Evolving Systems
https://doi.org/10.1007/s12530-022-09478-6

ORIGINAL PAPER

Autonomous acquisition of arbitrarily complex skills using locality based graph theoretic features: a syntactic approach to hierarchical reinforcement learning

Zeynep Kumralbaş[1] · Semiha Hazel Çavuş[1] · Kutalmış Coşkun[1] · Borahan Tümer[1]

Received: 18 July 2022 / Accepted: 7 December 2022

Abstract
With the growing state/action space, learning a satisfactory policy for regular Reinforcement Learning (RL) algorithms such as flat Q-learning becomes quickly infeasible. One possible solution to handle such cases is to employ hierarchical RL (HRL). In this work, we present two methods to autonomously construct (1) skills (*ASKA*) and (2) arbitrarily elaborate *superskills* or *complexes* through defining an arbitrary number of hierarchies in HRL (*ASKAC*) over a graph-based iteratively-growing environment model. We employ dynamic community detection (DCD) in detecting subgoals since DCD considers local changes only over the partially growing graphs and lowers the time complexity of the subgoal detection where groups of environment states (i.e., subenvironments) are modeled by *communities* from the graph theory. DCD's drawback is oversegmentation where it mispartitions a subenvironment further into smaller components. To maintain the robustness of *ASKAC* against DCD's possible oversegmentation we introduce the concept of *skill coupling*. Skill coupling does not only robustly solve the oversegmentation issue, but it also improves HRL by building up more elaborate complexes (i.e., skill compositions) obtained at an arbitrary number of hierarchies and reduces the number of decisions leading to the goal employing these complexes. In addition to the experiments that investigate the effect of parameters, proposed methods are experimentally evaluated in grid world and taxi driver benchmark environments.

Keywords Reinforcement learning · Hierarchical reinforcement learning · Skill construction · Skill coupling · Temporal abstraction · Community detection · Dynamic community detection

1 Introduction

RL is a branch of machine learning inspired from mammals' sequential problem-solving capabilities. Learning to solve their daily tasks composed of sequential components requires the execution of a series of actions. As a powerful tool for sequential analysis, RL may remedy complex real world problems of sequential nature. RL is a paradigm where an agent learns from interactions with its environment to take consecutive actions at each specific state of the environment. Learning involves attaining a goal state. The goal state is represented by the maximum of an expected total reward, which is some cumulative form of the individual responses provided by the environment per visited state-action pair.

As the environment (its state/action space) grows too large, converging to a satisfactory policy for regular RL algorithms such as model-free *Q-learning* or model-based *prioritized sweeping* becomes quickly infeasible. So, splitting up the environment (hence its state space) into *subenvironments* (i.e., clusters of states) based upon the structure of the environment and forming the policy from a set of *subpolicies* each learned in one of these subenvironments is an effective solution to RL with a large state space. Learning each component called an *option/skill/macro-action* and learning the main policy using these skills occur in

✉ Zeynep Kumralbaş
zeynep.kumralbas@mind-rg.com

Semiha Hazel Çavuş
hazel.cavus@mind-rg.com

Kutalmış Coşkun
kutalmis.coskun@mind-rg.com

Borahan Tümer
borahan.tumer@marmara.edu.tr

[1] Faculty of Engineering, Marmara University, Istanbul, Turkey

Published online: 04 January 2023

two different levels or *hierarchies* of learning. Hence, this process of splitting the environment into subenvironments divides, in fact, learning into hierarchies. In each such subenvironment, one or more subtasks are defined depending upon the *subgoals* selected and each of these subtasks is considered as an individual RL problem at a smaller scale in the first[1] hierarchy. Then in the following hierarchy, the same (or other) RL algorithms may be employed to solve the original problem now using the skills learned in the former hierarchy as the building blocks to compose the main policy. This advanced technique is called HRL (Sutton et al. 1999). In HRL, there are skills as well as *primitive actions*. A skill representing a specific subpolicy is learned to solve a certain subtask by reaching a subgoal from some initial state within a certain subenvironment.

A memory-based RL agent progressively constructs a partial model of the environment it interacts with in a dynamic fashion, where the partial model grows more similar to the actual environment as learning proceeds. Thus, subgoals are also detected in a dynamic manner (Xu et al. 2018). In RL, environments may be effectively modeled using graphs (Xu et al. 2018; Simsek and Barreto 2008; Davoodabadi and Beigy 2011a, b; Shoeleh and Asadpour 2017; Kazemitabar et al. 2018; Farahani and Mozayani 2019). The connection points of subenvironments may be good candidates for subgoals by their nature. Hence, subgoals may be considered as bottlenecks of a graph, and they may be isolated from other nodes using *betweenness centrality (BC)* (Simsek and Barreto 2008), a graph property that defines bottlenecks very well. However, calculation of BC values has a *cubic* time complexity with respect to the number of states. Further, since subgoals might change from one episode to another during the partial model construction, BC values should be calculated from scratch per episode which makes the time complexity $O(kn^3)$ with k the number of episodes throughout learning and n the number of states. Extending intervals among each BC computation from one episode to several might result in a possible improvement in computation speed only with the cost of potentially delayed detection of subgoals while the eventual time complexity of the process would still not change. The problem with using BC values is that its calculation brings a computational overhead.

Although this problem as explained appears to be a problem within the RL domain (improving the time complexity in HRL paradigm) this problem manifests much similarity to the *community detection* problem within the context of *graph theory*. Communities, a common practice in graph theory, may be considered as good candidates to model the subenvironments in RL where a community is defined as a group of nodes which, in a set, exhibit a relatively higher similarity in terms of their graph-theoretic properties (Newman and Girvan 2004). There are community detection algorithms that run on static graphs (Blondel et al. 2008; Traag et al. 2019; Waltman and Van Eck 2013) and others that do on dynamic graphs (Aktunc et al. 2015; Cordeiro et al. 2016; Zhuang et al. 2019).

The advantage of using communities is that they can be dynamically detected. When a partial graph is updated, there is no need to re-detect all communities from scratch. In *DCD* algorithms, only those communities affected by the addition of new nodes or edges are disbanded. Then, a local check is run that focuses only on these n^* newly added nodes and those within the disbanded communities where $n^* \ll n$. The remaining communities do not get involved in this check. Hence, a new community that forms up is not missed by this $O(n^*)$ local check, while a potential time waste due to the re-detection of already existing communities is avoided.

DCD is a convenient solution for detecting subgoals both from the aspects of accuracy and time complexity. One critical problem DCD algorithms are faced with is their high dependency on the resolution parameter while specifying the number and sizes of communities. A small change in the resolution parameter's value is likely to cause occasional changes both in the number and sizes of communities. This problem mostly manifests itself as an *oversegmentation* (and rarely as undersegmentation) of communities. Hence, the differences of the graph models with the suboptimally sized communities from that with the optimal community distribution (suboptimal in the sense that since they are oversegmented they do not yield or reflect the maximum modularity that the graph actually possesses) bring up a risk for the consistency of the policy learning for an RL environment; i.e., two different representations of the same RL environment may end up at two different policies learned and computation complexity of learning may be significantly different.

In this work, we present *Autonomous SKill Acquisition and Coupling (ASKAC)* methods (and *Autonomous SKill Acquisition (ASKA)* without skill coupling), an HRL algorithm that autonomously acquires higher level skills, *complexes* or *superskills*, at the second or higher hierarchies within its graph environment where communities are dynamically detected utilizing *Dynamic Community Detection by Incrementally Maximizing Modularity (DynaMo)* presented by Zhuang et al. (2019). To overcome the general oversegmentation problem in DCD algorithms (in particular in DynaMo) explained above due to their resolution parameter-dependent nature and come up with a robust approach, we introduce *skill coupling*, a novel concept in our work. With this capability, ASKAC unifies two successive skills (or superskills at the higher hierarchies) selected sufficiently frequently together in contiguous communities (i.e., subenvironments) to form a complex. Constructing a

[1] We denote the first and lowermost hierarchy as hierarchy #0 (H0), which involves single-step actions (or primitives).

complex by coupling two successive complexes at higher hierarchies (hence, several successive skills) the concept of skill coupling is analogous to the process of devising a skill out of multiple primitive actions from the first to the second hierarchy. Hence, skill coupling extends the idea of temporal abstraction in HRL, in terms of *robustness*, by adding an arbitrary number of hierarchies where more robust policies are obtained against the possibility of suboptimal community detection (i.e., oversegmentation) in DCD methods. Skill coupling neutralizes the differences among the representation alternatives for the same (sub) environment and from the viewpoint of the complexity of the policies learned. We show in Section 5 (Fig. 17) that the performance of ASKAC remains the same while that of ASKA (hence other state-of-the-art methods since ASKA outperforms them) experiences a reduction, which inevitably changes as a response to a potential change in the resolution parameter in the DCD algorithms since ASKA is incapable of skill coupling. Our novelties in this work are listed as follows:

- We introduce *skill coupling*, a conditional entropy based technique to deal with oversegmentation, which is capable of autonomously constructing higher-level skills, *complexes or superskills* (as stated by Sutton et al. 1999), stretched over an arbitrary number of hierarchies and hence an extension on HRL by significantly reducing the number of decisions needed to reach the goal. We will be using both terms, complexes or superskills, interchangeably in this work to mean higher level skills obtained through skill coupling at higher (i.e., > 1) hierarchies.
- We improve the robustness of DCD based HRL approaches against the resolution parameter-dependent nature of DCD methods potentially ending up at different suboptimal communities detected even for slightly different values of the resolution parameter (i.e., oversegmentation).
- We present ASKA and ASKAC, the former an HRL method using DynaMo for the first time as a DCD technique, and the latter ASKA's version equipped with skill coupling to increase its robustness against DCD methods' oversegmentation issue.

In the rest of this paper, we continue with the preliminaries and a review of relevant literature in Sections 2 and 3, respectively. Following a discussion of our methodology in Section 4 that provides the details about the subgoal detection, skill construction (Section 4.1) and coupling (Section 4.2), the experiment setup and results are discussed in Section 5. We conclude with the final remarks and the future work in Section 6.

2 Preliminaries

In this section, we introduce some basic concepts used in the rest of this study.

2.1 Reinforcement learning

RL is a sequential decision-making paradigm where an agent learns from interactions with its environment based upon evaluative feedbacks provided by the environment (Sutton and Barto 2018). Agent chooses an action a_t at time t, at state s_t, moves to state s_{t+1} with probability $P(s_t, a_t, s_{t+1})$, and receives an immediate reward r_{t+1}. So, it can learn a satisfactory (hopefully optimal or possibly a near optimal) policy, π^*, that leads the agent to a goal state characterized by a maximum cumulative reward of the environment, $\mathbb{E}\{\sum_{t=0}^{\infty} \gamma^t r_t\}$. RL may also be modeled with a Markov Decision Process (MDP). An MDP is described with a quintuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state transition function, $0 < \gamma \leq 1$ is the discount factor, $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function. The action-value function for policy π is indicated as in Eq. (1):

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right] \tag{1}$$

where k is the number of time steps elapsing between s_t and s_{t+k+1}. One possible method to approximate the Q^* is Q-Learning (Watkins and Dayan 1992) which is a model-free and off-policy Temporal Difference (TD) control algorithm with an update rule defined as,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2}$$

where the learning rate $\alpha \in [0, 1]$ and the discount factor $\gamma \in [0, 1]$. In Q-Learning, the Q value of a single state-action pair is updated in each step. A model-based algorithm, Prioritized Sweeping (PS), is an improved version of the Q-Learning where the agent is capable of planning using the partial model of the environment that it progressively constructs and updates. In PS, multiple eligible state-action pairs are updated using N steps backtracking. The Q values of the state action pairs are prioritized, and those with higher priority, i.e., those whose Q value is non-zero, are updated sooner.

As the environment grows too large, converging to a satisfactory policy becomes quickly infeasible for regular RL algorithms such as flat Q-Learning or model-based algorithm PS. One possible solution to handle such cases is to employ HRL while dividing the problem into a set of subproblems and using temporal abstraction in the *option*

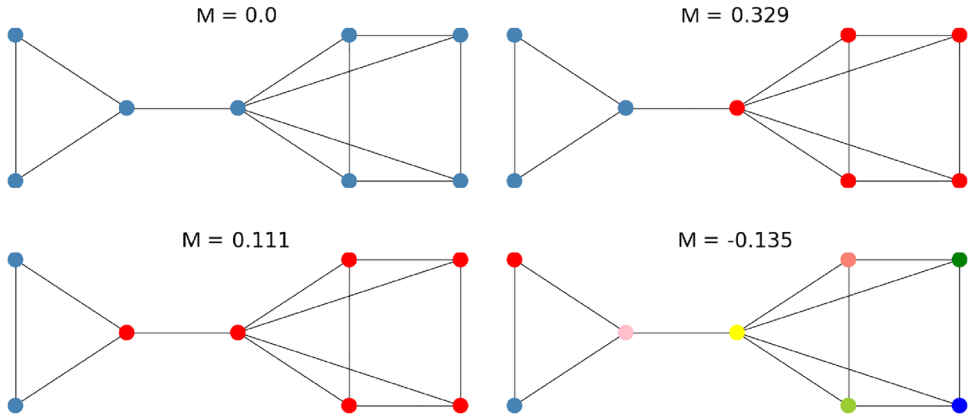


Fig. 1 Different modularities for a graph. Communities that nodes belong to are indicated by colors

(Sutton et al. 1999) framework. An HRL agent can make use of skills, temporally extended actions, that the agent has learned at a first hierarchy as well as primitive actions to construct the main policy at the next hierarchy. A skill $o := \langle \mathcal{I}, \pi, \beta \rangle$ consists of (1) a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, (2) an initiation set $\mathcal{I} \subseteq \mathcal{S}$, and (3) a termination condition $\beta : \mathcal{S}^+ \rightarrow [0, 1]$.

When the agent chooses a skill o at state s, the policy of the skill is followed until the termination condition is met. The skill value function is updated as in Eq. (3) once the skill is terminated:

$$Q(s_t, o_t) \leftarrow Q(s_t, o_t) + \alpha[r_{t+k+1} + \gamma^k \max_{o' \in O_{s_{t+k+1}}} Q(s_{t+k+1}, o') - Q(s_t, o_t)] \tag{3}$$

where the starting state of the skill o_t is s_t, k is the number of time steps elapsing between s_t and s_{t+k+1}, r_{t+k+1} is the cumulative discounted reward over this time.

In this study, we focus on autonomous construction of superskills (i.e., the components of the $\langle \mathcal{I}, \pi, \beta \rangle$ triplet) and their coupling.

2.2 Modularity and community detection in graphs

Modularity, denoted by M and given in Eq. (4), is a metric that reflects the strength of the commitment of nodes to its community (Newman and Girvan 2004), and it is strictly between -1 and 1 (Blondel et al. 2008). If the number of intra-community edges is no better than random, we will get $M = 0$. Values approaching $M = 1$, the upper limit, indicate strong community structure.

$$M = \frac{1}{2m} \sum_{i,j} \left[A_{i,j} - \rho \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{4}$$

In Eq. (4), $A_{i,j}$ is the number of edges between nodes i and j, ρ is the resolution parameter ($\rho > 0$), k_i is the degree of node i, k_j is the degree of node j, $\delta(c_i, c_j)$ is the Kronecker delta (1 if nodes i and j belong to the same community, 0 otherwise) and $2m$ is the total number of edges in the network.

A community is defined as a group of nodes which in a set exhibit a relatively higher similarity, i.e., based on various measures of similarity or strength of connectivity between vertices (Newman and Girvan 2004). Each community has similar nodes with dense interconnections, and these communities are connected sparsely with each other. Since modularity is an effective metric in identifying communities, it is widely used in community detection algorithms. In Fig. 1, the relation between community structure and modularity is visualized using different partitions on the same graph.

Detecting communities in a graph provides valuable information. The way this information is used may change depending upon the discipline and the problem. It is commonly used in social network analysis, network science and graph mining fields.

Community detection algorithms use the modularity function as an objective function to maximize. Maximizing modularity function is an NP-hard problem, so some heuristics are applied. These heuristics are based on (1) static community detection algorithms (Blondel et al. 2008; Traag et al. 2019; Waltman and Van Eck 2013) used for graphs that do not change in any timestamp and (2) dynamic community detection algorithms (Aktunc et al. 2015; Cordeiro et al. 2016; Zhuang et al. 2019) used for graphs that change in a timestamp. True communities may not always be found correctly. Two or more communities might be detected as a community and vice-versa. Oversegmentation problem, which divides a community into more communities, is more common in studies in RL. We discuss this problem in detail in Section 4.2.

3 Related work

For a skill to be obtained autonomously, the components of the triplet $\langle \mathcal{I}, \pi, \beta \rangle$ are needed. Therefore, there are many HRL studies that focus on detecting subgoals and/or defining initiation sets (Xu et al. 2018; Simsek and Barreto 2008; Davoodabadi and Beigy 2011a, b; Shoeleh and Asadpour 2017; Kazemitabar et al. 2018; Farahani and Mozayani 2019; Machado et al. 2017; McGovern and Barto 2001; Stolle and Precup 2002; Şimşek and Barto 2004a; Ghafoorian et al. 2013; Daniel et al. 2016; Cockcroft et al. 2020). Graph theoretic approaches are widely used for subgoal detection (Xu et al. 2018; Simsek and Barreto 2008; Davoodabadi and Beigy 2011a, b; Shoeleh and Asadpour 2017; Kazemitabar et al. 2018; Farahani and Mozayani 2019; Machado et al. 2017). Some of these studies apply different community detection algorithms to identify initiation sets besides detecting subgoals (Xu et al. 2018; Davoodabadi and Beigy 2011a, b; Shoeleh and Asadpour 2017; Farahani and Mozayani 2019). For example, *Label Propagation (LP)* (Raghavan et al. 2007), an agglomerative clustering approach, is used to detect communities in Davoodabadi and Beigy (2011a), Farahani and Mozayani (2019). Oversegmentation, which is the problem that forms the motivation of this study, is also encountered in Davoodabadi and Beigy (2011a), Farahani and Mozayani (2019). The approaches proposed for oversegmentation problem in Davoodabadi and Beigy (2011a) and Farahani and Mozayani (2019) are very similar and utilize the idea of merging the communities which will yield a higher modularity gain when merged. This approach is called *reformed LP* in Davoodabadi and Beigy (2011a) and *Modularity based LP (MBLP)* in Farahani and Mozayani (2019). Additionally, Farahani and Mozayani (2019) propose four different skill evaluation algorithms that mark a skill as good or bad, which enables the removal of useless skills after evaluation. In our work, instead of merging communities, we propose merging skills (which we call *skill coupling*) to solve the oversegmentation problem and we argue that this approach has some advantages over the previously proposed one, namely (1) since it can preserve the components of coupled skills (which are also skills from lower hierarchies) it is more robust to cases when a coupled skill is not useful anymore, and (2) it can in principle (and in practice in some cases) decrease the number of decisions to one, which is not plausible in community merging, since merging all the communities results in the same problem before community detection. Moreover, reformed LP and MBLP are only performed once after a predefined number of episodes, hence the community detection approach is static. Applying community detection at the early stages of learning might cause incorrect communities and, hence, inappropriate subgoals and initiation sets detected based upon the incorrect communities. On the other hand, waiting for the transition graph that represents the partial model to complete to perform community detection may delay the construction of skills. Since we use a DCD algorithm in our work, the starting point of DCD is not as important as in Davoodabadi and Beigy (2011a), Farahani and Mozayani (2019). The time complexity of reformed LP and MBLP is $O(mk + mL)$, where m is the number of edges, k is the number of iterations required for LP and L is the number of communities detected after running LP. Both studies found subpolicies of skills using *Experience Replay (ER)* (Lin 1992) and *intra-option Q-learning* (Sutton et al. 1998), which improve the subpolicies of skills throughout the learning process. In our work, when a skill is selected for the first time, its subpolicy is learned as a small RL problem and is used throughout learning.

Shoeleh and Asadpour (2017) proposed a graph based skill learning approach which extracts high-level skills in a continuous state space. First, a connectivity graph is built, which can be defined with a transition graph and a distance graph. Since the states are non-Markov (i.e., the probability for the agent to visit the same state again is zero), abstract states are used to build the transition graph and the distance graph is used to support the transition graph. Then, static community detection algorithms (Louvain (Blondel et al. 2008), InfoMap (Bohlin et al. 2014)) based on the connectivity graph are used to partition the state space into regions. Skills are then constructed on the basis of partitioned regions. Learning subpolicies is considered as another independent RL problem, as in our work.

DCD approach is also used by Xu et al. (2018). Louvain algorithm here is employed to find the initial communities (partitions). Then, communities are updated based on the initial communities using *Incremental Community Detection Algorithm (ICDA)*. ICDA is a rule-based algorithm that dynamically updates the communities. Each community is characterized as a macro-state (aka. aggregated), and skills are constructed between each of these macro-states. Subpolicies of skills are initialized using ER. Then, subpolicies are improved with intra-option Q-learning. If communities are changed during the learning, ICDA updates communities. Skills are reconstructed from the updated communities. If the previous and updated list of skills have the same instances, the learned subpolicies for those skills are retained. This study and ours have similar approaches (ICDA and DynaMo) to retaining subgoals and initiation sets since both of them use DCD algorithms.

The difference between DynaMo (Zhuang et al. 2019) and ICDA is in handling an intracommunity edge addition. In ICDA, adding an intracommunity edge is considered as strengthening the local modularity measure of the relevant community, and there is no need to change the existing community structure. In DynaMo, adding an intracommunity

edge results either in the same existing community structure as in ICDA, or in splitting the communities into two to have a better modularity gain. DynaMo might further evaluate edge and vertex removal, which we do not consider in our transition graph changes.

Kazemitabar et al. (2018) utilize both frequency-based and partition-based approaches to detect community-like structures. Since subgoals are in the border states of strongly connected regions and the visit frequency of border states is lower than that of a state within the same region, they remove less frequently visited edges to find such strongly connected regions. Then, the transition graph is divided into some clusters called *strongly connected components (SCCs)* using a linear time algorithm based on depth first search. Then the states connecting each SCC pair, namely subgoal states, were found considering border states of SCCs as potential candidates of subgoal states. This approach is also static and the components are only found at a predefined episode. Subpolicies of skills are learned using ER and intra-option Q-learning as in other studies.

A categorization of applying hierarchies in RL is proposed by Setyawan et al. (2022), which are hierarchies (1) in actions, (2) in learning agents, and (3) in environments. The studies discussed so far are based on hierarchy in actions. More recently, Setyawan et al. (2022) proposed *Micro-Macro States Combination (MMSC)* considering applying hierarchies in environments. A hierarchy is constructed based on (1) tasks, that are related to the original state of the environment, microstates, and (2) subtasks, that are represented by some collections of the microstate, macrostates. Macrostates are formed as the collection of adjacent microstates, and each microstate can be included in multiple macrostates. The agent attempts to solve the given problem by learning the best combination of microstates and macrostates. A macrostate has three components, $\langle I, \beta, \pi_M \rangle$, where I is the initial state, β is the termination condition, and π_M is the policy of a macrostate M. The subgoals are the states where β is 1, and they are set automatically based on the maximum Q-Value for each macrostate. Since they do not include any microstate with an obstacle (i.e., state that the agent can not be located at or pass through) in a macrostate, each macrostate is obstacle-free. Also, the size of the macrostate is determined empirically. The policy on macrostates is designed hand-coded as primitive actions that move the agent to the subgoal. In our work, subgoals are detected autonomously based on DCD algorithm. Since this work introduces hierarchy in environments, we may not be able to mention directly about skills here. However, a macrostate may be relatable to a skill in our work. A macrostate has components similar to skills, but with an I component as the initial state of the macrostate instead of the initiation set of the skill. Also, the size of a macrostate which is determined empirically may be relatable to the initiation set of a skill that is determined autonomously as the communities of subgoals in our work. Furthermore, the policy of a macrostate is hand-coded whereas the policy of the skills is found by solving independent RL problems in our work.

Machado et al. (2017) implicitly use Proto-Value Functions (PVFs) which are representations of learning to define options. PVFs are the eigenvectors of *the combinatorial graph Laplacian matrix* where it consists of the adjacency matrix of the transition graph. Since PVFs are not task dependent, they do not need to know about reward functions and they can speed-up learning in different tasks. Smoothest eigenvectors (i.e., the smallest eigenvalues) are used to construct options, and the options that are constructed with this method are called *eigenoptions*. Eigenoptions do not always lead to the bottlenecks, and they support exploration to corners of rooms. Although the number of options are given as a model parameter, the authors state that the method is fairly robust for different number of options.

Jinnai et al. (2019) improve the learning for tasks with sparse rewards. The proposed algorithm autonomously finds options that minimize the expected cover time which is the required time for a random walk to visit all the nodes in a graph. The method aims to find the two most distant nodes based on the current state transition graph in a greedy manner. The authors consider only point options. Since the point option has a single state for each initiation set and termination condition, an edge is added to the transition graph for the point option. Thus they decreased the required number of steps to walk between these two states to 1 using options. The number of options is given as a parameter. The minimization of cover time is done by maximizing the algebraic connectivity with an approximation method. Firstly, the second smallest eigenvalue and its corresponding eigenvector (the Fiedler vector (Fiedler 1973)) of the state transition graph are calculated in certain episodes. The largest and smallest values in the eigenvector are used to construct two options. This operation is repeated until the given number of options is reached.

Zhu et al. (2022) propose a new option discovery method, *Min Degree and Max Distance (MDMD)* options, that accelerate the exploration by reducing the expected cover time of the environment in sparse reward problems. Unlike (Jinnai et al. 2019), MDMD does not compute the Laplacian matrix's eigenvector. It selects two disconnected vertices with the smallest degree and largest distance to generate an option between them. Firstly, the vertex that has the minimum *scalability centrality index*, which is the sum of the degrees of the vertices connected with it, is chosen as the

min-degree vertex. Secondly, the max-distance vertex is found by breadth-first traversal from the min-degree vertex as the root node. These steps are repeated until the number of options reaches a predetermined value. The computational time complexity depends on the *breadth-first traversal* which has $O(|V| + |E|)$, where V is the number of vertices, E is the number of edges of the transition graph, the sparsity of the graph affects $O(|E|)$ in the range of $(O(1), O(|E|^2)$. Since the Laplacian matrix's eigenvector is not computed, the time complexity is better than Jinnai et al. (2019). Obtaining the min-degree and max-distance vertices and adding an edge between them makes the discovered options to achieve smaller the environment's expected cover time than (Jinnai et al. 2019).

Studies that construct point options (Machado et al. 2017; Jinnai et al. 2019; Zhu et al. 2022) utilize a subclass of the option framework. Two states for the initiation set and the termination condition are sufficient for constructing a point option whereas two sets of states should be found for constructing options. Also, point options do not need a subpolicy as in options. The transition between two states is achieved by adding an edge between these states. Machado et al. (2017) focus on the utilization of options for different tasks and exploration whereas the others focus on minimizing the covering time. Conversely, our study aims to autonomously find the option components and improve learning by decreasing the number of decisions. Although these studies and ours are proposed for autonomously constructing options, there are major differences in terms of scope and aims.

Skills, by their nature, are valuable outputs that can be transferred between tasks as well as between agents. Hence, HRL approaches are also used to solve transfer learning (Shoeleh and Asadpour 2017) and multiagent RL (Shafipour Yourdshahi et al. 2022) problems.

4 Methodology

In this section, we discuss ASKA and ASKAC in detail. ASKA is an HRL method with autonomous skill acquisition capability on environments modeled by graphs where ASKA obtains skills employing DynaMo that executes DCD on transition graphs that represent the partial model. ASKAC is the complex (or superskill) acquisition method obtained by extending ASKA incorporating skill coupling that functions at the second and higher hierarchies. Using skill coupling, ASKAC autonomously acquires complexes whereby it improves the performance of ASKA erasing the effects of the suboptimal communities obtained due to the resolution-dependent nature of DynaMo (Fig. 2). DynaMo splits the graph into a suboptimal number of (too many or too little) communities, caused by the resolution parameter that adjusts the scale of communities. In Section 4.1, we describe how the community information is utilized for subgoal detection and skill construction by ASKA. In Section 4.2, we discuss skill coupling that is merged to ASKA to extend it to ASKAC and to solve the oversegmentation problem.

The overall system is illustrated in Fig. 2. DynaMo achieving DCD as a component of ASKA, the skill coupling module and ASKAC, the extended version of ASKA with the integration of the skill coupling module are all shown in Fig. 2. The transition graph that represents the partial model constructed during the agent's moves within the environment is sent to DCD module DynaMo (Zhuang et al. 2019) within specific periods (selected in an application-dependent fashion) as long as changes occur in the communities.

DCD module detects the changes between the previous and the new transition graph that represents the partial model, and dynamically forms communities without detecting them from scratch. The communities detected are forwarded to ASKA or ASKAC and if there is a change in the community structure the set of skills is updated.

At each step, in ASKAC only, two successive skills at the second or complexes at higher hierarchies are coupled if they are selected by the agent significantly frequently.

4.1 Subgoal detection & skill construction

To maintain the generality of ASKA/ASKAC, we assume that the environment is mostly silent or irresponsive, i.e., totally unknown to the agent. The agent progressively constructs a partial model of and interacts with the environment. The partial model grows more similar to the actual environment as learning proceeds.

The environment in the RL framework is modeled by a state transition graph such that nodes and edges represent the states and actions, respectively. Nodes ranking high in terms of centrality (i.e., the frequency with which a node appears on the shortest paths between node pairs among all such shortest paths), in other words, nodes characterizing a bottleneck constitute good instances for a subgoal in an HRL environment (Simsek and Barreto 2008). Inspired by this fact, the connecting states of communities with significantly higher centrality are suitable candidates of subgoals in such a graph representation of an HRL environment.

The partial model changes over time as in Fig. 3a–d and so do the subgoals. The construction of the partial model of the environment is an iterative process during learning, where the model grows more similar to the real environment. Thus, a dynamic approach that realizes this progressive construction of the model is inevitable. This iterative approach starts with a newly created transition graph representing the initial partial model presented to DynaMo. DynaMo

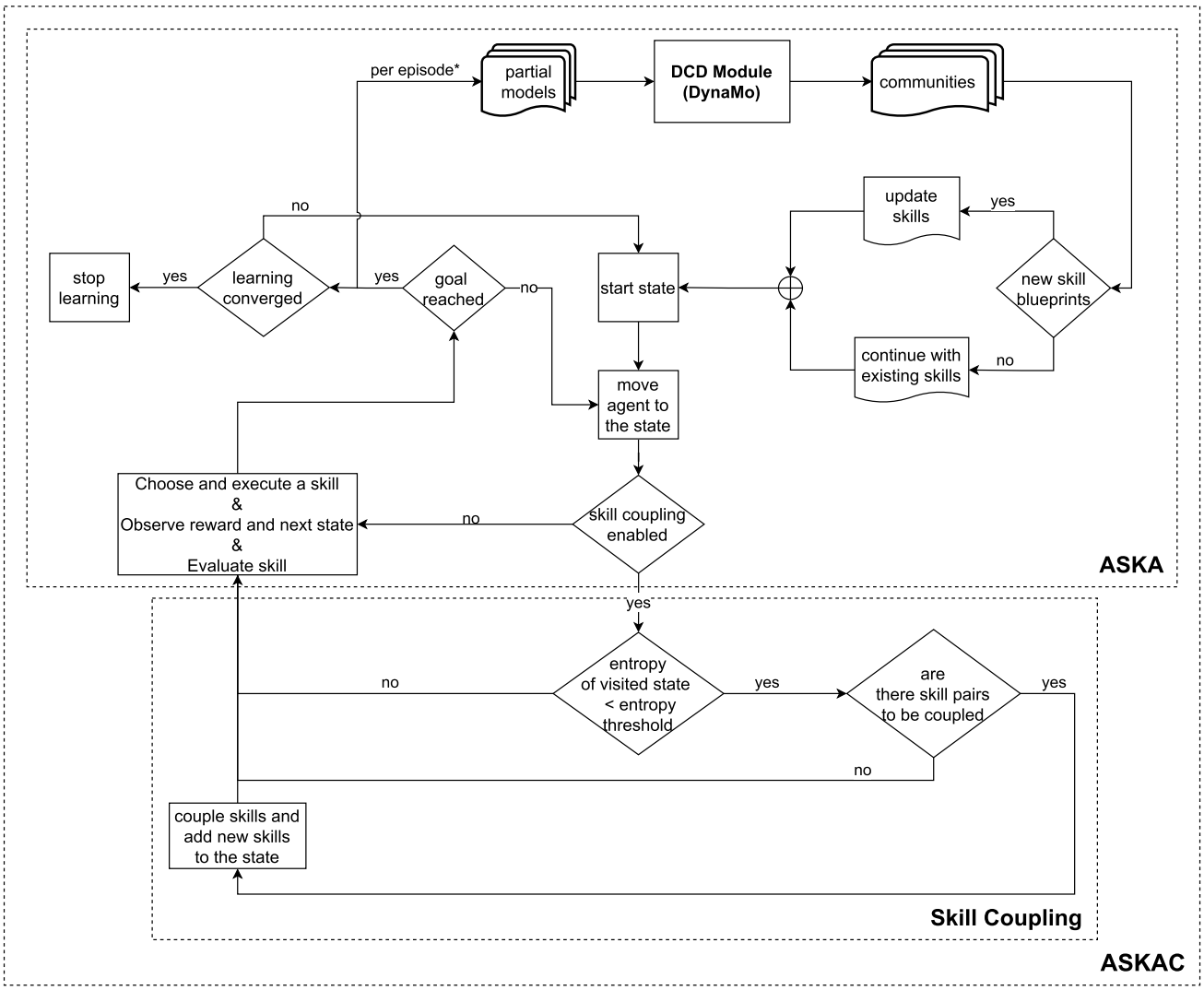


Fig. 2 System flow diagram

finds new communities based on the previous community information and the current transition graph as in Fig. 3. DynaMo's output, the community information, is sent back to ASKA/ASKAC. The interaction of ASKA/ASKAC with DynaMo is shown in the system diagram in Fig. 2.

After communities are formed, the detection of subgoals and the construction of the initiation sets are realized as shown in Algorithm 1. Subgoals are determined as nodes each connecting a pair of communities. If an environment region suffers from oversegmentation (i.e., the region is split into a suboptimal number of communities) the nodes along the borders of neighbor communities turn into subgoals as shown in Fig. 4. Therefore, an excessive number of skills are defined. In the flow of learning, only a few skills are used more frequently than all others. The initiation set is composed of all nodes in the community of a subgoal and all nodes in the community of a neighbor of the subgoal. Hence, the initiation set of a skill consists of states in more than one community. To construct the policy of a skill, an RL task is run in the environment which consists of the states in the initiation set (edges to other communities are not considered) of the skill and where the start state is chosen randomly in each episode and the goal state is the subgoal state. Now that the components of a skill ($\langle \mathcal{I}, \pi, \beta \rangle$ triplet) are determined, the skill becomes available for selection at the states in its initiation set lasting to the relevant subgoal state.

The subgoal detection and initiation set construction phase has $O(e)$ time complexity, where e is the number of edges in the graph that represents HRL environment.

Algorithm 1 : Subgoal Detection and Initiation Set Construction

Input: edge list E in graph G, community information from DynaMo
Output: SubgoalList, InitiationSets

for each node pair $(node1, node2)$ in E **do**
  state1 = corresponding state for $node1$
  state2 = corresponding state for $node2$
  if community($node1$) $\neq$ community($node2$) **then**
    Add $node1$ to SubgoalList
    Add $node2$ to SubgoalList
    InitiationSet[$node1$] + = nodes in community($node1$) + nodes in community($node2$)
    InitiationSet[$node2$] + = nodes in community($node1$) + nodes in community($node2$)
  else
    if state1 isGoal **then**
      Add $node1$ to SubgoalList
      InitiationSet[$node1$] + = nodes in community($node1$)
    else if state2 isGoal **then**
      Add $node2$ to SubgoalList
      InitiationSet[$node2$] + = nodes in community($node2$)
    end if
  end if
end for

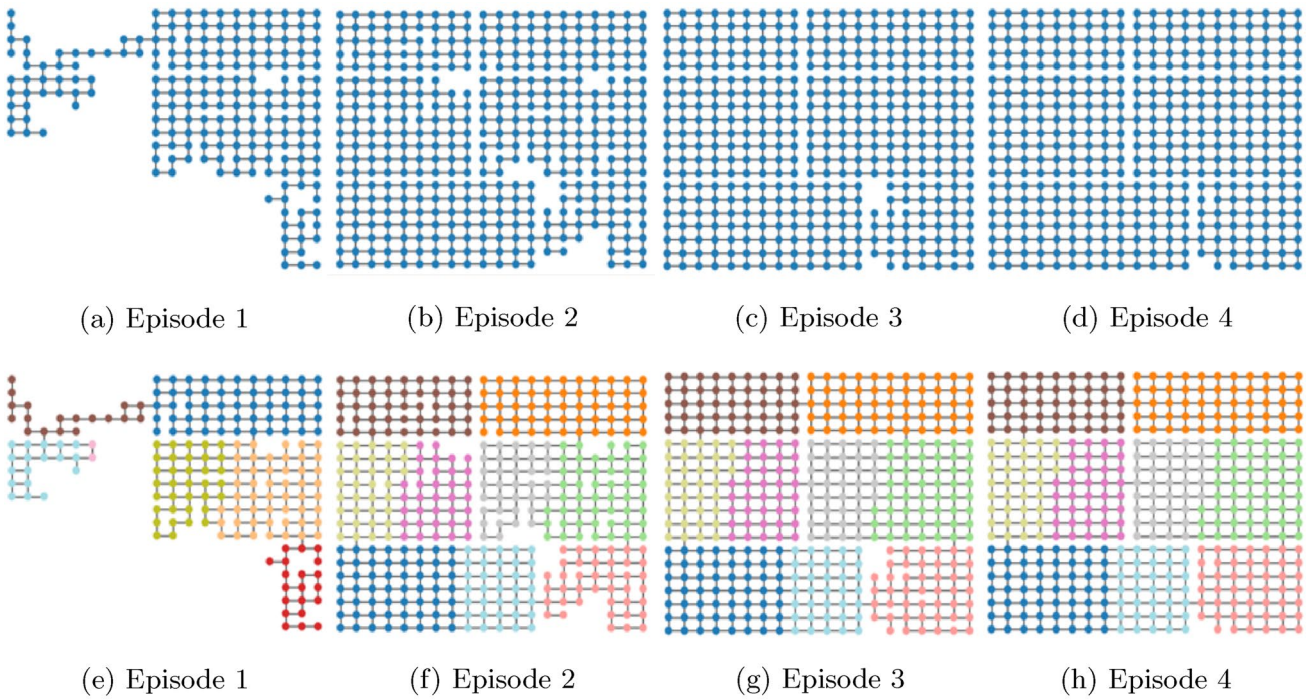


Fig. 3 The growth of a partial transition graph, (**a**)–(**d**), and detected communities, (**e**)–(**h**), in the first four episodes. The community structure is updated as new regions of the environment are explored

4.2 Skill coupling and hierarchy construction

We may define the oversegmentation (or rarely undersegmentation) problem as the partitioning procedure of a graph by a DCD algorithm into superfluously smaller (or in some cases larger), hence a suboptimal number of, communities. The emergence of oversegmentation depends upon the modularity aspect of the graph structure. An example that illustrates both the undersegmentation and oversegmentation problems is given in Fig. 5. In Fig. 5b, the DCD

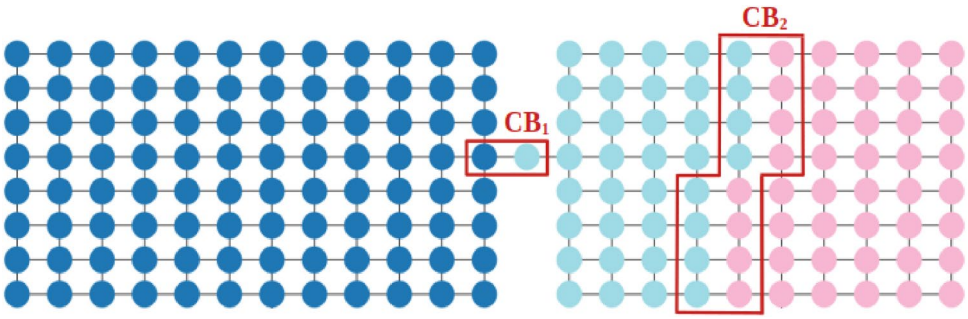


Fig. 4 As a result of oversegmentation, community borders span across wide regions instead of bottlenecks, which causes an unnecessarily large number of subgoals. If communities of the graph are detected correctly, there are 2 subgoals as in CB_1. However, there are 16 subgoals in CB_2, since the region on the right side is split into two communities

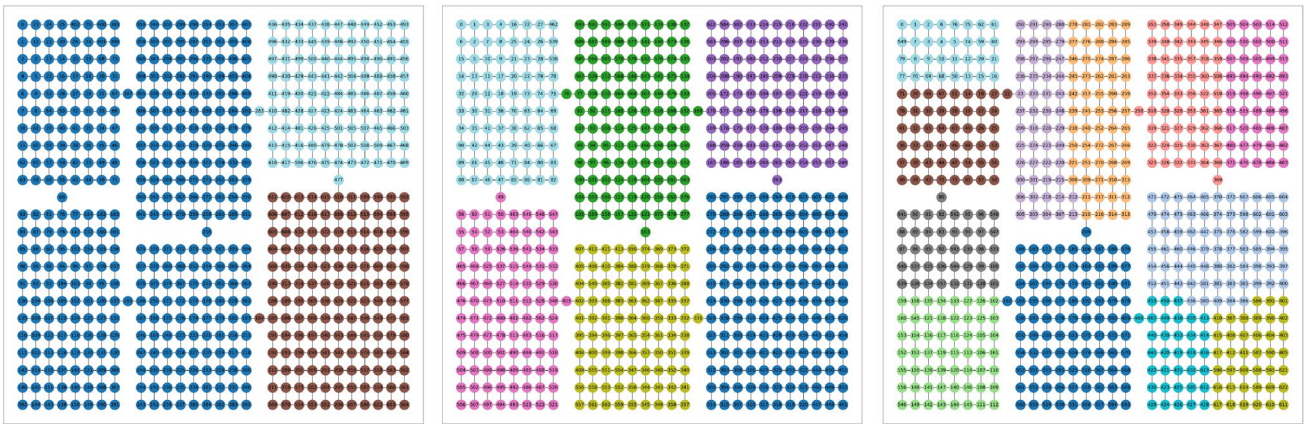


Fig. 5 Detected communities in the six-room environment with different values of the resolution parameter (ρ). Selection of ρ plays an important role in the resulting community structure. In this case, $\rho = 0.5$ correctly segments the graph into communities that correspond to individual regions. However, $\rho = 0.05$ and $\rho = 1$ lead to suboptimal segmentation, namely undersegmentation and oversegmentation, respectively

algorithm has achieved a proper distribution of nodes into communities that correctly models the distribution of states among the regions of the six-room grid world environment and detected an optimal number of communities while in Fig. 5a and 5c, the number of communities detected is too few (underestimation) and too many (overestimation) in respective order. One possible solution of the problem is the resolution parameter that is used for adjusting the scale of the communities to be detected (Reichardt and Bornholdt 2006). The corresponding resolution parameter value selected for each detection procedure is also shown in Fig. 5. For the underestimated, perfect and overestimated cases, the resolution parameter values are 0.05, 0.5 and 1, respectively. However, setting the resolution parameter right for different graph topologies is another challenging problem. Hence, with this aspect, DCD algorithms are the weak end of such graph based HRL methods. A potential remedy to this weakness may be a fault-tolerant method that endures the impact of and remains robust against the oversegmentation problem and the suboptimal communities detected by the DCD algorithms.

To robustly solve this oversegmentation issue, we present *skill coupling*, a novel solution, where, at an arbitrary number of hierarchies we introduce as an extension to HRL, skills (or complexes at higher hierarchies; for the rest of the discussion, by skill coupling we mean both coupling skills at the second and complexes at higher hierarchies) successively executed with prevailing frequencies are coupled into complexes to mend the potentially imperfect results of the DCD algorithms and improve the eventual results of ASKA (the improved version is ASKAC). As we discuss later in Section 4.2 one hierarchy is inserted to the hierarchy structure per skill in a complex (Fig. 7).

The method is designed to remain fault-tolerant throughout the learning procedure. We have inspired from the Syntactic Pattern Recognition (SPR) (Fu 1977) principles where

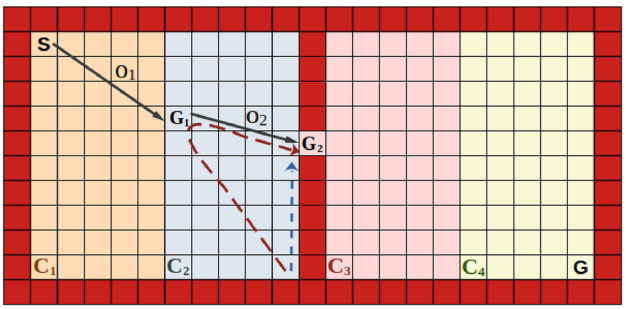


Fig. 6 An example community structure to support initiation set coupling given in Eq. (9). In this case, a simple set union of $\mathcal{I}_1$ and $\mathcal{I}_2$ would cause the agent to visit subgoal G_1 first before reaching G_2 (indicated with red dashed arrow), instead of directly heading towards G_2 (indicated with blue dashed arrow)

the *parsing*, the second phase of SPR, is devised adaptively so as to eliminate the effect of a possible suboptimal step taken in the *lexical analysis*, the first phase of SPR. Say, a clustering technique implemented in the lexical analysis to extract an alphabet of primitives from a given sequential dataset may not end up with an optimal alphabet cardinality (i.e., one that minimizes the reconstruction error while keeping the alphabet size as small as possible). We can still devise a parsing phase - among other techniques, by (a) reconstructing the sequential data so as to augment its dimensionality to cover several best representing cluster exemplars instead of the single best one and (b) choosing a model such as fuzzy (Tumer et al. 2003) or stochastic automata (Sürmeli and Tümer 2020) that diversifies the alternatives for a good representative for the original data token - that may eliminate the negative effects of suboptimal clustering. The idea of skill coupling is an analogous step to the adaptive parsing process taken in the direction of robustness to tolerate a detection of possibly suboptimal communities as a result of oversegmentation. The oversegmentation problem potentially leads to the detection of too many communities and, hence, the generation of premature skills (since obtained communities of these skills are smaller) that execute on these suboptimally detected communities. The skills are premature in the sense that they do not end up at the bottlenecks (the actual subgoals) of the environment (e.g., G_2 in Fig. 6) but at some intermediate state (e.g., G_1 in Fig. 6). These negative effects of the oversegmentation problem may be suppressed by coupling these premature skills into complexes. Here a criterion is required to decide on which skills to be coupled. We utilize the *entropy* of two successive skills at a state. The entropy of a discrete random variable X with outcomes $x_1, \ldots, x_n$ and their probabilities $P(x_1), \ldots, P(x_n)$ is given in Eq. (5).

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i) \tag{5}$$

Entropy of a state reveals whether any skill pair from the state is worth coupling. Let $\mathcal{S} = \{s_0, s_1, \ldots, s_n\}$ be the set of states, with $|\mathcal{S}| = n$. Each state has a set of skills $\mathcal{O}_{s_i} = \{o_{i1}, o_{i2}, \ldots, o_{ik}\}$ and these skills have Q-Values $\mathcal{Q}_{s_i} = \{q_{i1}, q_{i2}, \ldots, q_{ik}\}$ where i identifies the state and k is the number of skills for the state.

To calculate entropy, the probabilities of the selected skills are needed. If Q-Values are used directly, close probability values will be obtained, since the difference between Q-Values is relatively smaller than the values themselves. Nevertheless, instead of directly using Q-Values, Gibss-Boltzmann distribution can be used to obtain probabilities (Sutton and Barto 2018). Gibbs-Boltzmann distribution enables to obtain different probability distributions by adjusting the parameter τ. The probabilities of skills for a state are denoted as $\mathcal{P}_{s_i} = \{p_{i1}, p_{i2}, \ldots, p_{iK}\}$, and p_{ik} is given as in Eq. (6).

$$p_{ik} = \frac{e^{(q_{ik}/\tau)}}{\sum_{j=1}^{K} e^{(q_{ij}/\tau)}} \tag{6}$$

When events occur independently of each other, the entropy in Eq. (5) is employed.

In our case, the agent interacts with the environment at discrete time steps $t = 0, 1, 2, \ldots$ and chooses a skill $o \in O$ at its current state $s \in S$. Based on the chosen skill, environment responds with a reward r and the next state information s'. Since the states are dependent only on the immediately preceding states, so do the skills depend only on the immediately preceding skills, the *conditional entropy* in Eq. (7) is more convenient. Let us assume the agent is at state s_i and can select skills from the set $\mathcal{O}_{s_i} = \{o_{i1}, o_{i2}, \ldots, o_{il}\}$. The agent has l different choices for taking a skill, and it may end up with l different destination states. So there are l skill sets for selecting the next skill based on destination states. Let us denote all skills originating from one of these l destination states as $\mathcal{O}'_{s_j} = \{o'_{j1}, o'_{j2}, \ldots, o'_{jk_j}\}$ where k_j is the number of possible skills of the state s_j. Hence, the entropy of state s_i is attained as in Eq. (7).

$$
\begin{aligned}
H(s_i) &\equiv H(\mathcal{O}'_{s_j} \mid \mathcal{O}_{s_i}) \\
&\equiv \sum_{o_i \in \mathcal{O}_{s_i}} p(o_i) H(\mathcal{O}'_{s_j} \mid \mathcal{O}_{s_i} = o_i) \\
&= -\sum_{o_i \in \mathcal{O}_{s_i}} p(o_i) \sum_{o'_j \in \mathcal{O}'_{s_j}} p(o'_j \mid o_i) \log(p(o'_j \mid o_i)) \\
&= -\sum_{o_i \in \mathcal{O}_{s_i}} \sum_{o'_j \in \mathcal{O}'_{s_j}} p(o_i) p(o'_j \mid o_i) \log(p(o'_j \mid o_i)) \\
&= -\sum_{o_i \in \mathcal{O}_{s_i}} \sum_{o'_j \in \mathcal{O}'_{s_j}} p(o_i, o'_j) \log\left(\frac{p(o_i, o'_j)}{p(o_i)}\right)
\end{aligned}
\tag{7}
$$

The conditional entropy of each state in the environment is computed and updated per episode. Skill pairs with a considerably higher conditional probability outshine others, and hence, cause the entropy to decrease. The outstanding skill pairs are found if the entropy is reduced below a prespecified *entropy threshold*. The skill pair successively chosen with the highest frequency (hence, contributing most to lowering the entropy of the selected state) gets coupled.

Skill coupling does not only couple two skills, but is capable of coupling a multiple number of successive skills into a nested complex at subsequent hierarchies. Learning takes place at several hierarchies instead of only two. This significantly reduces the number of decisions; sometimes down to a single complex composed of a multiple number of skills. Thus, skill coupling method does not only solve the problem of the oversegmentation, but it also improves HRL in terms of the number of decisions. A complex o_C obtained by skill coupling is denoted using a triplet just like a skill and coupling two skills o_1 and o_2 into $o_C = \langle \mathcal{I}_C, \beta_C, \pi_C \rangle$ is shown in Eq. (8), where $\cup$ symbolizes skill coupling.

$$
\begin{aligned}
o_C = o_1 \cup o_2 &:= \langle \mathcal{I}_1, \beta_1, \pi_1 \rangle \cup \langle \mathcal{I}_2, \beta_2, \pi_2 \rangle \\
&:= \langle \mathcal{I}_1 \cup \mathcal{I}_2, \beta_1 \cup \beta_2, \pi_1 \cup \pi_2 \rangle
\end{aligned}
\tag{8}
$$

In the following, we discuss the details of the union operations within the resulting complex's triplet in Eq. (8) obtained by skill coupling. The initiation set $\mathcal{I}_C$ of the complex o_C is obtained during skill coupling from the initiation sets $\mathcal{I}_1$ and $\mathcal{I}_2$ as given in Eq. (9).

$$
\mathcal{I}_C = \mathcal{I}_1 \cup \mathcal{I}_2 := \begin{cases} \mathcal{I}_1, & \text{if } \mathcal{I}_1 \backslash \mathcal{I}_2 = \emptyset \\ \mathcal{I}_1 \backslash \mathcal{I}_2, & \textit{otherwise} \end{cases}
\tag{9}
$$

Coupling both skills' initiation sets as in Eq. (9) rather than a simple set union is to discard complexes leading to suboptimal policies. As a simple example, Fig. 6 depicts a two-room environment where each room is suboptimally segmented into two communities by the DCD algorithm and two skills, o_1 and o_2, are defined in $C_1 \cup C_2$ and $C_2 \cup C_3$, respectively. To endure the oversegmentation the two skills o_1 and o_2 with initiation sets $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively, are coupled into o_C. The initiation set $\mathcal{I}_C$ of o_C is formed as that of the first skill o_1 with possibly common states with that of o_2 excluded, as given in Eq. (9), since otherwise, a possible start of the complex o_C at one of these excluded states would lead to an indirect path to the subgoal G_2 of o_2, through the subgoal G_1 of o_1 and not directly to G_2.

Considering the termination condition, we use binary β_C, that is, $\beta_C : \mathcal{S} \to \{0, 1\}$. Therefore, we define the coupling of two termination conditions as shown in Eq. (10), which simply applies statewise logical AND operation to determine the new β_C. Also, β_C is set to 0 and 1 for the subgoals of o_1 and o_2, respectively.

$$
\beta_C = \beta_1 \cup \beta_2 := \beta_1 \wedge \beta_2
\tag{10}
$$

Note that, two skills are coupled, the initiation set of the resulting complex o_C becomes a subset of states that the complex does not terminate at, that is, $\mathcal{I}_C \subset \{s : \beta_C(s) < 1\}$, where $\mathcal{I}_C$ is the initiation set of the complex. This is slightly different from the $\{s : \beta(s) < 1\} \subseteq \mathcal{I}$ condition stated in Sutton et al. (1999), however it is still consistent with the skill definition and provides more flexibility, since this way it is possible to represent two subsets of $\mathcal{S}$ in one skill.

Finally, two subpolicies are coupled as,

$$
\pi_C(s, o) := \begin{cases} \pi_1(s, o), & \text{if } o_1 \text{is being executed} \\ \pi_2(s, o), & \text{otherwise} \end{cases}
\tag{11}
$$

When the skill o_C is selected, the agent recursively runs the policies that π_C is composed of.

The Q-Value of the complex o_C is initialized with the Q-Value obtained as if the two skills, o_1 and o_2, are taken. Therefore, the complex o_C is neither boosted nor starts with any disadvantages. Then, the Q-Value of this new complex is calculated for each state in the initiation set and the complex is added to each of these states.

The skill definition provided in Sutton et al. (1999) also covers primitive actions, namely each primitive is a single-step skill defined on the state that it is available. The initiation set of such a skill consists of only the state that the primitive action is starting from. Similarly, β is 1 everywhere except $\mathcal{I}$. The subpolicy is then a mapping $\mathcal{S} \times \mathcal{A} \to [0, 1]$ where the probability of the relevant primitive is 1, and others are 0. Thus, the coupling operations we defined above are

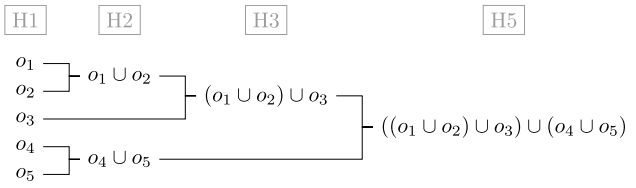


Fig. 7 Skill coupling operation creates a binary tree structure. Single-step actions are at H0 and are not shown here. Skills at H1 are constructed from the community structure obtained from DCD. In this example, when $((o_1 \cup o_2) \cup o_3) \cup (o_4 \cup o_5)$ is selected at a state, a depth-first traversal is applied to recursively run previously coupled components, which leads to execution of o_1, o_2, o_3, o_4 and o_5 respectively. Note that, this is a single-decision process, which leads the agent to the subgoal of the superskill $((o_1 \cup o_2) \cup o_3) \cup (o_4 \cup o_5)$ (also o_5 in this case)

also applicable to primitive actions, resulting in a bottom-up approach to iteratively combine and build more complex skills (i.e., complexes). It should be noted that, DCD is an approach that significantly speeds up the construction of skills at hierarchy #1 as shown in Fig. 7. Instead of coupling single-step skills one by one at the beginning, we directly construct more comprehensive skills thanks to the communities detected autonomously during learning. After that, obtained skills are started to be coupled.

Hierarchies are structured such that they manifest the number of skills that build a complex. A complex composed of, say, *i* skills is coupled at the *i*th hierarchy. This idea of hierarchy ordering tends to reflect the elaborateness of a complex. That is; a complex with $i+1$ skills is more elaborate than one with i skills. An alternative hierarchical structure may be to set up a hierarchy per skill coupling, where any coupling defines a different hierarchy. Such hierarchy structure would be time based in the sense that the hierarchy order pinpoints the ordering of time instances, where each coupling associates with a specific time instance.

In Fig. 6 a two-room environment suffering from oversegmentation is illustrated. Three skills o_1, o_2 and o_3 are learned by the HRL agent at hierarchy #1 (H1) in communities C_1, C_2, C_3, respectively. To cover up the negative effects of oversegmentation, first o_1 and o_2, are coupled at H2. Then the complex $o_1 \cup o_2$ may be coupled with o_3 at H3 to form a complex of three skills.

Figure 7 describes how skills are coupled and executed, and the algorithm is given in Algorithm 2.

Algorithm 2 : Skill Coupling

Input: currentState

if $lastEntropy$ of the state $\leq entropyTreshold$ **then**
  for skill1, skill2 in skillTuple **do**
    if probability(skill1, skill2) == max probability in skillTuple **then**
      $s \leftarrow$ currentState, $s_1 \leftarrow$ destStateOfSkill1, $s_2 \leftarrow$ destStateOfSkill2
      $k_1 \leftarrow$ numberOfSteps(s, s_1), $k_2 \leftarrow$ numberOfSteps(s_1, s_2), $k_{coupled} \leftarrow k_1 + k_2$
      if $newSkill$ is not in s **then**
        $Q_{temp} = Q(s_1, skill2) + \varphi[r_{\text{cumulative}} + \gamma^{k_2} max_{a'} Q(s_2, a') - Q(s_1, skill2)]$
        $Q(s, newSkill) = Q(s, skill1) + \varphi[r_{\text{cumulative}} + \gamma^{k_1} max(Q_{temp}, max_{a'} Q(s_1, a')) - Q(s, skill1)]$
        newInitiationSet = $\mathcal{I}_1$ if $\mathcal{I}_1 \setminus \mathcal{I}_2 = \varnothing$ else $\mathcal{I}_1 \setminus \mathcal{I}_2$
        newPolicy = couplePolicies(skill1, skill2)
        addSkill(s, newInitiationSet, newPolicy, Q(s, newSkill))
        for state in newInitiationSet **do**
          calculate Q(state, newSkill) with corresponding k values
          addSkill(state, newInitiationSet, newPolicy, Q(state, newSkill))
        end for
      end if
    end if
  end for
end if

Table 1 Comparison of the proposed methods and the other existing graph-based methods. DCD has $O(|\Delta e| + |e|^*)$ complexity in the best case, and $O(|\Delta e|.\frac{|e|}{|n|} + |e|^*)$ complexity in the worst case, where Δe is the set of edges being changed and $|e|^* \ll |e|$. For the overall time complexity of ASKAC, the time complexities of subgoal detection, skill coupling, and individual RL tasks should be considered

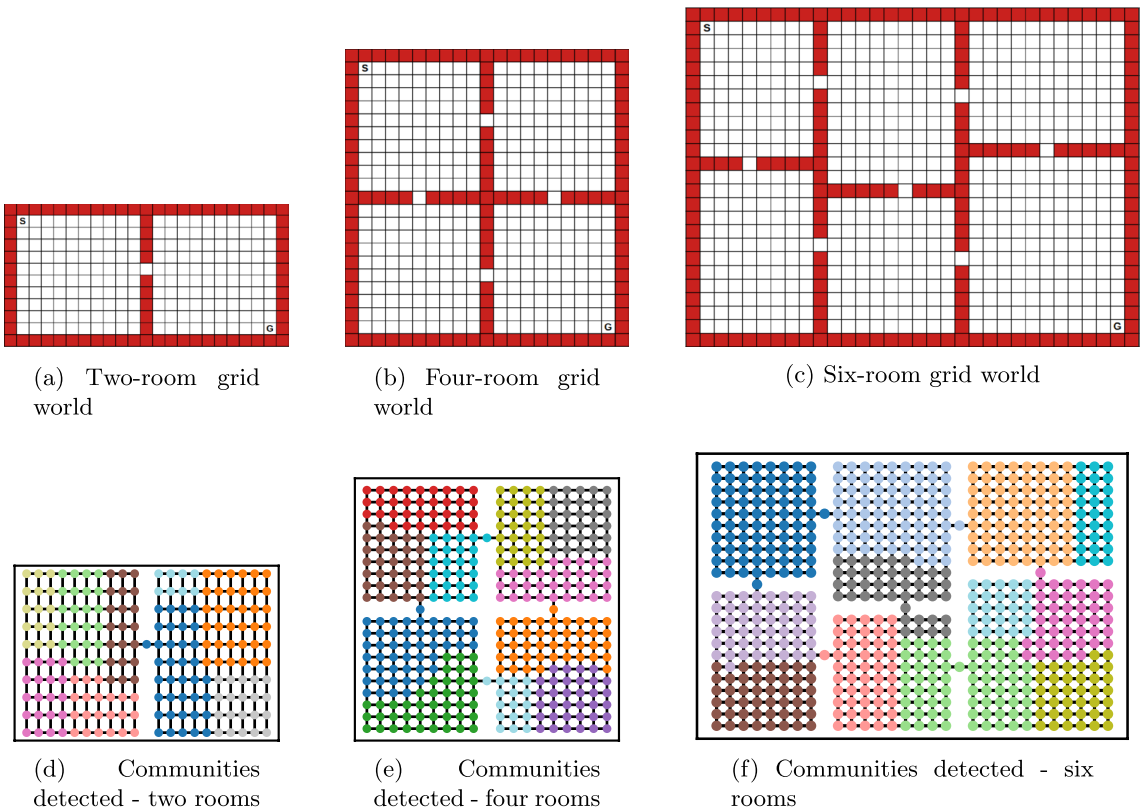
Method	Parameters	Time complexity for subgoal detection	Time complexity for skill coupling
BC	–	$O(n^3)$	–
SCC	t_t	$O(e+n)$	–
ASKA	ρ	DCD + $O(e)$	–
ASKAC	ρ, entropy threshold	DCD + $O(e)$	O(# steps $*$ (# skills in state)^2)

5 Experiments and results

We categorize the experiments in two groups regarding the claims we make about our method's capabilities: (1) experiments for displaying the performance achieved by skill coupling (2) experiments for illustrating the robustness gained by skill coupling. We use two benchmark environments: (1) the two-, four- (with two variations) and six-room grid worlds, and (2) taxi driver environment. In particular, we use all environments to evaluate the performance of ASKA and ASKAC compared with state-of-the-art in Section 5.1. To exhibit the performance of ASKAC on robustness, we use the six-room environment due to its larger state space. The effects of skill coupling on solution or policy complexity and robustness are discussed, respectively, in Sections 5.2 and 5.3.

We compare the proposed algorithms with Q-Learning (Watkins and Dayan 1992), BC (Simsek and Barreto 2008), SCC (Kazemitabar et al. 2018), and MMSC (Setyawan et al. 2022). Since the main goal of point option methods (Machado et al. 2017; Jinnai et al. 2019; Zhu et al. 2022) is not making the agent reach the goal faster (in terms of steps or decisions), we argue that these studies are not directly comparable to ours. For SCC and BC, experiments are run with the same RL parameters as those given in Section 5.1.1. In Section 5.1.2, we compare ASKA and ASKAC with MMSC on a special environment introduced by Setyawan et al. (2022).

The comparison of the proposed method and the other existing graph-based subgoal detection methods in terms of the number of parameters and time complexity are given in Table 1. The common parameter for all graph-based



(a) Two-room grid world

(b) Four-room grid world

(c) Six-room grid world

(d) Communities detected - two rooms

(e) Communities detected - four rooms

(f) Communities detected - six rooms

Fig. 8 Suboptimal communities of grid worlds with varying numbers of rooms (each color refers to a different community)

methods, which is not given in Table 1, is the number of episodes required by the agent to construct the partial model for detecting subgoals. Since DCD is used in ASKA and ASKAC, this parameter is considered as the minimum number of episodes required to start constructing the partial model. Note that the partial model grows more similar to the actual environment as learning proceeds, so the related changes to the skills are performed in the proposed method.

While BC does not have any parameters, SCC and ASKA have one parameter and ASKAC has two parameters. The effect of the resolution parameter, ρ, the parameter of DCD, is eliminated in ASKAC by coupling skills. ASKAC has one more parameter than ASKAC, the entropy threshold.

Another criterion, the time complexity, for all methods are also given in Table 1. Let n be the number of nodes and e the number of edges in the partial graph. Here, DCD used for ASKA and ASKAC has $O(|\Delta e| + |e|^*)$ complexity in the best case, and $O(|\Delta e|.\frac{|e|}{|n|} + |e|^*)$ complexity in the worst case, where Δe is the set of edges currently modified and $|e|^* \ll |e|$, to detect communities dynamically (Zhuang et al. 2019). After obtaining communities, subgoals are detected in $O(e)$ as given in Algorithm 1. In ASKAC, skill coupling which is an extension to ASKA has $O(\#\text{steps} * (\#\text{ skills in state})^2)$ time complexity, since the entropy of a state is calculated as described in Section 4.2.

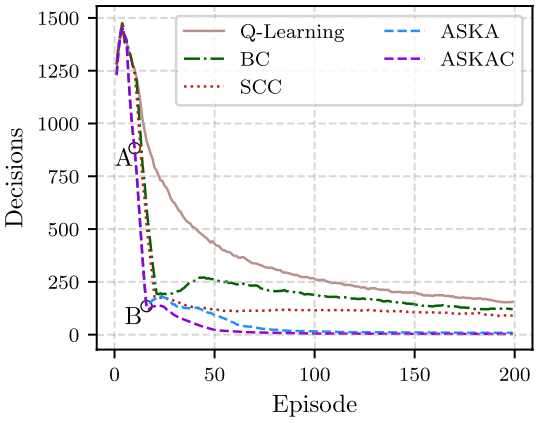
5.1 Benchmark environments

In this section, ASKA, ASKAC and the other methods are compared in two standard benchmark environments: two-dimensional grid worlds and taxi driver.

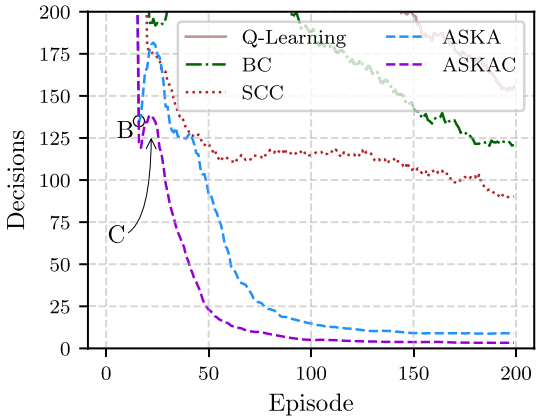
5.1.1 Two-/Four-/Six-room grid worlds

For the two-dimensional grid world environments, there are four primitive actions: going north, west, south, and east, in each state. If the move of the agent is blocked by a wall, it stays at the current state. Each episode where the agent starts at the same initial state terminates when the agent reaches the goal state. The immediate reward for taking each action is -1, $+1000$ for reaching the goal state. We use ϵ-greedy policy with a decaying ϵ of an initial value of $\epsilon = 0.2$, a discount factor $\gamma = 0.9$, and a learning rate $\alpha = 0.05$. For ASKAC, the temperature parameter in Gibbs-Bolztmann $\tau = 0.3$, entropy threshold 0.8, and skill coupling is available after episode 10. The experiments are run in two-room (Fig. 8a), four-room (Fig. 8b) and six-room (Fig. 8c) grid worlds. Each experiment is run for 3200 episodes and each one averaged over 40 runs.

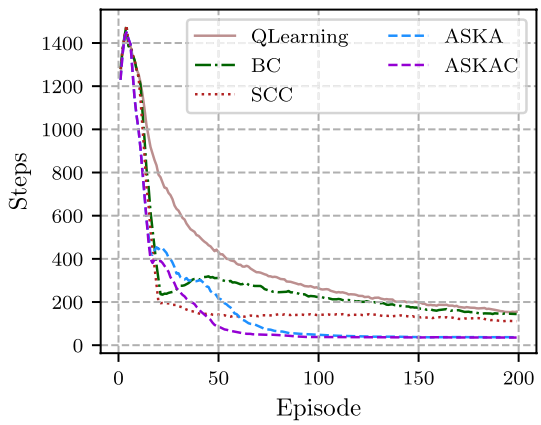
5.1.1.1 Results and discussion For all grid worlds, communities detected by DCD module are suboptimal (the



(a) Number of decisions to goal



(b) Number of decisions to goal (zoomed from (a) so $x : [0, 200]$ and $y : [0, 200]$)



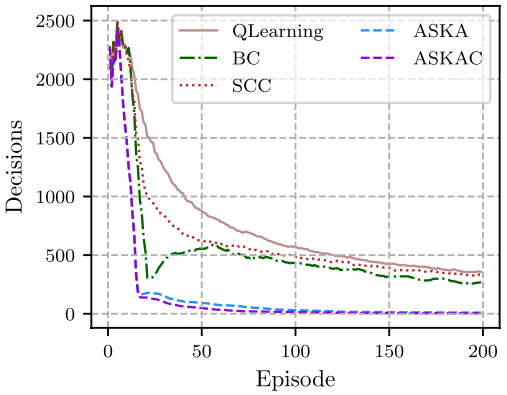
(c) Number of steps to goal

Fig. 9 Comparison of the proposed methods with Q-Learning (Watkins and Dayan 1992), BC (Simsek and Barreto 2008) and SCC (Kazemitabar et al. 2018) in two-room environment. Entropy threshold is 0.8 for ASKAC. The average of 40 runs with different seeds for Pseudorandom Number Generator (PRNG) is shown with a sliding window of size 10. Point A in (**a**) indicates the spot where skill coupling is started. Point B in (**a**) and (**b**) is the spot where the effect of skill coupling becomes visible. The lag between A and B is approximately 10 episodes and caused by the time passed before agent's revisit to the state that the coupled skill is available. The peak indicated as C in (**b**) is the first local peak of ASKAC after skill coupling

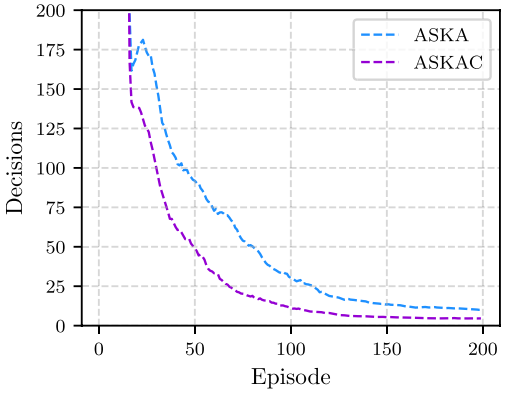
environments are split into a higher number of communities than expected; i.e., more than the number of rooms). The oversegmentation problem discussed in Section 4.2 is illustrated for all three grid worlds in Figs. 8d–f corresponding in respective order to Figs. 8a–c. For each of the two-, four- and six-room grid worlds, the curves for the number of decisions (not considering the number of primitive actions chosen inside a skill) and steps to goal are illustrated for the first 200 episodes in Figs. 9, 10 and 11, respectively, to better see the speed of decay of the methods. Further in Table 2, both the average number of decisions and steps for both 200 and 3200 episodes are shown for all five methods along with the corresponding percent reductions in the last column. Both ASKA and ASKAC outperform other methods (Q-Learning, BC and SCC) where ASKAC achieves a minimum number of decisions at all three environments for both 200 and 3200 episode runs and hence performs better than ASKA while ASKA yields the minimum number of steps (with the only exception at the two-room environment at 3200 episodes where BC has a reduction in the number of steps of 6.53% compared with that of ASKA). With skill coupling, ASKAC is able to significantly reduce its number of decisions. The goal is attained by 3.3, 4.1 and 35.4 decisions in two-, four- and six-room environments, respectively. ASKAC's remaining behind ASKA in the number of steps is highly likely to be attributed to ASKAC's learning suboptimal complexes. There may be multiple complexes that lead to the goal. ASKAC may have learned one or more of them based on the progress of learning. Since some of them may be experienced sooner than others, ASKAC uses these complexes obtained sooner to reach the goal. If sufficient time is provided, all complexes may be learned. The percent reductions for decisions point to the relative error between ASKAC and ASKA to emphasize the influence of skill coupling while those for steps denote the relative error between ASKA and state-of-the-art's best. The reduction in the number of decisions between ASKAC and state-of-the-art's best is more dramatic than the figures for steps.

As seen in Figs. 9a, 10a and 11a and Table 2, with the growing state space more communities, hence, more skills and a higher number of more elaborate complexes show up. While skill coupling can cope with the growing state space by building policies of a smaller number of more complex decisions through an arbitrary number of hierarchies, the number of decisions for other methods grows faster.

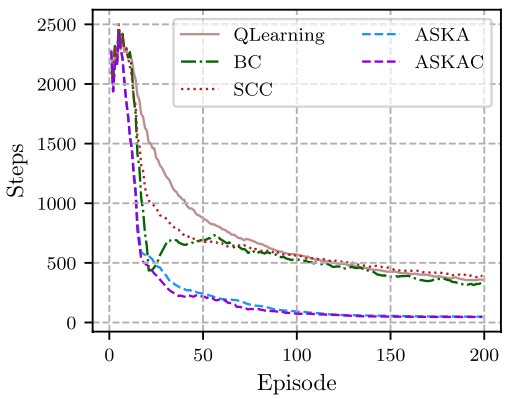
There are peaks (up to episode 50) in Figs. 9b and 10b in curves for the number of decisions. Skill coupling first starts at episode 10. For instance, the point in Fig. 9b where the two learning curves of ASKA and ASKAC branch out and that of ASKA forms the local peak (point B in Fig. 9b) is a little after episode 10. The effect of using coupled skills (i.e., complexes) manifests itself on ASKAC's curve as keeping the steepness of the decay in the number of decisions



(a) Number of decisions to goal



(b) Number of decisions to goal (zoomed from (a) so $x : [0, 200]$ and $y : [0, 200]$). Only ASKA and ASKAC are within the selected interval of decisions (y-axis) in the zoomed figure. That is why other methods' graphs are not in the figure. They are already noticeably above those of ASKA and ASKAC in (a)



(c) Number of steps goal

Fig. 10 Comparison of the proposed methods with Q-Learning (Watkins and Dayan 1992), BC (Simsek and Barreto 2008) and SCC (Kazemitabar et al. 2018) in the four-room environment. Entropy threshold is 0.8 for ASKAC. The average of 40 runs with different seeds for PRNG is shown with a sliding window of size 10

whereas ASKA is affected by relatively more frequent exploring selections and changes in the community structure in the early stages of learning. ASKAC may also be affected by these and form local peaks (point C in Fig. 9b) which we also observe at the notch on ASKAC's curve below right after the branch-out (Fig. 10b). However, these local peaks are typically not as high as those of ASKA as seen in Fig. 9b. Hence, as a result of skill coupling ASKAC exhibits a steeper falling learning curve.

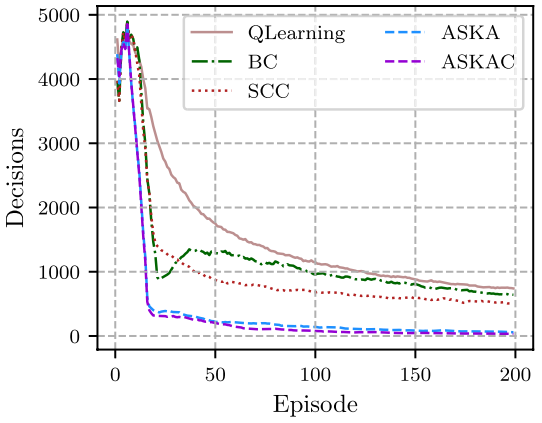
In Figs. 9c, 10c and Table 2, the difference among the average number of steps for ASKA and ASKAC is less than 10% for two- and four-room environments at the end of 200 episodes. Due to likely premature learning, the same rule does not apply to the number of steps for six-room environment for 200 episodes; but at the end of 3200 episodes that of ASKAC is 2.88% better than that of ASKA while they are both still far better than other methods (66.9%, 84.61% and 74.45% reduction for two-, four- and six-room environments, respectively).

5.1.2 Four-room grid world with terminating collision: a special case

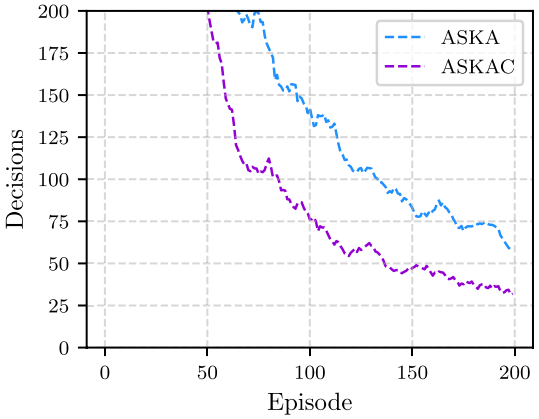
In this section, proposed methods ASKA and ASKAC are run on a slightly different four-room environment to compare them with a recent state-of-the-art method MMSC (Setyawan et al. 2022). The main difference of this special case from the four-room grid world discussed in Section 5.1.1 is that the episode ends if the agent hits an obstacle (including walls). Therefore, an additional metric, *success rate* (ratio of episodes in which the agent reached the goal), is also considered in addition to the number of decisions.

The experimental setup in (Setyawan et al. 2022) includes six variations of the four-room grid world with different amounts of obstacles, each with three different starting states. We run experiments on three of these variations, namely four-room without any additional obstacles (Fig. 12a), four-room with the maximum amount of additional obstacles (Fig. 12b) and the variation without any obstacles or walls (Fig. 12c) to cover both extrema. We also consider Q-Learning (Watkins and Dayan 1992) and Options (Sutton et al. 1999) methods as non-hierarchical and hierarchical RL baselines.

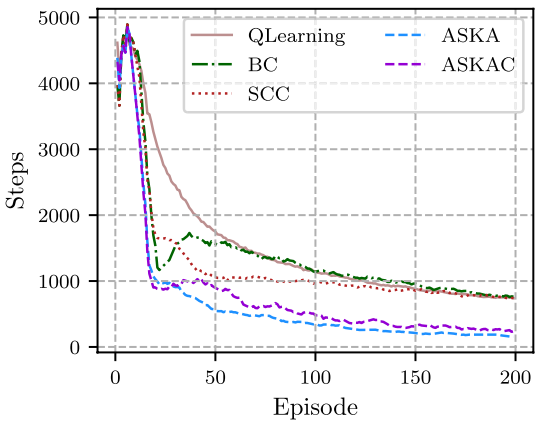
For all methods, the immediate reward for moving to a new state is -0.1, hitting an obstacle is -1 and reaching the goal state is $+1$. Epsilon $\epsilon = 0.999^{\#\text{episode}}$, discount factor $\gamma = 0.9$, and a learning rate $\alpha = 0.1$ are used for all methods. For ASKAC, the temperature parameter in Gibbs-Boltzmann



(a) Number of decisions to goal



(b) Number of decisions to goal (zoomed from (a) so $x : [0, 200]$ and $y : [0, 200]$)



(c) Number of steps to goal

Fig. 11 Comparison of the proposed methods with Q-Learning (Watkins and Dayan 1992), BC (Simsek and Barreto 2008) and SCC (Kazemitabar et al. 2018) in six-room environment. Entropy threshold is 0.8 for ASKAC. The average of 40 runs with different seeds for PRNG is shown with a sliding window of size 10

Table 2 The number of decisions and steps at episode 200 and 3200. Each number is the average of 40 runs

Metric	Env.	Episode	Algorithm					Reduction (%)
			Q-Learning	BC	SCC	ASKA	ASKAC	
Decisions	Two-room	200	172.85	97.55	83.18	8.53	**3.3**	61.31
		3200	36.25	25.4	21.9	8.45	**4.43**	47.57
	Four-room	200	357.98	245.73	305.68	9.03	**4.1**	54.60
		3200	47.45	30.35	32.63	7.88	**3.75**	52.41
	Six-room	200	730.7	636.43	500.25	72.8	**35.4**	51.37
		3200	62.48	42.63	40.3	16.1	**8.35**	48.14
Steps	Two-room	200	172.85	115.65	104.93	**34.73**	37.73	66.90
		3200	36.25	**32.9**	33.55	35.05	35.58	−6.53
	Four-room	200	357.98	305.33	361.48	**47.0**	51.75	84.61
		3200	47.45	46.55	44.55	**43.48**	43.73	2.40
	Six-room	200	730.7	754.05	764.43	**186.68**	266.825	74.45
		3200	62.48	65.85	62.98	61.23	**60.68**	2.88

For each row, the best result is indicated with **bold** numbers. For decisions, percent reduction illustrates the decrease from the second-best result. For steps, on the other hand, it manifests the decrease of the best of our methods (ASKA or ASKAC) from the state-of-the-art's best

$\tau = 0.3$, entropy threshold 0.8, and skill coupling is available after episode 10. Similar to (Setyawan et al. 2022), the number of decisions and success rate are calculated for ASKA and ASKAC from episodes between 9000 and 10000, which are then averaged over 40 runs.

5.1.2.1 Results and discussion The average number of decisions and the average success rate between episodes 9000 and 10000 are given in Table 3. Setyawan et al. (2022) define the number of steps taken by the agent as the number of decisions made, which is slightly different than how we use the term step in this study. However, it matches the number of decisions metric we use. Hence, we report it as the number of decisions in Table 3.

Similar to previous experiments, less number of decisions that agent has to make to reach the goal indicates more inclusive and useful skills. Considering this special case of the four-room environment, less number of decisions could also be due to the episodes in which the agent hits an obstacle. Hence, the performance of the methods should be evaluated considering both the number of decisions and the success rate.

As seen in Table 3, ASKA manages to achieve smaller average number of decisions and higher success rates than MMSC, Options, and Q-Learning in all environments. This indicates that the options constructed by ASKA are more useful for the agent to reach the goal. Moreover, the effect of skill coupling mechanism we propose is visible in the number of decisions made by ASKAC, which manages to further improve ASKA in terms of number of decisions with almost equal success rates.

The significantly high success rates achieved by ASKA and ASKAC (even when extra obstacles are added to the environment) could be linked to how we construct subpolicies by considering options as separate small RL tasks. Naturally, with subpolicies constructed this way, the agent moves from one subgoal to another and learn to avoid hitting obstacles.

5.1.3 Taxi driver

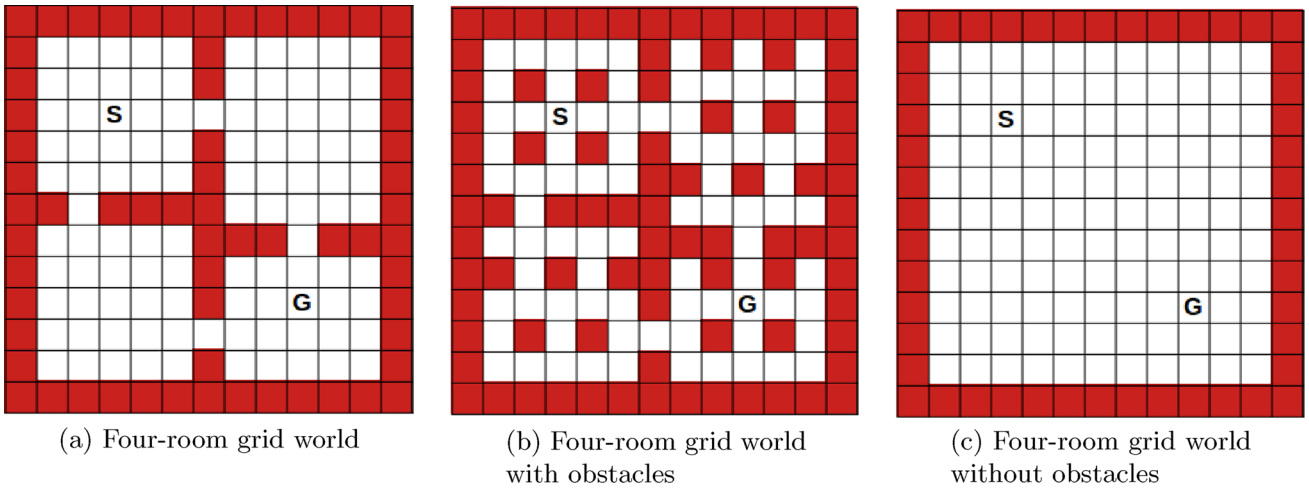
The taxi driver task is to pick up and deliver the passenger to the destination in a 5×5 grid environment given in Fig. 13a. The possible locations of the passenger and the destination are one of the 4 states marked by R, G, B and Y. In each episode, passenger and destination locations are chosen from the marked states, and the initial location of the taxi is chosen among the 25 states randomly. The taxi has six primitive actions: north, west, south, east, pick up and put down at each state. If the movement is blocked by a wall, the taxi stays at its current state. If the taxi is at the same location as the passenger, the pick up action places the passenger in the taxi; if the passenger is in the taxi and the taxi is at the destination, the put down action delivers the passenger; otherwise these actions have no effect. The agent receives a reward of -1 for each action, $+20$ for passenger delivery, and -10 for an unsuccessful pick up or put down action.

This domain has 25 grid locations, 5 passenger locations (including in-taxi), 4 destinations, and hence 500 ($25 \times 5 \times 4$) states in total (Şimşek and Barto 2004b). The graph representation of the taxi driver environment is given in Fig. 13b. We consider each 25-state passenger and destination combination as a subgraph, and the connections between these subgraphs are established through directed edges when the valid actions, i.e., pick up and put down, are

Table 3 The average number of decisions (# dec.) and average success rate (SR) in episodes from 9000 to 10000

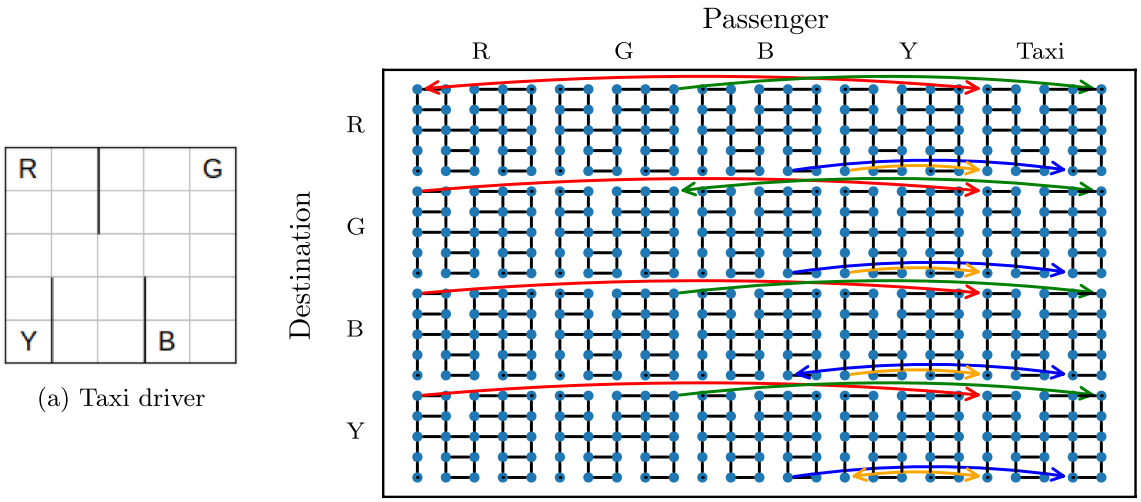
Env.	Algorithm									
	Q-Learning		Options		MMSC		ASKA		ASKAC	
	# of dec.	SR (%)	# of dec.	SR (%)	# of dec.	SR (%)	# of dec.	SR (%)	# of dec.	SR (%)
Fig. 12a	12.67	81.23	11.25	67.97	7.44	89.99	5.96	99.99	**2.75**	**99.99**
Fig. 12b	12.75	98.10	12.54	45.49	12.75	98.10	4.63	99.99	**3.83**	**99.99**
Fig. 12c	12.69	92.60	6.49	94.92	5.78	91.22	5.43	**100**	**3.64**	99.99

For ASKA and ASKAC, each number is the average of 40 runs. In all environments, the agent starts the episode from the state at (2, 2). In each row, the best results of both metrics are indicated with **bold** numbers



(a) Four-room grid world (b) Four-room grid world with obstacles (c) Four-room grid world without obstacles

Fig. 12 Four-room grid world environment with different obstacle levels. **a** to **c** correspond to the environments given in Figs. 4a, 4d and 4f in Setyawan et al. (2022) respectively



(a) Taxi driver (b) Taxi driver graph

Fig. 13 Taxi driver environment and its graph representation. The transitions between subgraphs are represented by colored edges. For each subgraph, the edges between the passenger location and the taxi location are shown in red, green, blue, and orange for the passenger locations R, G, B and Y, respectively. The self loops that originate from unsuccessful pick up and put down actions are not shown here for the simplicity of the graph

executed. For unsuccessful pick up and put down actions, a self-loop will occur at the related node.

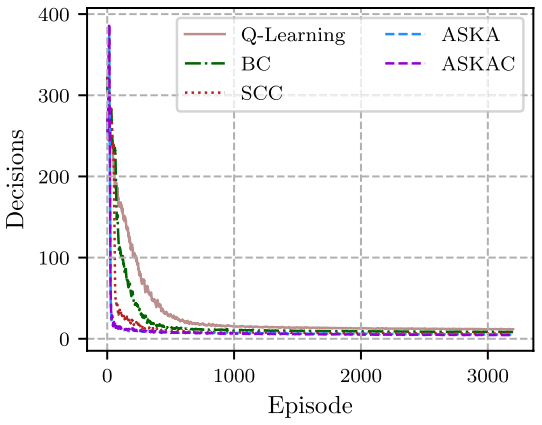
In our experiments, the episode ends when the taxi delivers the passenger to the destination state. We use ϵ-greedy policy with a decaying ϵ of an initial value of $\epsilon = 0.4$, a discount factor $\gamma = 0.95$, a learning rate $\alpha = 0.2$. For ASKAC, the temperature parameter in Gibbs-Bolztmann $\tau = 0.3$, entropy threshold 0.8, skill coupling is available after episode 10. Experiments are run for 3200 episodes and each averaged over 40 runs.

5.1.3.1 Results and discussion For the taxi driver environment, the curves for the number of decisions (not considering the number of primitive actions chosen inside a skill) and of steps to goal are illustrated for 3200 and 200 episodes, respectively, in Fig. 14. Further in Table 4, both the average number of decisions and steps for both 200 and 3200 episodes are shown for all five methods, along with the corresponding percent reductions at the last column. In Fig. 14a, b and in Table 4 both ASKA and ASKAC outperform other methods. The difference in the number of decisions between ASKA and ASKAC is not significant (3.87% from Table 4) as in the results of grid worlds, since there are less coupled skills in the taxi driver environment. However, ASKAC still achieves the minimum number of decisions.

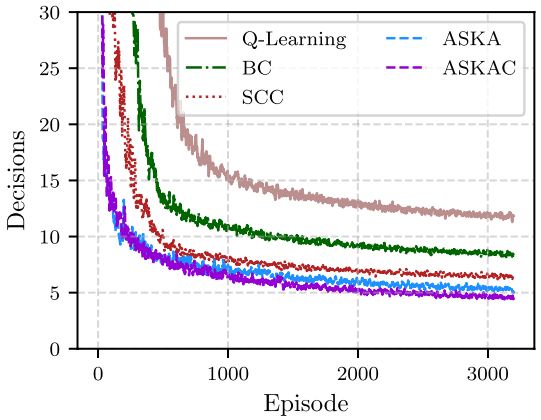
In terms of the number of steps, ASKA and ASKAC have almost the same performance as in Fig. 14c, but ASKA has slightly less steps than ASKAC in some episodes due to the incorrect communities detected. Incorrect communities lead to unexpected subgoals, and the subgoals bring about different skills. Hence, choosing such a skill may end up at a state that is a few steps away from the real subgoal. In case such skills are coupled, the number of decisions can be decreased, but the number of steps can be higher than optimal.

In taxi driver experiments, we observed that learning the subpolicies of skills as independent RL problems instead of using intra-option Q-learning may be a disadvantage. In ASKA and ASKAC, each subpolicy is learned as an RL problem when skills are selected for the first time. Then, whenever a skill is selected, ASKA and ASKAC agents go directly to the subgoal using the learned subpolicy, which causes a skill to be executed until its subgoal regardless of whether it is a good choice. Hence, both agents may move away from the goal, and need to take more steps to reach it. This is likely the reason for ASKA and ASKAC's rarely converging to a policy with a slightly higher number of steps than other methods.

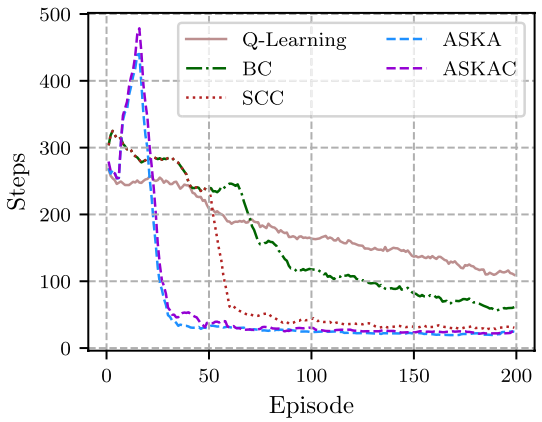
Although the taxi driver environment has less states than six-room grid world, the learning process takes longer. Since the graph of taxi driver environment consists of four disconnected subgraphs, in each episode, ASKAC is able to move



(a) Number of decisions to goal



(b) Number of decisions to goal (zoomed from (a))



(c) Number of steps to goal

Fig. 14 Comparison of the proposed methods with Q-Learning (Watkins and Dayan 1992), BC (Simsek and Barreto 2008) and SCC (Kazemitabar et al. 2018) in taxi driver environment. Entropy threshold is 0.8 for ASKAC. The average of 40 runs are shown with a sliding window of size 10

Table 4 Number of steps and decisions both at episodes 200 and 3200

Env.	Metric	Episode	Algorithm					Reduction (%)
			Q-Learning	BC	SCC	ASKA	ASKAC	
Taxi driver	Decisions	200	129.9	56.88	29.43	**10.43**	10.85	64.56
		3200	12.98	8.93	6.65	5.75	**4.5**	32.33
	Steps	200	129.9	60.33	35.6	**20.4**	21.73	42.7
		3200	12.98	12.53	**12.03**	12.55	13.3	−4.32

Each number is the average of 40 runs. For each row, the best result is indicated with **bold** numbers. For both metrics, percent reduction manifests the decrease of the best of our methods (ASKA or ASKAC) from the state-of-the-art's best

and learn only at a limited number of states. In this respect, intra-option Q-learning may be preferable, because it provides more updates for state-action pairs once a skill is executed.

5.2 Effects of skill coupling on solution complexity

In this set of experiments, we investigate the effect of entropy threshold parameter on skill coupling and the relation between skill coupling and the solution complexity. The entropy threshold parameter is the only parameter of skill coupling. Therefore, the experiments in this section are intended to establish the relation between the solution complexity and the entropy threshold. The policy or solution complexity is a combination of the elaborateness of the decisions or complexes (i.e., the number of skills composing a decision) and the number of decisions it takes an agent to the goal (i.e., the size of the policy).

5.2.1 Experiment setup

The experiments are conducted with two entropy threshold values $\{0.5, 0.8\}$ in the six-room environment in Fig. 8c. The learning parameters are as in Section 5.1.1. For the temperature parameter in Gibbs-Bolztmann, $\tau = 0.3$ is used, and skill coupling is effective after episode 10. Each experiment is run for 3200 episodes.

5.2.2 Results and discussion

The detected communities are identical for all entropy threshold values and the partitioning of the environment is shown in Fig. 15. Dendrograms showing the coupled skills for two threshold values are given in Fig. 16a and b. As the entropy threshold increases, the total number of skills coupled (the skill densities in Fig. 16a and b) and the number of skills involved in coupling (the number of colors in Fig. 16a and b expand. In Fig. 16a, the hierarchy level can go up at most to 3, and the coupled skills are not sufficient to take the agent to the goal state by a single complex. In Fig. 16b, there are much more coupled skills and the number of hierarchies

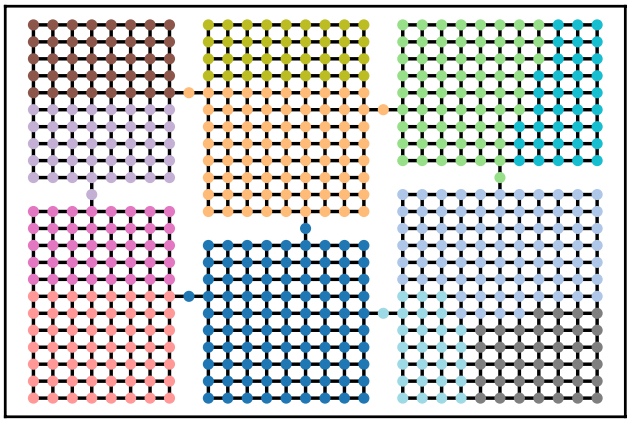


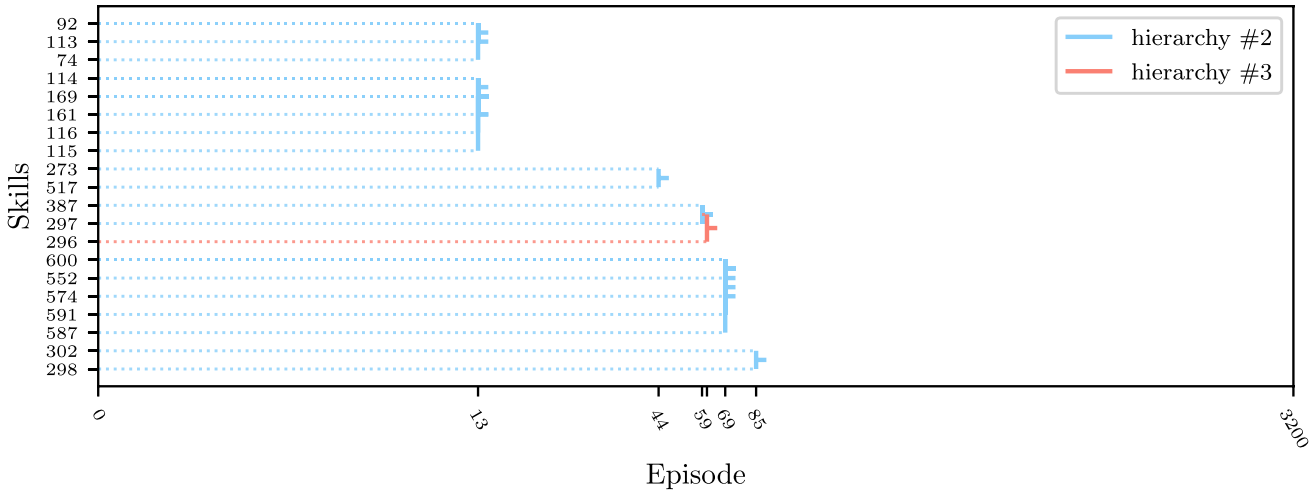
Fig. 15 Detected communities in six-room grid world

rises to as many as 10. The coupled skills formed in the hierarchy #8 and hierarchy #10 can take the agent to the goal state by a single complex. However, since the number of steps is higher in the coupled skill at hierarchy #10 (because it is coupled with a skill in the reverse direction to the goal state), the agent prefers the skill at the hierarchy #8. This selection of the agent indicates the fault tolerance capability of ASKAC as a result of skill coupling.

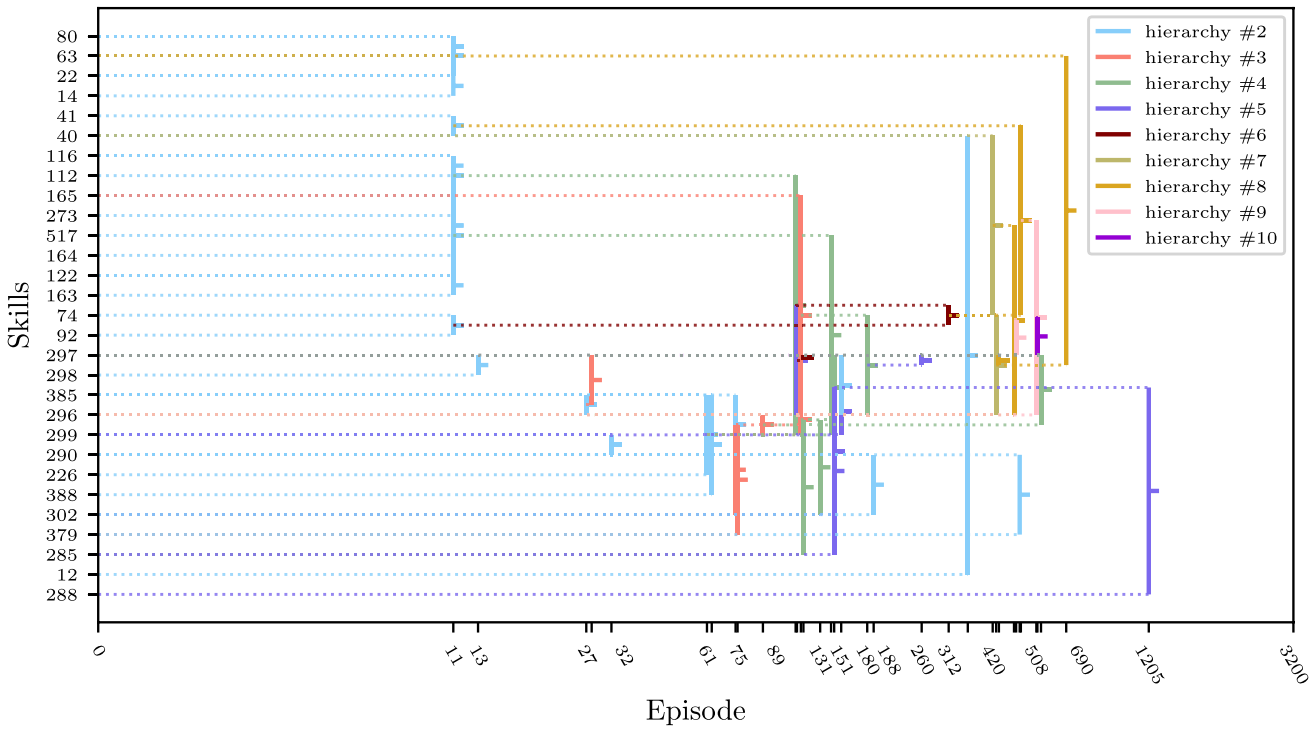
The empirical results obtained point out that with the growing entropy threshold the decision complexity (i.e., the number of hierarchies, hence, the number of skills coupled to form the complex) grows while the number of decisions leading to the goal reduces. The entropy threshold is a heavily application dependent parameter and depends on the number of skills defined per state; hence normalization may be a good idea.

5.3 Effects of skill coupling on robustness

In this set of experiments, we investigate the effect of the resolution parameter, that causes the partitioning of the environment into different communities, on the learning. With various values of this parameter, we compare ASKA and ASKAC in terms of their number of decisions and cumulative reward, and analyze the robustness.



(a) Dendrogram when entropy threshold is set to 0.5

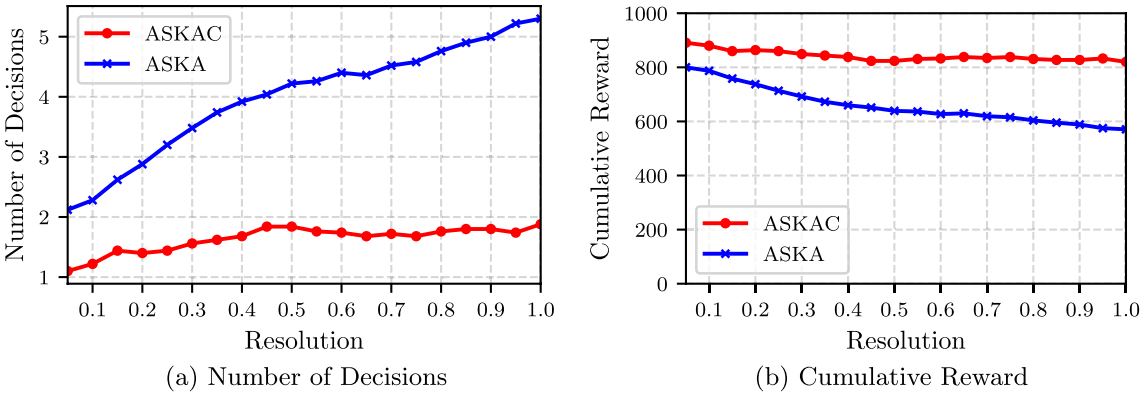


(b) Dendrogram when entropy threshold is set to 0.8

Fig. 16 Dendrograms of coupled skills for different entropy threshold values. Episode numbers are shown in logarithmic ($\log_2$) scale to improve visibility. As expected, higher entropy threshold values lead to more coupling operations, and hence higher number of hierarchies are formed. Note that, this representation is different from Fig. 7, as this also considers the coupling order (i.e., time axis). Therefore, hierarchies are not in ascending order as the episode number increases

5.3.1 Experiment setup

The experiments are conducted with various values of the resolution parameter ρ uniformly distributed and ranging within (0, 1] in the benchmark six-room environment in Fig. 8c. The learning parameters are as in the Section 5.1.1. Following values are used: $\tau = 0.3$ for the temperature parameter in Gibbs-Boltzmann, the entropy threshold 0.8, skill coupling is effective after episode 10. 50 experiments are conducted per ρ value. Each experiment is run for 90000 episodes.



(a) Number of Decisions (b) Cumulative Reward

Fig. 17 The effect of resolution parameter ρ to the average number of decisions (**a**) and average cumulative reward (**b**). In this experiment, for each ρ value from the set $\{0.05, 0.1, \ldots, 1\}$, the average of 50 runs with different seeds for PRNG are shown. Stability of the number of decisions and cumulative reward while ρ changes is an indication of the robustness capability of ASKAC

5.3.2 Results and discussion

Graphs for the number of decisions and cumulative reward are given in Fig. 17a and b respectively. An example of the detected communities for each of the resolution parameter values of 0.05, 0.5 and 1 is given in respective order in Fig. 5a–c. The size of the communities detected decreases as their number rises with the growing resolution. ASKA is affected by the partitioning resulting in an increase in the number of decisions in Fig. 17a and a decrease in the cumulative reward in Fig. 17b with the growing resolution, while in both cases the ASKAC remains stable.

ASKA starts ($\rho = 0.05$) and ends ($\rho = 1$) with an average number of decisions of 2.12 and 5.3, respectively, while ASKAC with 1.1 and 1.88. Hence, ASKAC has an average increase of 0.78 while ASKA 3.18 (i.e., ASKA grows about 4 times faster than ASKAC (a 75% improvement)). Similarly, ASKA's cumulative reward values for $\rho = 0.05$ and $\rho = 1$ in respective order are 800.01 and 570.79, while those of ASKAC's 890.91 and 820.01. So, ASKAC's cumulative reward has an average percent reduction bounded at 8.38% while that of ASKA grows up to 30.16% (i.e., ASKA's cumulative reward falls down about 3.6 times faster than that of ASKAC (a 72% improvement)). Both of these results justify for the higher robustness of ASKAC than that of ASKA.

6 Conclusion

With this study, we presented a novel method to deal with the oversegmentation issue that locality based community detection approaches are facing. By a conditional entropy based metric we recognize skills that are consecutively executed with a significantly high frequency and build arbitrarily elaborate complexes through a multiple number of hierarchies so that graph models with overly-segmented communities cause either minimal or no negative effects on attaining satisfactory policies. As a result, ASKAC turned out to (a) improve skill construction so that it is more robust to different resolution parameter values, and (b) significantly reduce the number of decisions (in some cases even to a single decision) needed to reach the goal by skill coupling that builds arbitrarily complex skills through multiple hierarchies.

Further research would probably be directed towards, (a) exploiting the graph model for planning, (b) including actions in skill coupling method, (c) dealing with the non-stationary environments (and hence community detection in non-stationary graphs), and (d) skill sharing and transfer for multi-agent HRL.

Author contributions All authors contributed equally.

Funding No funding was received for conducting this study.

Availability of data and materials The datasets generated during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Ethical standard Not applicable

Consent to participate Not applicable

Consent for publication Not applicable

Code availability The software used in this study is available from the corresponding author on reasonable request.

References

Aktunc R, Toroslu IH, Ozer M, et al (2015) A dynamic modularity based community detection algorithm for large-scale networks: Dslm. In: Proceedings of the 2015 IEEE/ACM international conference on advances in social networks analysis and mining 2015, pp 1177–1183

Blondel VD, Guillaume JL, Lambiotte R et al (2008) Fast unfolding of communities in large networks. J Stat Mech-Theory Exp 2008:P10.008

Bohlin L, Edler D, Lancichinetti A, et al (2014) Community detection and visualization of networks with the map equation framework. In: Measuring scholarly impact. Springer, pp 3–34

Cockcroft M, Mawjee S, James S, et al (2020) Learning options from demonstration using skill segmentation. In: 2020 International SAUPEC/RobMech/PRASA Conference, pp 1–6. https://doi.org/10.1109/SAUPEC/RobMech/PRASA48453.2020.9040988

Cordeiro M, Sarmento RP, Gama J (2016) Dynamic community detection in evolving networks using locality modularity optimization. Soc Netw Anal Min 6(1):15

Daniel C, van Hoof H, Peters J et al (2016) Probabilistic inference for determining options in reinforcement learning. Mach Learn 2016:104. https://doi.org/10.1007/s10994-016-5580-x

Davoodabadi M, Beigy H (2011a) A new method for discovering subgoals and constructing options in reinforcement learning. In: Proceedings of the 5th Indian international conference on artificial intelligence, IICAI 2011 pp 441–450

Davoodabadi M, Beigy H (2011b) A new method for discovering subgoals and constructing options in reinforcement learning. In: IICAI, pp 441–450

Farahani MD, Mozayani N (2019) Automatic construction and evaluation of macro-actions in reinforcement learning. Appl Soft Comput 82(105):574

Fiedler M (1973) Algebraic connectivity of graphs. Czechoslovak Math J 23(2):298–305

Fu KS (1977) Introduction to syntactic pattern recognition. In: Syntactic pattern recognition, applications. Springer, p 1–30

Ghafoorian M, Taghizadeh N, Beigy H (2013) Automatic abstraction in reinforcement learning using ant system algorithm. IN: AAAI Spring Symposium—Technical Report, pp 9–14

Jinnai Y, Park JW, Abel D, et al (2019) Discovering options for exploration by minimizing cover time. In: International Conference on Machine Learning, PMLR, pp 3130–3139

Kazemitabar SJ, Taghizadeh N, Beigy H (2018) A graph-theoretic approach toward autonomous skill acquisition in reinforcement learning. Evol Syst 9(3):227–244

Lin LJ (1992) Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach Learn 8(3):293–321

Machado MC, Bellemare MG, Bowling M (2017) A laplacian framework for option discovery in reinforcement learning. In: International conference on machine learning, PMLR, pp 2295–2304

McGovern A, Barto AG (2001) Automatic discovery of subgoals in reinforcement learning using diverse density. In: Computer Science Department Faculty Publication Series, p 8

Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026113

Raghavan N, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E Stat Nonlinear Soft Matter Phys 76(036):106. https://doi.org/10.1103/PhysRevE.76.036106

Reichardt J, Bornholdt S (2006) When are networks truly modular? Phys D: Nonlinear Phenom 224(1–2):20–26

Setyawan GE, Sawada H, Hartono P (2022) Combinations of micro-macro states and subgoals discovery in hierarchical reinforcement learning for path finding. Int J Innov Comput Inf Control 2022:447–462. https://doi.org/10.24507/ijicic.18.02.447

Shafipour Yourdshahi E, do Carmo-Alves MA, Varma A et al (2022) On-line estimators for ad-hoc task execution: learning types and parameters of teammates for effective teamwork. Autonomous Agents Multi-Agent Syst 36(2):1–49

Shoeleh F, Asadpour M (2017) Graph based skill acquisition and transfer learning for continuous reinforcement learning domains. Pattern Recogn Lett 87:104–116

Simsek O, Barreto AS (2008) Skill characterization based on betweenness. In: Advances in neural information processing systems, pp 1497–1504

Şimşek Ö, Barto AG (2004a) Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, p 95

Şimşek Ö, Barto AG (2004b) Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, p 95

Sürmeli BG, Tümer MB (2020) Multivariate time series clustering and its application in industrial systems. Cybern Syst 51(3):315–334. https://doi.org/10.1080/01969722.2019.1691851

Stolle M, Precup D (2002) Learning options in reinforcement learning. In: International symposium on abstraction, reformulation, and approximation. Springer, pp 212–223

Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT press, Cambridge

Sutton RS, Precup D, Singh SP (1998) Intra-option learning about temporally abstract actions. In: ICML, pp 556–564

Sutton RS, Precup D, Singh S (1999) Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. Artif Intell 112(1–2):181–211

Traag VA, Waltman L, Van Eck NJ (2019) From louvain to leiden: guaranteeing well-connected communities. Sci Rep 9(1):1–12

Tumer M, Belfore L, Ropella K (2003) A syntactic methodology for automatic diagnosis by analysis of continuous time measurements using hierarchical signal representations. IEEE Trans Syst Man Cybern Part B (Cybern) 33(6):951–965. https://doi.org/10.1109/TSMCB.2002.804365

Waltman L, Van Eck NJ (2013) A smart local moving algorithm for large-scale modularity-based community detection. Eur Phys J B 86(11):1–14

Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8(3):279–292

Xu X, Yang M, Li G et al (2018) Constructing temporally extended actions through incremental community detection. Comput Intell Neurosci 2018:156

Zhu X, Zhang R, Zhu W (2022) Mdmd options discovery for accelerating exploration in sparse-reward domains. Knowl-Based Syst 241(108):151

Zhuang D, Chang MJ, Li M (2019) Dynamo: dynamic community detection by incrementally maximizing modularity. IEEE Trans Knowl Data Eng 2019:1–1. https://doi.org/10.1109/TKDE.2019.2951419