Graph Learning based Generation of Abstractions for Reinforcement Learning.

Yuan Xue
L3S Research Center
yuan@l3s.de

Daniel Kudenko
L3S Research Center
kudenko@l3s.de

Megha Khosla
L3S Research Center
khosla@l3s.de

ABSTRACT

The application of Reinforcement Learning (RL) Algorithms is often hindered by the combinatorial explosion of the state space. Previous works have leveraged *abstractions* which condense large state spaces to find tractable solutions, however they assumed that the abstractions are provided by a domain expert. In this work we propose a new approach to automatically construct Abstract Markov Decision Processes (AMDPs) for Potential Based Reward Shaping to improve the sample efficiency of RL algorithms. Our approach to construct abstract states is inspired by *graph representation learning methods* and effectively encodes topological and reward structure of the ground level MDP. We perform large scale quantitative experiments on Flag Collection domain. We show improvements of up to 6.5 times in sample efficiency and up to 3 times in run time over the baseline approach. Besides, with our qualitative analyses of the generated AMDP we demonstrate the capability of our approach to preserve topological and reward structure of the ground level MDP.

KEYWORDS

Reinforcement Learning, Abstract MDP, State Representations, Graph Representations

1 INTRODUCTION

Reinforcement Learning Algorithms often suffer from a combinatorial explosion of the state space [2]. One way to tackle this problem is to leverage *abstraction* [1, 13] to condense large state spaces such that essential information is preserved, and consequently, solutions are tractably computable. Abstraction methods reduce ground level MDPs with large state spaces to *abstract MDPs* (AMDPs) with smaller state spaces by aggregating states according to some notion of similarity. For example, abstractions can be realised by aggregating states with equal values of particular quantities, for example, optimal Q-values. We focus on exploiting state abstractions to build informative Abstract Markov Decision Processes (AMDPs), and use the resulting value function for *Potential Based Reward Shaping* [16] to improve the convergence speed of RL algorithms.

Domain experts can sometimes build high quality AMDPs by manually aggregating ground states into abstract states. An example is shown in Figure 1 where a domain expert would create abstractions based on the intuitive concept of "Rooms". However, such manual approaches can not be scaled to large environments with high dimensionality.

In an attempt to automatically generate abstractions, [3] proposed to uniformly partition the state space to build abstract states. While this has been shown to improve the convergence rate of RL, it first needs to know the boundaries of the state features of the environment to perform uniform partitioning. Moreover, the resulting abstract states completely ignore the connectivity and reward structure of the underlying MDP. Consider for example (see Figure 2) two states which are spatially close but are separated by an obstacle and it takes a large number of steps to move between the states.

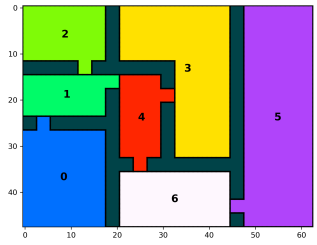


Figure 1: Manually created abstraction

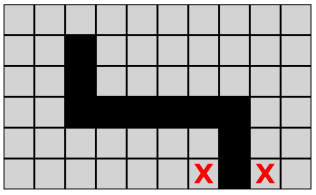


Figure 2: The two states in label "X" are spatially close, but topologically quite far from each other. They are likely to be aggregated into same abstract state using uniform partitioning.

We argue that the state aggregation for reward shaping should be based on properties of *topological* and *value function* similarity. Intuitively, the ground level states belonging to the same abstract state should be topologically close and should appear more frequently together in the high reward paths from the start state to goal state. Moreover, if two states are in the same abstract state their long-term expected discounted reward (i.e. value function) should also be similar.

To satisfy the above two properties, we propose a new approach to generate state abstractions which exploit the underlying topology and reward structure of the ground level MDP. Borrowing ideas from graph representation learning, we first extract latent representations of the states encoding similarity among states in terms of topological and reward structure. The state representations are then clustered to generate the AMDP. Our experimental results show vast improvements in convergence speed as compared to the previous state-of-the-art [3] which constructs AMDPs by uniformly

partitioning each state-dimension into a predefined number of abstract cells. We attribute the performance gains to an improved cluster structure which qualitatively and intuitively agrees well with the cluster structure induced by the state value functions of the ground level MDP.

To sum up our main contributions are as follows.

(1) We introduce a new approach to construct abstract states to preserve properties of topological and value function proximity among the ground level states.
(2) We show that reward shaping with our constructed AMDP results in improvements of up to (i) 6.5 times in *convergence speed* and *sample efficiency* and (ii) 3 times in *run time* of the RL algorithm over several different maze classes.
(3) With our qualitative analysis of the generated AMDP we demonstrate that our approach is able to preserve topological and reward structure of the ground level MDP.

We will publish our code for reproducibility and further development at the time of publication.

2 BACKGROUND AND RELATED WORK

2.1 Reinforcement Learning

Reinforcement Learning consists of an environment and an agent which is driven by a policy to interact with the environment. For each interaction, the agent observes the response of the environment to its actions and update its behaviour. The goal for the agent is to maximize the accumulated reward given by the environment in limited numbers of interactions. The environment is typically represented by a Markov Decision Process(MDP). The standard notation for MDPs:

$$\mathcal{M} = (S_{\mathcal{M}}, A_{\mathcal{M}}, R_{\mathcal{M}}, P_{\mathcal{M}}) \tag{1}$$

where $S_{\mathcal{M}}$ is a set of available states the agent may find itself in the environment, $A_{\mathcal{M}}$ denotes available actions for each state, $R_{\mathcal{M}}(s, a, s')$ defines the immediate reward after the agent transitions from s to s' via a, $P_{\mathcal{M}}$ denotes the probability of reaching s' when performing action a at state s.

When the complete knowledge of the environment's MDP is unknown, algorithms based on utilizing temporal-difference is often used. Q-Learning is one of the fundamental TD-Learning algorithms, which update the value of a state-action pair after each transitions:

$$Q(s, a) = Q(s, a) + \alpha(R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{2}$$

where a denotes the learning rate and γ is the discount factor.

In order to improve the learning rate of Q-Learning, multiple updates can be made to the Q function after each transition. Since an action made by the agent will likely influence not only immediate but also future rewards, it is reasonable to give more credit to more recently experienced state-action pairs. The extent of updates to those state-action pairs is quantified by their eligibility. After each transition, the current state-action pair has its eligibility to 1, the eligibility of other state-action pairs in the same trajectory will decay in contrast. Q-Learning utilising eligibility is referred to as Q(λ) algorithm. A more detailed introduction regarding reinforcement learning refers to [19].

2.2 Reward Shaping

Traditional TD-Learning algorithms still suffer from sparse reward in large-scale environments and a slow convergence rate. Reward shaping helps alleviate the problem by giving the agent an extrinsic reward $F(s, a, s')$. This additional reward imparts the external domain knowledge to the agent and stimulates the agent towards desired behaviours.

It has been proven in [16] that if $F(s, a, s')$ is defined as the difference of potential between two states, the optimal policy will remain unchanged.

$$F\left(s, a, s'\right) = \gamma\phi\left(s'\right) - \phi(s) \tag{3}$$

In most past work, the potential function $\phi(s)$ was defined by a domain expert, but in cases where this is difficult it is preferable to automatically construct such a function.

2.3 Reward Shaping with Abstract Markov Decision Processes

A potential function can be generated by solving an abstract version of task, represented by an Abstract Markov Decision Process(AMDP):

$$\mathcal{A} = (S_{\mathcal{A}}, A_{\mathcal{A}}, R_{\mathcal{A}}, P_{\mathcal{A}}) \tag{4}$$

where each element is an abstraction corresponding to its counterpart in an MDP. A mapping function Z is also constructed between ground states and abstract states. AMDPs can induce Potential Based Reward Shaping. Once an AMDP is built, dynamic programming is used to compute an optimal value function V over the AMDP. Then V is used to build the potential function $\phi(s) \leftarrow \omega V(Z(s))$, where ω is a scaling constant.

Since we have multiple agents operating on different levels of abstractions, we refer to the agent interacting with the AMDP as the abstract agent and the agent interacting with ground level MDP as the ground agent. We denote this similarly for abstract and ground learning process.

2.4 Related Work

Marthi [13] first introduced the idea to automatically modify the reward function of a Markov Decision Process by defining an abstract MDP so as to speed up standard reinforcement learning algorithms. In past work (e.g. [5] and [7]), AMDPs were manually built by an domain expert. However, generating AMDPs in this way often requires utilising external domain knowledge which can be hard to acquire or encode. When facing complex domains, human experts could be inefficient and even fail to build AMDPs. In [3, 4] a conceptually simple method for constructing state abstractions was proposed. In this approach, the state space is partitioned uniformly along each dimension, and each partition forms an abstract state. While the results showed a speed-up of RL, the state abstraction generated in this way doesn't match the topological and reward structure of the underlying MDP (i.e. environment). Consequently, the reward shaping derived from corresponding AMDPs could be inaccurate and mislead the ground learning process. Our approach can handle this limitation.

Automated discovery of options [20] or skills is another related active research area, useful options are proven to speed up RL learning. Similar with constructing an AMDP, hand-crafted options can be quite time consuming as well. In [8, 12, 18, 21], clustering algorithms are applied on a MDP model or graph estimated from states trajectories to identify abstract states or bottlenecks and subsequently generate options. Also, this method struggles to robustly and effectively generate abstractions aligned with the underlying topological and reward structure.

Note that given the state transition history , there is no unique way to construct the corresponding graph. It is therefore inevitable to lose some information at this approximation step. These approaches could require huge storage for the estimated graph when facing large and complex environments. Our approach, on the other hand, directly learns low-dimensional latent state representations from the state transitions which are then clustered to generate abstract states.

Our approach employs graph representation learning or node embedding approaches [6, 9, 10, 17]. The main goal there is to learn low dimensional representations of the nodes such that the topological structure of the graph is preserved. In the area of Reinforcement Learning, [11] used node embeddings as basis functions in a generalized version of representation policy iteration (RPI). Other work [22] showed that graph representation learning techniques can be leveraged to learn informative state representations for DeepRL Networks.

3 OUR APPROACH

Solving dynamic programming over AMDP to achieve reward shaping is a well-known paradigm to boost the convergence rate of model-free RL algorithms. It is crucial to build appropriate AMDP which can preserve topological and reward structure of the environment. Inappropriate abstraction can result in inaccurate reward shaping, which could slow down the convergence rate or even destroy the learning process.

In order to get useful reward shaping, we formulate two important properties for state abstraction namely, (i)*topological proximity* and (ii)*value-function proximity*. **First**, the generated abstract states should preserve *topological proximity* among the ground level states. We say that two states are topological proximal if the states are reachable from each other in a small number of steps. In other words the construction of abstract states should obey the underlying topology of the ground level MDP. **Second**, any two states in an abstract state should have similar value functions. To understand the intuition behind the second property, note that the member states of the same cluster receive the same shaped reward. Now, if the actual value functions of the states differ too much then providing the same shaped rewards would have a negative impact on the learning of the ground level policy.

Towards incorporating the desired properties, we first generate latent representations of the states preserving the topology as well as the reward structure of the ground level MDP. In particular, we design a novelty and reward exploration strategy, which generates experiences with a bias towards making states sharing similar reachability and ground value functions co-occur more frequently. The experiences are then used to learn state representations. The more co-occurring the states are, the closer would the states be embedded in the the low-dimensional representation space. These representations are then clustered into abstract states for constructing AMDP. Finally, the AMDP is exactly solved to compute the reward shaping for the ground level agent. In the following subsections, we further elaborate our approach which we refer to as the Topology approach.

3.1 Learning Abstract States

Let a graph $G = (V, E)$ represents the MDP where each node $s \in V$ corresponds to a state and two nodes (s, s') have an edge between them if the s' can be reached from s in one step transition. We adopt the ideas from graph representation learning [9, 17] to learn continuous representations of the nodes in V such that the topology of G is preserved. As G is not known in advance we devise a novelty based SARSA strategy to collect sequences of state. The co-occurrence frequency of nodes in one episode within a predefined window range then quantifies the similarity between the nodes. Given the state sequences, we use the Skip-gram model [14, 15] with negative sampling to learn state representations. States will be closer embedded, if they co-occur more frequently. Let $\mathcal{N}(s)$ denote the higher order neighborhood or the co-occurring states of state s. We are interested in learning state and context representations $\Phi(.)$ and $\Theta(.)$ such that the following objective is maximized

$$\sum_{s \in V} \sum_{s' \in \mathcal{N}(s)} \left(\log(\sigma(\Phi(s) \cdot \Theta(s')) + \sum_{k=1}^{K} \mathbb{E}_{s'' \sim P} \log(\sigma(-\Phi(s) \cdot \Theta(s'')) \right) \tag{5}$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function, K is some constant denoting the number of negative samples and P denotes a random distribution over the observed node set. The state representations are then used to construct the abstract states.

3.1.1 Exploring the environment. We here describe in detail our novelty based SARSA strategy employed to explore the ground level MDP and generate state sequences used in state representation learning. Our exploration strategy is motivated by two main objectives: (i) explore the environment as much and uniformly as possible with limited cost. (ii) more similar states (in terms of their reachability and ground value functions) should co-occur more frequently during exploration.

The pseudocode of our ground level exploration policy algorithm is provided in Algorithm 1. To enforce our above two objectives, we develop a dynamic reward to favour certain transitions over others. In particular for a transition (s, a, s') a dynamic reward r (see line 12 in the pseudocode) is computed by taking into account both novelty(visit count) of s' and change of ground reward function (compared to recent mean reward) for this transition.

We then learn Q-values for state-action pairs using a SARSA based feedback mechanism (see lines 15-18). We remark that we do not require Algorithm 1 to converge and are only interested in using the learnt Q-values to weakly guide our exploration.

We fix the the length (M) and number (N) of episodes to a small number sothat the size of *experiences* will be $(M + 1) \cdot N$. The time complexity of this phase is therefore $O(MN)$. In our experiment, we are able to keep the time consumption for exploration acceptable so

that the total time of solving tasks is still substantially reduced, as compared to uniform partitioning approach. Detailed comparison of run time is discussed in section 4.2.2.

Algorithm 1 Exploration-SARSA

Input: Length (M) and number (N) of episodes, hyperparameters $\alpha, \beta, \theta, \rho$

```
experiences = []
Initialize Q(s, a)
Initialize starting state s randomly
Visit(s) ← 0                          ▷ Initialize visit counts for all states
for episode = 1, 2, ... N do
    Choose a from s using policy derived from Q (ϵ greedy)
    track = [s]
    m ← 0   ▷ Initialize mean reward (m) of recent transitions
    for step = 1, 2, ... M do
        Visit(s) ← Visit(s) + 1
        Take action a, observe s', r_env
        r ← −β Visit(s') − θ |r_env − m|
        m ← ρ m + (1 − ρ) r_env
        track.append(s')
        Choose a' from s' using policy derived from-
            Q (ϵ greedy)
        Q(s, a) ← Q(s, a) + α [r + γQ (s', a') − Q(s, a)]
        s ← s'; a ← a'
    experiences.append(track)
return experiences
```

3.1.2 Constructing the Abstract States. We employ K-Means clustering to cluster the learned state representations into k clusters, as k abstract states, where k determines the granularity of state abstraction. A mapping function Z between ground states and abstract states is built then.

3.2 Dynamics of AMDP

Once the abstract states are determined, we complete an AMDP which describes the dynamic of the explored environment on the abstract level(Algorithm 2).

In case there are some states in the environment which are undesirable(having quite low ground value functions) to visit for ground agent, those states can be memorized into a set T during exploration phase. And Algorithm 2 can ensure that, value function of generated AMDP agrees with the ground value function on abstract level. Consequently, reward shaping will guide the ground agent to avoid states in T.

The transitions of AMDP are determined by observations happened during exploration phase. For each step $(s \rightarrow s')$, if $Z(s)$ and $Z(s')$ don't belong to the same abstract state, then it is transitable from $Z(s)$ to $Z(s')$, namely $P_{\mathcal{A}}(Z(s), Z(s) \rightarrow Z(s'), Z(s')) = 1$, where $Z(s) \rightarrow Z(s')$ denotes abstract action a. Not observed abstract transitions are assigned probabilities of 0. During exploration phase the ground level goal state s_g should be also observed. In AMDP, from abstract state containing goal state can also transit to itself, formally $P_{\mathcal{A}}(Z(s_g), Z(s_g) \rightarrow Z(s_g), Z(s_g)) = 1$.

The abstract reward function is conceptually simple as well. We set $R(Z(s_g), Z(s_g) \rightarrow Z(s_g), Z(s_g)) = 0$ sothat $Z(s_g)$ will get highest value once the AMDP is solved. In addition, for transitions from abstract states containing ground states memorized in T, a huge negative reward is given. That ensures these abstract states get lower values than their neighbours over the solved AMDP. For rest normal transitions, reward -1 is assigned. Since our approach can automatically construct abstract states, transition and reward function of AMDP based on stored experiences from exploration stage, no external domain knowledge is required.

Algorithm 2 AMDP Construction

Input: $experiences, T$
Output: $AMDP = (S_{\mathcal{A}}, A_{\mathcal{A}}, R_{\mathcal{A}}, P_{\mathcal{A}})$

```
States representation: G_S ← train_model(experiences)
S_A ← Clustering(G_S)
Create mapping function Z : S → S_A
Initialize J ≫ |S_A|
Initialize abstract transitions: P_A (s, a, s') ← 0
Initialize abstract rewards: R_A (s, a, s') ← 0
for all track ∈ experiences do
    for each adjacent state pair (s, s') in track do
        if Z (s) ≠ Z (s') then
            P (Z (s), Z (s) → Z (s'), Z (s')) = 1
            if s ∈ T then
                R (Z (s), Z (s) → Z (s'), Z (s')) = −J
            else
                R (Z (s), Z (s) → Z (s'), Z (s')) = −1
        if s == s_goal then
            P (Z (s), Z (s) → Z (s), Z (s)) = 1
return AMDP = (S_A, A_A, R_A, P_A)
```

3.3 Exploiting AMDP

Once a complete AMDP is constructed, we use dynamic programming (Value Iteration) to solve the AMDP exactly, then a Value $V(t)$ for each state in AMDP can be computed. With this value function over abstract states, we can achieve the Potential Based Reward Shaping and further boost an existing RL algorithm. More specifically, after each ground level step $s \rightarrow s'$, if we observe $Z(s) \neq Z(s')$, which means an abstract transition happens. Consequently the agent gets a combined reward, r as follows

$$r = r_{env} + \omega \cdot F(t, t'), \tag{6}$$

where $t = Z(s), t' = Z(s')$, $F(t, t') = \gamma V(t') - V(t)$ and ω is a weight parameter. The reward given by the environment plus reward shaping can be treated as an augmented reward to update the agent's policy. Except abstract states of T, those abstract states topologically closer to the abstract goal state will have higher values over $V(t)$, like potential. This intuitively ensures that, the shaped rewards can always guide the agent towards abstract states with higher potential.

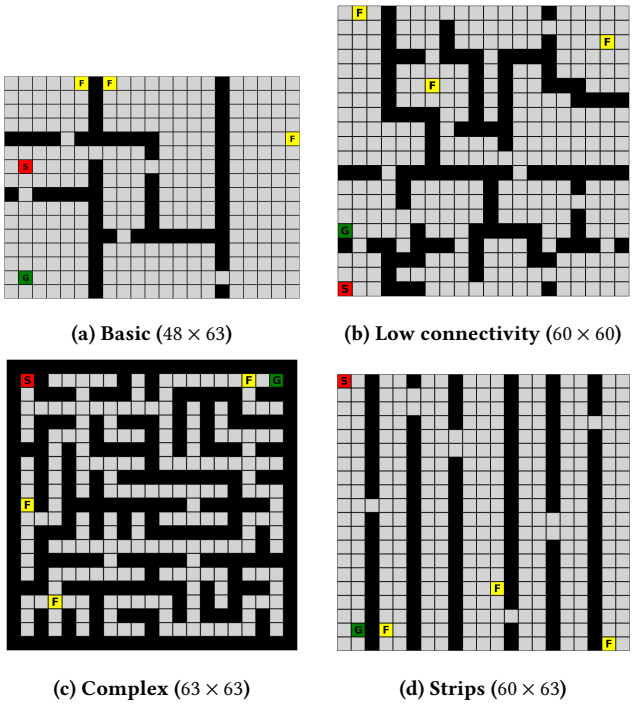
4 EXPERIMENTAL EVALUATION

Our evaluation is divided into two parts. **First**, we perform *quantitative* evaluation in which we show the superiority of our approach

in terms of sample efficiency, convergence speed and run times. Besides, we conduct sensitivity experiments in which we show that our approach is robust (as compared to the baseline) towards the number of clusters or the granularity of state abstraction. We used the Flag Collection Domain to evaluate the performance of our approach as compared to the uniform partitioning approach[3] which we refer to as UNIFORM in our experiments.

Second, we perform a *qualitative* analysis of the constructed AMDPs on the Grid World and Flag Collection domains. In particular, through visualizations of value function heatmaps of ground level MDP and generated AMDPs, we demonstrate that our approach in fact achieves the desired two properties of topological and value function proximity.

4.1 Setup



(a) Basic (48×63) **(b) Low connectivity** (60×60)

(c) Complex (63×63) **(d) Strips** (60×63)

Figure 3: Flag Collection variations

We use the following two navigational domains for our experiments.

(1) *Flag Collection Domain*: We quantitatively evaluated our approach on 4 different variants (shown in Figure 3) of the Flag Collection domain. The actual sizes of the maze variants are provided in the captions. As an augmentation of the normal Grid World Navigation domain, the Flag Collection domain tasks the agent with traversing a gridworld environment, collecting flags marked as F and bringing them to the goal marked as G. The agent can't move out of environment's boundary and reappear on the opposite side.

Table 1: Hyper-parameters used in experiments

Parameter	*Explore agent*	*Ground agent*
α	0.1	0.1
γ	0.999	0.999
ϵ	0.01	$1 \longrightarrow 0.1$
ω	–	300
λ	–	0.9
Episodes	30000	500
Max steps per episode	150	∞

There are impassable walls spreading throughout the environments, which determine the connectivity of the environments. Each state has up to 4 actions to choose, corresponding to 4 orthogonal directions. Each episode begins from the starting state marked as S and terminates when the agent reaches the goal coordinate, no matter how many flags are collected. In all experiments, the number of flags is fixed to 3. For each ordinary step, where no flag is collected and the goal state is not reached, the agent gets a step penalty of -1. For steps picking up a flag, a reward of 10,000 is given. For reaching the goal state $(x_{goal}, y_{goal}, 1, 1, 1)$, the agent receives a reward of 0. Here (x_{goal}, y_{goal}) corresponds to the coordinates of goal state in the maze, the last three binary dimensions $(1, 1, 1)$ indicate that all three flags are collected.

(2) *Grid World with traps* : For the qualitative analysis we also used a grid world maze as shown in Figure 4, which is 33×48 in actual size. The task is to reach the green cell marked as G starting from red cell marked as S. The agent should avoid getting into purple cells(traps) marks as T, which cause huge negative rewards. For each ordinary step, a reward of -1 is given; for steps to the goal state $s_{goal} = (x_{goal}, y_{goal})$, a reward of 0 is given, and for steps getting into the traps, a reward of -1000 is given.

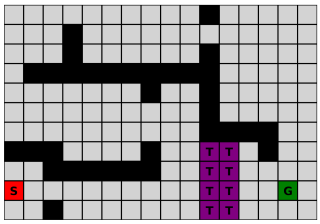


Figure 4: Grid World with traps (33×48)

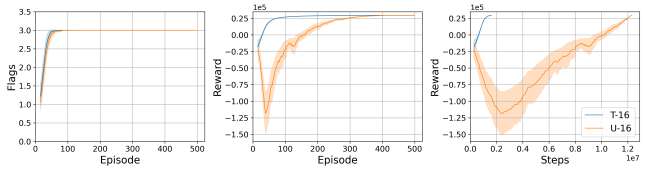
4.1.1 Hyper-parameter settings. For the Flag Collection domain, settings of Hyper-parameters for the exploration agent and the ground agent are listed in Table 1. For the domain of Grid World with traps, the number of exploration episodes is reduced to 5,000, since the actual state space of the Grid World with traps is much smaller than that of Flag Collection domain. More specifically, a state of Flag Collection domain has 3 more binary dimensions (indicating how many flags are carried) than a state of Grid World with

traps. A visualization of this can be found in Section 4.3. Moreover, parameter γ for solving AMDP is set to 0.99 for both experiment domains. In each Flag Collection domain, we quantitatively evaluate performance based on mean results of 25 repetitions for both the Topology approach and the Uniform approach.

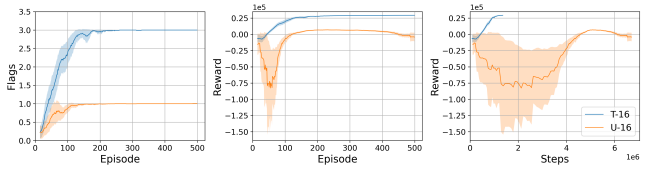
To prepare the data for training state representations, each collected state sequence is 50% downsampled at random, while still maintaining the original sequence order. We observed that such downsampling allowed the reduction of training time without any adverse affects on the quality of learned abstractions. We use a window size of 75 to collect the context states.

4.2 Quantitative Results

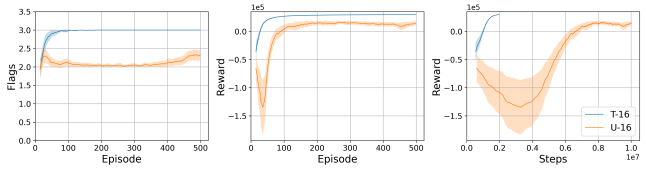
4.2.1 Convergence Speed and Sample Efficiency. We compare the Topology and Uniform approaches over the 4 variants of the Flag Collection domain. In Figure 5, we compare the two approaches with respect to (i) the number of flags collected (ii) the cumulative reward with respect to number of episodes and (iii) the cumulative reward with respect to the number of total steps.



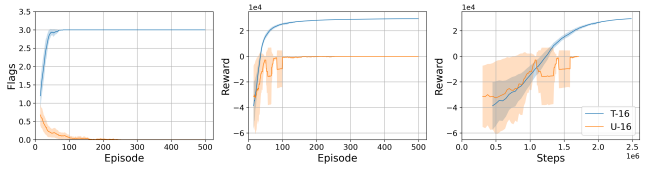
(a) Basic



(b) Low connectivity



(c) Complex



(d) Strips

Figure 5: Comparison of performance between Topology approach(T) and Uniform approach(U), using same number of abstract states(on average there are 16 abstract states in each subspace)

Our Topology approach (denoted as T) is more stable and faster than the Uniform approach (denoted as U). The Topology approach successfully converges (collecting all 3 flags) in 500 episodes, while the Uniform approach even fails to collect all flags in some mazes. Meanwhile, to achieve the same or even higher reward, the Topology approach costs much fewer total steps (up to 6.5 times fewer steps achieving same reward) than the Uniform approach to learn the optimal policy in 500 episodes, thereby showcasing better sample efficiency than the Uniform approach. More comparisons with different abstraction granularities have been done and similar results are observed.

Besides, note that the way the Uniform approach partitions mazes might cause conflicts between the layout of abstract states and the topology of the environment. For example: two states separated by an obstacle might correspond to the same abstract state, but are topologically pretty far from each other. Consequently the guidance of reward shaping could likely convey noise and be partially misleading.

4.2.2 Run Time Comparison. In Figure 6, a detailed run time comparison for the Flag Collection task over maze Basic is presented. In Basic maze, both approaches can converge to the same reward. Our approach has three phases before solving the AMDP: exploration, state representation learning and state clustering, which the Uniform approach doesn't have. Despite of that our approach is in total up to 3 times faster than the Uniform approach while achieving the same total reward.

We remark that the time taken to solve the AMDP through dynamic programming is negligible for both approaches as in the Flag Collection domain state transitions are deterministic. However, in other experimental domains, it might be important to consider the impact of abstraction granularity on the time required to solve AMDP. In principle, run time would increase when the number of abstract states gets higher. We therefore study the effect of different abstraction granularities on the performance of both approaches.

4.2.3 Effect of Abstraction Granularity. Figure 7 shows the performance of sample efficiency using different numbers of abstract states, which depict different abstraction granularities. As shown in Figure 7(left column), Topology approach shows robustness towards different abstraction granularities in different mazes. In contrast, as shown on the right side of Figure 7, the performance of Uniform approach under fixed abstraction granularity is quite unstable among different mazes. In particular for the Uniform approach, the abstraction granularity achieving the best performance in one maze is not necessarily the best one in another or even fails the task.

Another advantage of the Topology approach is that the efficacy of reward shaping from AMDP is predictable as the abstraction granularity varies. As shown on the left side of Figure 7, for the each flag collection task, when the abstraction granularity becomes finer, the extent of acceleration increases correspondingly, and vice versa. This can be observed among all mazes.

However, for the Uniform approach, finer abstraction granularity doesn't necessarily increase the convergence time and may even fail at the tasks. For some mazes, the best uniform abstraction

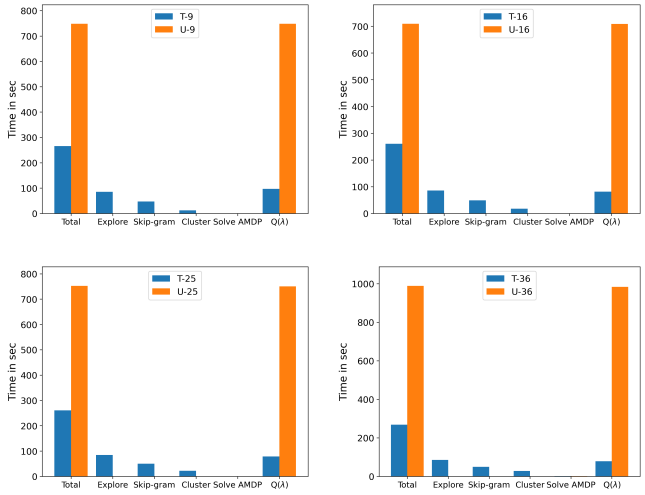


Figure 6: Run time comparisons of both approaches in Flag Collection domain over Maze Basic for four abstraction granularities from coarser(9 abstract states per subspace in average) to finer(36 abstract states per subspace in average)

granularity turns out to be a coarser one. Therefore, a proper abstraction granularity needs to be determined experimentally, which costs extra time.

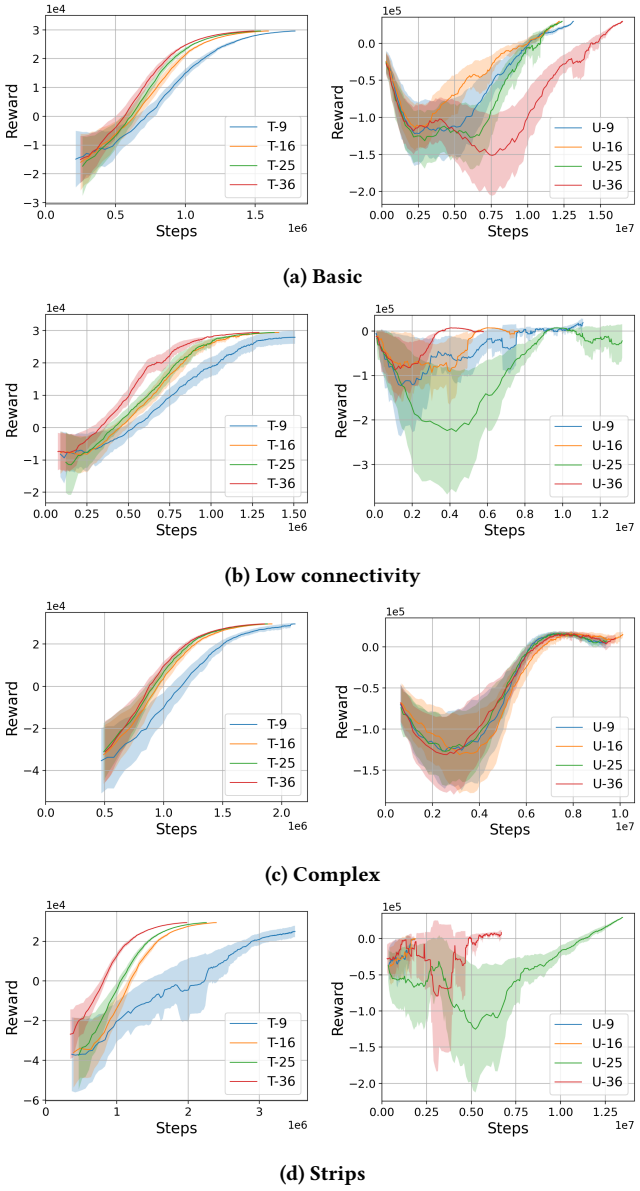
4.3 Qualitative Analysis

4.3.1 Preserving Topological and Value Function Proximity. We use the Grid World domain with traps (see Figure 4) to demonstrate through visualizations that our Topology approach can preserve both topological and reward structure of the ground level MDP into the AMDP. Towards that we first solve the ground level MDP by dynamic programming to obtain the optimal policy, which certainly avoids the traps to reach the goal state. The corresponding value function as a heatmap for the ground level MDP is shown in Figure 8b.

According to Algorithm 1, the exploration agent always prefers to visit states with higher novelty (lower visit counts). Meanwhile the exploration agent tries to avoid traps, as once it gets into them, then it is relatively hard to come out. Consequently, the whole state space can be uniformly explored and the states sharing similar topological structure and ground value functions will co-occur more frequently, as shown in Figure 8a.

Generated abstract states by clustering together with the heatmaps corresponding to solved AMDPs are shown in Figures 8c and 8d for the Topology and Uniform approaches respectively. Clearly the layout of abstract states for the Topology approach agrees with both topology and value function of the ground level MDP(Figure 8b). More prominently our approach was able to cluster the adjoining trap states together in a single abstract state. By comparison, in Figure 8d, the Uniform approach constructs abstract states ignoring the topological structure and traps in the environment.

In particular, in Figure 8c we also note that the abstract state containing traps gets a much lower value than its neighbours, other



(a) Basic

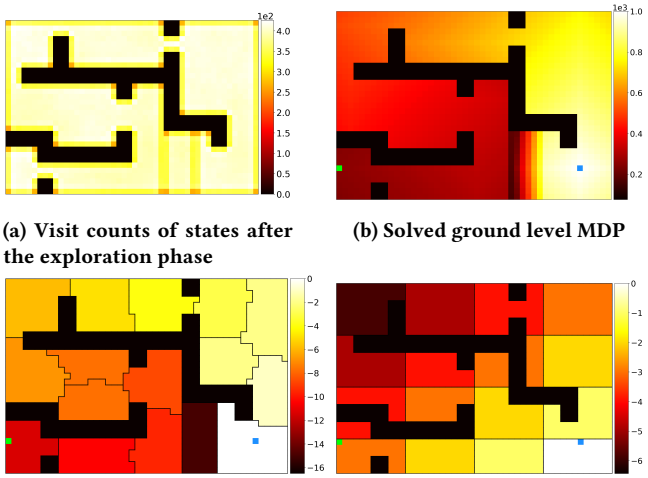
(b) Low connectivity

(c) Complex

(d) Strips

Figure 7: Comparison of performance with respect to sample efficiency under different abstraction granularities. The left column and right column present the performance of Topology approach and Uniform approach respectively.

abstract states get potential-like values. The closer an abstract state is to the goal state, the higher is its value. The difference (scaled by ω) between two abstract states can be taken as an additional reward for the ground level agent. Therefore the guidance of reward shaping will roughly agree with the optimal policy derived from the solved ground level MDP (Figure 8b). One can always simulate a successful path to the goal state by following the brightness of the heatmap from darker to lighter regions. In contrast, the reward shaping from the Uniform approach guides the agent straight



(a) Visit counts of states after the exploration phase

(b) Solved ground level MDP

(c) Solved AMDP of Topology approach

(d) Solved AMDP of Uniform approach

Figure 8: Comparison of solved AMDPs for both approaches in Grid World with traps. Green cell on the left side denotes starting state, goal state is the blue cell on the right side.

towards the goal state, directly traversing the region of traps, as indicated in Figure 8d. This looks correct in the abstract MDP, but actually causes huge negative rewards in the environment. Moreover, reward shaping in Figure 8d is actually hindering the ground level agent from learning the optimal policy in certain area (from bottom right to top left) of the maze.

4.3.2 Visualization in Flag Collection Domain. Figure 9 visualizes how our Topology approach works over the actual state space of the Flag Collection domain. In our Flag Collection domain, a state is in form of $s = (x, y, a, b, c)$, where x, y indicate coordinates and a, b, c are binary variables indicating whether the corresponding flag is collected or not. Since the last three dimensions are binary, we are able to visualize and convert the 5-dimensional state space into 8 2-dimensional subspaces, corresponding to each status of flag collection. Certain transitions are only uni-directional, for example from state $(x, y, 0, 0, 0)$ one can transit to state $(x, y, 0, 1, 0)$, but the transition in the reverse direction is not possible.

As shown in Figure 9, the regions with irregular boundary are the abstract states generated by Topology approach and they all abide by the topology of the environment. It is possible that one abstract state containing a flag could span across two subspaces, because the Topology approach generates abstract states following the topology of the complete 5-dimensional state space. This characteristic enables our approach to build the transition and reward function of AMDPs automatically, based on stored experiences without external domain knowledge (according to Algorithm 2). That is to say, the Topology approach is able to generalize to various state spaces, in which each dimension could have very different properties.

In Figure 9, value functions for abstract states are also presented as a heatmap, where lighter colors indicate higher values, and vice versa. Starting from the white cell(left bottom corner) in subspace(0-0-0), reward shaping can always steer the agent towards abstract states with higher values, then approach the goal state (blue cell on the left bottom side) in subspace(1-1-1) efficiently.

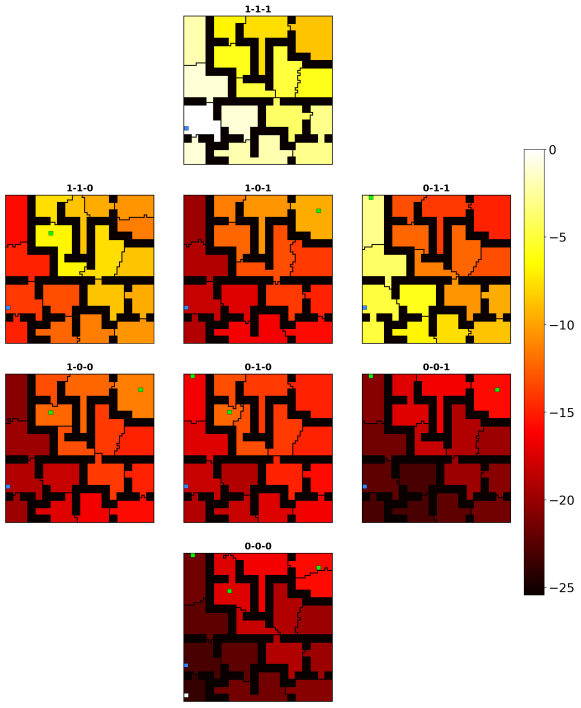


Figure 9: Solved AMDP for Topology approach in Flag Collection domain over maze: Low connectivity. The agent starts from the white cell(left bottom corner) in subspace(0-0-0) and tries to reach the goal state(blue cell on left bottom side) in subspace(1-1-1). Green cells located in the upper half of the maze are flags.

5 CONCLUSION

We proposed a novel approach for generating high-quality AMDPs that help accelerate existing model-free RL algorithms. Our approach to construct abstract states is inspired by *graph representation learning methods* and effectively encodes topological and reward structure of the ground level MDP. Meanwhile, it requires little external domain knowledge and generalizes well to various state spaces. We showed impressive performance improvements over the Uniform approach in Flag Collection domain in terms of convergence speed, sample efficiency and run time consumption. In our qualitative analysis, we showcased that our approach can generate AMDPs that preserve the topological and reward structure of the underlying MDP.

ACKNOWLEDGEMENT

This work was partially funded by the Federal Ministry of Education and Research (BMBF), Germany under the project LeibnizKILabor (grant no. 01DD20003).

REFERENCES

[1] David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, pages 2915–2923. PMLR, 2016.

[2] Richard Bellman. *Dynamic Programming*. Princeton University Press, USA, 2010.

[3] John Burden and Daniel Kudenko. Using uniform state abstractions for reward shaping with reinforcement learning. In *Workshop on Adaptive Learning Agents (ALA) at the Federated AI Meeting*, volume 18, 2018.

[4] John Burden and Daniel Kudenko. Uniform state abstraction for reinforcement learning. *arXiv preprint arXiv:2004.02919*, 2020.

[5] Kyriakos Efthymiadis and Daniel Kudenko. A comparison of plan-based and abstract mdp reward shaping. *Connection Science*, 26(1):85–99, 2014.

[6] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[7] Marek Grzes and Daniel Kudenko. Plan-based reward shaping for reinforcement learning. In *2008 4th International IEEE Conference Intelligent Systems*, volume 2, pages 10–22. IEEE, 2008.

[8] Ghorban Kheradmandian and Mohammad Rahmati. Automatic abstraction in reinforcement learning using data mining techniques. *Robotics and Autonomous Systems*, 57(11):1119–1128, 2009.

[9] Megha Khosla, Jurek Leonhardt, Wolfgang Nejdl, and Avishek Anand. Node representation learning for directed graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 395–411. Springer, 2019.

[10] Megha Khosla, Vinay Setty, and Avishek Anand. A comparative study for unsupervised network representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[11] Sephora Madjiheurem and Laura Toni. Representation learning on graphs: A reinforcement learning application. *AISTATS*, 2019.

[12] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71, 2004.

[13] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th International Conference on Machine learning*, pages 601–608, 2007.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[16] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[17] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *SIGKDD*, pages 701–710. ACM, 2014.

[18] Balaraman Ravindran and RISE IIL. Hierarchical reinforcement learning using spatio-temporal abstractions and deep neural networks.

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

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

[22] Daniel Kudenko Vikram Waradpande and Megha Khosla. Graph-based state representation for deep reinforcement learning. In *Proceedings of the 16th International Workshop on Mining and Learning with Graphs (MLG)*, 2020.