Creating Multi-Level Skill Hierarchies in Reinforcement Learning

Joshua B. Evans **Özgür Şimşek**
Department of Computer Science
University of Bath
Bath, United Kingdom
{jbe25|o.simsek}@bath.ac.uk

Abstract

What is a useful skill hierarchy for an autonomous agent? We propose an answer based on the graphical structure of an agent's interaction with its environment. Our approach uses hierarchical graph partitioning to expose the structure of the graph at varying timescales, producing a skill hierarchy with multiple levels of abstraction. At each level of the hierarchy, skills move the agent between regions of the state space that are well connected within themselves but weakly connected to each other. We illustrate the utility of the proposed skill hierarchy in a wide variety of domains in the context of reinforcement learning.

1 Introduction

How can an agent autonomously develop an action hierarchy as it interacts with its environment? This is a fundamental open question in artificial intelligence. Before answering this algorithmic question, it is useful to first ask a conceptual question: What is a useful action hierarchy? Here we focus on this conceptual question to provide a useful foundation for future algorithmic development.

We propose a characterisation of a useful action hierarchy based on a graphical representation of an agent's interaction with its environment. We first partition this graph at multiple scales, exposing the structure of the environment at various levels of granularity, then define actions that efficiently move an agent between neighbouring clusters in the partitions. The outcome is an action hierarchy that enables the agent to efficiently interact with its environment at multiple time scales. In a diverse set of environments, the proposed characterisation translates into action hierarchies that are intuitively appealing, improve learning performance, and has desirable scaling properties.

Our approach has two key characteristics. First, we partition the interaction graph by maximising *modularity*, a measure of partition quality that generates clusters that are strongly connected within themselves but weakly connected to each other. Secondly, our approach produces a multi-level hierarchy, where lower-level actions are naturally composed into higher-level actions. Multi-level action hierarchies naturally support the ability to act, plan, explore, and learn over varying timescales, offering many concrete benefits over unstructured collections of actions that are not organised into a hierarchy.

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 $\gamma \in [0, 1]$

Preprint. Under review.

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 a numerical reward, $r_{t+1} \in \mathcal{R}$, and observes the next state, $s_{t+1} \in \mathcal{S}$. The *return* at decision stage t is the discounted sum of future rewards, $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. A policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is a mapping from state-action pairs to probabilities. The agent's 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 represent the states of the MDP and whose edges represent possible transitions between these states via primitive actions. 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 $a \in \mathcal{A}(u)$. Unless stated otherwise, 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*, or *skills*, whose execution can take a variable number of decision stages. Primitive and abstract actions can be combined to form complex action hierarchies. We represent skills using the options framework [1]. 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, $\pi_o : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the option policy, and $\beta_o : \mathcal{S} \rightarrow [0, 1]$ is the termination condition, specifying the probability of option termination in a given state. An option policy is ultimately defined over primitive actions—because these 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 operate with hierarchies of primitive and abstract actions.

3 Related Work

Although many existing approaches to skill discovery use the state-transition graph, all such approaches produce skill hierarchies with only a single level of skills above primitive actions. Multi-level skill hierarchies are essential when solving complex tasks, which often require agents to act, plan, explore, and learn over varying timescales. They offer many concrete benefits to reinforcement learning agents. First, when using a multi-level hierarchy, an agent can learn about not only the skill it is currently executing but also about any lower-level skills it calls upon. Secondly, arranging skills into multi-level hierarchies allows them to be updated in a modular fashion. For instance, any improvements to the internal policy of a lower-level skill would be immediately reflected in all higher-level skills that call it. Thirdly, the ability to form multi-level skill hierarchies supports a continual learning process, where an agent combines existing skills to produce new skills in an open-ended manner.

While existing graph-based methods do not learn multi-level hierarchies, policy-gradient methods have made some progress towards this goal. Bacon et al. [2] extended policy-gradient theorems [3] to allow the learning of option policies and termination conditions in a two-level hierarchy. Riemer et al. [4] further generalised these theorems to support multi-level hierarchies. Fox et al. [5] 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. [6] 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 also make simplifying assumptions, such as that all skills are available in all states, and target different types of problems than we do, such as imitation-learning or goal-directed problems with high-dimensional or continuous state-spaces.

Our approach is most directly related to skill discovery methods that use graph partitioning [7–12]. Three of these methods use the concept of modularity, which is also central to our approach. 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 [8]. 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 [12]. In these two approaches, the final partition will maximise modularity with respect to only the initial clusters identified by other methods; in other words, the final partition will generally not be the one that maximises modularity overall. The label propagation method runs in near-linear time, whereas the edge betweenness method has a time complexity of $O(m^2 n)$ on a graph with m edges and n nodes. A third approach is by Xu et al.

[11], 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. Using only the highest-level partition produces skills for navigating the state-space at only a high level, whereas our approach produces skills for navigating the state-space at varying timescales by using the full cluster hierarchy.

Another approach to skill discovery is to identify useful subgoals and define skills for navigating to them. Suggestions have often been inspired by the concept of "bottleneck" states. They include states that are on the border of strongly-connected regions of the state-space [13], states that allow transitions between different regions of the state-space [14], and states that lie on the shortest path between many other pairs of states [15]. To identify such states, several approaches use graph centrality measures [15–19]. Others use graph partitioning algorithms to identify meaningful regions of the state-space, then identify subgoals as the nodes bordering them [9, 13, 20–25]. The bottleneck concept has also inspired non-graphical approaches to skill discovery, such as seeking to identify states that are visited frequently on successful trajectories but infrequently on unsuccessful ones [26]. 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 [27].

Several approaches have used the graph Laplacian [28, 29] to identify skills that are specifically useful for efficiently exploring the state space. These methods produce skills that aim to minimise the expected number of actions required to navigate between any two states, allowing efficient navigation between distant areas of the state space. It is unclear how to arrange such skills to form multi-level skill hierarchies. In contrast, we propose a principled approach to characterising skills that are both useful and naturally form a multi-level hierarchy.

4 Proposed Approach

We identify partitions of the state-transition graph that maximise modularity [30, 31]. A *partition* of a graph is a division of its nodes into mutually exclusive groups, called *clusters*. Given a set of clusters $C = \{c_1, c_2, \ldots, c_k\}$ forming a partition of a graph, the *modularity* of the partition 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 ρ 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. So, as ρ increases, large clusters with many inter-cluster edges are penalised to a higher degree, leading to partitions with smaller clusters.

A partition that maximises modularity will have dense connections within its clusters but sparse connections between them. In the context of reinforcement learning, these clusters correspond to regions of the state space that are easy to navigate within but difficult to navigate between. The proposed skill hierarchy gives an agent the ability to efficiently move between such regions.

Finding a partition that maximises modularity for a given graph is NP-complete [32]. Therefore, when working with large graphs, approximation algorithms are needed. The most widely used approximation algorithm is the *Louvain algorithm* [33], an agglomerative hierarchical graph clustering algorithm. While no formal analysis exists, the runtime of the Louvain algorithm has been observed empirically to be linear in the number of graph edges [34].

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. This 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 that 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. For more details please refer to Blondel et al. [33], Arenas et al. [35], and the pseudocode in Appendix B.

The algorithm's output is a series of partitions of the input graph. This series of partitions has a useful structure: multiple clusters found in one partition are merged into a single cluster in the next partition. In other words, the output is a *hierarchy* of clusters, with earlier partitions containing many smaller clusters which are merged into fewer larger clusters in later partitions. This hierarchical structure forms the basis of the multi-level skill hierarchy we propose for reinforcement learning.

Let h denote the number of partitions returned by the algorithm. We use each of the h partitions to define a single layer of skills, resulting in a hierarchy with h levels above primitive actions. Each level of the hierarchy contains one or more skills for efficiently navigating between neighbouring clusters. 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. That is, at each level of the hierarchy, option policies will be defined over only the actions (options or primitive actions) in the level below, with options at only the first level of the hierarchy using policies defined over primitive actions. We call the resulting set of skills the *Louvain skill hierarchy*.

5 Empirical Analysis

We analyse the Louvain skill hierarchy in six environments: Rooms, Grid, Maze [27], Office, Taxi [36], and Towers of Hanoi. The environments are depicted in Figure 1 and fully described in Appendix D. In all environments, the agent receives a reward of -0.001 for each action and an additional $+1.0$ for reaching a goal state. In Rooms, Grid, Maze, and Office, there are 4 primitive actions: north, south, east, and west. In Taxi, there are two additional primitive actions: pick-up-passenger and put-down-passenger. Taxi also features irreversible actions. For instance, after picking up the passenger, the agent cannot return to a state where the passenger has not yet been picked up. All experimental details are in Appendix F.

Our analysis is directed by the following questions: What is the Louvain skill hierarchy produced in each environment? How does this skill hierarchy impact the learning performance of the agent? How are the results impacted as the environment gets larger? We report results using resolution parameter $\rho = 0.05$ (unless stated otherwise) but also examine the impact of ρ on the results.

Louvain Skill Hierarchy. Figure 2 shows the Louvain cluster hierarchy in Rooms, Office, Taxi, and Towers of Hanoi obtained by applying the Louvain algorithm to the state-transition graph of each environment. Appendix E shows the Louvain cluster hierarchies in Grid and Maze.

In Rooms, the hierarchy has four levels. At the third level, each room is placed in its own cluster. Moving up the hierarchy, at the fourth level, two of these rooms are joined together into a single cluster. Moving down the hierarchy, each room is divided further into smaller clusters at level 2, and then into even smaller clusters at level 1. It is easy to see 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 to each other by corridors. As we move lower down the hierarchy, these clusters are divided into increasingly smaller regions. At level 3, we see many rooms that form their own cluster. At level 2, most rooms are divided into multiple clusters. Once again, it is relatively easy to see how the corresponding skill hierarchy would enable efficient navigation of the space at multiple levels of granularity.

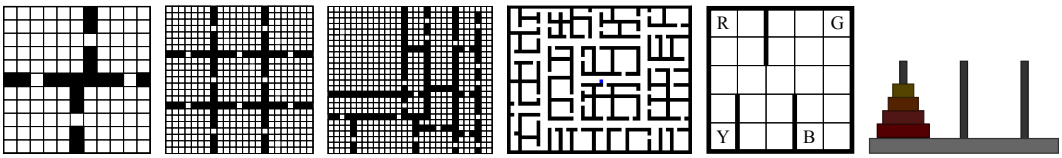


Figure 1: The environments.

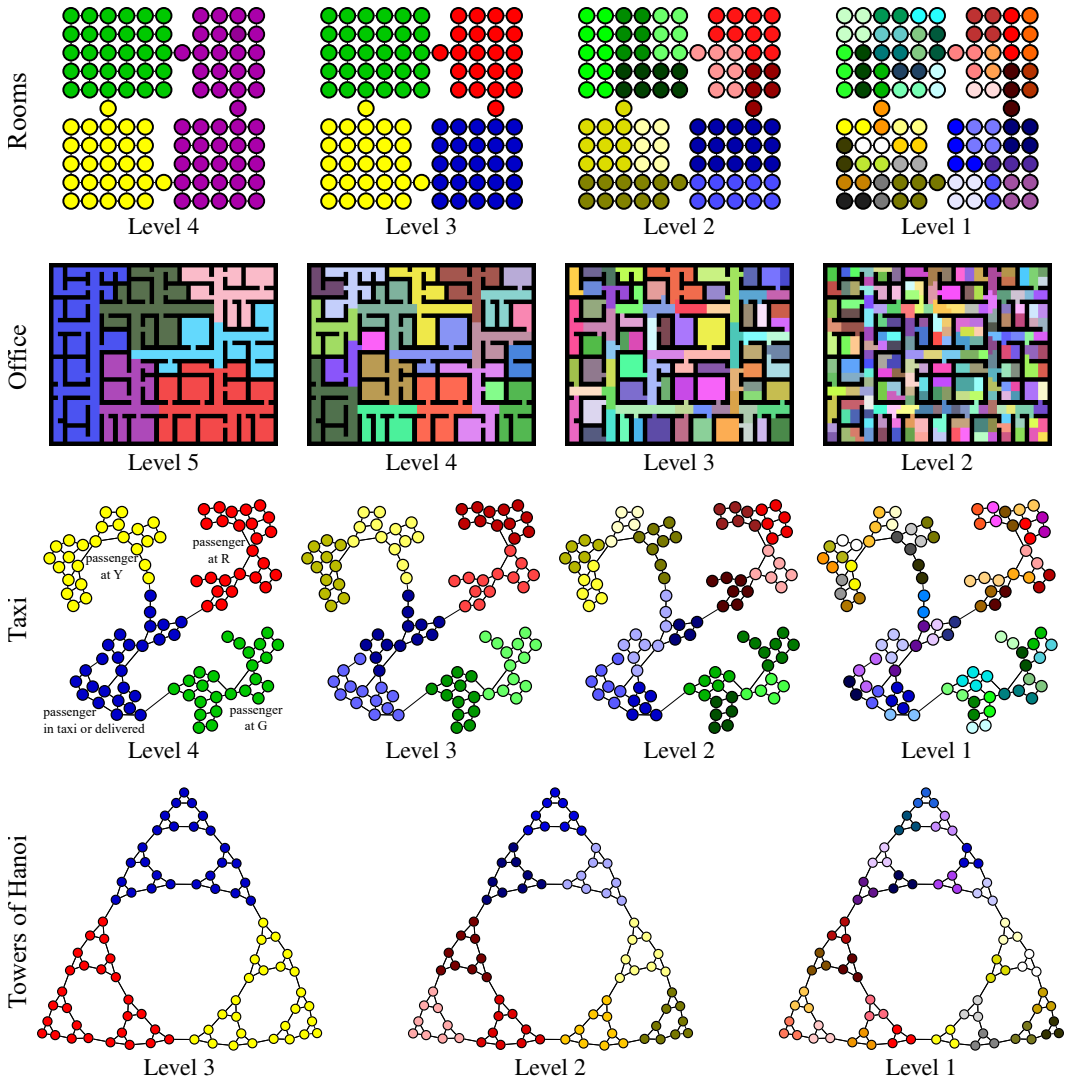


Figure 2: The cluster hierarchies produced by the Louvain algorithm when applied to the state-transition graphs representing Rooms, Office, Taxi, and Towers of Hanoi. For Taxi and Towers of Hanoi, the graph layout was determined by using a force-directed algorithm that models nodes as charged particles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

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 Louvain hierarchy has four levels. At the top level, we see three clusters where the passenger is waiting at R, G, or Y, and a fourth cluster where the passenger is either in-taxi or delivered to the destination. Navigation between these clusters is unidirectional, with only three possibilities, and the three corresponding skills navigate the taxi to the passenger location *and* pick up the passenger. Moving one level down the hierarchy, the clusters correspond to skills that move the taxi between the left and the right side of the grid, which are connected by a bottleneck state in the middle of the grid.

In Towers of Hanoi, the hierarchy has three levels. Moving between the top-level, middle-level, and bottom-level clusters corresponds to moving, respectively, the largest, the second-largest, and the third-largest disc between the different poles.

In each domain, (1) the Louvain skill hierarchy closely matches human intuition, and (2) it is clear how skills at one level of the hierarchy can be composed to produce the skills at the next level.

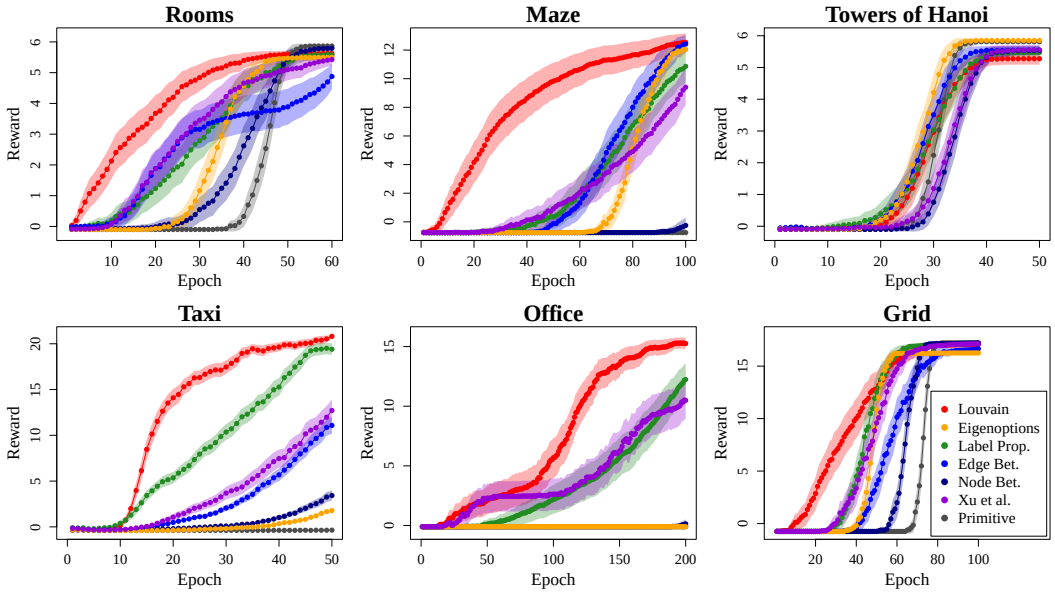


Figure 3: Learning performance. An epoch corresponds to 100 decision stages in Rooms and Towers of Hanoi, 300 in Taxi, 750 in Maze and Grid, and 1000 in Office.

Learning Performance. We compare learning performance with the Louvain skill hierarchy to the `Edge Betweenness` [8] and `Label Propagation` [12] methods, and to the method proposed by `Xu et al.` [11]. These methods are the most directly related to the proposed approach because they make use of modularity maximisation to define their two-level skill hierarchies. In addition, we compare to options that navigate to local maxima of `Node Betweenness` [15], a state-of-the-art subgoal-based approach that captures the bottleneck concept that characterises many existing subgoal-based methods. We also compare to `Eigenoptions` [28] derived from the graph Laplacian, a state-of-the-art Laplacian-based approach that encourages efficient exploration of the state space. Finally, we also include a `Primitive` agent that uses only primitive actions. Primitive actions are available to all agents.

For all methods, we generated options using the complete state-transition graph and learned their policies offline using macro-Q learning [37]. We trained all hierarchical agents using macro-Q learning and intra-option learning [38]. 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 [39]. The shaded regions on the learning curves represent the standard error over 40 random seeds. When creating the Louvain skill hierarchy, we discarded any lower level of the cluster hierarchy that had a mean number of nodes per cluster of less than a threshold value, c. We used $c = 4$ in all experiments; 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.

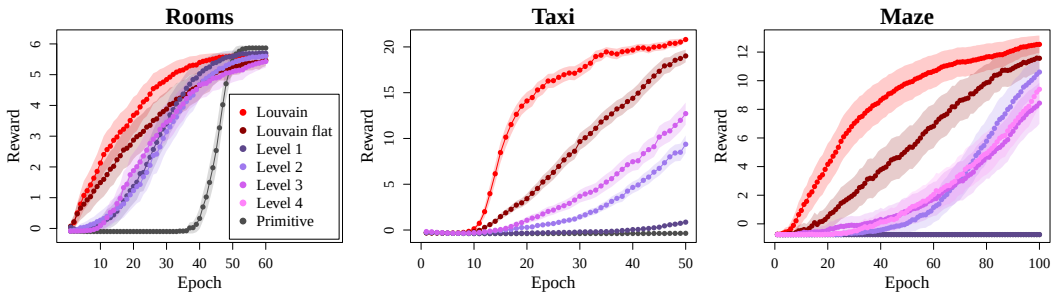


Figure 4: Learning curves comparing various different Louvain agents. An epoch corresponds to 100 decision stages in Rooms, 300 in Taxi, and 750 in Maze.

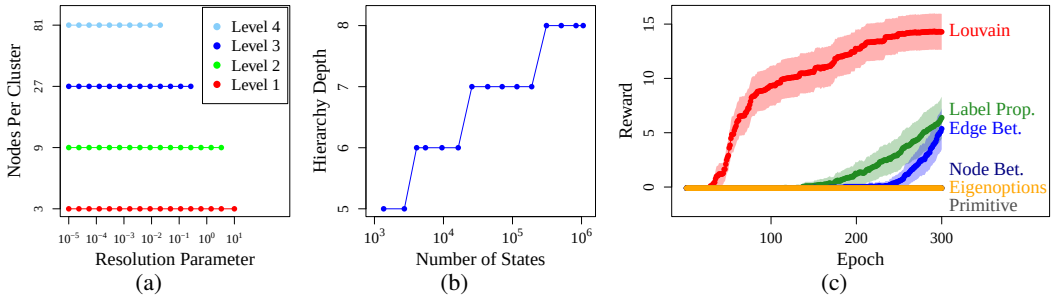


Figure 5: (a) How the Louvain algorithm's output when applied to Towers of Hanoi changes with the resolution parameter. (b) How the Louvain skill hierarchy's depth scales with the size of the state space. (c) Learning performance in Office with two floors containing 2537 states.

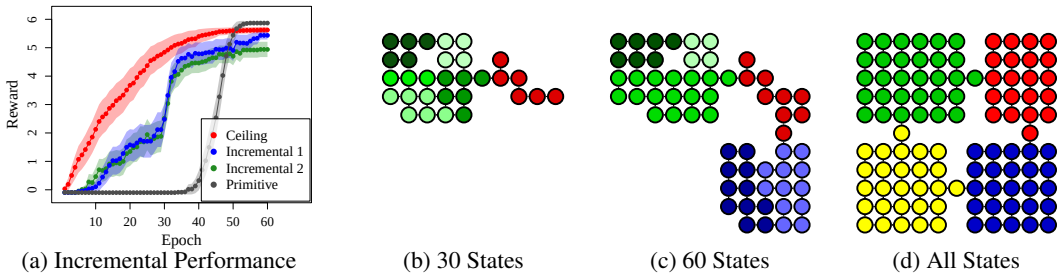
We show learning curves in Figure 3. The Louvain agent has a clear and substantial advantage over other approaches in all domains except for Towers of Hanoi, where its performance was much closer to that of the other hierarchical agents, with none of them performing much better than the primitive agent. This is consistent with existing results reported in this domain (e.g., by Jinnai et al. [29]).

Hierarchical versus flat arrangement of skills. An alternative to the Louvain skill hierarchy is a flat arrangement of the same skills, where the skills have the same behaviour but they call primitive actions directly rather than indirectly through other (lower-level) skills. We expect the multi-level hierarchy to lead to faster learning than the flat hierarchy due to the additional macro-Q and intra-option learning updates enabled by the hierarchical relationship between the skills. Figure 4 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, with option policies defined over primitive actions; these are depicted in the figure by the label `Level 1`, 2, 3, or 4. 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 ρ. Figure 5a shows how changing ρ impacts the cluster hierarchy produced by the Louvain algorithm in Towers of Hanoi. At $\rho = 10$, the output is a single level containing many small clusters comprised of three nodes. At $\rho = 3.3$, a two-level cluster hierarchy is produced: level 1 is identical to the partition produced at $\rho = 10$, but level 2 contains larger clusters, each formed by merging three of the clusters from level 1. Further decreasing ρ produces additional levels, each containing progressively fewer, larger clusters. As ρ is reduced, the clusters identified at a given level of the hierarchy generally remain stable: the lowest-level partition found using a higher value of ρ will typically be the same as the lowest-level partition found using a lower value. In other words, decreasing ρ may add levels to the hierarchy, but it generally does not impact the existing levels. A sensitivity analysis on the value of ρ showed that a wide range of ρ values led to useful skills being produced, and that performance gradually decreased to no worse than that of a primitive agent at higher values of ρ. Please refer to Appendix A for this sensitivity analysis and a more detailed discussion on the choice of ρ.

How do Louvain skills scale to larger domains? The Louvain algorithm has been successfully applied to graphs with millions of nodes and billions of edges in minutes [33], and has been observed empirically to have a time complexity that is linear in the number of graph edges [33, 34].

Also important is how the Louvain skill hierarchy changes with the size of the state space. 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 adjacent floors. The size of the state space can be varied by adjusting parameters such as the number of office floors. We generated a series of fifteen offices of increasing size, with the smallest office having a single floor ($\sim 10^3$ states), and the largest office having one thousand floors ($\sim 10^6$ states). Figure 5b shows that hierarchy depth increased very gradually with the size of the state space. At 1000 states, the hierarchy contained 5 levels, with the top-level skills allowing high-level navigation of each office floor, as well as from regions near an elevator to adjacent floors. At $\sim$4000 states, a sixth level was added, containing skills for efficiently moving from anywhere on one floor to an adjacent floor.



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

Figure 6: (a) Performance of Incremental 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 Incremental 2 method 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.

Subsequently, as the state space size grew to $\sim$25000 and $\sim$300000 states, two levels were added that allowed the agent to move, respectively, two and four office floors at a time. Figure 5c shows that the Louvain agent learns much more quickly than other approaches in a two-floor Office, even while some alternatives, including the Eigenoptions agent, fail to achieve any learning.

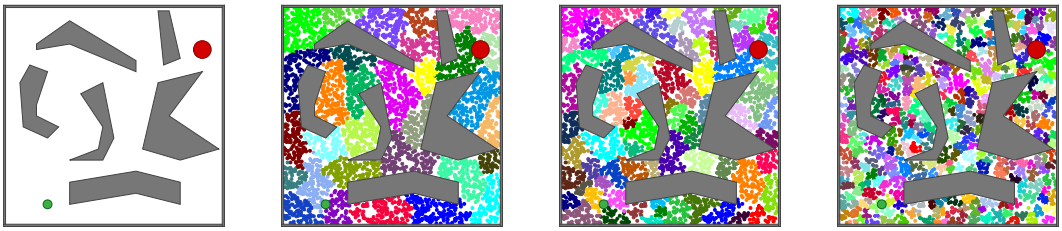
6 Discussion and Future Work

Our results show that the Louvain skill hierarchy is a useful answer to the conceptual question of what constitutes a useful skill hierarchy. This hierarchy is intuitively appealing, improves agent performance (more so than alternatives in the literature), and shows desirable scaling properties.

An important research direction for future work is incremental learning of Louvain skill hierachies as the agent is interacting with its environment—the state-transition graph will not always be available in advance. We explored the feasibility of incremental learning in the Rooms environment and present the results in Figure 6. 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 in one of two ways. In the first approach, the agent applied the Louvain algorithm to create a new skill hierarchy from scratch. In the second approach, the agent incorporated the new information into its existing skill hierarchy, using an algorithm similar to the Louvain algorithm. This algorithms starts by assigning each new node to its own cluster; it is 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 then repeated on the aggregate graph, and the next aggregate graph, and so on, until an iteration is reached with no modularity gain. Aside from existing clusters being merged into new higher-level clusters, the cluster membership of existing nodes stays fixed; only new nodes have their cluster membership updated. The result is a revised set of partitions from which a revised hierarchy of Louvain skills are derived. Pseudocode for these incremental approaches is in Appendix C.

Figures 6b–6d show the evolution of the partitions using the second approach as the 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. After 30 states, the top-level skills allow low-level movement within and between two of the rooms. After 60 states, another level was added, allowing slightly higher-level navigation of the state-space. After observing all possible transitions, the top level contained skills enabling efficient movement between the four rooms.

Figure 6a 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 following the two approaches discussed above. 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. Each incremental agent 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 the first approach reached a higher level of asymptotic performance than the second approach. The reason is that partitions produced early in training are based on incomplete information; the first approach discards



(a) Pinball Layout (b) Level 3 (c) Level 2 (d) Level 1

Figure 7: (a) Pinball domain, with the green ball in its initial position, the red goal, and several obstacles. (b–d) The cluster hierarchy produced by the Louvain algorithm.

the early imperfections while the second approach carries them forward. But there is a trade-off: the first approach has a higher computational cost than the second one. 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 advantages and disadvantages. 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 graph-based skill discovery methods due to the inherently discrete nature of the state-transition 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 [40, 23, 10] to examine the Louvain hierarchy in a variant of the Pinball domain [41], which involves manoeuvring a ball around a series of obstacles to reach a goal, shown in Figure 7a. The state is represented by two continuous values: the ball's position in the x and y 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 each edge (u, v) a weight of $e^{-\|u-v\|^2/\sigma}$. We then applied the Louvain algorithm to the resulting graph and derived the Louvain skill hierarchy from the partitions produced. The result was the three-level cluster hierarchy shown in Figure 7. The highest-level clusters yield skills for 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. Skills derived from the lower-level partitions enable more local navigation. Once again, we see a skill hierarchy that is intuitive, and it can easily be seen how the lower-level skills can be composed to produce the higher-level skills in the hierarchy.

Currently, Louvain skills at one level of the hierarchy are composed of skills from only the immediately previous level. While such skills may be optimal with respect to the skills from the previous level, they may not be optimal with respect to primitive actions. Future work could consider higher-level skills composed of skills from all lower levels, including primitive actions.

Because we derive Louvain skills solely from the connectivity of the state-transition graph, not considering the reward function, we expect them to be suitable for solving a range of tasks in a given domain. Examining their use for transfer learning is a useful direction for future work. Additionally, future work can examine how to use the reward function when defining Louvain skills to tailor the resulting skills for solving specific tasks.

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 of the problem. Future work should consider how best to manage large skill hierarchies. One solution that has been explored in the context of two-level hierarchies is "pruning" less useful skills from the hierarchy [24, 42].

Lastly, we point out 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 studying how different ideas on skills discovery can be brought together to enable agents that can autonomously develop complex and varied skill hierarchies.

Acknowledgments and Disclosure of Funding

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. We would like to thank the members of the Bath Reinforcement Learning Laboratory for their constructive feedback.

References

[1] 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.

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

[3] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[4] M. Riemer, M. Liu, and G. Tesauro. Learning abstract options. *Advances in neural information processing systems*, 31, 2018.

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

[6] A. Levy, G. Konidaris, R. Platt, and K. Saenko. Learning multi-level hierarchies with hindsight. In *Proceedings of International Conference on Learning Representations*, 2019.

[7] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 71–78, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015355.

[8] M. Davoodabadi and H. Beigy. A new method for discovering subgoals and constructing options in reinforcement learning. In *IICAI*, pages 441–450, 2011.

[9] J. H. Metzen. Online skill discovery using graph-based clustering. In *European Workshop on Reinforcement Learning*, pages 77–88. PMLR, 2013.

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

[11] X. Xu, M. Yang, and G. Li. Constructing Temporally Extended Actions through Incremental Community Detection. *Computational Intelligence and Neuroscience*, 2018. ISSN 16875273. doi: 10.1155/2018/2085721.

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

[13] 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, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44036-4.

[14] Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 95–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: 10.1145/1015330.1015353.

[15] Ö. Şimşek and A. G. Barto. Skill characterization based on betweenness. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1497–1504. Curran Associates, Inc., 2009.

[16] 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.

[17] 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.

[18] 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. ISSN 2077-1207.

[19] 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.

[20] 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*, pages 816–823, 2005.

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

[22] 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.

[23] 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.

[24] N. Taghizadeh and H. Beigy. A novel graphical approach to automatic abstraction in reinforcement learning. *Robotics and Autonomous Systems*, 61(8):821 – 835, 2013. ISSN 0921-8890.

[25] 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.

[26] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1.

[27] 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*, pages 3304–3310, 2019.

[28] 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 - Volume 70*, ICML'17, pages 2295–2304. JMLR.org, 2017.

[29] Y. Jinnai, J. W. Park, D. Abel, and G. Konidaris. Discovering options for exploration by minimizing cover time. In *International Conference on Machine Learning*, pages 3130–3139. PMLR, 2019.

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

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

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

[33] 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):P10008, 2008.

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

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

[36] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13(1):227–303, November 2000. ISSN 1076-9757.

[37] 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.

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

[39] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[40] 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.

[41] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1015–1023, 2009.

[42] 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 **Özgür Şimşek**
Department of Computer Science
University of Bath
Bath, United Kingdom
{jbe25|o.simsek}@bath.ac.uk

A Sensitivity to the Resolution Parameter

Here we explore how changing the resolution parameter ρ impacts the partitions produced by the Louvain algorithm and the performance of agents with the Louvain skill hierarchy.

The resolution parameter ρ controls the relative importance placed on the two terms that make up the definition of modularity: (1) e_{ii}, the proportion of total edge weight in the graph that connects two nodes that are in the same cluster c_i, and (2) a_i, the proportion of total edge weight in the graph where at least one end of the edge is a node in cluster c_i. A lower value of ρ favours the e_{ii} term, rewarding large clusters with many intra-cluster edges. This leads the Louvain algorithm to run for more iterations, producing more partitions and resulting in a deeper skill hierarchy. Conversely, a higher value of ρ favours the a_i term, penalising inter-cluster edges and overall cluster size more harshly. This leads the Louvain algorithm to run for fewer iterations, producing fewer partitions and resulting in a shallower skill hierarchy.

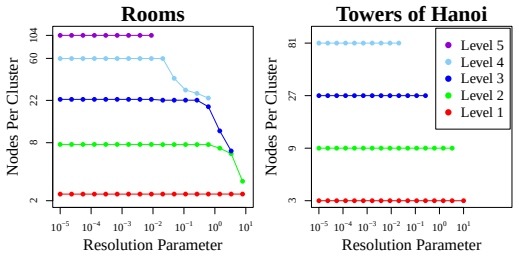


Figure 1: How the output of the Louvain algorithm changes with the resolution parameter in Rooms and Towers of Hanoi.

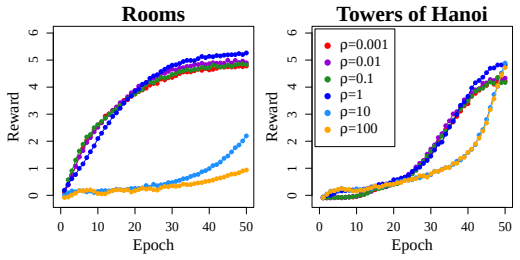


Figure 2: How agent performance changes when using Louvain skills created using different values of the resolution parameter ρ.

Figure 1 shows how changing ρ impacts the partitions produced in Rooms and Towers of Hanoi. At higher ρ values, fewer partitions are produced, each containing many small clusters. As ρ decreases, more partitions are produced because the Louvain algorithm performs additional iterations and merges the clusters found in earlier partitions to produce larger clusters. For example, in Towers of Hanoi, at $\rho = 10$, only a single partition containing many small clusters is produced. At $\rho = 0.1$, three partitions are produced, each containing progressively fewer, larger clusters. As ρ decreases, the clusters identified at a given level of the hierarchy generally remain stable: the lowest-level partition found using a higher value of ρ will typically be the same as the lowest-level partition found using a lower value. In other words, decreasing ρ may add levels to a hierarchy but it generally does not impact the existing levels of the hierarchy.

When using extremely high values of ρ, the Louvain algorithm terminates without producing any partitions. In contrast, when using extremely low values of ρ, the Louvain algorithm runs until a partition is produced where all nodes are merged into a single cluster.

Figure 2 shows the performance of agents in Rooms and Towers of Hanoi using Louvain skills created by using different values of ρ. We found that Louvain skills created using lower values of ρ consistently outperformed those created by using higher values, and that very high values ($\rho \geq 10$) generally led to performance similar to that obtained by using primitive actions only. This makes sense because 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, and all levels will be made up of skills that are useful and intuitively appealing.

B Louvain Algorithm Pseudocode

Algorithm 1 shows pseudocode for the Louvain algorithm. It takes a graph $G_0 = (V_0, E_0)$ as input and outputs a set of partitions of that graph into clusters, with one partition being produced for each iteration before termination. This series of partitions forms the basis of our characterisation of a useful skill hierarchy for reinforcement learning.

Algorithm 1: The Louvain Algorithm for Hierarchical Graph Clustering

parameters: resolution parameter $\rho \in \mathbb{R}^+$
input: $G_0 = (V_0, E_0)$ `// e.g., the state-transition graph (STG) of an MDP`
$i \leftarrow 0$
repeat
  $C_i \leftarrow \{\{u\} \mid u \in V_i\}$ `// define singleton partition`
  $Q_{\text{old}} \leftarrow$ modularity from dividing G_i into partition C_i
  repeat
    $C_{\text{before}} \leftarrow C_i$
    foreach $u \in V_i$ **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 $Q_{new} > Q_{old}$ **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 $C_0, \ldots, C_{i-1}$

C Incremental Method Pseudocode

Algorithm 2 shows pseudocode for a high-level algorithm where an agent, starting with only primitive actions and no information about its environment, incrementally builds a state-transition graph and creates Louvain skills from it.

Algorithm 2: Incrementally Discovering and Using 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 shows an incremental version of the Louvain algorithm that integrates new nodes into an existing Louvain cluster hierarchy.

Algorithm 3: Incremental Louvain Algorithm for Hierarchical Graph Clustering

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 $C_0, \ldots, C_{i-1}$

D Environments

Gridworlds. In Rooms, Grid, Maze [27], and Office, the agent learns to navigate from a starting grid-square to a goal grid-square. Four primitive actions are available in each state: north, south, east, and west. Moving into a wall causes the agent to remain in the same state. The agent receives a reward of -0.001 per action taken, and an additional $+1.0$ for reaching the goal state.

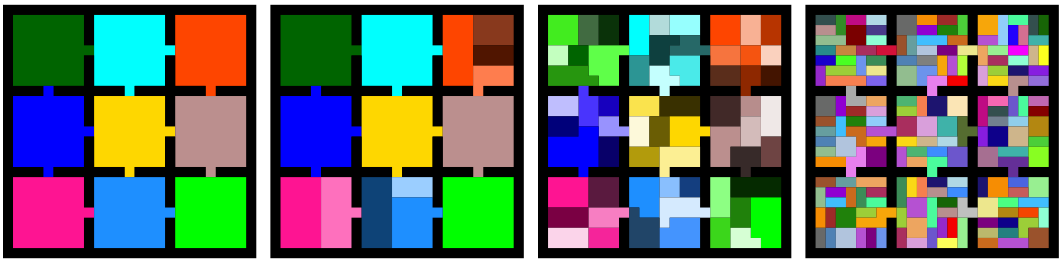
Multi-Floor Office. Multi-Floor Office is an extension of the Office environment to multiple floors, with each floor having a central elevator grid-square that allows the agent to move between adjacent floors. When standing on an elevator grid-square, the agent has access to two additional primitive actions: up and down. The dynamics of Multi-Floor Office are otherwise identical to Office.

Taxi. The taxi agent transports passengers between four taxi-ranks (R, G, B, and Y) in a gridworld [36]. The agent starts in a random grid-square, with a passenger waiting at one taxi-rank to be transported to a different taxi-rank. Six primitive actions are available in each state: north, south, east, west, pick-up-passenger, and put-down-passenger. Moving into a wall or inappropriately trying to pick up or put down a passenger cause the agent to remain in the same state. The agent receives a reward of -0.001 for each action, and $+1.0$ for delivering a passenger to their destination taxi-rank.

Towers of Hanoi. An agent starts with four different-sized discs stacked on the leftmost of three pegs, and its goal is to move them to all be stacked on the rightmost peg. In each state, the agent has access to primitive actions that move the top disc from one pole to any other pole, with the constraint that a larger disc cannot be placed on top of a smaller disc. The agent receives a reward of -0.001 for action action, with an additional reward of $+1.0$ for reaching the goal state.

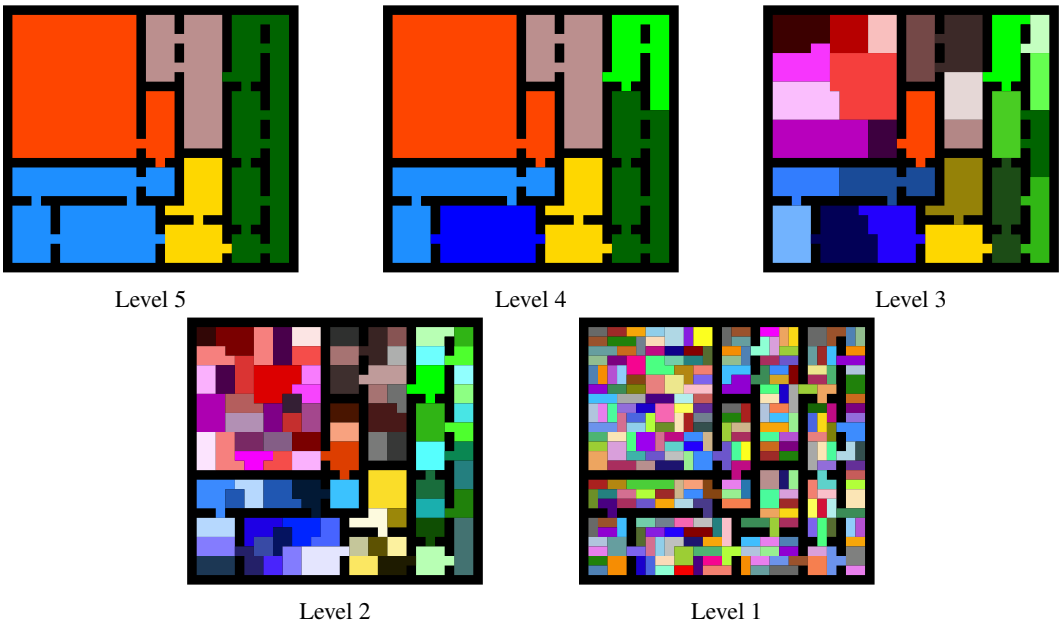
E Full Cluster Hierarchies Produced in Sample Domains

Here we show the Louvain cluster hierarchies in Grid and Maze environments. We also show the first level of the hierarchy in Office, which is omitted in the main paper due to space limitations.



Level 4 Level 3 Level 2 Level 1

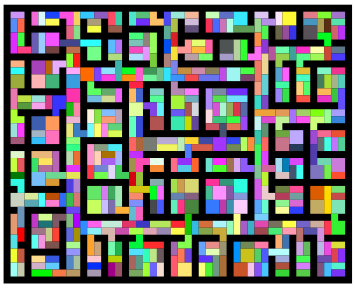
Figure 3: Partitions produced in Grid.



Level 5 Level 4 Level 3

Level 2 Level 1

Figure 4: Partitions produced in Maze.



Level 1

Figure 5: Lowest-level partition produced in Office.

F Methodology

How we produced options in each environment. For Louvain options, we started by applying the Louvain algorithm ($\rho = 0.05$) to the state-transition graph. We discarded any lower-level partition that had a mean number of nodes of less than 4. For each cluster c_i in the remaining partitions, we defined options for navigating to each neighbouring cluster c_j.

To produce skills using the Label Propagation [12], Edge Betweenness [8], and Xu et al. [11] methods, we applied the appropriate graph partitioning method to the state-transition graph and defined skills for moving between neighbouring clusters in the resulting partition.

We produced Eigenoptions [28] 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, we used the first 32 eigenvectors (and their negations). In all other environments, we used the first 16 eigenvectors (and their negations).

Finally, we produced options for navigating to local maxima of betweenness [15]. We computed the betweenness centrality of each node on the state-transition graph and selected all local maxima as subgoals. We defined options for navigating to each subgoal from the nearest 30 states.

We arranged the Louvain options into a multi-level hierarchy, where options for navigating between clusters in partition i were able to call skills for navigating between clusters in partition $i - 1$, and only options at the lowest level of the hierarchy were able to call primitive actions. All of the options produced using alternative methods called primitive actions directly.

How we trained option policies. For all methods except Eigenoptions, we trained option policies using macro-Q learning [37]. We used 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 the clustering-based methods, we started the agent in a random state in the source cluster, and gave them rewards of -0.01 for each action taken and an additional $+1.0$ for reaching a state in the goal cluster. For node betweenness, we started the agent in a random state in the initiation set, and gave it a reward of -0.01 each decision stage, an additional $+1.0$ for reaching the subgoal state, and an additional -1.0 for leaving the initiation set. For Eigenoptions, we used value iteration to produce policies that maximised each Eigenoption's pseudo-reward function.

How we trained our agents. We trained all agents using macro-Q and intra-option learning updates, which were performed every time an option at any level of the hierarchy terminated. We used a learning rate of $\alpha = 0.4$, discount factor of $\gamma = 1$, and initial action-values of $Q_0 = 0$ in all experiments. All agents used an ϵ-greedy exploration strategy with $\epsilon = 0.1$.

All learning curves we present show evaluation performance. After training each agent for one epoch, we evaluated the learned greedy policy by disabling exploration and learning (i.e., by setting $\epsilon = 0$ and $\alpha = 0$) and recording the rewards collected for one epoch in a separate instance of the environment.

To account for the stochasticity in the training process and the graph clustering algorithms used, we report the mean performance and standard error over 40 runs.

How we trained the incremental agents. The training procedure for the incremental agents was largely identical to that of the non-incremental agents. The only difference is that the agent started with access to only primitive actions and, after a set number of decision stages, updated its state-transition graph and option hierarchy. A high-level overview of this training procedure is shown in Algorithm 2.

How we derived Louvain skills in Pinball. We used the existing graph construction method of Mahadevan and Maggioni [40] to derive Louvain skills in the Pinball environment [41]. We started by randomly sampling 4000 states, and added edges between nodes representing each state and its 10 nearest neighbours. We then assigned each edge (u, v) a weight of $e^{-\|u-v\|^2/\sigma}$, with $\sigma = 0.25$. Finally, we applied the Louvain algorithm ($\rho = 0.05$) to the resulting graph.

G Comparison to Betweenness Skills

Here, we explore the similarities and differences between Louvain skills and skills that take the agent to local maxima of betweenness [15]. This is a state-of-the-art subgoal-based approach that captures the bottleneck concept. Like us, this earlier work put forward an explicit skill characterisation, but presented only a single layer of skills defined over the primitive actions.

Many Louvain skills involve traversing bottlenecks in the state-space, such as navigating between rooms in a gridworld, or picking up the passenger and navigating between the walled sections in the Taxi domain. These bottlenecks are local maxima of betweenness. In Towers of Hanoi, all Louvain skills traverse states also identified as local maxima of betweenness: the highest local maxima of betweenness correspond to states separating the level 3 clusters, and the second-highest local maxima of betweenness correspond to states separating the level 2 clusters. However, not all Louvain skills traverse bottlenecks. In Rooms, lower levels of the cluster hierarchy divide each of the rooms into smaller regions, and skills for navigating between these regions do not correspond to traversing bottlenecks.

Louvain skills and betweenness skills also differ in how they can be arranged hierarchically. Even in Towers of Hanoi, where there is clearly hierarchical structure present between the larger and smaller local maxima of betweenness, it is unclear how to exploit the betweenness metric to form a multi-level hierarchy. In contrast, the modularity metric approximated by the Louvain algorithm provides us with a clear and principled way of building a multi-level hierarchy.

H Compute resource usage

Our experiments were run using a shared internal CPU cluster with specifications shown in Table 1.

Table 1: CPU cluster specifications

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

We utilised approximately 40 CPU cores for approximately 336 hours when producing the final set of results presented in the paper. Prior to this, we utilised approximately 20 CPU cores for approximately 168 hours during preliminary testing.

We did not make use of GPU acceleration because our experiments involved tabular reinforcement learning methods. We do not have access to estimates of our CO_2 emissions at this time.