Louvain Skills: Building Multi-Level Skill Hierarchies in Reinforcement Learning

Joshua B. Evans
Department of Computer Science
University of Bath
Bath, United Kingdom
jbe25@bath.ac.uk

Özgür Şimşek
Department of Computer Science
University of Bath
Bath, United Kingdom
o.simsek@bath.ac.uk

Abstract

What is a useful skill hierarchy for an autonomous agent? We propose an answer based on a graphical representation of how the interaction between an agent and its environment may unfold. Our approach uses modularity maximisation as a central organising principle to expose the structure of the interaction graph at multiple levels of abstraction. The result is a collection of skills that operate at varying time scales, organised into a hierarchy, where skills that operate over longer time scales are composed of skills that operate over shorter time scales. The entire skill hierarchy is generated automatically, with no human intervention, including the skills themselves (their behaviour, when they can be called, and when they terminate) as well as the hierarchical dependency structure between them. In a wide range of environments, this approach generates skill hierarchies that are intuitively appealing and that considerably improve the learning performance of the agent.

1 Introduction

One of the most important open problems in artificial intelligence is how to make it possible for autonomous agents to develop useful action hierarchies on their own, without any input from humans, including system designers and domain specialists. Before addressing this algorithmic question, it is useful to first consider a conceptual one: *What constitutes a useful action hierarchy?* Here we focus on this conceptual question with the aim of providing a useful foundation for algorithmic development.

Our primary contribution is a characterisation of a useful action hierarchy. In defining this action hierarchy, we use no information other than a graphical representation of how the interaction between the agent and its environment may unfold. When this information is not known a priori, it would be discovered naturally by the agent as it operates in its environment. Beyond this interaction graph, no particular domain knowledge is needed. Hence, our approach is applicable broadly.

Multiple strands of earlier work have used the interaction graph as a basis for defining collections of actions. The main novelty in our approach is our use of *modularity maximisation* as a central organising principle to expose the structure of the interaction graph at multiple levels of abstraction. The outcome is an action hierarchy that enables the agent to explore its environment efficiently at multiple time scales.

Our approach yields an action hierarchy with four desirable properties. First, it contains actions that operate at a wide range of time scales. This is necessary to solve complex problems, which

The scientific content of this paper was originally published as a NeurIPS 2023 conference paper under the title "Creating Multi-Level Skill Hierarchies in Reinforcement Learning" [1].

17th European Workshop on Reinforcement Learning (EWRL 2024).

require agents to be able to act, learn, and plan at varying time scales. Secondly, the actions are naturally organised into a hierarchy, with actions that operate over longer time scales being composed of actions that operate over shorter time scales. This hierarchical structure offers substantial benefits over unstructured collections of actions. For example, it allows an agent to learn about not only the action it is currently executing but also all lower-level actions that are called in the process. In addition, a hierarchical structure allows actions to be updated in a modular fashion. For instance, any improvements to an action would be immediately reflected in all higher-level actions that call it. Thirdly, the action hierarchy is fully specified. This includes when each action can be selected for execution, how exactly it behaves, and when it terminates. It also includes the number of levels in the hierarchy and the exact dependency structure between the actions. Fourthly, the action hierarchy is generated automatically, with no human intervention.

In a diverse set of environments, the proposed approach translates into action hierarchies that are intuitively appealing. When evaluated within the context of reinforcement learning, they substantially improve learning performance compared to alternative approaches, with the largest performance improvement observed in the largest environment tested.

An important question for future research is how such an action hierarchy may be learned when the agent has no prior knowledge of the dynamics of the environment. We present an initial exploration of how this may be achieved, with positive results.

2 Background

We use the reinforcement learning framework, modelling an agent's interaction with its environment as a finite Markov Decision Process (MDP). An MDP is a six-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{D}, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function, $\mathcal{D} : \mathcal{S} \to [0, 1]$ is an initial state distribution, and $\gamma \in [0, 1]$ is a discount factor. Let $\mathcal{A}(s)$ denote the set of actions available in state $s \in \mathcal{S}$. At decision stage t, $t \geq 0$, the agent observes state $s_t \in \mathcal{S}$ and executes action $a_t \in \mathcal{A}(s_t)$. Consequently, at decision stage $t + 1$, the agent receives reward $r_{t+1} \in \mathbb{R}$ and observes new state $s_{t+1} \in \mathcal{S}$. The *return* at decision stage t, denoted by G_t, is the discounted sum of future rewards, $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. A policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ specifies the probability of selecting action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. The objective is to learn a policy that maximises the expected return.

The *state transition graph* of an MDP is a weighted, directed graph whose nodes correspond to the states of the MDP and whose edges correspond to possible transitions between states. Specifically, an edge (u, v) exists on the graph if it is possible to transition from state $u \in \mathcal{S}$ to state $v \in \mathcal{S}$ by taking some action $a \in \mathcal{A}(u)$. In this paper, we use uniform edge weights of 1.

The actions of an MDP take exactly one decision stage to execute. We refer to them as *primitive actions*. Using primitive actions, it is possible to define *abstract actions*, also known as *skills*, whose execution can take a variable number of decision stages. Furthermore, primitive and abstract actions can be combined to form complex action hierarchies. In this work, we represent abstract actions using the options framework [2, 3]. An option o is a three-tuple $(\mathcal{I}_o, \pi_o, \beta_o)$, where $\mathcal{I}_o \subset \mathcal{S}$ is the initiation set, specifying the set of states in which the option can start execution, $\pi_o : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the option policy, and $\beta_o : \mathcal{S} \to [0, 1]$ is the termination condition, specifying the probability of option termination in a given state. An option policy is ultimately defined in terms of primitive actions—because primitive actions are the fundamental units of interaction between the agent and its environment—but this can be done indirectly by allowing options to call other options, making it possible for agents to act, learn, and plan with hierarchies of primitive and abstract actions.

3 Proposed Approach

To define a skill hierarchy, we use modularity maximisation as a central organising principle, applied at multiple time scales. Specifically, we represent the possibilities of interaction between the agent and its environment as a graph and identify partitions of this graph that maximise modularity [4–6].

A *partition* of a graph is a division of its nodes into mutually exclusive groups, called *clusters*. The *modularity* of a partition composed of a set of clusters $C = \{c_1, c_2, \ldots, c_k\}$ is

$$\sum_{i=1}^{k} e_{ii} - \rho a_i^2 \,,$$

where e_{ii} denotes the proportion of total edge weight in the graph that connects two nodes in cluster c_i, and a_i denotes the proportion of total edge weight in the graph with at least one end connected to a node in cluster c_i. A resolution parameter $\rho > 0$ controls the relative importance of e_{ii} and a_i. Intra-cluster edges contribute to both e_{ii} and a_i while inter-cluster edges contribute only to a_i. A partition that maximises modularity will have relatively dense connections within its clusters and relatively sparse connections between its clusters.

Finding a partition that maximises modularity for a given graph is NP-complete [7]. Therefore, when working with large graphs, approximation algorithms are needed. The most widely used approximation algorithm is the *Louvain algorithm* [8], which is an agglomerative hierarchical graph clustering approach. While no formal analysis exists, the runtime of the Louvain algorithm has been observed empirically to be linear in the number of graph edges [9]. It has been successfully applied to graphs with millions of nodes and billions of edges [8].

An important feature of the Louvain algorithm is that, as a *hierarchical* graph clustering method, it exposes the structure of a graph at multiple levels of granularity. Specifically, the output of the Louvain algorithm is a sequence of partitions of the input graph. This sequence has a useful structure: multiple clusters found in one partition in the sequence are merged into a single cluster in the next partition in the sequence. In other words, the output is a *hierarchy* of clusters, with earlier partitions containing many smaller clusters that are merged into fewer larger clusters in later partitions. This hierarchical structure forms the basis of our characterisation of a useful multi-level skill hierarchy.

The Louvain algorithm starts by placing each node of the graph in its own cluster. Nodes are then iteratively moved locally, from their current cluster to a neighbouring cluster, until no gain in modularity is possible. This results in a revised partition corresponding to a local maximum of modularity with respect to local node movement. The revised partition is used to define an *aggregate graph* as follows: each cluster in the partition is represented as a single node in the aggregate graph, and a directed edge is added to the aggregate graph if there is at least one edge that connects neighbouring clusters in the corresponding direction. This process is then repeated on the aggregate graph, and then on the next aggregate graph, and so on, until an iteration is reached with no modularity gain. Pseudocode for the Louvain algorithm is presented in Section H of the supplementary material.

Let h denote the number of partitions returned by the Louvain algorithm when applied to the state transition graph. We use each of the h partitions to define a single layer of skills, resulting in an action hierarchy with h levels of abstract actions (skills) above primitive actions. Each level of the hierarchy contains one or more skills for efficiently navigating between neighbouring clusters of the state transition graph. Specifically, we define an option for navigating from a cluster c_i to a neighbouring cluster c_j as follows: the initiation set consists of all states in c_i; the option policy efficiently takes the agent from a given state in c_i to a state in c_j; the option terminates with probability 1 in states in c_j, with probability 0 otherwise.

Taking advantage of the natural hierarchical structure of the partitions produced by the Louvain algorithm, we compose the skills at one level of the hierarchy to define the skills at the next level up. That is, at each level of the hierarchy, option policies call actions (options or primitive actions) from the level below, with primitive actions being called directly by option policies from only the first level of the hierarchy. We call the resulting set of skills the *Louvain skill hierarchy*.

4 Related Work

There have been earlier approaches to skill discovery using the state transition graph. The approach we propose here differs from them in two fundamental ways. First, it is novel in its use of modularity maximisation as a central organising principle for skill discovery. Secondly, it produces a multi-level hierarchy, whereas existing graph-based approaches produce hierarchies with only a single level of skills above primitive actions.

Many existing approaches to skill discovery use the state transition graph to identify useful subgoal states and define skills that efficiently take the agent to these subgoals. Suggestions for useful subgoals have often been inspired by the concept of a "bottleneck". They include states that are on the border of strongly-connected regions of the state space [10], states that allow transitions between different regions of the state space [11], and states that lie on the shortest path between many pairs of states [12]. To identify such states, several approaches use graph centrality measures [12–16]. Others use graph partitioning algorithms [10, 17–23]. Alternatively, it has been proposed that "landmark" states found at the centre of strongly-connected regions of the state-space can be used as subgoals [24].

The proposed approach is most directly related to skill discovery methods that make use of graph partitioning to identify meaningful regions of the state transition graph and define skills for navigating between them [17, 25–29]. Three of these methods use the concept of modularity. One such approach is to generate a series of possible partitions by successively removing the edge with the highest edge betweenness from a graph, then selecting the partition with the highest modularity [26]. A second approach is to generate a partition using the label propagation algorithm and then to merge neighbouring clusters until no gain in modularity is possible [29]. In these two approaches, modularity maximisation is applied after first producing candidate partitions using different criteria. Consequently, the final partition does not maximise modularity overall. The time complexity of the label propagation method is near-linear in the number of graph edges, whereas the edge betweenness method has a time complexity of $O(m^2n)$ on a graph with m edges and n nodes. A third approach is by Xu et al. [28], who use the Louvain algorithm to find a partition that maximises modularity but, unlike our approach, define skills only for moving between clusters in the highest-level partition, discarding all lower-level partitions. All three methods produce a single level of skills above primitive actions.

Several approaches have used the graph Laplacian [30, 31] to identify skills that are specifically useful for efficiently exploring the state space. It is unclear how to arrange such skills to form multi-level skill hierarchies. In contrast, the proposed approach produces a set of skills that are naturally arranged into a multi-level hierarchy.

While existing graph-based methods do not learn multi-level hierarchies, policy-gradient methods have made some progress towards this goal. Bacon et al. [32] extended policy-gradient theorems [33] to allow the learning of option policies and termination conditions in a hierarchy with a single level of skills above primitive actions. Riemer et al. [34] further generalised these theorems to support multi-level hierarchies. Fox et al. [35] propose an imitation learning method that finds the multi-level skill hierarchy most likely to generate a given set of example trajectories. Levy et al. [36] propose a method for learning multi-level hierarchies of goal-directed policies, with each level of the hierarchy producing a subgoal for the lower levels to navigate towards. However, these methods are not without their limitations. Unlike the approach proposed here, they all require the number of hierarchy levels to be pre-defined instead of finding a suitable number automatically. They do not judiciously define initiation sets, instead making all skills available in all states. They also target different types of problems than we do, such as imitation-learning or goal-directed problems.

5 Empirical Analysis

We analyse the Louvain skill hierarchy in the six environments depicted in Figure 1: Rooms, Grid, Maze [24], Office, Taxi [37], and Towers of Hanoi. In all environments, the reward is -0.001 for each action and an additional $+1.0$ for reaching a goal state. In Rooms, Grid, Maze, and Office, there are four primitive actions: north, south, east, and west. In Taxi, there are two additional primitive actions: pick-up-passenger and put-down-passenger. Some decisions in Taxi are irreversible. For instance, after picking up the passenger, the agent cannot return to a state where the passenger is still waiting to be picked up. We describe each of these environments fully in Section A of the supplementary material.

When generating partitions of the state transition graph using the Louvain algorithm, we used a resolution parameter of $\rho = 0.05$, unless stated otherwise. When converting the output of the Louvain algorithm into a concrete skill hierarchy, we discarded all levels of the cluster hierarchy where the mean number of nodes per cluster was less than 4. Our reasoning is that skills that navigate between such small clusters execute for only a very small number of decision stages (often only one or two) and are not meaningfully more abstract than primitive actions. For all methods used in our

comparisons, we generated options using the complete state transition graph and learned their policies offline using macro-Q learning [38]. We trained all hierarchical agents using macro-Q learning and intra-option learning [39]. Although these algorithms have not previously been applied to multi-level hierarchies, they both extend naturally to this case. The primitive agent was trained using Q-Learning [40]. The shaded regions on the learning curves represent the standard error in 40 independent runs. All experiments are fully described in Section B of the supplementary material.

Our analysis is directed by the following questions: What is the Louvain skill hierarchy generated in each environment? How does this skill hierarchy impact the learning performance of the agent? How do the results change as the number of states increases? Does arranging skills into a multi-level hierarchy provide benefits over a flat arrangement of the same skills? What is the impact of varying the value of the resolution parameter ρ?

Louvain Skill Hierarchy. We first examine the cluster hierarchies generated by applying the Louvain algorithm to the state transition graphs of various environments. Figure 2 shows the results in Rooms, Office, Taxi, and Towers of Hanoi. Section D of the supplementary material shows additional results in Grid and Maze.

In Rooms, the hierarchy has four levels. At level three, we see that each room has been placed in its own cluster. Moving up the hierarchy, at level four, two of these rooms have been joined together into a single cluster. Moving down the hierarchy, each room has been divided into smaller clusters at level two, and then into even smaller clusters at level one. The figure illustrates how the corresponding skill hierarchy would enable efficient navigation between and within rooms.

In Office, at the top level, we see six large clusters connected by corridors. As we move lower down the hierarchy, we see that these large clusters have been divided into increasingly smaller regions. At level three, there are many rooms that form their own cluster. At level two, most rooms have been divided into multiple clusters. The figure reveals how the corresponding skill hierarchy would enable efficient navigation of the environment at multiple time scales.

In Taxi, the state transition graph has four disconnected components, each corresponding to one particular passenger destination: R, G, B, or Y. In Figure 2, we show only one of these components, the one where the passenger destination is B. The results for the other components are similar. The Louvain cluster hierarchy has four levels. At the top level, we see four clusters. In three of these clusters, the passenger is waiting at their starting location (R, G, or Y). In the fourth cluster, the passenger is either in-taxi or has been delivered to their destination (B). Navigation between these clusters is unidirectional, with only three possibilities. The three corresponding skills navigate the taxi to the passenger location *and* pick up the passenger. Moving one level down the hierarchy, the clusters produce skills that move the taxi between the left and the right-hand side of the grid, which are connected by the bottleneck state at the centre of the taxi navigation grid.

In Towers of Hanoi, the hierarchy has three levels. At level three, moving between the three clusters corresponds to moving the largest disk to a different pole. Each of these clusters has been divided into three smaller clusters at level two and then into three even smaller clusters at level one. A similar structure exists at levels two and one. The smaller clusters at level two correspond to moving the second-largest disc between different poles; the smallest clusters at level one correspond to moving the third-largest disc between different poles.

In all of these domains, the Louvain skill hierarchy closely matches human intuition. In addition, it is clear how skills at one level of the hierarchy can be composed to produce the skills at the next level.

Learning Performance. We compare learning performance with the Louvain skill hierarchy to approaches based on `Edge Betweenness` [26] and `Label Propagation` [29], as well as to the

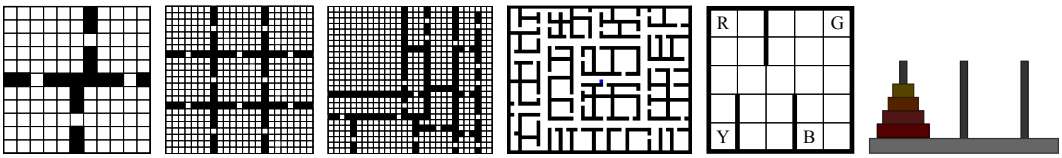


Figure 1: The environments.

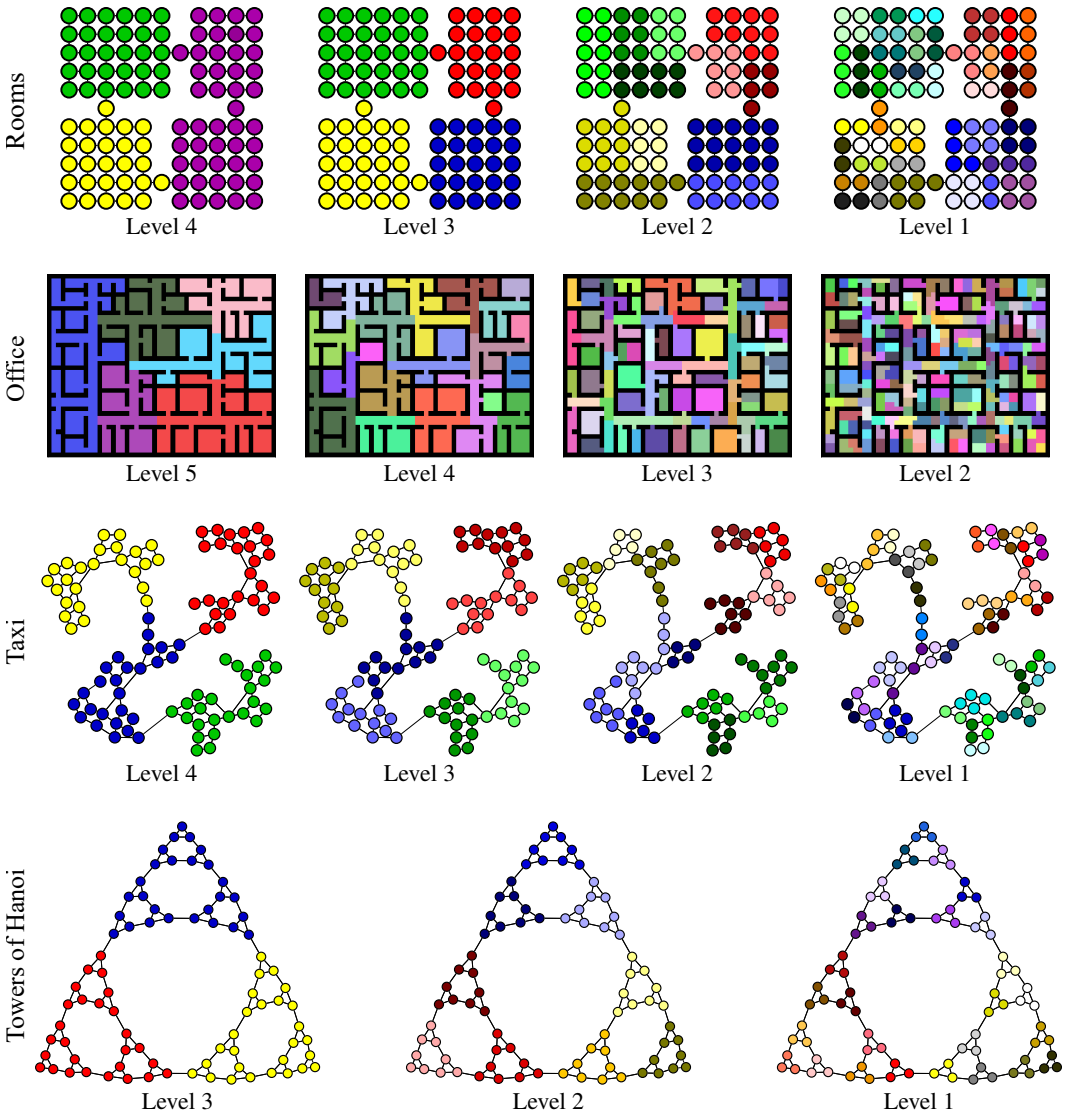


Figure 2: The cluster hierarchies produced by the Louvain algorithm when applied to the state transition graphs representing Rooms, Office, Taxi, and Towers of Hanoi. For Taxi and Towers of Hanoi, the graph layout was determined by using a force-directed algorithm that models nodes as charged particles that repel each other and edges as springs that attract connected nodes.

method proposed by `Xu et al.` [28]. These methods are the most directly related to the proposed approach, as discussed in Section 4. In addition, we compare to options that navigate to local maxima of `Node Betweenness` [12], a subgoal-based approach that captures the many different notions of the bottleneck concept in the literature. Similarly to the proposed approach, it is also one of the few approaches in the literature to explicitly characterise a target skill hierarchy for an agent to learn. We also compare to `Eigenoptions` [30], which are derived from the Laplacian of the state transition graph and encourage efficient exploration. Finally, we include a `Primitive` agent that uses only primitive actions. Primitive actions are available to all agents.

We present learning curves in Figure 3. The Louvain agent has a clear and substantial advantage over other approaches in all domains except for Towers of Hanoi, where its performance is much closer to that of the other hierarchical agents, with none of them performing much better than the primitive agent. This is consistent with existing results reported in this domain (e.g., by Jinnai et al. [31]).

Section C of the supplementary material contains further analysis comparing Louvain skills and skills based on node betweenness.

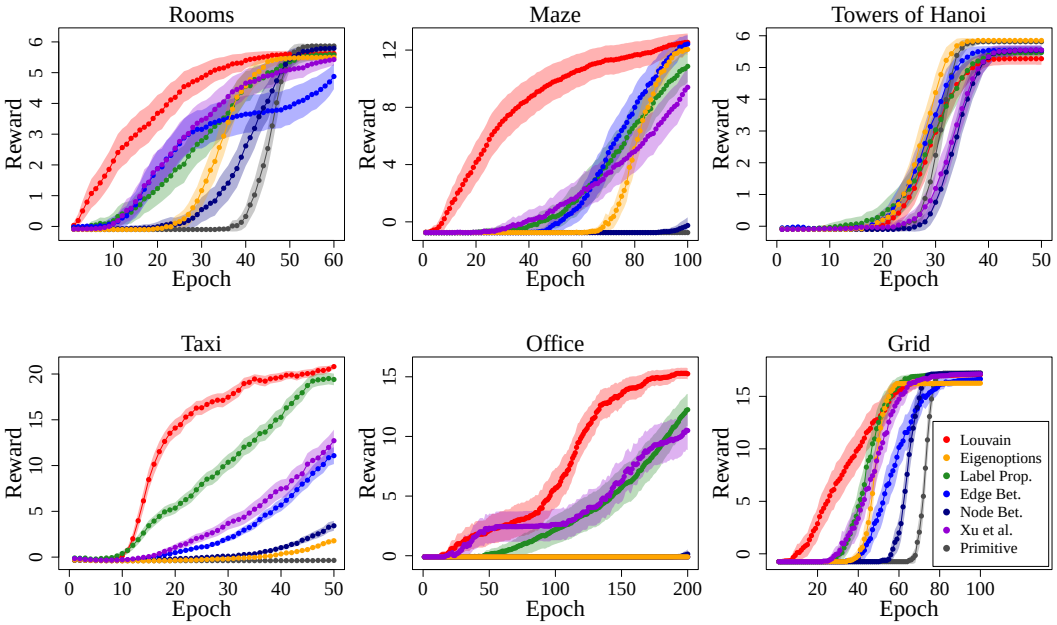


Figure 3: Learning performance. An epoch corresponds to 100 decision stages in Rooms and Towers of Hanoi, 300 in Taxi, 750 in Maze and Grid, and 1000 in Office.

Scaling to larger domains. We experimented with a multi-floor version of the Office environment, with floors connected by a central elevator, where two primitive actions move the agent up and down between two adjacent floors. The number of states in this domain can be varied by adjusting parameters such as the number of office floors. We use this environment to explore how the performance of the agent and the Louvain skill hierarchy change as the number of states in the environment increases.

Figure 4a presents results in an Office environment with 2537 states. It shows that the Louvain agent learns much more quickly than all other agents. Some alternatives, including the Eigenoptions agent, fail to achieve any learning.

Figure 4b shows how the Louvain skill hierarchy changes with the size of the state space in the Office environment. The figure shows the depth of the skill hierarchy in a series of fifteen Office environments of increasing size, ranging from a single floor ($\sim 10^3$ states) to one thousand floors ($\sim 10^6$ states). The depth of the hierarchy increased very gradually with the number of states in the environment. The maximum hierarchy depth reached was eight in the largest environment tested.

Hierarchical versus flat arrangement of skills. An alternative to the Louvain skill hierarchy is a flat arrangement of the skills, where each skill calls primitive actions directly rather than indirectly

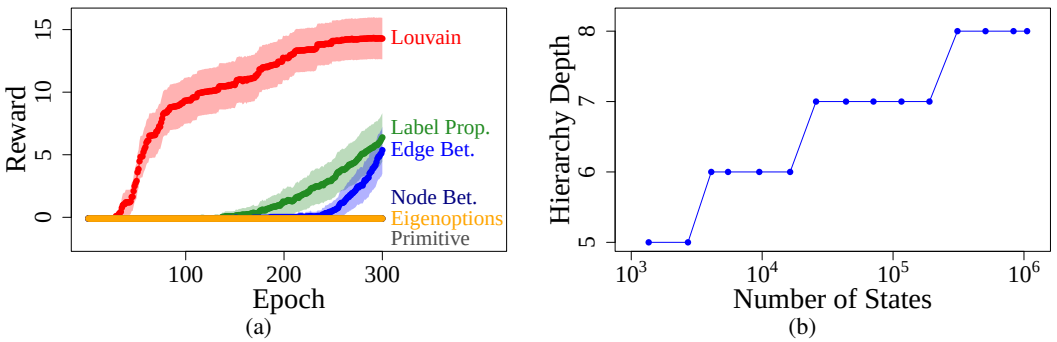


Figure 4: (a) Learning performance in a two-floor Office containing 2537 states. (b) How the depth of the skill hierarchy scales with the size of the state space in multi-floor Office.

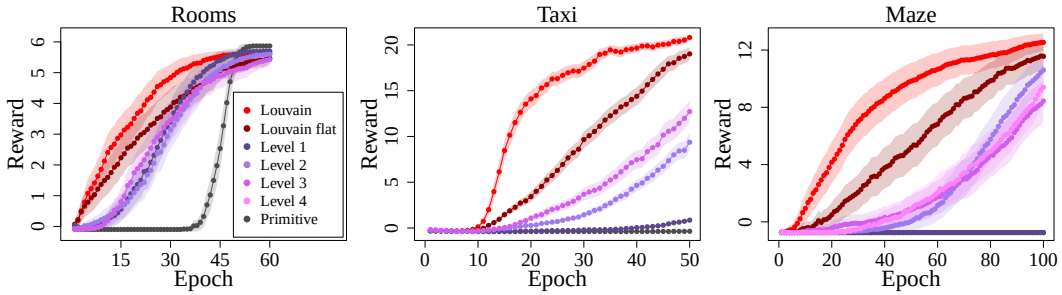


Figure 5: Learning curves comparing various different Louvain agents. An epoch corresponds to 100 decision stages in Rooms, 300 in Taxi, and 750 in Maze. The skill hierarchy contained three levels in Rooms and Taxi, and four levels in Maze.

through other (lower-level) skills. We expected the hierarchical arrangement to lead to faster learning than the flat arrangement due to the additional learning updates enabled by the hierarchical relationship between the skills. Figure 5 shows that this is indeed the case. In the figure, `Louvain` shows an agent that uses the Louvain skill hierarchy while `Louvain flat` shows an agent that uses the Louvain skills but where the skill policies call primitive actions directly rather than through other skills. In addition, the figure shows a number of agents that use only a single level of the Louvain hierarchy (`Level 1`, `Level 2`, `Level 3`, or `Level 4`), with option policies that call primitive actions directly. Primitive actions were available to all agents. The figure shows that the hierarchical agent learns more quickly than the flat agent. Furthermore, the agents using individual levels of the Louvain hierarchy learn more quickly than the primitive agent but not as quickly as the agent using the full Louvain hierarchy.

Impact of the resolution parameter. When using very high values of the resolution parameter ρ, the Louvain algorithm terminates without producing any partitions. On the other hand, when using very low values of ρ, it runs until a partition is produced where all nodes are merged into a single cluster.

Figure 6 shows how changing the resolution parameter ρ impacts the Louvain cluster hierarchy in Towers of Hanoi and Rooms. In Towers of Hanoi, at $\rho = 10$, the output is a single level containing many small clusters comprised of three nodes. As ρ gets smaller, the cluster hierarchy remains the same until $\rho = 3.3$, at which point a second level is added. The first level remains identical to the single level produced at $\rho = 10$. The second level contains larger clusters, each formed by merging three of the clusters from the first level. As ρ is reduced further, additional levels are added to the hierarchy. When a new level is added to the hierarchy, the existing levels generally remain the same.

A sensitivity analysis on the value of ρ showed that a wide range of ρ values led to useful skills, and that performance gradually decreased to no worse than that of a primitive agent at higher values of ρ. Section E of the supplementary material contains full details of this sensitivity analysis and a more detailed discussion on the choice of ρ.

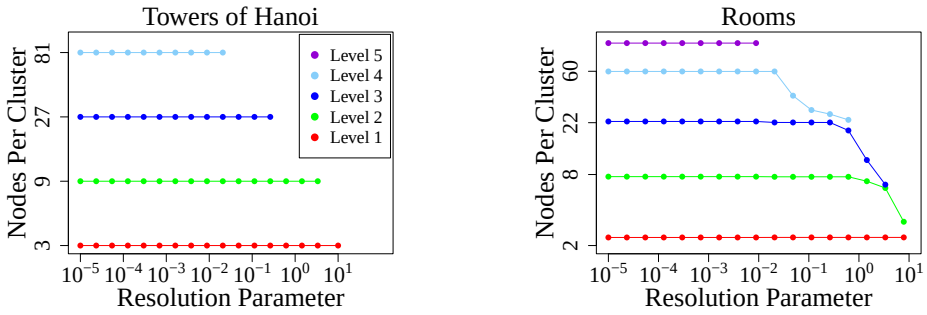
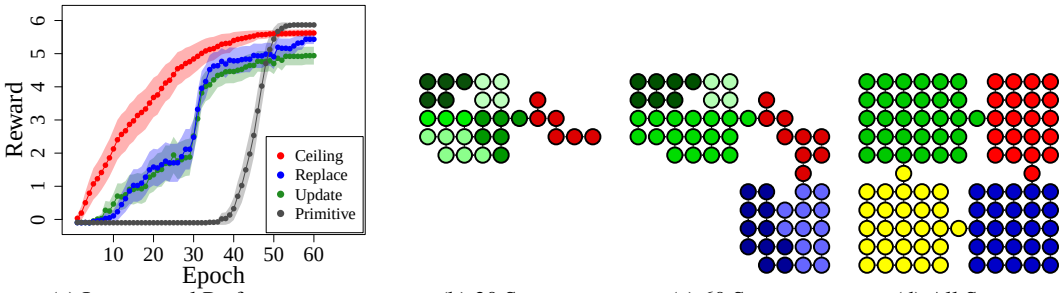


Figure 6: How the Louvain algorithm's output when applied to Towers of Hanoi and Rooms changes with the resolution parameter. The cluster hierarchy had a maximum of four levels in Towers of Hanoi and five levels in Rooms.



(a) Incremental Performance (b) 30 States (c) 60 States (d) All States

Figure 7: (a) Performance when incrementally discovering Louvain options in Rooms. An epoch corresponds to 100 decision stages. (b-d) How the state transition graph and top-level partitions produced by the `Update` approach evolved as an agent explored Rooms. The hierarchy contained 2 levels after visiting 30 states, 3 levels after visiting 60 states, and 5 levels after observing all possible transitions.

6 Discussion and Future Work

An important research direction for future work is incremental learning of Louvain skill hierachies as the agent interacts with its environment—because the state transition graph will not always be available in advance. We explored the feasibility of incremental learning in the Rooms environment. The results are shown in Figure 7. The agent started with an empty state transition graph and no skills. Every m decision stages, it updated its state transition graph with new nodes and edges, and it revised its skill hierarchy using one of two approaches: `Replace` or `Update`.

In the first approach (`Replace`), the agent applied the Louvain algorithm to create a new skill hierarchy from scratch. In the second approach (`Update`), the agent incorporated the new information into its existing skill hierarchy, using an approach similar to the Louvain algorithm. This second approach starts by assigning each new node to its own cluster; the new nodes are then iteratively moved locally, between neighbouring clusters (both new and existing) until no modularity gain is possible. This revised partition is used to define an aggregate graph and the entire process is repeated on the aggregate graph, and the next aggregate graph, and so on, until an iteration is reached with no modularity gain. The cluster membership of existing nodes stays fixed; only new nodes have their cluster membership updated. The result of both approaches is a revised set of partitions, from which a revised hierarchy of Louvain skills is derived. Section I of the supplementary material presents pseudocode for the two incremental approaches.

Figure 7a shows the performance of the incremental agents. The agents started with only primitive actions; after decision stages 100, 500, 1000, 3000, and 5000, the state transition graph was updated and the skill hierarchy was revised. The figure compares performance to a `Primitive` agent and an agent using the full Louvain skill hierarchy, whose performance acts as a `Ceiling` for the incremental agents. Both incremental agents learned much faster than the primitive agent and only marginally slower than the fully-informed Louvain agent. The two incremental agents had similar performance throughout training but `Replace` reached a higher level of asymptotic performance than `Update`, as expected. The reason is that partitions produced early in training are based on incomplete information; `Replace` discards these early imperfections while `Update` carries them forward. But there is a trade-off: building a new skill hierarchy from scratch has a higher computational cost than updating an existing one.

Figures 7b–7d show the evolution of the partitions as the `Update` agent performed a random walk in Rooms. The partitions were updated after observing 30 states, 60 states, and all possible transitions. The figure shows that, as more nodes were added, increasingly higher-level skills were produced.

These results demonstrate the feasibility of learning Louvain skills incrementally. A full incremental method for learning Louvain skills may take many forms, and different approaches may be useful under different circumstances, with each having its own strengths and weaknesses. We leave the full development of such algorithms to future work.

Another direction for future work is extending Louvain skills to environments with continuous state spaces such as robotic control tasks. Such domains present a difficulty to all skill discovery methods

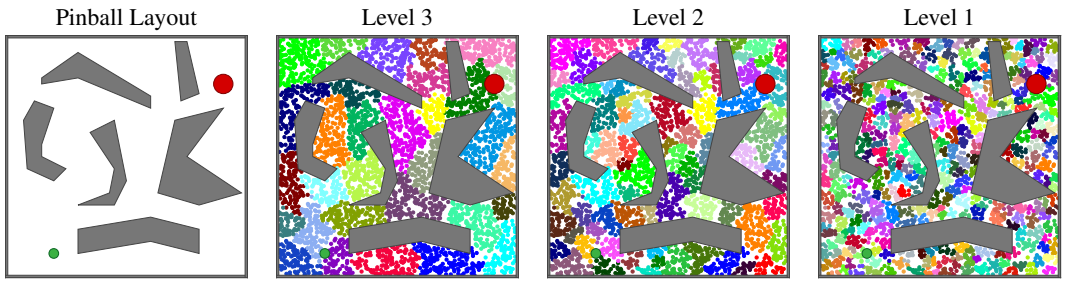


Figure 8: The layout of the Pinball environment—with the green ball in its initial position, the red goal, and several obstacles—and the Louvain cluster hierarchy produced by the Louvain algorithm.

that use the state transition graph due to the inherently discrete nature of the graph. If the critical step of constructing an appropriate graphical representation of a continuous state space can be achieved, all graph-based methods would benefit. Some approaches have already been proposed in the literature. We use one such approach [41, 21, 27] to examine the Louvain hierarchy in a variant of the Pinball domain [42], which involves manoeuvring a ball around a series of obstacles to reach a goal, as shown in the leftmost panel of Figure 8. The state is represented by two continuous values: the ball's position in the horizontal and vertical directions. At each decision stage, the agent can choose to apply a small force to the ball in each direction. The amount of force applied is stochastic and causes the ball to roll until friction causes it to come to a rest. Collisions between the ball and the obstacles are elastic.

We sampled 4000 states, added them to the state transition graph, then added an edge between each node and its k-nearest neighbours according to Euclidean distance, assigning an edge weight of e^{-4d^2} to an edge that connects two locations u and v with Euclidean distance d between them. We then applied the Louvain algorithm to the resulting graph. The result was the cluster hierarchy shown in Figure 8. It had three levels. Moving between clusters in the top level corresponds to skills that enable high-level navigation of the state space, and take into account features such as the natural bottlenecks caused by the obstacles, allowing the agent to efficiently change its position. Moving between clusters in the lower-level partitions enable more local navigation.

Currently, Louvain skills at one level of the hierarchy are composed of skills from only the previous level. Such skills may not provide the most efficient navigation between clusters. Future work could consider composing skills from all lower levels, including primitive actions.

Because Louvain skills are derived from the connectivity of the state transition graph, we expect them to be suitable for solving a range of problems in a given environment. Examining their use for transfer learning is a useful direction for future work.

A possible difficulty with building multi-level skill hierarchies is that having a large number of skills can end up hurting performance by increasing the branching factor. One solution that has been explored in the context of two-level hierarchies is pruning less useful skills from the hierarchy [22, 43]. Further research is needed to address this potential difficulty.

A key open problem for graph-based skill discovery methods generally is how to best make use of state abstraction. We suggest that the most natural place to introduce state abstraction is when constructing the state transition graph itself. Instead of a concrete state, each node could represent an abstract state based on some learned representation of the environment. The proposed method—and any other existing graph-based method—could then directly use this abstract state transition graph to define a skill hierarchy.

Lastly, we note that the various characterisations of useful skills proposed in the literature, including the one proposed here, are not necessarily competitors. To solve complex tasks, it is likely that an agent will need to use many different types of skills. An important avenue for future work is exploring how different ideas on skills discovery can be used together to enable agents that can autonomously develop complex and varied skill hierarchies.

Acknowledgments and Disclosure of Funding

We would like to thank the members of the Bath Reinforcement Learning Laboratory for their constructive feedback. This research was supported by the Engineering and Physical Sciences Research Council [EP/R513155/1] and the University of Bath. This research made use of Hex, the GPU Cloud in the Department of Computer Science at the University of Bath.

References

[1] J. B. Evans and Ö. Şimşek. Creating multi-level skill hierarchies in reinforcement learning. *Advances in Neural Information Processing Systems*, 36:48472–48484, 2023.

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

[3] D. Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.

[4] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004.

[5] E. A. Leicht and M. E. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11):118703, 2008.

[6] A. Arenas, J. Duch, A. Fernández, and S. Gómez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176, 2007.

[7] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *arXiv preprint:physics/0608255*, 2006.

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

[9] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80(5):056117, 2009.

[10] I. Menache, S. Mannor, and N. Shimkin. Q-Cut — Dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 295–306. Springer-Verlag, 2002.

[11] Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, ICML '04, pages 95–102. ACM, 2004.

[12] Ö. Şimşek and A. G. Barto. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems 21*, NeurIPS '09, pages 1497–1504. Curran Associates, Inc., 2009.

[13] P. Moradi, M. E. Shiri, and N. Entezari. Automatic skill acquisition in reinforcement learning agents using connection bridge centrality. In *International Conference on Future Generation Communication and Networking*, pages 51–62. Springer, 2010.

[14] A. A. Rad, M. Hasler, and P. Moradi. Automatic skill acquisition in reinforcement learning using connection graph stability centrality. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 697–700. IEEE, 2010.

[15] M. A. Imanian and P. Moradi. Autonomous subgoal discovery in reinforcement learning agents using bridgeness centrality measure. *International Journal of Electrical and Computer Sciences*, 11(5):54–62, 2011.

[16] P. Moradi, M. E. Shiri, A. A. Rad, A. Khadivi, and M. Hasler. Automatic skill acquisition in reinforcement learning using graph centrality measures. *Intelligent Data Analysis*, 16(1): 113–135, 2012.

[17] J. H. Metzen. Online skill discovery using graph-based clustering. In *Proceedings of the Tenth European Workshop on Reinforcement Learning*, EWRL '13, pages 77–88. PMLR, 2013.

[18] Ö. Şimşek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd International Conference on Machine learning*, ICML '05, pages 816–823. ACM, 2005.

[19] S. J. Kazemitabar and H. Beigy. Using strongly connected components as a basis for autonomous skill acquisition in reinforcement learning. In *6th International Symposium on Neural Networks*, ISSN '09, pages 794–803. Springer, 2009.

[20] N. Entezari, M. E. Shiri, and P. Moradi. A local graph clustering algorithm for discovering subgoals in reinforcement learning. In *International Conference on Future Generation Communication and Networking*, pages 41–50. Springer, 2010.

[21] P.-L. Bacon and D. Precup. Using label propagation for learning temporally abstract actions in reinforcement learning. In *Proceedings of the Workshop on Multiagent Interaction Networks*, MAIN '13, 2013.

[22] N. Taghizadeh and H. Beigy. A novel graphical approach to automatic abstraction in reinforcement learning. *Robotics and Autonomous Systems*, 61(8):821–835, 2013.

[23] S. J. Kazemitabar, N. Taghizadeh, and H. Beigy. A graph-theoretic approach toward autonomous skill acquisition in reinforcement learning. *Evolving Systems*, 9(3):227–244, 2018.

[24] R. Ramesh, M. Tomar, and B. Ravindran. Successor options: an option discovery framework for reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI '19, pages 3304–3310. AAAI Press, 2019.

[25] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the 21st International Conference on Machine Learning*, ICML '04, pages 71–78. ACM, 2004.

[26] M. Davoodabadi and H. Beigy. A new method for discovering subgoals and constructing options in reinforcement learning. In *Proceedings of the 5th Indian International Conference on Artificial Intelligence*, pages 441–450, 2011.

[27] F. Shoeleh and M. Asadpour. Graph based skill acquisition and transfer learning for continuous reinforcement learning domains. *Pattern Recognition Letters*, 87:104–116, 2017.

[28] X. Xu, M. Yang, and G. Li. Constructing temporally extended actions through incremental community detection. *Computational Intelligence and Neuroscience*, 2018.

[29] M. D. Farahani and N. Mozayani. Automatic construction and evaluation of macro-actions in reinforcement learning. *Applied Soft Computing*, 82:105574, 2019.

[30] M. C. Machado, M. G. Bellemare, and M. Bowling. A Laplacian framework for option discovery in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, ICML '17, pages 2295–2304. PMLR, 2017.

[31] Y. Jinnai, J. W. Park, D. Abel, and G. Konidaris. Discovering options for exploration by minimizing cover time. In *Proceedings of the 36th International Conference on Machine Learning*, ICML '19, pages 3130–3139. PMLR, 2019.

[32] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[33] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, NeurIPS '99. MIT Press, 1999.

[34] M. Riemer, M. Liu, and G. Tesauro. Learning abstract options. In *Advances in Neural Information Processing Systems 31*, NeurIPS '18. Curran Associates, Inc., 2018.

[35] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. *arXiv preprint:1703.08294*, 2017.

[36] A. Levy, G. Konidaris, R. Platt, and K. Saenko. Learning multi-level hierarchies with hindsight. In *Proceedings of the 7th International Conference on Learning Representations*, ICLR '19, 2019.

[37] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Resesearch*, 13(1):227–303, 2000.

[38] A. McGovern, R. S. Sutton, and A. H. Fagg. Roles of macro-actions in accelerating reinforcement learning. In *Grace Hopper Celebration of Women in Computing*, volume 1, pages 13–18, 1997.

[39] R. S. Sutton and D. Precup. Intra-option learning about temporally abstract actions. In *Proceedings of the 15th International Conference on Machine Learning*, pages 556–564. Morgan Kaufman, 1998.

[40] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[41] S. Mahadevan and M. Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8(10), 2007.

[42] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22*, NeurIPS '09, pages 1015–1023. Curran Associates, Inc., 2009.

[43] D. M. Farahani and N. Mozayani. Evaluating skills in hierarchical reinforcement learning. *International Journal of Machine Learning and Cybernetics*, 11(10):2407–2420, 2020.

Creating Multi-Level Skill Hierarchies in Reinforcement Learning

SUPPLEMENTARY MATERIAL

Joshua B. Evans
Department of Computer Science
University of Bath
Bath, United Kingdom
jbe25@bath.ac.uk

Özgür Şimşek
Department of Computer Science
University of Bath
Bath, United Kingdom
o.simsek@bath.ac.uk

A Environments

Gridworlds include Rooms, Grid, Office, and Maze [24]. They had four primitive actions: north, south, east, and west. These actions move the agent in the intended direction (unless the intended direction faces a wall, in which case the agent remains in the same state). The reward is -0.001 for each action and an additional $+1.0$ for reaching the goal state. There is a single start state and a single goal state, selected for each run from a list of possibilities.

Multi-Floor Office is an extension of Office to multiple floors. All foors are connected by an elevator, which occupies the same grid square on each floor, where the agent has two additional primitive actions: up and down. These actions move the agent to the adjecent floor in the intended direction (unless there is no other floor in that direction).

Taxi is a 5×5 grid with four special squares (R, G, B, and Y) that serve as possible pick-up and drop-off locations for a passenger. An episode starts with the taxi at a random square, the passenger at a random special square, and the destination another random special square. Six actions are available in each state: north, south, east, west, pick-up, and put-down. The navigation actions are identical to those in the gridworlds, as described above. Pick-up and put-down have the intended effect when appropriate; otherwise they do not change the state. The reward is -0.001 for each action and an additional $+1.0$ when the passenger is put down at the destination.

Towers of Hanoi contains four discs of different sizes, placed on three poles. An episode starts with all discs stacked on the leftmost pole. Primitive actions move the top disc from one pole to any other pole, with the constraint that a disc cannot be placed on top of a disc smaller than itself. The reward is -0.001 for each action and an additional reward of $+1.0$ at the goal state, where when all three discs are stacked on the rightmost pole.

B Methodology

Generating options. To generate Louvain options, the Louvain algorithm ($\rho = 0.05$) was applied to the state transition graph, resulting in a set of partitions. Any partition where the mean number of nodes per cluster was smaller than 4 was discarded. For all remaining partitions, options were defined for efficiently taking the agent from a cluster c_i to each of its neighbouring clusters c_j if a directed edge existed from a node in c_i to a node in c_j. The Louvain options were arranged into a multi-level hierarchy, where options for navigating between clusters in partition i could call skills for navigating between clusters in partition $i - 1$. Only options at the lowest level of the hierarchy could call primitive actions. Options generated using alternative methods called primitive actions directly.

Eigenoptions [30] were generated by computing the normalised Laplacian of the state transition graph and using its eigenvectors to define pseudo-reward functions for each Eigenoption to maximise.

In Office, the first 32 eigenvectors (and their negations) were used. In all other environments, the first 16 eigenvectors (and their negations) were used. Options for navigating to local maxima of betweenness [12] were generated by selecting all local maxima as subgoals and defining options for navigating to each subgoal from the nearest 30 states.

Learning option policies. For all methods except Eigenoptions, option policies were learned using macro-Q learning [38], with learning rate $\alpha = 0.6$, initial action values $Q_0 = 0$, and discount factor $\gamma = 1$. All agents used an ϵ-greedy exploration strategy with $\epsilon = 0.2$. For options based on clustering, the agent started in a random state in the source cluster. It received a reward of -0.01 for each action taken and an additional $+1.0$ for reaching a state in the goal cluster. For options based on node betweenness, the agent started in a random state in the initiation set. It received a reward of -0.01 at each decision stage, an additional $+1.0$ for reaching the subgoal state, and an additional -1.0 for leaving the initiation set. For Eigenoptions, value iteration was used to produce policies that maximised the pseudo-reward function of each Eigenoption.

Producing learning curves. All agents were trained using macro-Q [38] and intra-option learning [39] updates, which were performed every time an option at any level of the hierarchy terminated. A learning rate of $\alpha = 0.4$, discount factor of $\gamma = 1$, and initial action values of $Q_0 = 0$ were used in all experiments. All agents used an ϵ-greedy exploration strategy with $\epsilon = 0.1$. All learning curves show evaluation performance. After training each agent for one epoch, the learned policy was evaluated (with exploration and learning disabled) in a separate instance of the environment.

Applying the Louvain algorithm in Pinball. The state transition graph was constructed by following the method used by Mahadevan and Maggioni [41]. Initially, 4000 states were randomly sampled and a node representing each state was added to the graph. Edges were then added between each node and its 10 nearest neighbours. An between two locations u and v with Euclidean distance d between them was assigned a weight of e^{-4d^2}. Finally, the Louvain algorithm ($\rho = 0.05$) was applied to the resulting graph.

C Comparison to Skills that Navigate to Local Maxima of Betweenness

Here we explore the relationship between Louvain skills and the skill chracterisation proposed by Şimşek and Barto [12]. The latter is a subgoal-based approach that captures various definitions of the bottleneck concept. It defines skills that navigate to local maxima of betweenness. Both approaches address the conceptual problem of what makes a useful skill, explicitly defining a target set of skills for the agent to learn.

There is substantial overlap between Louvain skills and skills that navigate to local maxima of betweenness. They both include skills that traverse bottleneck states, including those that navigate between rooms in Rooms, picking up the passenger in Taxi, and navigating different parts of the grid in Taxi. In Towers of Hanoi, all Louvain skills traverse states that are also identified as local maxima of betweenness. The highest local maxima of betweenness correspond to the states that separate Louvain clusters at level 3; the second highest local maxima of betweenness correspond to the states that separate Louvain clusters at level 2.

On the other hand, there are many Louvain skills that do not correspond to navgating to local maxima of betweenness. Examples include the Louvain skills that navigate within a single room in Rooms.

Most importantly, Louvain skills and skills that navigate to local maxima of betweenness differ in how they can be arranged hierarchically. Even in Towers of Hanoi, where there is a clear hierarchical structure between the larger and smaller local maxima of betweenness, it is not clear how to exploit the betweenness metric to form a multi-level hierarchy. In contrast, the modularity metric approximated by the Louvain algorithm provides a clear and principled way of building a multi-level hierarchy.

D Cluster Hierarchies

Figure 1 shows the cluster hierarchies produced by the Louvain algorithm when applied to the state transition graphs of Grid and Maze. It also shows the first level of the cluster hierarchy in Office, which was omitted in the main paper due to space limitations.

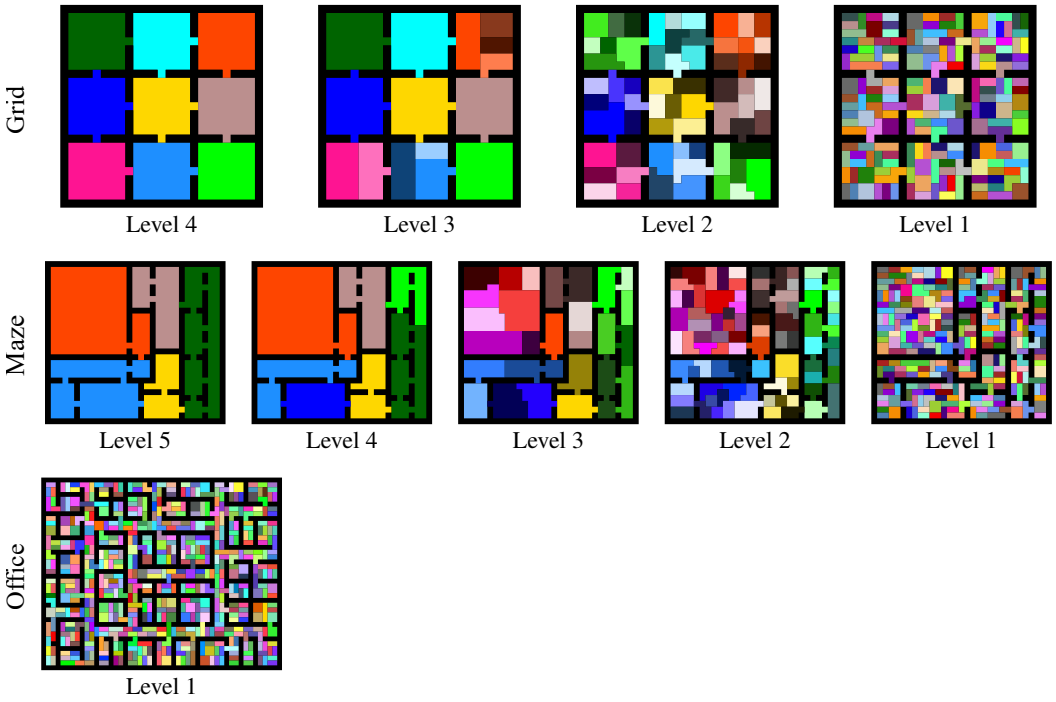


Figure 1: Top two rows: Cluster hierarchies produced by the Louvain algorithm in Grid and Maze. Bottom row: The lowest level of the cluster hierarchy in Office.

E Sensitivity to the Resolution Parameter

Figure 2 shows the performance of agents in Rooms and Towers of Hanoi using Louvain skills generated using different values of ρ. Louvain skills created using lower values of ρ consistently outperformed those created by using higher values, and very high values ($\rho \geq 10$) generally led to performance similar to that obtained by using primitive actions only. Lower ρ values lead to deeper hierarchies that contain skills for navigating the environment over varying timescales. In contrast, higher ρ values result in shallower skill hierarchies that contain few—or, in the extreme, no—levels of skills above primitive actions. While there are better and worse values of ρ, it may be argued that there is no "bad" choice; lowering ρ will result in deeper hierarchies, but existing levels of the hierarchy will remain intact.

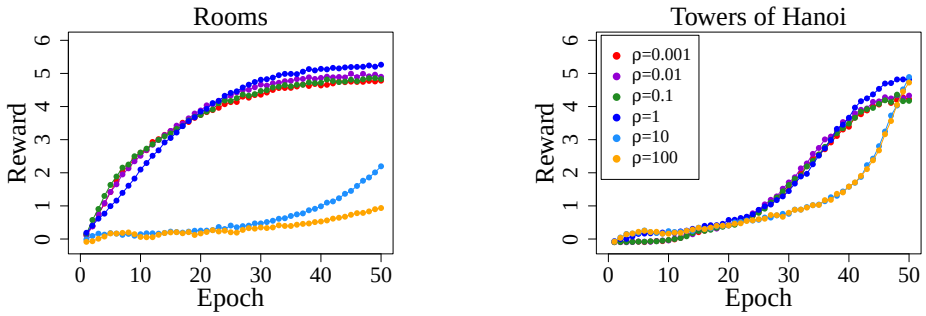


Figure 2: Agent performance with Louvain skills generated using different settings of the resolution parameter ρ.

F Compute Resource Usage

The experiments were run using a shared internal CPU cluster with the specifications shown below. Approximately 40 CPU cores were utilised for approximately 336 hours when producing the final set of results presented in the paper. Prior to this, approximately 20 CPU cores were utilised for approximately 168 hours during preliminary testing. GPU acceleration was not used because the experiments involved tabular reinforcement learning methods.

Processor	2× AMD EPYC 7443
Cores per Processor	24 Cores
Clock Speed	2.85GHz–4GHz
RAM	512 GB
RAM Speed	3200MHz DDR4

G Source Code

An implementation of the proposed approach, implementations of the test environments, and the code used to generate all results presented in the paper can be found in the following GitHub repository: https://github.com/bath-reinforcement-learning-lab/Louvain-Skills-NeurIPS-2023

H Louvain Algorithm

Algorithm 1 shows pseudocode for the Louvain algorithm [8]. It takes a graph $G_0 = (V_0, E_0)$ as input and outputs a set of partitions of that graph into clusters. Each iteration of the algorithm (lines 4–25) produces one partition of the graph.

Algorithm 1: Louvain Algorithm

1 **parameters:** resolution parameter $\rho \in \mathbb{R}^+$
2 **input:** $G_0 = (V_0, E_0)$ `// e.g., the state transition graph of an MDP`
3 $i \leftarrow 0$
4 **repeat**
5 $C_i \leftarrow \{\{u\} \mid u \in V_i\}$ `// define singleton partition`
6 $Q_{\text{old}} \leftarrow$ modularity from dividing G_i into partition C_i
7 **repeat**
8 $C_{\text{before}} \leftarrow C_i$
9 **foreach** $u \in V_i$ **do**
10 find clusters neighbouring u, $N_u \leftarrow \{c \mid c \in C_i, v \in V_i, v \in c, (u, v) \in E_i\}$
11 compute the modularity gain from moving u into each cluster $c \in N_u$
12 update C_i by inserting u into cluster $c \in N_u$ that maximises modularity gain
13 **end foreach**
14 $C_{\text{after}} \leftarrow C_i$
15 **until** $C_{before} = C_{after}$ `// no nodes changed clusters during an iteration`
16 $Q_{\text{new}} \leftarrow$ modularity from dividing G_i into revised partition C_i
17 **if** $Q_{new} > Q_{old}$ **then**
18 $V_{i+1} \leftarrow \{c \mid c \in C_i\}$
19 $E_{i+1} \leftarrow \{(c_j, c_k) \mid c_j \in C_i, c_k \in C_i, (u, v) \in E_i, u \in c_j, v \in c_k\}$
20 $G_{i+1} \leftarrow (V_{i+1}, E_{i+1})$ `// derive aggregate graph from current partition`
21 $i \leftarrow i + 1$
22 **else**
23 break
24 **end if**
25 **end**
26 **output:** partitions $C_0, \ldots, C_{i-1}$

I Incremental Discovery of Louvain Skills

Algorithm 2 presents an approach for incrementally developing a hierarchy of Louvain skills over time, starting with only primitive actions. To update the agent's skill hierarchy, Algorithm 2 calls upon either Algorithm 1 or Algorithm 3, depending on whether the agent's existing skill hierarchy is to be replaced or updated. Algorithm 3 presents an approach for incrementally updating Louvain partitions. It integrates new nodes into an existing cluster hierarchy.

Algorithm 2: Incremental Discovery of Louvain Skills

input: variant $\in \{1, 2\}$ `// which variant of the incremental algorithm to use`
input: $N = \{n_1, n_2, \ldots\}$ `// decision stages to revise skill hierarchy after`
$V \leftarrow \emptyset$ `// initialise empty set of nodes`
$E \leftarrow \emptyset$ `// initialise empty set of edges`
$G \leftarrow (V, E)$ `// initialise empty state transition graph (STG)`
$C \leftarrow \emptyset$ `// initialise empty set of partitions of the STG`
$V_{\text{new}} \leftarrow \emptyset$ `// initialise empty set for recording novel states`
$E_{\text{new}} \leftarrow \emptyset$ `// initialise empty set for recording novel transitions`
initialise $Q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily, with $Q(\text{terminal state}, \cdot) = 0$
$t \leftarrow 0$
repeat
  initialise environment to state S
  if $S \notin V$ **then**
    $V_{\text{new}} \leftarrow V_{\text{new}} \cup \{S\}$
  end if
  while S *is not terminal* **do**
    choose A from S using policy derived from Q (e.g., ϵ-greedy)
    take action A, observe next-state S' and reward R
    perform macro-Q and intra-option updates using (S, A, S', R)
    $S \leftarrow S'$
    if $S' \notin V$ **then**
      $V_{\text{new}} \leftarrow V_{\text{new}} \cup \{S'\}$ `// add novel state to set of new nodes`
    end if
    if $(S, S') \notin E$ **then**
      $E_{\text{new}} \leftarrow E_{\text{new}} \cup \{(S, S')\}$ `// add novel transition to set of new edges`
    end if
    if $t \in N$ **then**
      add each state $u \in V_{\text{new}}$ to V
      add each transition $(u, v) \in E_{\text{new}}$ to E
      $V_{\text{new}} \leftarrow \emptyset$
      $E_{\text{new}} \leftarrow \emptyset$
      if *variant* $= 1$ **then**
        $C \leftarrow$ partitions of the STG derived from (V, E) using Algorithm 1
        replace existing skill hierarchy with skills derived from C
      else if *variant* $= 2$ **then**
        $C \leftarrow$ partitions of the STG derived from (V, E, C) using Algorithm 3
        revise existing skill hierarchy based on skills derived from C
      end if
      initialise entries in Q for all new skills arbitrarily, with $Q(\text{terminal state}, \cdot) = 0$
      remove entries from Q for all skills that no longer exist in the revised hierarchy
    end if
    $t \leftarrow t + 1$
  end while
end

Algorithm 3: Update Louvain Partitions

parameters: resolution parameter $\rho \in \mathbb{R}^+$
input: $G_0 = (V_0, E_0)$ `// e.g., the state transition graph (STG) of an MDP`
input: $C = \{C_0, C_1, \ldots, C_n\}$ `// an existing set of partitions of the STG`
$i \leftarrow 0$
repeat
  $V_{\text{new}} \leftarrow$ nodes in V_i not assigned to any cluster in C_i
  $C_i \leftarrow C_i \cup \{\{u\} \mid u \in V_{\text{new}}\}$ `// define singleton partition over new nodes`
  $Q_{\text{old}} \leftarrow$ modularity from dividing G_i into partition C_{i}
  repeat
    $C_{\text{before}} \leftarrow C_i$
    foreach $u \in V_{new}$ **do**
      find clusters neighbouring u, $N_u \leftarrow \{c \mid c \in C_i, v \in V_i, v \in c, (u, v) \in E_i\}$
      compute the modularity gain from moving u into each cluster $c \in N_u$
      update C_i by inserting u into cluster $c \in N_u$ that maximises modularity gain
    end foreach
    $C_{\text{after}} \leftarrow C_i$
  until $C_{before} = C_{after}$ `// no nodes changed clusters during an iteration`
  $Q_{\text{new}} \leftarrow$ modularity from dividing G_i into revised partition C_i
  if $Q_{new} > Q_{old}$ *or* $i < |C|$ **then**
    $V_{i+1} \leftarrow \{c \mid c \in C_i\}$
    $E_{i+1} \leftarrow \{(c_j, c_k) \mid c_j \in C_i, c_k \in C_i, (u, v) \in E_i, u \in c_j, v \in c_k\}$
    $G_{i+1} \leftarrow (V_{i+1}, E_{i+1})$ `// derive aggregate graph from current partition`
    $i \leftarrow i + 1$
  else
    break
  end if
end
output: partitions $C_0, \ldots, C_{i-1}$