Abstract: With the ability to incorporate prior skills into reinforcement learning, skill composition aims to accelerate the learning process on new tasks. Previous works have given insight into combining pre-trained task-agnostic skills for new tasks, where skills are transformed into sequential representation. However, these methods are incapable of capturing potential complex skill relations that would be beneficial for learning better skill representation and further achieving superior skill composition. In this paper, we propose a Graph-based method for Skill Composition (GSC). To learn rich structural information, a carefully designed skill graph is taken as the first-order graph, where skill representations are taken as nodes and skill relations dynamically masked by prior knowledge are utilized as edges. Rather, to allow it to be trained efficiently on large-scale skill set, a transformer-style graph updating method is employed to achieve high-order information aggregation. Our experiments show that GSC outperforms the state-of-the-art methods on extensive challenging tasks.

Keywords: Skill composition, Graph, Deep reinforcement learning

1 Introduction

Reinforcement learning (RL) algorithms coupled with powerful function approximator have recently achieved great success [1, 2, 3]. Unfortunately, such approaches usually require a large number of interactions with exterior environment because they typically learn from scratch rather than leverage prior experience. In contrast, human beings only require comparatively short time to learn new tasks because they can quickly generate skill combination from prior skills to solve complex problems [4, 5]. Inspired by this, researchers have spent much time to acquire combinatorial generalization for agents to reuse skills for handling new tasks [6].

There have been a few works presented for addressing the challenge of composing skills in an efficient and scalable manner. Recent skill composition methods can be categorized into two groups: sequential composition and simultaneous composition. Sequential composition, also known as skill selection, attempts to select one primitive skill at a time and executes such skill until its time horizon. And this process is repeated until reaching the final goal or having the maximum step [7]. In terms of simultaneous composition, the action is generated by combining actions of primitive skills. Previous RNN-based methods [8] explicitly address the composition by linearly weighting such primitive skills, where skills are taken as sequential input with a fixed-order. However, we notice that RNN is more suitable for processing sequential inputs, such as speech signals or natural language. This seems to be less applicable for skills, because there is no clear temporal correlation between any two skills. Intuitively, we want to view skills as an unordered set rather than an ordered list. To overcome this drawback, we give insight into graph neural network (GNN), which has been effective at tasks with rich data type, like social analysis [9], traffic prediction [10] and biology [11]. In our setting, the agents learn a variety of skills against complicated tasks, then all skills are organized as a topology form where each of the nodes in the graph represents an individual skill, and each of the edges represents skill relation. Next, one crucial question remains to be addressed: How can we construct graph nodes and edges for skills? As some neural network parameters, skills themselves are difficult to be used as nodes directly. In addition, the skills in our settings are not homologous.
resulting in incomparable internal representations of neural networks. However, the dynamic action changes of skill output can be regarded as the representation of skills, which has been proved in the previous literature [8]. Therefore, we take the latent representation obtained by projecting the policy output as the graph node. On the other hand, the outputs of skills are probability distributions, so the relationship between them can be measured by Kullback-Leibler divergence (KLD) [12, 13]. Then, these measurements will be taken as the values on the graph edge. Therefore, base on the information mentioned above, we obtain an initial skill graph.

However, another drawback of the previous approach emerges when the input sequence becomes longer as the number of skills increases. RNN-based methods are more difficult to deal with the long-term dependence of the head and tail entries. In contrast, GNN imports all the information for combination equally and simultaneously, instead of taking them in as a fixed-order sequence. Hence it can preserve global-structure information of skill set in graph embeddings. In order to accelerate the information propagation, we propose a transformer-style [14] information propagation to capture high-order neighborhood information, which can benefit large-capacity skill composition. Full details can be found in Section 4. To summarize, our contributions are as follows:

• We are the first to model a whole skill set as a graph and learn skill embeddings with graph neural networks jointly.
• We conduct a simple but effective skill graph in GSC, where each of nodes represents an individual skill and each of edges represents skill relation masked by KLD-based prior knowledge.
• We propose a tailored transformer-style graph architecture to capture superior skill representations by aggregating high-order neighborhood information.
• We conduct extensive experiments to prove the superiority of GSC compared with state-of-the-art methods, especially the robustness when the task difficulty and the number of skills increase.

2 Related Work

Skill Reuse. As RL agents are tasked with challenging and diverse goals, learning policies from scratch often requires tremendous amounts of time and computation power. The paper [15] propose to jointly learn an embedding space of skills and a prior over skills from unstructured data to accelerate the learning of new tasks. Recent approaches [7, 16, 17] improve the learning efficiency by incorporating prior information into the learning process. In these cases, it is generally necessary to learn the high-level policy to select the previously trained policies. Hierarchical reinforcement learning (HRL) [18] can learn a two-level structure, where high-level module decides which low-level policy should be activated. Options framework [19] and macro-action models [20] share similar strategy for combining action sequences, where high-level planner is learned based on extrinsic reward function. However, these methods usually activate only one policy at each time step, which will affect the scope of application. They are not suitable for tasks that need to activate multiple policies synchronously [16].

Concurrent works have explored new structures to simultaneously model the relationship between multiple skills to improve combination efficiency. MCP [21] adopts simultaneous composition using an multi-layer perceptron (MLP), which simultaneously generates primitive skills and a gate function with selective ability. In [8], a Bidirectional LSTM (BiLSTM)-based ensemble composition has been proposed to combine a set of independent and task-agnostic primitive skills for solving the given RL tasks. These methods directly learn assigned weights for action composition, where skills are encoded by using MLP or BiLSTM. The prominent difference between these approaches and ours is that we propose a skill graph with much richer structural information for more effective skill composition. We also explored some prior information to define the initial graph, following prior works using KL divergence to represent skill similarity [22, 12].

GNN-based RL. GNN has been proved to be capable of successfully tackling problems with irregular structured data. Among all previous works, graph convolutional network (GCN) [23] and graph attention network (GAT) [24] are two representative models since they show powerful ability on various tasks like node classification and node clustering. GCN treats neighbor nodes equally when gathering information from neighborhood nodes while GAT adopts self-attention mechanism and
3 Preliminaries

Generally, we assume the interaction between an agent and exterior environment can be modelled as a Markov decision process (MDP) $\mathcal{M} \equiv (\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $p(s_{t+1}|s_t, a_t)$ gives the next-state distribution upon taking action $a_t$ in $s_t$, $r$ specifies the reward associated with the transition $s_t \xrightarrow{a_t} s_{t+1}$, and $\gamma \in [0, 1)$ is the discount factor. The general objective of RL is to find a policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expectation of return $G_t = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$.

As we know, training a policy from scratch to accomplish a complicated new task is time-consuming, especially when the rewards are sparse. Hence during the problem formulation, instead of learning from the beginning, we can acquire some task unrelated policy from pre-training. These pre-trained policies are considered as the innate ability of the agent, also known as primitive skill. And our goal is to learn a composition policy that can utilize the primitive skills of the agent to accomplish the tasks with higher efficiency. Their detailed definitions are as follow:

**Primitive Skills:** The pre-trained policies are called primitive skills, denoted as $\{\pi_i\}_{i=1}^N$, where $N$ is the number of skills. These skills can be trained via standard RL [31, 32] with a given external or intrinsic reward [33]. For simplicity, we set all primitive skills from the same agent share common state and action spaces.

**Composition Policy:** The high-level policy trained to compose primitive skills is called composition policy $\hat{\pi}$ with state space $\tilde{\mathcal{S}} = [\mathcal{S}, \mathcal{G}]$, where the space $\mathcal{G}$ could contain any task-specific information such as the target’s location. For new tasks given prior primitive skills, the specific form of composition policy is $\hat{\pi}(\hat{a}_t|\hat{s}_t, \{\pi_i(a_t|s_t)\}_{i=1}^N)$. Note that primitive skills take as input $s_t$ instead of $\hat{s}_t = [s_t, g_t]$ due to the fact that the primitive skills are task-agnostic.

4 Graph-based Skill Composition

In this section, we present our GSC framework. Two essential mechanisms mentioned in the previous part, skill graph and transformer-style information aggregation, will also be explicitly discussed here. As shown in Figure 1, there are 3 components in our architecture: Graph Construction, Graph Computing, and Skill Composition. In Graph Construction, on the one hand, the action output of the primitive skill goes through a series of transformations, resulting in new representation, which serve as node in the graph. On the other hand, the dynamic mask of the edge features will be obtained based on the prior information between the nodes. Thus, we build a first-level graph, also known as a skill graph. In Graph Computing, we designed a transformer-style module to update and aggregate information from high-level nodes, with the purpose of obtaining a more optimized skill representation. In the Skill Composition, we use an attention mechanism to calculate the associated weight between the current task and every primitive skill, and the final composition action $\hat{a}_t$ can be obtained by weighting these primitive skills. Then, the agent reach the next state $s_{t+1}$ after executing $\hat{a}_t$ in the environment.

4.1 Graph Construction

More formally, we denote the graph structure of primitive skills as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ ($|\mathcal{V}| = N$) and $\mathcal{E}$ denote nodes (skill representations) and edges (skill relations), respectively. We use $h^i$
Graph construction
Graph computing

Figure 1: The schematic of our GSC framework. (a) GSC architecture. (b) Graph construction: In the left branch, actions from primitive skills are projected as the node features. In the right branch, the correlation calculated between two primitive skills are taken as mask features of the edges between nodes. Finally, the skill graph can be constructed. (c) Graph computing: A transformer-style graph update mechanism is designed to aggregate the feature of higher-order nodes. The node representation after the update will be used for the downstream task.

to represent the feature of node \(i\), \(u^{ij}\) to represent the feature of edge \((i, j)\), and \(u^{ij}\) implies the relationship between node \(i\) and \(j\). The essential process of the GNN involves message passing (also called neighborhood aggregation) between the nodes. This message passing operation iteratively updates feature \(h^t_i\) of node \(i\) (red node in Figure 1(a)) by aggregating the features of neighboring nodes (blue node in Figure 1(a)). Next, we use the method shown in Figure 1(b) to construct the graph, and once the construction is finished, all the skills will be embedded into the nodes.

**Graph Initialization** After the mapping transformation, the actions sampled from the primitive skill output distributions are aggregated to the nodes in the graph. Thus, node features (skill representation) can be expressed as:

\[
h^t_i = \tanh(W_f \cdot a^t_i + b_f),
\]

where \(a^t_i \sim \pi_i(a^t_i \mid s_t)\), \(h^t_i \in \mathbb{R}^F\), \(F\) is the dimension of node features and \(W_f, b_f\) are learnable parameters. Without any prior, the initial connection relationship between nodes is unknown, so we temporarily construct a fully connected graph in which the values on edges are equal, such as 1.0.

**Prior Knowledge Injection** To further excavate the relationship between the nodes and build a more compact graph, we try to construct the graph with artificial computation rules. As a prior knowledge, these rules can reduce the training cost of subsequent neural networks.

Mathematically, the Kullback-Leibler divergence (KLD) is a measure of how a probability distribution differs from another. Meanwhile, in RL, the action outputs of skills can be considered as probability distribution, such as Gaussian Distribution \(N_G(\mu, \Sigma)\), where \(\mu, \Sigma\) are mean and variance produced by the policy network. These existing facts inspired us to use KLD to calculate the similarity between the skills. Hence the difference between skills can be measured by:

\[
D_{KL}(\pi_i(a^t_i \mid s_t)||\pi_j(a^t_i \mid s_t)) = \frac{1}{2} \left( \log \frac{\Sigma_j}{\Sigma_i} + \text{tr}(\Sigma_j^{-1} \Sigma_i) + (\mu_j - \mu_i)^T \Sigma_j^{-1} (\mu_j - \mu_i) - l \right),
\]

where \(a^t_i \sim N_G(\mu_i, \Sigma_i), a^t_j \sim N_G(\mu_j, \Sigma_j)\), and \(l\) is the dimension of actions. However, KLD is asymmetric, so the relationship between skill \(i\) and \(j\) is not equal to that between \(j\) and \(i\). To address the symmetry problem\(^1\), we consider **resistor average distance** [34] and propose to use

\[
x^{ij} = \frac{1}{2} \left( \frac{1}{D_{KL}(\pi_i||\pi_j)} + \frac{1}{D_{KL}(\pi_j||\pi_i)} \right)
\]

as the mask feature of edge between nodes. Then, together with \(u^{ij}\), \(x^{ij}\) is applied to graph information propagation in the following **Graph Computing** (Eq. 9).

---

\(^1\) We consider proposing \(u\) in Eq. (3) instead of Jensen-Shannon Divergence (JSD) to avoid expensive computational complexity as JSD does not have a closed expression that need to be approximated at every time step.
### 4.2 Graph Computing

In previous section, we have construct a skill graph that takes skill representations $H = (h^1_t, h^2_t, ..., h^N_t)$ as nodes and KLD-based skill relation matrix as edges. Next, combining the advantages of GAT [24] and Transformer [14], we construct a transformer-style module to calculate information spread in our graph. The key component therein is a graph multi-head attention mechanism, as shown in Figure 1(c), which is tailored for structural input data. For brevity, only a single-time update is described below and the module could extract vectors as queries $q^m$, keys $k^m$, and values $v^m$ from nodes feature $h^i$ of the skill graph, using the following formulas:

$$v^m = \tanh(w_{mv}h^i + b_{mv}),$$

$$k^m = \tanh(w_{mk}h^i + b_{mk}),$$

$$q^m = \tanh(w_{mq}h^i + b_{mq}),$$

where $m \in \{1, ..., M\}$ and $M$ is a hyperparameter for the number of heads, and $w_{mv}, w_{mk}, w_{mq}, b_{mv}, b_{mk}, b_{mq}$ are all learnable parameters.

The inner product of query $q^m$ and key $k^m$ is used to indicate the importance of node $j$ to node $i$:

$$e_{ij} = \frac{q^m_k^m}{\sqrt{F}}.$$  (7)

To make the importance easily comparable across different nodes, we apply softmax function to normalize $e_{ij}$ and then get the attention weight (edge feature) as

$$u_{ij} = \alpha = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})},$$

where $N_i$ is the neighborhood set of node $i$, indicating which nodes have an influence on node $i$, and can be obtained by thresholding $U$. Based on such attention weights and prior knowledge in Eq. (3), the new feature of node $i$ is then obtained as

$$h^i = \frac{1}{M} \sum_{m=1}^{M} \sum_{j \in N_i} u_{ij}^m \cdot k^m_j v^m_j,$$  (9)

which is a linear combination of neighbor-node features, and the whole updated skill representation can be expressed as $H' = (h^1_t', h^2_t', ..., h^N_t')$. By repeating the above process, the node features in the graph can be updated iteratively.

### 4.3 Skill Composition

Here we need to take the task-related goal $g$ into consideration, and form another new input $\hat{s}_t$ with $s_t$. We need to first solve the correlation between $\hat{s}_t$ and each node feature in the skill map, and describe it with the following formula:

$$q_t = W \lambda \cdot \tanh(W_s \cdot f(\hat{s}_t) + W_h \cdot h^i); \forall i \in [1, N],$$  (10)

where $\hat{s}_t = [s_t, g_t]$, $f$ is a forward mapping function and $W_{\lambda}, W_s, W_h \in \mathbb{R}^{d \times d}$ are learnable parameters. Then, the composition weights $\{\omega_i \in [0, 1]\}^N_{i=1}$ for skill composition are determined using gumbel-softmax denoted as softmax ($q/T$), where $T$ is the temperature term [35]. The composite action can be considered as the weighted sum of all primitive skills, so we have $\tilde{a}_t \sim \sum_{i=1}^{N} w_i \pi_i$.

In this paper, we consider off-policy soft-actor critic (SAC) [31] for the training of our policy function. SAC maximizes the expected entropy $H(\cdot)$ in addition to the expected reward objective, i.e.

$$J(\pi) = \sum_{t=0}^{\infty} E[r(s_t, a_t) + \lambda H(\pi(\cdot | s_t))],$$  (11)

where $\lambda$ is a hyperparameter. The overall process can be summarized in Algorithm 1.

### 5 Experiments

In this part of the paper, we will demonstrate the advantages of our method and further explore its characteristics. Through the experiments, we will answer the following questions: 1) Does our GSC outperform the state-of-the-art methods? 2) Can GSC be adapted to tasks that require a lot of skills? 3) How does prior knowledge have an influence on the performance of GSC? 4) How do composition policy and primitive skills affect each other?
Algorithm 1 Graph-based skill composition (GSC)

Input: Primitive skills \( \{\pi_i\}_{i=1}^N \)

1: Initialize composition policy \( \hat{\pi} \) and empty replay buffer \( R \leftarrow \emptyset \).

2: for each iteration do
3:   for each environment step \( t \) do
4:     Sample primitive actions \( \{a_{i_t}\}_{i=1}^N \sim \{\pi_i(\cdot|s_t)\}_{i=1}^N \).
5:     Sample composite actions \( \hat{a}_t \sim \hat{\pi}(\cdot|\hat{s}_t, \{a_{i_t}\}_{i=1}^N) \), where \( \hat{s}_t = [s_t, g_t] \).
6:     Sample next state \( s_{t+1} \sim p(s_{t+1}|s_t, \hat{a}_t) \).
7:     Add sample \( \{\hat{s}_t, \hat{a}_t, \{a_{i_t}\}_{i=1}^N, r(\hat{s}_t, \hat{a}_t), \hat{s}_{t+1}\} \) to buffer \( R \).
8:   end for
9:   Drawing samples from \( R \), update composition policy \( \hat{\pi} \) to maximize reward with SAC.
10: end for

Environment: We pick 2 agents and design 4 environments to challenge our framework with a numerous variety of skills and a rising slope of task difficulties. The first agent is a quadruped ant that moves in a 2D plane and the second agent is a half cheetah that could perform in a one-dimensional direction. Figure 2 present the environments that we construct in mujoco simulation engine (http://www.mujoco.org). Ant Random Goal: In this environment, an ant will navigate to a randomly generated goal within a pre-defined circle. Ant Simple Maze: We introduce a ‘Z’ shape maze in the environment, the ant will navigate through the maze to reach the goal at the end. Ant Cross Maze: We increase the complexity of the maze by changing the shape to a cross. The ant has to navigate to the given goal sampled from 3 pre-determined target locations. HalfCheetah Hurdle In this environment, a half cheetah will move along a straight line while jump over 3 hurdles to reach the destination.

Baselines. We pick three well-developed methods as baselines to evaluate our proposed method in each environments above. BiLSTM [8]: Taking primitive skills as a sequence input, this method consists of two modules: encoder-decoder network and attention network. The former is mainly used to extract the latent representation of primitive skills, while the latter is used to calculate the relevant weights of the current task and primitive skills. The final output of the policy is the weighted sum of these skills. MLP [21]: A multilayer perceptron like-gating function, taking goal information and primitive skills as input, is trained to output weighted sum of primitive skills. SAC [31]: The agent is trained to complete tasks from scratch without any priori.

In all tasks, only reaching the goal position or maximum steps can stop the learning process, and all experimental tasks will be executed over 10 times to obtain stable mean and variance. Besides, we use dense rewards with both positional and actuation costs. Due to the page limitation, more experimental details and real robot experiment can be found in the Appendix.

5.1 Comparative Analysis

Figure 3: The results of our method against BiLSTM, MLP and SAC.

The purpose of this section is to prove that GSC has advantages over the baseline methods. Here we choose the most basic primitive skills for composition. All these skills were pre-trained in free space without mazes, hurdle or targets, and are sufficient for the agent to finish the tasks in our...
environments. For ant, four primitive skills of moving to the left, up, right and down. For half cheetah, we choose the primitive skills of walking and jumping. The methods GSC, BiLSTM and MLP are provided with primitive skills while SAC, as a standard RL method, trains the model from scratch without skill composition.

**Comparison to baselines:** The results are shown in Figure 3, where the average reward is used as the evaluation criterion, and a higher reward indicates better performance. Our method outperforms other skill composition methods and standard RL method in all types of scenarios with a higher data efficiency by learning in fewer steps.

When the agent is in a simple environment (Figure 6(a)), our method appears to have a similar performance with BiLSTM while converging at a higher reward than other baselines. But the lead of GSC has become more obvious as the task complexity increases. From Ant Random Goal (Figure 6(a)) to Ant Simple Maze (Figure 3(b)) to Ant Cross Maze (Figure 3(c)), the difference between the reward of GSC and the second-highest method has become larger. When the environment changes to the HalfCheetah hurdle, the convergence of the MLP remains at a low value and BiLSTM becomes unstable. GSC can still finish the current task with a higher reward than all baselines. It should be noted that MLP directly outputs the mixture weights without conditioning on the latent encoding of primitive actions gives inferior performance compared to our method. Our method utilizes all information (states, goals, and primitive skills) in a structured way through skill representation, and therefore, leads to better performance.

**Importance of skill composition:** Observing from Figure 3, we notice that almost all composition methods (GSC, BiLSTM and MLP) perform better than SAC. The experiment results illustrate the superiority of skill composition methods over the standard RL method. Except in Figure 3(d) the MLP has the lowest convergence. The main reason is that only two primitive skills are considered with very little prior information in this environment, which is a serious challenge for MLP.

### 5.2 Skills Amount Increase

In this section, we will explore the adaptation ability of our method when facing skill amount increase. For ant, we hand-designed the reward function to allow it to move in 30 (or 15) degrees intervals to cover the 360-degree range. In this way, the number of primitive skills for the ant rise to 16. As for half cheetah, due to the fact that this agent only contains 20 state dimensions, we believe that 6 primitive skills are sufficient. Moreover, it is difficult to train extra skills with manually designed reward when half cheetah can only move in 1D environment, so we turn to use an unsupervised method DIAYN [36]. It is important to note that the primitive skills developed in this situation are uncontrolled or even useless, which will pose major challenges for downstream tasks. The details of the primitive skills are in the Appendix.

Figure 4 shows the results of GSC and BiLSTM under different environments. Take notice that as the skill amount increase for the same task, the reward of GSC converges to a higher value. While on the contrary, the performance of BiLSTM remains the same in Figure 4(a), but suffer great degradation in Figure 4(b). The impressive outcome of the GSC in these experiments thoroughly demonstrates that our method can extract the relation between skills by utilizing the structural and semantic information representation. For BiLSTM, all skills are forced to be entered in a fixed order, but in fact there is no intuitive sequential logical relationship between them. This relationship becomes weaker, especially when some primitive skills generated by unsupervised methods have very weak relationships with each other (Figure 4(b)). Additionally, when the number of primitive

---

2 The results of the MLP and SAC methods are not plotted because these methods are less effective for skill composition and we want to show the most important results in the diagram.
skill increases, the long-term forgetting problem of LSTM itself will further weaken the information extraction ability.

5.3 Prior Knowledge Impact

In this section, we design an ablation experiment to explore the impact of prior knowledge on the framework. In the first model, GSC, we use the complete implementation, and in the second model, GSC-w/o-KLD, we partially implement the GSC without the KLD-based prior mask. We use 8 and 16 primitive skills in Ant Cross Maze and 4 and 6 primitive skills in HalfCheetah Hurdle environment to apply the ablation in a wide variety of conditions. The primitive skills used for composition are the same as those introduced in Section 5.2. From Figure 5, one can observe that the reward of the GSC converges to a higher value compared to the GSC-w/o-KLD under the same experimental conditions. This clarifies the indispensable role that the KLD-mask played in our framework.

5.4 Visual Analysis of Composition Attention

To further assess the merit of graph used in our framework, we visualize attention weights from Skill Composition in Figure 6. Take GSC as an example, for task 1, Figure 6(b) shows that the attention weight of up skill reaches the highest while others get lower value throughout the process. This is consistent with our intuition that only the up skill is mainly activated when the agent moves upward. For task 2, as shown in Figure 6(e), the trajectory contains two phases of moving up (step 0 - 130) and up right (step 130 - 210). The attention weight of the up skill reaches the highest in the first phase, and the right skill gradually becomes more prominent in the second phase.

In contrast, BiLSTM and MLP need to take more steps to achieve the goal, or their attention weights seem to be more chaotic as shown in Figure 6(c), 6(d), 6(f) and 6(g). These experiments show that the node in the graph is more discriminative to help the downstream module to quickly assign weights for the specific task.

Figure 6: The visualization of attention weights for skills composition in Ant Cross Maze. (a) Task description: The red trajectory represents task 1 where the ant goes up, the green trajectory represents task 2 where the ant first goes up and then goes up right. (b) - (d) and (e) - (g) present the weighting “strength” of primitive skills for task 1 and 2, respectively. Task 1: (b) GSC-take 120 steps; (c) BiLSTM-take 184 steps; (d) MLP-take 275 steps. Task 2: (e) GSC-take 210 steps; (f) BiLSTM-take 338 steps; (g) MLP-take 598 steps. The x-axis stands for primitive skills and the y-axis stands for rollout steps. The vertical bar at the right side of the attention weight plot represents the weight value, lighter color refers to larger weight.

6 Conclusion and Future Work

In this paper, we propose a novel GSC method for effective skill composition to deal with new downstream tasks. Specifically, we explore and exploit skill relations by building a skill graph, where skill representations and skill relations are nodes and edges, respectively. KLD-based prior injection and transformer-style information propagation are applied to extract superior skill representations. Extensive experiments demonstrate that our GSC is able to efficiently compose primitive skills and solve challenging tasks compared to state-of-the-art methods.

As the paper [8] (our baseline) points out that, there is still need constructive approaches that simultaneously combine and transfer past skills into new skills. To the best of our knowledge, we are the first to investigate GNN in this domain and have proved its superiority experimentally. Although a classical GNN is used in this paper, it has constructed the foundation for future research. To allow the graphical network containing more skills, a direction worth further exploring is to reduce redundant skills like Graph Polling [37], or learn the structure between skills automatically like Graph Structure Learning [38]. We believe exploration in skill graph is meaningful and it will better bring robots with more skills into the real world.
References


