0 *update is therefore standard tabular Q-learning over the ground MDP, and* Q_0 *converges to* Q^* *provided that every state-action pair is visited infinitely often and the learning rates satisfy the Robbins–Monro conditions (Watkins and Dayan, 1992; Sutton and Barto, 2018).*

Sufficient visitation comes from three properties of the method: the behaviour policy is soft; every option policy reaches an abstract state that satisfies its termination condition with probability 1; and the primitive actions are retained. Together, these properties ensure that the agent reaches a decision point in finite time (or the episode terminates) with a non-zero probability of selecting each primitive action at a decision point. Since primitive actions terminate after one step, the next state is also a decision point, and so, in the limit, every state-action pair is updated infinitely often.



Taxi (404 states) | Maze (617 states) | Towers of Hanoi (2187 states) | Rush Hour (2332 states) | Office (2 Floors) (2698 states) | London Streets (5000 states)

Figure 2: The environments used in the empirical analysis, ordered by size of state space.

The choice set, behaviour policy, and two update rules together define the learning procedure (pseudocode in Appendix A). The agent starts in ground state s, samples an option from the choice set at s, executes it through any nested sub-options until it terminates at ground state s', then samples a new option. Intra-option updates are applied after every ground transition, and SMDP updates are applied at option termination, all using the restricted bootstrap value of Equation (1). Illustrative abstract state values found with these update rules on the abstraction hierarchy of Figure 1 are shown in Appendix B.

4 Related Work

Ravindran and Barto (2003) introduced SMDP homomorphisms as a formal framework for state-action equivalence under temporal abstraction. Konidaris, Kaelbling and Lozano-Pérez (2014) established that an agent's skills determine the state abstractions it requires for planning, and Konidaris (2016) extended this result into an iterative skill-symbol loop that builds successive abstraction layers. Gopalan et al. (2017) formalised abstract MDPs as a hierarchical planning framework, and Rodriguez-Sanchez and Konidaris (2024) learn a continuous state abstraction sufficient for value-preserving planning given a fixed option set. Abel et al. (2020) formalise ϕ-relative options as a class of state-action abstraction and prove that, under structural conditions, they preserve the ability to represent near-optimal policies. We build on the ϕ-relative formalism, considering how to autonomously discover and learn with ϕ-relative state-action abstractions.

The options framework (Sutton, Precup and Singh, 1999; Precup, 2000) formalises temporally extended actions, and a substantial literature considers how to discover useful options. Amongst these, graph-theoretic approaches (Menache, Mannor and Shimkin, 2002; Mannor et al., 2004; Şimşek and Barto, 2004; Şimşek, Wolfe and Barto, 2005; Şimşek and Barto, 2008) locate subgoals at structurally important nodes such as bottlenecks between weakly connected regions of the state space, while spectral and geometric variants use Laplacian eigenvectors (Machado, Bellemare and Bowling, 2017), manifold structure (Bar, Talmon and Meir, 2020) or successor representations (Ramesh, Tomar and Ravindran, 2019). Methods targeting exploration minimise cover-time (Jinnai et al., 2019) or use Laplacian-based novelty (Theophilou and Şimşek, 2025), while skill chaining constructs goal-directed option chains (Konidaris and Barto, 2009; Bagaria and Konidaris, 2020). Closest to our setting, Evans and Şimşek (2023) build a multi-level option hierarchy using the Louvain algorithm to find modularity-maximising partitions of the state-transition graph. They use these partitions only to define action abstractions, and their options do not meet the definition of ϕ-relative options because they terminate on reaching a target cluster rather than on leaving their initiation cluster. In contrast, we use the same partitioning algorithm but treat the resulting partitions as the basis for a multi-level state-action abstraction, in the form of a ϕ-relative hierarchy.

5 Empirical Analysis

The following questions motivate our empirical analysis: Does multi-level state-action abstraction improve learning? How do the results change as the number of states in the environment increases? What is the impact of multi-level abstraction? What is the impact of the partitioning method? What is the impact of the restricted bootstrap rule? Are hierarchical state-action abstractions useful when they are discovered through online interaction?

We evaluate the proposed approach in six environments: `Taxi` (Dietterich, 2000), `Maze` (Ramesh, Tomar and Ravindran, 2019), `Towers of Hanoi`, `Rush Hour`, `Office` (Evans and Şimşek, 2023), and `London Streets` (adapted from Theophilou and Şimşek, 2025). Environment visuali-

sations and state-space sizes are shown in Figure 2, and full environment details are in Appendix C. In each environment, the agent receives a reward of -0.001 per ground transition, and an additional reward of $+1$ for reaching a goal state, with discount factor $\gamma = 0.99$ throughout. We compare our approach against established option discovery methods: Louvain (Evans and Şimşek, 2023), betweenness (Şimşek and Barto, 2008), eigenoptions (Machado, Bellemare and Bowling, 2017), diffusion options (Bar, Talmon and Meir, 2020), and covertime options (Jinnai et al., 2019). We also compare to an agent with only primitive actions. The Louvain baseline uses the same multi-level partitioning as our method, and therefore the same options, but uses action abstraction only. Thus, the Louvain baseline serves as an ablation, isolating the contribution of learning with state abstraction and restricted bootstrapping. All methods share learning rates, exploration parameters, and the evaluation protocol, with the exception of a uniform-random behaviour policy for eigenoptions (Machado, Bellemare and Bowling, 2017). The primary hyperparameter of our method, the Louvain resolution ρ, is held fixed across environments and baselines, with sensitivity reported in Appendix E.4. Full details of parameters and baselines are given in Appendix D.

Each agent is trained for a number of epochs, where an epoch is a fixed number of transitions in the ground MDP. We evaluate the agent's current greedy policy at the end of each epoch, with no learning updates applied during evaluation. Evaluation episodes are run until termination or a maximum episode length is reached. In `Taxi`, `Maze`, `Office` and `London Streets`, we train and evaluate the agent on 50 tasks, each a randomly sampled start-goal state pair, and record the evaluation performance for each task. In `Rush Hour` and `Towers of Hanoi`, we use the single canonical task defined by the environment. Every agent is trained and evaluated on the same set of tasks in each environment, with Q-values reset to 0 for each task. We use epoch lengths of 250 transitions for `Maze`, `Rush Hour`, `Office`, and `London Streets`; 100 transitions for `Towers of Hanoi`; and 25 transitions for `Taxi`. We use 20 agent seeds. For each seed, we generate a new abstraction hierarchy (state-action for our architecture, action-only for baselines) which is held fixed for all tasks. For all methods, we discover and pre-train option policies offline in a reward-free setting, with no goal states except `Taxi`'s canonical sink-state after passenger drop-off. This separation ensures the options reflect the environment structure rather than task structure. We report performance as the interquartile mean of the normalised discounted return achieved during evaluation across all tasks, where the uniform-random policy achieves 0 and an optimal policy achieves 1. We report 95% confidence intervals on the interquartile mean with stratified bootstrapping using the `rliable` library (Agarwal et al., 2021). We use the label $(\phi, \mathcal{O})$ for state-action abstraction (our proposed method) and $(\mathcal{O})$ for action-only abstraction.

Does multi-level state-action abstraction improve learning? Figure 3 compares our method against all baselines across the six environments. Compared with the corresponding action-only abstraction baseline, which uses the same options, learning with state-action abstraction is more sample-efficient in four of six environments, including the two largest, and has similar sample efficiency in `Taxi`, the smallest environment, consistent with our scaling results (Figure 4), and in `Rush Hour`, where both variants reach their converged performance very early in training, leaving little room for improvement. Converged performance is slightly lower in `Rush Hour` and `London Streets`, consistent with the averaging effect of state abstraction, under which an option may appear greedy at ground states where it is not the optimal choice. Compared with all baselines, learning with state-action abstraction is among the two most sample-efficient in every environment, and the most sample-efficient in `Office` and `London Streets`, the two largest. The baselines are inconsistent across environments. Covertime is the most sample-efficient method in `Maze`, `Towers of Hanoi`, and `Rush Hour`, but is substantially less sample-efficient than our method in `Taxi` and `London Streets`, and performs poorly in `Office`. Eigenoptions is the most sample-efficient method in `Taxi`, but fails to learn within the training budget in `Towers of Hanoi`, `Office`, and `London Streets`. Diffusion and betweenness options learn more slowly than our method in every environment, although both reach higher asymptotic performance in `Rush Hour` and `London Streets`. No single baseline is competitive with our method across all six environments.

What is the impact of environment size? We explore the impact of the size of the environment state space on learning performance by running scaling experiments in `Towers of Hanoi` and `Office`. In `Towers of Hanoi`, we scale the state space by adding an extra disc at each scaling step, starting from 6 discs; in `Office`, we increase the number of floors, starting at one floor. In `Towers of Hanoi`, the task is always to move all the discs from the left-most pole to the right-most one, while in `Office` we select one random state in the top-left room of floor one and one state

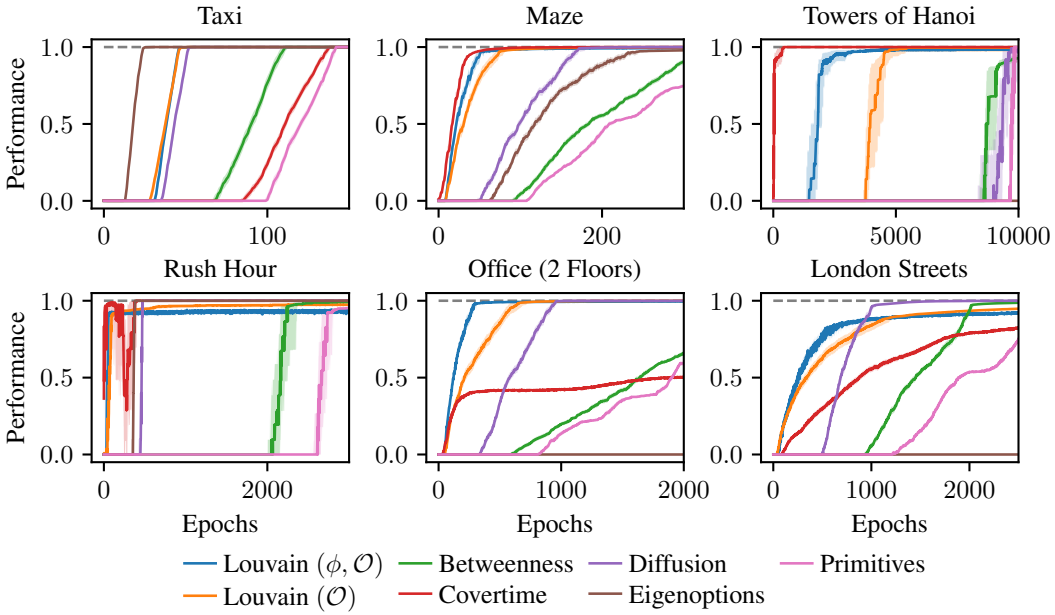


Figure 3: Learning performance as measured by the interquartile mean of the normalised discounted return. Shaded regions are 95% confidence intervals on the interquartile mean. The symbol $(\phi, \mathcal{O})$ denotes the full state-action abstraction, $(\mathcal{O})$ the action-only abstraction.

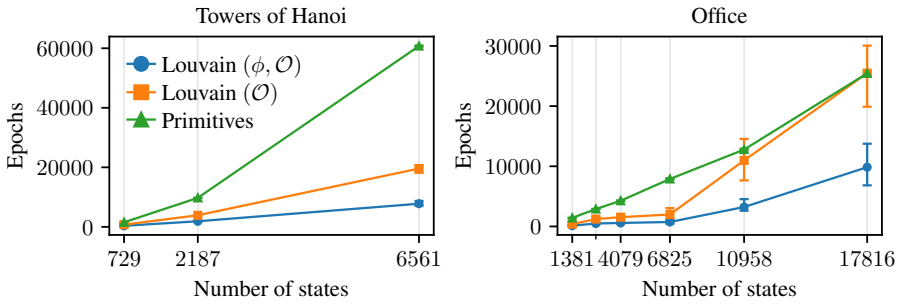


Figure 4: Scaling results, measured by the interquartile mean number of epochs required to reach a performance of at least 0.8 over three consecutive epochs. Bars are 95% confidence intervals on the interquartile mean. $(\phi, \mathcal{O})$ denotes full state-action abstraction, $(\mathcal{O})$ action-only abstraction.

in the bottom-right room of the highest floor as the initial and goal states for all agents at that scale. We use multi-level modularity-maximising partitions and compare state-action abstraction against the corresponding action-only abstraction and against primitives. Figure 4 shows the number of epochs required for the agent to achieve a normalised discounted return of at least 0.8 in the evaluation of three consecutive epochs. The results indicate that the sample-efficiency benefit of state-action abstraction over the corresponding action-only abstraction increases as the state space grows. The modularity of the discovered partitions is stable across these scaling steps (Appendix E.1), so the growing benefit is attributable to learning with state abstraction rather than to changes in partition quality.

What is the impact of multi-level abstraction? We build single-level state-action abstractions using each layer of the Louvain partition individually, each defined over the ground states and primitive actions, and compare them against the full multi-level abstraction. The multi-level state-action abstraction outperforms every corresponding single-level abstraction, and at each layer, the single-level state-action abstraction outperforms the corresponding action-only abstraction in nearly all cases (Appendix E.2).

What is the impact of the partitioning method? Our architecture is not tied to modularity-maximising partitions. We use spectral clustering (Ng, Jordan and Weiss, 2001) as an alternative basis for the abstraction. Spectral clustering produces partitions that approximately minimise the

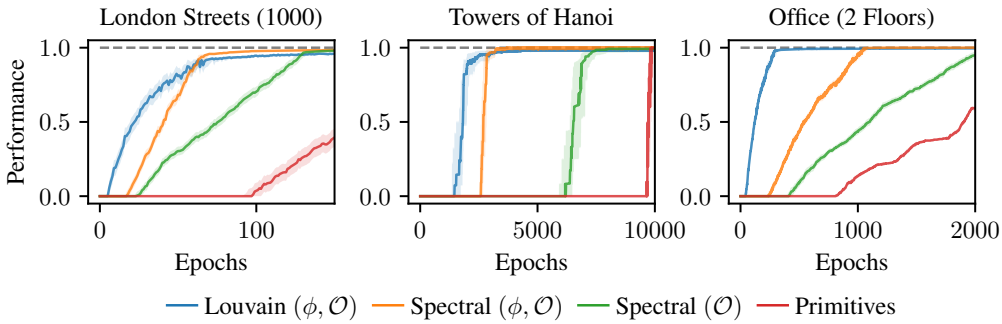


Figure 5: Learning performance with spectral clustering, measured by the interquartile mean of the normalised discounted return. $(\phi, \mathcal{O})$ denotes full state-action abstraction, $(\mathcal{O})$ action-only abstraction.

Normalised Cut (Shi and Malik, 2000), rather than maximising modularity. Unlike the Louvain algorithm, which naturally returns a hierarchy of nested partitions, spectral clustering returns a single flat partition and has no canonical multi-level form. Therefore, we construct a single-level state-action abstraction. Figure 5 compares the learning performance of an agent using state-action abstraction against the corresponding action-only abstraction, both using the same spectral clustering as the basis for abstraction. We present results for `Towers of Hanoi`, `Office`, and a 1000-node subgraph of `London Streets`. With spectral clustering partitions, the state-action abstraction substantially outperforms the action-only abstraction, but the multi-level modularity-maximising abstraction outperforms both. Finally, we consider the granularity of the partitioning by varying the Louvain algorithm's resolution parameter. Sample efficiency varies with the resolution, but learning with the state-action abstraction substantially outperforms the primitives in every case (Appendix E.4).

What is the impact of the restricted bootstrap rule? We compare learning with the restricted bootstrap rule of Equation (1) against an unrestricted variant that bootstraps from the full choice set $\mathcal{O}(s)$. We use a `Four Rooms` environment, and both agents use the same hand-coded single-level state-action abstraction. Full details and results are in Appendix E.3. The unrestricted agent learns substantially more slowly than both the restricted agent and an agent with only primitive actions, and its performance is unstable throughout learning (Figure 10), demonstrating that the restricted bootstrap rule plays a key role in the speed and stability of learning with state-action abstraction. Without the restricted bootstrap rule, the state-action abstraction is actively harmful in this setting.

Is learning with incremental state-action abstraction useful? Constructing hierarchical state-action abstractions through online interaction is an important open problem. We adapt the incremental approach of Evans and Şimşek (2023) to the state-action abstraction setting, providing empirical evidence that the proposed method remains beneficial when the abstraction is discovered incrementally. The agent starts with only primitive actions and no prior knowledge of the environment, storing its experience in a replay buffer. Every k epochs, this experience is used to update a state-transition graph. A node is added for each newly encountered state, a directed edge is added for each newly observed transition between states, and transition counts are updated. This graphical representation persists across iterations. The Louvain algorithm is then applied to the updated graph to produce a sequence of partitions. These partitions are used to construct a new ϕ-relative hierarchy as described in Section 3. Each new abstraction replaces the previous one: the existing abstraction and the action value estimates associated with it are discarded, and the action values for all skills in the new abstraction are initialised to 0. The values of primitive actions persist across updates, and the restricted bootstrap rule means that they will converge to the optimal action values.

Figure 6 compares the performance of the incremental agent against two baselines: an agent with access only to primitive actions, and an agent given the full hierarchical abstraction a priori. In all cases, the incremental agent learns substantially faster than the primitive agent. In `Four Rooms` and `Maze`, it learns more slowly than the agent given the full state-action abstraction, while in `Towers of Hanoi` it initially outperforms that agent but converges to a lower performance. The dotted vertical lines indicate epochs where the hierarchy updates, producing a noticeable change in performance. These results demonstrate that hierarchical state-action abstractions can be discovered online and can substantially improve learning.

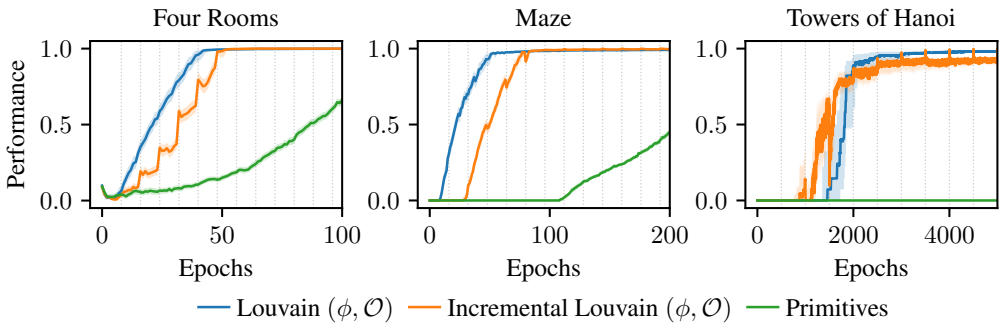


Figure 6: Learning performance of the incremental agent with state-action abstraction $(\phi, \mathcal{O})$, measured by the interquartile mean of the normalised discounted return. Vertical lines indicate the epochs where abstraction hierarchies are replaced: every 8, 16, and 500 epochs in `Four Rooms`, `Maze`, and `Towers of Hanoi`, respectively.

6 Discussion

We have introduced an architecture for learning with multi-level state-action abstractions, and instantiated two versions of it using graph partitioning methods. Our empirical analysis illustrates three key benefits of the proposed architecture. First, learning with multi-level state-action abstraction is more sample-efficient than learning with the corresponding action-only abstraction in most environments, and this benefit increases with the size of the state space. Our intuition is that state-action abstraction propagates value across space as well as time. Action abstraction propagates value across trajectories via SMDP updates, while state abstraction updates values at the scale of abstract states, rather than ground states. In Figure 7, for example, values are learned for whole rooms, while the action-only abstraction must learn each option's value at every ground state. Second, the benefit of state abstraction is not tied to the particular partitions used: across multi-level and single-level modularity-maximising partitions and single-level spectral partitions, learning with state-action abstraction outperforms the corresponding action-only abstraction in most cases. Third, the multi-level state-action abstraction improves on the corresponding single-level abstraction in every domain we tested. Our intuition is that coarser layers pool experience over larger regions of the state space and propagate value over longer trajectories, so their estimates form quickly. Finer layers learn more slowly but at higher resolution, and the restricted bootstrap rule prevents the coarse estimates from biasing the finer ones. The multi-level hierarchy maintains estimates at every scale, allowing behaviour to be guided by coarse estimates early in learning and by finer estimates as they improve.

We presented a tabular instantiation of the architecture, but it is applicable to reinforcement learning in continuous and very large discrete state and action spaces. A useful observation is that the architecture discretises both spaces at every layer above the ground. For every layer $i \geq 1$, the state abstraction ϕ_i maps to a finite set of abstract states $\mathcal{S}_i$, and the option set $\mathcal{O}_i$ is finite regardless of the ground action space. Only the ground layer value estimates and the layer 1 option policies, which act over ground states and primitive actions, touch the continuous spaces. Achieving this in practice requires a partitioning method for continuous or very large discrete state spaces, and function approximation for the ground layer value estimates and the layer 1 option policies. Existing work addresses each component. Partitions can be formed without enumerating the state space by clustering states with similar transition structure (Mendonça, Ziviani and Barreto, 2018), by successor-feature similarity (Hoang et al., 2021), or on learned temporal distance metrics (Baek et al., 2025; Evans, 2026). Value-based deep reinforcement learning provides function approximation for value estimates (e.g. Mnih et al., 2015), and a variety of methods learn option policies with function approximation (e.g. Bacon, Harb and Precup, 2017; Bagaria and Konidaris, 2020; Cannon and Şimşek, 2025).

Three limitations offer directions for future research. First, although Remark 1 establishes that action values converge to optimal at the ground layer, we do not establish bounds on the value-loss of the agent's greedy policy over options. Second, abstractions are discovered in the reward-free setting, reflecting only transition dynamics. This makes them useful across a variety of tasks in a given environment, but they are not aligned with any particular task. Third, although we demonstrate learning with incremental discovery of state-action abstractions, we do not learn option policies online, which would introduce non-stationarity into the learning process.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing methods in code, and to write and edit supporting code for experiments and figures. We have not used generative AI tools to develop the conceptual framework, formulate or prove mathematical claims, propose hypotheses, design the methodology or experiments, or interpret results. The rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to help draft and revise the paper. All AI-assisted code was reviewed and tested by authors. We take responsibility for the final content of this work, including any text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Abel, D., Umbanhowar, N., Khetarpal, K., Arumugam, D., Precup, D. and Littman, M., 2020. Value preserving state-action abstractions. *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*. PMLR, pp.1639–1650.

Agarwal, R., Schwarzer, M., Castro, P.S., Courville, A.C. and Bellemare, M.G., 2021. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., pp.29304–29320.

Bacon, P.L., Harb, J. and Precup, D., 2017. The option-critic architecture. *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. pp.1726–1734.

Baek, S., Park, T., Park, J., Oh, S. and Kim, Y., 2025. Graph-assisted stitching for offline hierarchical reinforcement learning. *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, pp.2391–2408.

Bagaria, A. and Konidaris, G., 2020. Option discovery using deep skill chaining. *International Conference on Learning Representations*.

Bar, A., Talmon, R. and Meir, R., 2020. Option discovery in the absence of rewards with manifold analysis. *Proceedings of the 37th International Conference on Machine Learning*. PMLR, pp.664–674.

Blondel, V.D., Guillaume, J.L., Lambiotte, R. and Lefebvre, E., 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10), p.10008.

Cannon, T.P. and Şimşek, Ö., 2025. Accelerating task generalisation with multi-level skill hierarchies. *International Conference on Learning Representations*.

Delvenne, J.C., Yaliraki, S.N. and Barahona, M., 2010. Stability of graph communities across time scales. *Proceedings of the National Academy of Sciences*, 107(29), pp.12755–12760.

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

Evans, J.B., 2026. *Identifying multi-level behaviour hierarchies using modularity maximisation*. Ph.D. thesis. University of Bath.

Evans, J.B. and Şimşek, Ö., 2023. Creating multi-level skill hierarchies in reinforcement learning. *Advances in Neural Information Processing Systems*, vol. 36. Curran Associates, Inc.

Gopalan, N., desJardins, M., Littman, M., MacGlashan, J., Squire, S., Tellex, S., Winder, J. and Wong, L., 2017. Planning with abstract Markov decision processes. *Proceedings of the International Conference on Automated Planning and Scheduling*, 27, pp.480–488.

Hoang, C., Sohn, S., Choi, J., Carvalho, W. and Lee, H., 2021. Successor feature landmarks for long-horizon goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, vol. 34.

Jinnai, Y., Park, J.W., Abel, D. and Konidaris, G., 2019. Discovering options for exploration by minimizing cover time. *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp.3130–3139.

Konidaris, G., 2016. Constructing abstraction hierarchies using a skill-symbol loop. *Proceedings of the 25th International Joint Conference on Artificial Intelligence*. New York, New York, USA: AAAI Press, pp.1648–1654.

Konidaris, G. and Barto, A.G., 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in Neural Information Processing Systems*, vol. 22. Curran Associates, Inc.

Konidaris, G., Kaelbling, L. and Lozano-Pérez, T., 2014. Constructing symbolic representations for high-level planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1).

Lambiotte, R., Delvenne, J.C. and Barahona, M., 2014. Random walks, Markov processes and the multiscale modular organization of complex networks. *IEEE Transactions on Network Science and Engineering*, 1(2), pp.76–90. Available from: https://doi.org/10.1109/TNSE.2015.2391998.

Li, L., Walsh, T.J. and Littman, M.L., 2006. Towards a unified theory of state abstraction for MDPs. *International Symposium on Artificial Intelligence and Mathematics, ISAIM*.

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

Mannor, S., Menache, I., Hoze, A. and Klein, U., 2004. Dynamic abstraction in reinforcement learning via clustering. *Proceedings of the 21st International Conference on Machine Learning*. ACM, p.71.

Menache, I., Mannor, S. and Shimkin, N., 2002. Q-cut—dynamic discovery of sub-goals in reinforcement learning. *Machine Learning: ECML 2002*. Springer Berlin Heidelberg, pp.295–306.

Mendonça, M.R.F., Ziviani, A. and Barreto, A.M.S., 2018. Abstract state transition graphs for model-based reinforcement learning. *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. IEEE.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540), pp.529–533.

Ng, A., Jordan, M. and Weiss, Y., 2001. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, vol. 14. MIT Press.

Precup, D., 2000. *Temporal abstraction in reinforcement learning*. Ph.D. thesis. University of Massachusetts Amherst.

Puterman, M.L., 1994. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.

Ramesh, R., Tomar, M. and Ravindran, B., 2019. Successor options: An option discovery framework for reinforcement learning. *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. Macao, China: International Joint Conferences on Artificial Intelligence Organization, pp.3304–3310.

Ravindran, B. and Barto, A.G., 2003. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. pp.1011–1016.

Rodriguez-Sanchez, R. and Konidaris, G., 2024. Learning abstract world models for value-preserving planning with options. *Reinforcement Learning Journal*, 1(1).

Shi, J. and Malik, J., 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), pp.888–905.

Şimşek, Ö. and Barto, A.G., 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proceedings of the 21st International Conference on Machine Learning*. p.95.

Şimşek, Ö. and Barto, A.G., 2008. Skill characterization based on betweenness. *Advances in Neural Information Processing Systems*, vol. 21. Curran Associates, Inc.

Şimşek, Ö., Wolfe, A.P. and Barto, A.G., 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proceedings of the 22nd International Conference on Machine Learning*. pp.816–823.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. 2nd ed. Cambridge, Massachusetts: MIT Press.

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), pp.181–211.

Theophilou, A. and Şimşek, Ö., 2025. Novel exploration via orthogonality. *Advances in Neural Information Processing Systems*, vol. 38. Curran Associates, Inc.

Watkins, C.J.C.H. and Dayan, P., 1992. Q-learning. *Machine Learning*, 8(3), pp.279–292.

A Learning Algorithm

Algorithm 1 gives the learning procedure of Section 3.2 in full. Option execution is recursive: a layer i option acts by sampling layer $(i-1)$ options from its policy, down to primitive actions at layer 0, until it terminates. Intra-option updates are applied after each ground transition, and the SMDP update for an option is applied when it terminates.

Algorithm 1 Restricted hierarchical Q-learning with state-action abstraction.

Require: ϕ-relative hierarchy H_L; behaviour policy μ; learning rates α_{smdp}, α_{intra}

Initialise $Q_i(\bar{s}_i, o)$ arbitrarily for all $i \in \{0, \dots, L\}$, $\bar{s}_i \in \mathcal{S}_i$, $o \in \mathcal{O}_i(\bar{s}_i)$
for each episode **do**
  while s is not terminal **do**
    Sample $o \sim \mu(\cdot \mid s)$ from the choice set $\mathcal{O}(s)$
    $(\cdot,\ s,\ \cdot) \leftarrow$ EXECUTE(o, s)
  end while
end for

procedure EXECUTE(o, s) ▷ returns $(G,\ s_{\text{end}},\ K)$
  $i \leftarrow \text{layer}(o)$; $s_\tau \leftarrow s$; $G \leftarrow 0$; $k \leftarrow 0$
  if $i = 0$ **then** ▷ primitive: $o = a \in \mathcal{A}$
    Execute primitive a; observe reward r and next state s'
    for each layer $i' = 0, 1, \dots, L$ **do**
      for all $o' \in \mathcal{O}_{i'}(\Phi_{i'}(s))$ with $\pi^{\Downarrow}_{o'}(s) = a$ **do** ▷ intra-option update
        $U \leftarrow (1 - \beta_{o'}(s'))\, Q_{i'}(\Phi_{i'}(s'), o') + \beta_{o'}(s')\, \overline{V}_{i'}(s')$
        $Q_{i'}(\Phi_{i'}(s), o') \leftarrow Q_{i'}(\Phi_{i'}(s), o') + \alpha_{\text{intra}}\big[r + \gamma U - Q_{i'}(\Phi_{i'}(s), o')\big]$
      end for
    end for
    return $(r,\ s',\ 1)$
  else ▷ layer i option o
    while o has not terminated **and** s is not terminal **do**
      Sample sub-option $o' \sim \pi_o(\cdot \mid s)$ ▷ $\pi_o\colon \mathcal{S}_{i-1} \to \Delta(\mathcal{O}_{i-1})$
      $(G',\ s,\ K') \leftarrow$ EXECUTE$(o',\ s)$
      $G \leftarrow G + \gamma^k\, G'$; $k \leftarrow k + K'$
    end while
    $Q_i(\Phi_i(s_\tau), o) \leftarrow Q_i(\Phi_i(s_\tau), o) + \alpha_{\text{smdp}}\big[G + \gamma^k\, \overline{V}_i(s) - Q_i(\Phi_i(s_\tau), o)\big]$ ▷ SMDP
    return $(G,\ s,\ k)$
  end if
end procedure

B Visualisation of Abstract State Values

Figure 7 shows the converged abstract state values at each layer of the ϕ-relative hierarchy of Figure 1, where the value of an abstract state is that of its greedy option: $\hat{V}_i(\bar{s}_i) = \max_{o \in \mathcal{O}_i(\bar{s}_i)} Q_i(\bar{s}_i, o)$. The initial state is shown with a cyan border, and the terminal goal state has a red border. Grey borders represent abstract state boundaries. Red arrows on the bottom row represent the actions taken by the

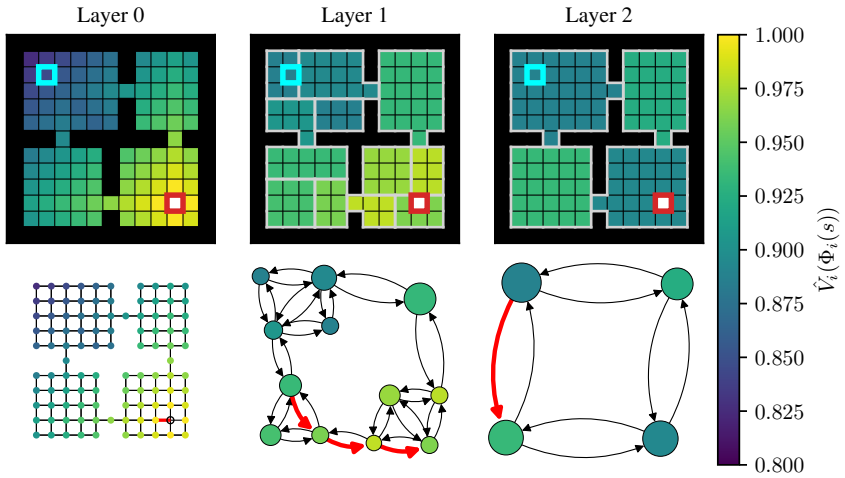


Figure 7: The converged abstract state values at each layer of the ϕ-relative hierarchy of Figure 1.

greedy policy $\tilde{\pi}$ after starting in the initial state. The reward is -0.001 per step, and an additional $+1$ if the agent reaches the goal state. The discount factor γ is 0.99.

The value functions at different layers are not different estimates of a single underlying quantity. A layer i option terminates only on leaving its initiating abstract state, so $Q_i(\bar{s}_i, o)$ estimates the return of leaving $\bar{s}_i$ with option o and then behaving greedily using options at layers i and below; the layers therefore estimate the values of different policies, and only the ground layer values estimate the optimal state values of the underlying problem (Remark 1). One visible consequence is that, at layers $i \geq 1$, the abstract state containing the goal does not have the highest value in its layer: any option initiating there first moves the agent away from the goal, so its value is lower than that of its neighbours, whose greedy options move the agent towards it.

The greedy policy selects over the options of every layer, so an option is executed only where it is greedy across the full choice set. In Figure 7, the greedy trajectory enters the goal room with a layer 1 option and reaches the goal with primitive actions; the room's layer 2 options are never selected.

C Environments

Taxi (Dietterich, 2000) is a game in which the agent moves on a 5×5 grid, picks up a passenger from four taxi rank locations, and drops them off at one of the other taxi rank locations. Each episode begins with the agent placed on a random cell in the grid and a passenger spawned at one of the taxi ranks. Each action allows the agent to move in one of the four cardinal directions, pick up a passenger, or drop off a passenger. Pick-up is available in all states but only leads to a different state when the passenger and the agent are at the same location, and the taxi is empty. Similarly, drop-off is available in all states but only leads to a different state when the passenger is in the taxi and the taxi is at the correct drop-off location. The pick-up and drop-off actions trigger rewards, whether or not they lead to a different state. The episode ends when the agent drops the passenger off at the correct drop-off location. The reward is -0.001 for each action, and an additional $+1$ on termination. Evaluation episodes have a maximum length of 500 steps.

Maze (Ramesh, Tomar and Ravindran, 2019) is a gridworld in which the agent navigates to a goal state. Each episode begins with the agent in a random unoccupied cell on the grid, other than the goal state. The state is an integer encoding of the agent's location. Each action moves the agent by one cell in one of the cardinal directions: North, South, East, and West. If the action would result in the agent colliding with a wall cell, then the action has no effect but still triggers a reward. The episode terminates if the agent reaches the goal cell. The reward is -0.001 for each action, and an additional $+1$ on termination. Evaluation episodes have a maximum length of 10000 steps.

Towers of Hanoi is a puzzle in which the agent moves seven discs of different sizes between three poles. Each episode begins with all discs on the first pole. The state is the current position of all discs on each pole. Each action moves the top disc from a source pole to one of the other poles, while

obeying the constraint that no disc can be placed above a disc of smaller size. The episode terminates when all discs are placed on the last pole. The reward is -0.001 for each action, and an additional $+1$ on termination. Evaluation episodes have a maximum length of 5000 steps.

Rush Hour is a sliding-block puzzle in which the agent manoeuvres a car block to the exit of a crowded 6×6 grid. Each episode begins with the car and eleven blocking vehicles (of length two or three) placed at fixed positions on the grid, with some cells additionally blocked by walls. Each action slides one vehicle by one cell along its axis of movement (horizontal vehicles slide left or right; vertical vehicles slide up or down), provided the destination cells are unoccupied. Actions trigger reward whether or not they are successful. The episode terminates when the car reaches the rightmost column of its row, exiting the grid. The reward is -0.001 for each action, and an additional $+1$ on termination. Evaluation episodes have a maximum length of 5000 steps.

Office (Evans and Şimşek, 2023) is a gridworld like `Maze` with two differences: first, the `Office` layout is intended to resemble an office building, with rooms, corridors, and multiple floors. Second, `Office` has an elevator state that takes the agent between floors. When standing on an elevator tile, the agent has two additional actions: moving up or down one floor. These additional actions change the state only if a floor exists in the intended direction but trigger a reward whether or not they are successful. The reward is -0.001 for each action, and an additional $+1$ on termination. Evaluation episodes have a maximum length of 10000 steps.

London Streets (adapted from Theophilou and Şimşek, 2025) is a real-world navigation problem based on the streets of London, UK. The agent navigates a graph representation of a subsection of London's street map, where each node represents a junction on the street map, and each edge is a street between two junctions. The state is the node on the graph where the agent is located. Each episode begins with the agent at a random state. Each action moves the agent down one of the available streets (edges) connected to the current state (node). The episode ends if the agent reaches the goal state. The reward is -0.001 for each action, and an additional $+1$ on termination. Evaluation episodes have a maximum length of 10000 steps.

D Experimentation Details

In this section, we provide details on the specific baseline methods and parameters used in the experiments.

D.1 Common Parameters

All methods use tabular Q-learning with combined SMDP and intra-option updates, and option policies pre-trained by value iteration. All methods use ϵ-greedy behaviour policies, except for eigenoptions, where the behaviour policy is uniform random (Machado, Bellemare and Bowling, 2017). Common parameters are listed in Table 1, all of which are constant throughout training and across environments.

Table 1: Common parameters for all methods.

Symbol	Description	Value
α_{smdp}	SMDP learning rate	0.4
α_{intra}	Intra-option learning rate	0.2
ϵ	Behaviour exploration rate	0.1
γ	Discount factor	0.99
q_{init}	Initial Q-values	0
ρ	Default Louvain resolution	0.05
θ	Value-iteration convergence threshold	10^{-6}

D.2 Baselines

We compare the proposed method against five established option discovery methods and an agent with access to only primitive actions.

Option pre-training. For every option discovery method, option initiation sets $\mathcal{I}_o$, termination conditions, and subgoals are discovered offline. Option policies π_o are precomputed with synchronous value iteration. In all cases except eigenoptions, the pseudo-reward is $+1$ for transitioning to the option's subgoal and 0 elsewhere, and $\gamma = 0.99$. Eigenoptions use the eigenpurpose as a pseudo-reward, as defined by Machado, Bellemare and Bowling (2017). Value iteration is run to convergence with $\theta = 10^{-6}$.

Louvain. Options are produced by running the Louvain modularity-maximisation algorithm on the state-transition graph with resolution $\rho = 0.05$, producing a multi-level partition hierarchy. Each layer defines a set of ϕ-relative options, as described in Section 3. The behaviour policy is ϵ-greedy over primitives and options. The agent learns Q-values over both primitive actions and options. The discovered options are the same for the state-action abstraction architecture and the action-only abstraction baseline (Evans and Şimşek, 2023).

Betweenness. Following Şimşek and Barto (2008), we compute betweenness centrality on the state-transition graph and define one option per local maximum. The number of options therefore depends on the graph structure of each environment. The initiation set of each option is the 30 states closest to its subgoal in path length on the transition graph. The behaviour policy is ϵ-greedy over primitives and options. The agent learns Q-values over both primitive actions and options.

Eigenoptions. Following Machado, Bellemare and Bowling (2017), we compute the bottom 16 eigenvectors of the graph Laplacian of the state-transition graph and define one option per eigenvector in each direction to give 32 options in total. The behaviour policy chooses uniformly at random from primitive actions and options. The agent learns Q-values over primitive actions only.

Diffusion. Following Bar, Talmon and Meir (2020), we identify subgoals via diffusion-based manifold analysis of the state-transition graph. We use a diffusion timescale $t = 8$. The behaviour policy is ϵ-greedy over primitives and options. The agent learns Q-values over primitive actions only.

Covertime. Following Jinnai et al. (2019), we discover 32 options whose subgoals minimise the expected cover time of the state-transition graph. The behaviour policy is ϵ-greedy over primitives and options. The agent learns Q-values over both primitive actions and options.

Primitives. The agent uses only the primitive action set. This serves as the no-abstraction reference. The behaviour policy is ϵ-greedy over primitives. The agent learns Q-values over primitives only, with a single learning rate $\alpha = 0.4$.

D.3 Memory Complexity

At each layer i our method stores a tabular action value function $Q_i : \mathcal{S}_i \times \mathcal{O}_i \to \mathbb{R}$, taking $\mathcal{O}_0 = \mathcal{A}$. The total memory required is:

$$M_{(\phi,\mathcal{O})} = O\left(|\mathcal{S}_0|\,|\mathcal{A}| + \sum_{i=1}^{L} |\mathcal{S}_i|\,|\mathcal{O}_i|\right).$$

For comparison, a primitive agent requires $M_{\text{primitive}} = O(|\mathcal{S}_0|\,|\mathcal{A}|)$, and an action-only abstraction agent using the same multi-level option set requires $M_{(\mathcal{O})} = O\big(|\mathcal{S}_0|(|\mathcal{A}| + \sum_{i=1}^{L} |\mathcal{O}_i|)\big)$, since it maintains a single Q-table indexed by ground states. By construction of the state abstraction, $|\mathcal{S}_i| \leq |\mathcal{S}_0|$ for all i, so $M_{(\phi,\mathcal{O})} \leq M_{(\mathcal{O})}$ term-by-term, with strict inequality at every layer that achieves any compression. State-action abstraction, therefore, never costs more memory than action-only abstraction over the same options, and is strictly cheaper whenever ϕ compresses the state space. The cost of state-action abstraction relative to primitives is the additional $\sum_{i=1}^{L} |\mathcal{S}_i|\,|\mathcal{O}_i|$ term.

E Additional Empirical Results

In this section, we provide additional details and results of our empirical analysis.

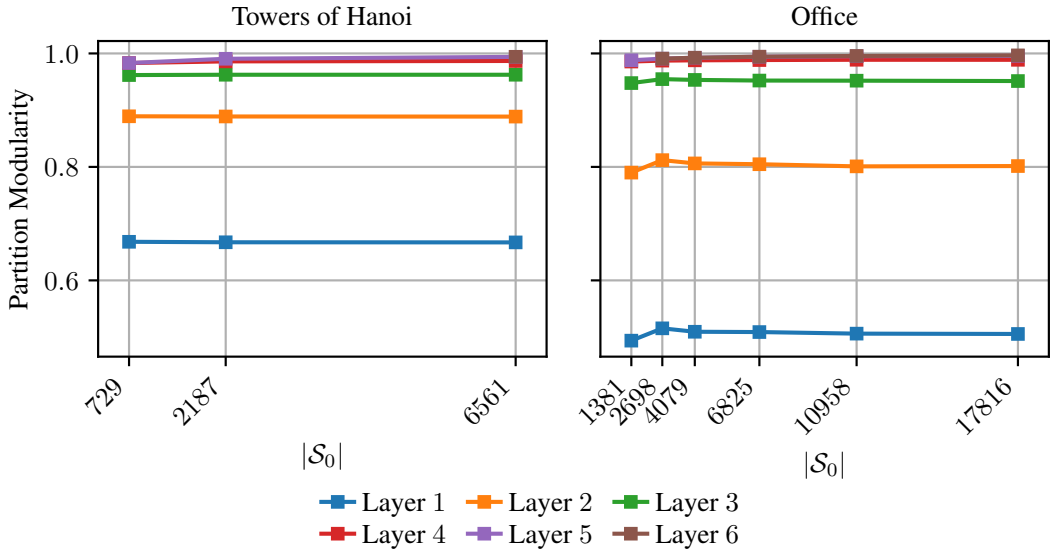


Figure 8: Modularity of the partition discovered at each layer of the hierarchy, at each scaling step of the experiments in Figure 4. Points are means over 20 seeds. Error bars show 95% confidence intervals on the mean; all are smaller than the markers. The depth of the hierarchy produced by the Louvain algorithm varies across seeds; layers present in fewer than 5 of the 20 seeds at a given scale are omitted.

E.1 Partition Quality with Environment Scale

The scaling experiments of Figure 4 show that the benefit of state-action abstraction grows with the size of the state space. Here we demonstrate that this benefit is not caused by changes to partition quality as the environments scale. Figure 8 shows the modularity of the partition discovered at every layer of the hierarchy, for each scaling step of the experiments in Figure 4. Per-layer modularity is stable across the full scaling range in both environments.

E.2 Single-Level Abstraction Ablation

Here we explore the impact of multi-level abstraction compared with single-level abstraction. We construct a single-level abstraction from each layer of the multi-level Louvain partition individually. The layer i single-level abstraction pairs the layer i state partition with a set of ϕ-relative options defined directly over the ground states and primitive actions, rather than over the preceding layer of the hierarchy. Figure 9 compares learning with each single-level state-action abstraction $(\phi_i, \mathcal{O}_i)$ against the corresponding action-only abstraction $(\mathcal{O}_i)$, alongside the full multi-level state-action abstraction $(\phi, \mathcal{O})$ and action-only abstraction $(\mathcal{O})$. The single-level state-action abstraction outperforms the corresponding single-level action-only abstraction in most cases, and the multi-level state-action abstraction outperforms every single-level state-action abstraction.

E.3 Restricted Bootstrap Ablation

We provide more details of the restricted bootstrap ablation. Instead of the restricted bootstrap rule of Equation (1), the unrestricted variant bootstraps from the full choice set $\mathcal{O}(s)$:

$$V(s) = \max_{0 \le i' \le L,\ o \in \mathcal{O}_{i'}(\Phi_{i'}(s))} Q_{i'}(\Phi_{i'}(s), o).$$

We use a hand-coded single-level state-action abstraction in the `Four Rooms` environment, with the two left-hand side rooms in one partition, and the two right-hand side rooms in the other. The initial and goal states are the top-left and bottom-right states respectively. Rewards are -0.001 per step, with an additional $+1$ on reaching the goal state, where the episode terminates. Epochs are 25 steps, and we use 60 agent seeds per variant. The learning performance of agents with and without the restricted bootstrap rule, and of a primitives agent, is shown in Figure 10. The unrestricted agent learns substantially more slowly than both other agents, and its performance is unstable.

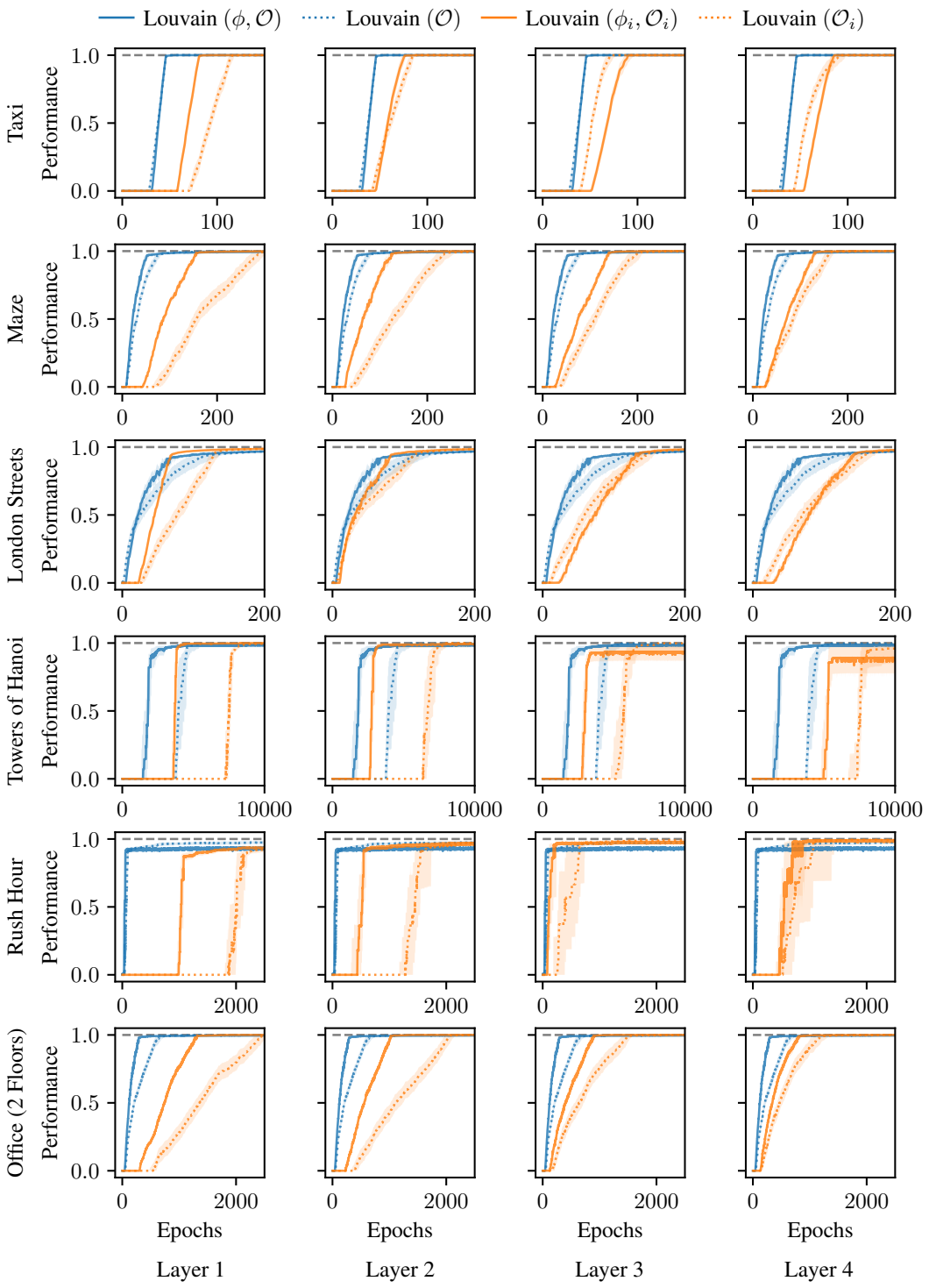


Figure 9: A comparison of learning performance for single-level abstractions from different layers of the hierarchy. Each panel shows a comparison of state-action abstraction $(\phi_i, \mathcal{O}_i)$ against action-only abstraction $(\mathcal{O}_i)$ with the single-level abstraction of layer i. Also shown are the full multi-level state-action abstraction $(\phi, \mathcal{O})$ and action-only abstraction $(\mathcal{O})$. Shaded regions are 95% confidence intervals on the interquartile mean. A value of 0 is the random baseline, 1 is optimal.

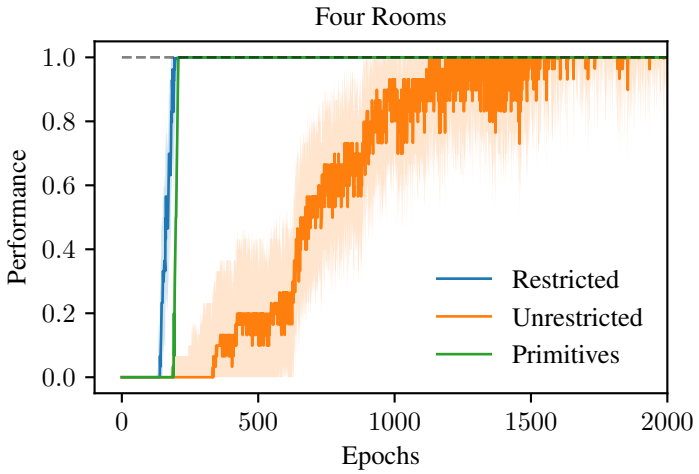


Figure 10: A comparison of learning performance with and without restricted bootstrapping, measured by the interquartile mean of the normalised discounted return. Shaded regions are 95% confidence intervals on the interquartile mean.

E.4 Impact of Louvain Resolution Parameter

Most of our results use the Louvain algorithm to find a multi-level state-action abstraction. The resolution parameter ρ is the primary hyperparameter of the Louvain algorithm. It influences the scale of the communities identified by the algorithm, with lower values resulting in fewer, larger communities and higher values producing more, smaller ones.

Figure 11 shows how the resolution parameter influences the learning performance of our agent in `Maze`, `Office` (one floor), and `Towers of Hanoi`. In each case, the learning performance is substantially better than that of the primitive agent under all resolutions.

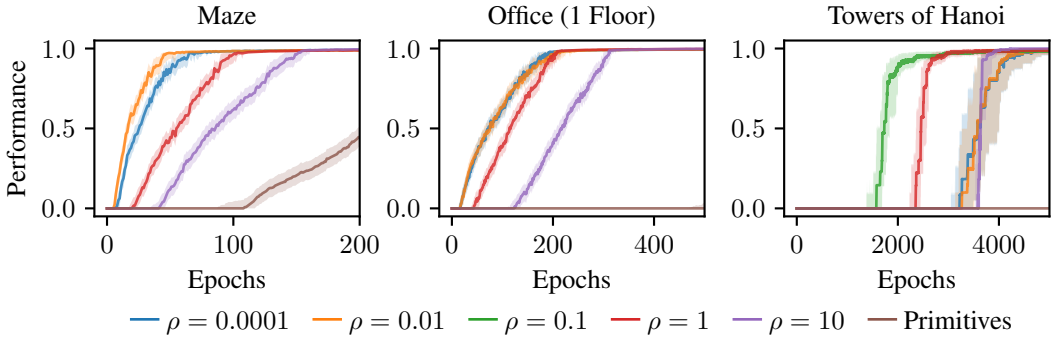


Figure 11: Learning performance across a range of values for the resolution parameter ρ of the Louvain algorithm. Shaded regions are 95% confidence intervals on the interquartile mean. A value of 0 is the random baseline, 1 is optimal.