Sample Efficient Reinforcement Learning Using Graph-Based Memory Reconstruction

Yongxin Kang, Enmin Zhao, Yifan Zang, Lijuan Li, Kai Li, *Member, IEEE*, Pin Tao, *Member, IEEE*, and Junliang Xing, *Senior Member, IEEE*

***Abstract*—Reinforcement learning (RL) algorithms typically require orders of magnitude more interactions than humans to learn effective policies. Research on memory in neuroscience suggests that humans' learning efficiency benefits from associating their experiences and reconstructing potential events. Inspired by this finding, we introduce a human brainlike memory structure for agents and build a general learning framework based on this structure to improve the RL sampling efficiency. Since this framework is similar to the memory reconstruction process in psychology, we name the newly proposed RL framework as graph-based memory reconstruction (GBMR). In particular, GBMR first maintains an attribute graph on the agent's memory and then retrieves its critical nodes to build and update potential paths among these nodes. This novel pipeline drives the RL agent to learn faster with its memory-enhanced value functions and reduces interactions with the environment by reconstructing its valuable paths. Extensive experimental analyses and evaluations in the grid maze and some challenging Atari environments demonstrate GBMRs superiority over traditional RL methods. We will release the source code and trained models to facilitate further studies in this research direction.**

***Impact Statement*—The problem of sampling efficiency has always been an essential issue in reinforcement learning. It is directly related to the interaction cost between an agent and the environment. However, previous approaches pay little attention to the experience storage structure and the utilization mechanism. Our proposed GBMR framework gives substantial innovation for both aspects. The framework is highly adaptable and offers various possible approaches to studying sampling efficiency problems in reinforcement learning methods.**

***Index Terms*—Experience replay (ER), graph model, memory reconstruction, reinforcement learning (RL), sample efficiency.**

Manuscript received 22 November 2022; revised 19 March 2023; accepted 8 April 2023. Date of publication 20 April 2023; date of current version 12 February 2024. This work was supported in part by the National Key R&D Program of China under Grant 2022ZD0116401, in part by the Natural Science Foundation of China under Grant 62076238, Grant 62222606, and Grant 61902402, and in part by the CCF-Tencent Open Fund. This article was recommended for publication by Associate Editor Robert Hunjet upon evaluation of the reviewers' comments. *(Corresponding author: Junliang Xing.)*

Yongxin Kang is with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China, and also with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: kangyongxin2018@ia.ac.cn).

Enmin Zhao and Yifan Zang are with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: zhaoenmin2018@ia.ac.cn; zangyifan2019@ia.ac.cn).

Lijuan Li and Kai Li are with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China (e-mail: lijuan.li@ia.ac.cn; kai.li@ia.ac.cn).

Pin Tao and Junliang Xing are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: taopin@tsinghua.edu.cn; jlxing@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TAI.2023.3268612

I. Introduction

REINFORCEMENT learning (RL) agents have achieved superhuman performances in various game environments, such as Atari 2600 games [1], [2], Go [3], [4], [5], and StarCraft [6]. Despite significant achievements in the last decade, these techniques usually require several orders of magnitude more interactions with the environment than humans to reach an equivalent performance. How can we as humans be so efficient? Researchers have long suggested that explicit, writable memory plays a vital role in the human learning process [7], [8]. Memories allow a system to reason in complex ways over relative information or perform tasks requiring storing and accessing information over longer timescales [9]. Machines and organisms that rapidly encode the experience into memory can later draw on those memories to quickly learn new knowledge from a few examples [10].

In neuroscience research on memory, Richard Semon introduced the term "engram" to describe the neural substrate for storing and recalling memories [11], [12]. Semon proposed that an engram is a subset of cells activated by an experience and that these cells undergo persistent chemical and physical changes. Recent experiments [13], [14], [15] have confirmed Semon's hypothesis. They experimentally showed that a collection of specifically connected engrams distributed across numerous brain regions constitutes memories. Subsequent reactivation of the engrams induces memory retrieval [16]. On the other hand, based on Semon's theory about the existence of memory, Hebb [17] proposed a hypothesis about memory learning and updating. Hebb pointed out the significance of synaptic plasticity (the increase in synaptic strength between neurons) in memory updating. This idea has also been recently reported to be scientifically correct by Tonegawa et al. [18] and Zhang et al. [19]. This memory learning process is called memory reconstruction by Bartlett [20] in social psychology, and humans recall an event or a story in this process. To reconstruct events, we have to pull from our knowledge of similar events to fill in the missing part of memory, enriching our memory [21], [22].

There are also many methods in RL that can be seen as *setting up memory* for the agent, such as storing the original data or the episodic trajectories. Experience replay (ER) methods [1], [2], [23], [24] "memorize" historical data directly and use it in the current environment by sampling. However, the ER method only stores data without considering their association. Model-based RL methods (e.g., [25], [26]) transform history into a model of the environment and then predict upcoming states. Episodic RL

methods (e.g., [27], [28], [29], [30]) store good experiences in a tabular-based nonparametric memory and rapidly latch onto past successful policies when encountering similar states. Although the methods mentioned above try to use experience storage to improve sample efficiency, these research works on the agent's memory have been mostly restricted to their limited storage structure. The lack of a reasonable storage structure leads to the inability to construct an efficient recall process.

In this work, we first introduce Semon's concept of connected engrams in neuroscience into RL to design an agent's memory structure. We imitate the netlike connected engrams in the human brain to establish an attributed graph, which models agents' experience and policy. In particular, we model states in RL as attributed nodes and model actions taken by the agent as the edges in the graph. The state–action values are used as weights on the edges. Agents with attributed graphs make decisions by reading similar scenes from current observations and memories from those scenes. After an episode of interactions, we write these experiences into the graph to update memory. This attributed graph will store states, the relationships among the states, and the agent's policies.

Based on this new memory structure, we proposed a graph-based memory reconstruction (GBMR) framework to mimic Bartlett's reconstruction mechanism of human memory. Following experience accumulation and reconstruction of the brain, we design two corresponding processes, 1) *Interaction* and 2) *Reflection*, for the GBMR framework. The *Interaction* process constructs a memory graph with our proposed structure by interacting with the environment. With this graph in mind, the agent reconstructs and updates memories to learn its policy, which we name the *Reflection* process. In this process, GBMR abstracts key nodes from the memory graph and updates the edge attributes of the reconstructed potential paths among these key nodes. To help the agent update the policy with as few interactions as possible, the *Interaction* process updates the policy through realistic actions, and the *Reflection* process updates the policy by reorganizing some new paths by these past fragments and re-estimating the crucial memory through the diffusion of information on the graph.

We in this work make three main original contributions as follows.

1) To our best knowledge, we propose for the first time that, just as memory structure plays a vital role in the human's learning process, designing appropriate memory structure, and reconstruction schemes also promote efficient learning for the RL agents.
2) We inventively design an attributed graph representation to model the agent's memory in RL, containing state transitions, and agent policy. This nonparametric model elegantly represents the RL agent's learning process and provides a suitable form for memory reconstruction.
3) We present GBMR, a general memory-driven RL framework, to reduce interactions and accelerate convergence by abstracting key nodes and reconstructing valuable paths on the memory graph. GBMR enables agents to converge to the same policy with few interactions.

Based on the earlier contributions, we implement GBMR to drive an RL agent in a range of environments to verify its effectiveness. In particular, we first perform some ablation experiments to validate the rationality of graph memory and the necessity of graph reconstruction. At the same time, the effect of different graph reconstruction implementations in different grid mazes will show the flexibility and generality of GBMR. Afterward, we utilize GBMR with the corresponding observation embedding network to reveal its superiority over the baseline algorithm on some challenging Atari games. In addition to providing essential insights into the GBMR mechanism, our visualization of the intermediate results shows that GBMR holds the potential to move toward explainable RL.

II. Preliminary

In the complex and ever-changing world, human beings are making choices all the time. Rational people seek to respond optimally to incentives within a limited time. In RL problems, the agent constantly interacts with the environment and makes decisions. Existing RL frameworks design corresponding incentives and operating mechanisms, hoping they can think and make decisions like humans.

Traditional RL algorithms typically assume that the RL problem is well modeled by an agent interacting with a finite Markov decision process (MDP) [31]. At each timestamp t, the agent receives a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ following a policy $\pi(a_t|s_t)$, where $\mathcal{S}$ represents the state space and $\mathcal{A}$ is the action space. Then, the agent receives a reward r_t from the reward function $\mathcal{R}(s, a)$ and transfers to the next state s_{t+1}, according to the environment dynamics model where the state transition probability is $\mathcal{P}(s_{t+1}|s_t, a_t)$. In an episodic problem, this process continues until the agent reaches a terminal state. The episodic problem means that the agent's task lasts a finite amount of time, as opposed to the continuous task, which never ends. The agent can obtain a trajectory with finite-length T for each episode. Here, we design a discounted, cumulative return in (1) with the discount factor $\gamma \in (0, 1]$:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}. \tag{1}$$

The state–action value $q_\pi(s, a) = E[R_t|s_t = s, a_t = a]$ is the expected return for selecting an action a in state s following policy π. Here, we denote the optimal policy as π^*, and the corresponding state–action value as q^*. In traditional RL, e.g., Q-learning [32], the state–action value q is updated by

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

where $\alpha \in (0, 1]$ is the step size. It often requires sufficient interactions to update the current q toward optimal $q*$.

The phenomenon that RL algorithms require many rounds of interaction with the environments to obtain satisfactory performance is often described as *sample inefficiency*. This problem is caused by a design defect in RL algorithms. In the traditional RL formulations, the agent usually saves policy only (or values that can deduce policy, e.g., Q-table). Therefore, if agents want to update their policies, they have to do more interactions. However, humans improve their policies through reasoning without interaction. The fundamental distinction is that the human brain

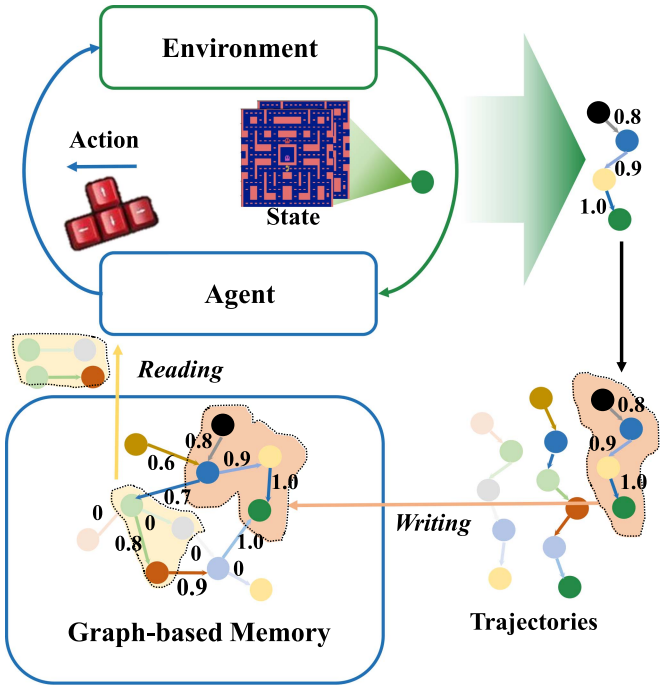


Fig. 1. Graph-based memory for RL and basic operations of memory *Reading* and *Writing*. The nodes with different colors represent different node attributes (i.e., the features of states). Different colored edges represent different edge attributes (i.e., value function of state–action pair). The *Writing* operation is to write the interaction trajectory into the memory graph, and the same node represents the same state. Each *Writing* operation completes the addition of new nodes or the updating of past edge attributes. The *Reading* operation is to get the connected edges (actions) and attributes (state–action values) according to the memory graph and apply them to the *Decision-making* process.

has a corresponding storage structure and access mechanism, which is skilled at remembering events and recalling them to accomplish new tasks. The agent lacks such a memory structure that efficiently stores experiences and reconstructs policies.

III. Graph-Based Memory

Inspired by the connected form of engrams in neuroscience, we introduce an attributed graph to provide a new formulation of the agent's memory, building a fundamental framework for policy updating in Section IV. We establish *Writing*, *Reading*, and *Decision-making* operations on this graph-based memory to imitate human memory's access and retrieval, as shown in Fig. 1. The *Writing* operation embeds the current events into the agent's historical memory, and the *Reading* operation retrieves similar events and decisions from memory based on the current observation. After *Reading*, the agent applies these reading results to a *Decision-making* process, just like humans take actions according to similar situations in memory.

A. Graph Formulation of the Agent's Memory

We build a directed attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{W})$ to model the agent's memory, where each **node** $v_i \in \mathcal{V}$ corresponds to a **state** s_i, **node attribute** $x_i \in \mathcal{X}$ denotes the **feature** $\phi(s_i)$ of state s_i, **edge** $e_{(i,j)} \in \mathcal{E}$ between the nodes v_i and v_j corresponds to an **action** a_i taken in state s_i that causes the transition to s_j, and

TABLE I
Notation Table of RL and Graph-Based RL (GBRL)

	Notation	Meanings
RL	$\mathcal{S}$	state space
	s_t	state at time step t, $s_t \in \mathcal{S}$
	$\mathcal{A}$	action space
	a_t	action at time step t, $a_t \in \mathcal{A}$
	$\pi(a_t \mid s_t)$	policy
	$\mathcal{R}(s,a)$	reward function
	$\mathcal{P}(s_{t+1} \mid s_t, a_t)$	state transition probability
	R_t	return
	$q_\pi(s,a)$	expected return for (s,a) in π
GBRL	$\mathcal{G}$	directed attribute graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{W})$
	$v_i \in \mathcal{V}$	node in graph, which corresponds to s_i
	$x_i \in \mathcal{X}$	attribute of v_i denotes the feature $\phi(s_i)$ of s_i
	$e_{(i,j)} \in \mathcal{E}$	edge between node v_i and v_j
	$w_{(i,j)} \in \mathcal{W}$	attribute of $e_{(i,j)}$ denotes $q_\pi(s_i, a_i)$
	p	K-steps path $p = \{e^0, \cdots, e^K\}$
	$r(e_{(i,j)})$	reward function in graph
	$\mathcal{G}^*$	graph to deduce optimal policy
	$ne[v_j]$	neighbors of v_j

edge attribute $w_{(i,j)} \in \mathcal{W}$ denotes the characteristic of state–action pair (s_i, a_i, s_j), i.e., the **state–action value** $q_\pi(s_i, a_i)$. A **path** $p = \{e^0, e^1, \ldots, e^K\}$ in the graph corresponds to a policy sampling with K steps. We denote the reward as a function of edge $r(e_{(i,j)})$. The **return** $R_t = \sum_{k=0}^{K} \gamma^k r(e^k)$ evaluates the value of a sampling path. The corresponding state–action value under current $\mathcal{G}$ is $q_{\mathcal{G}}(e_{(i,j)}) = E[\sum_{k=0}^{K} \gamma^k r(e^k)]$. In episodic RL, we deduce the agent's current policy π from the whole graph with edge attributes $\mathcal{W}$. The agent's goal in RL is to get an optimal attributed graph $\mathcal{G}^*$, by which the path with the max cumulative reward can be sampled:

$$p^*[v_i, v_j] = \underset{p_i \in p[v_i, v_j]}{\arg\max} \sum_{e_{(m,n)} \in p_i} r\left(e_{(m,n)}\right). \tag{3}$$

To make the following parts easier to follow, we summarize these denotations in Table I.

B. Memory Writing and Reading

To update and employ the memory graph, we design its basic operations, including the memory *Writing* operation and the memory *Reading* operation. *Writing* is the primary way to grow and update the graph memory along with the agent's exploration process. *Reading* obtains some similar or associated nodes from the graph memory. The read-out nodes will be used in the *Decision-making* process.

1) Writing: Memory *Writing* is executed after we get a trajectory. For each pair (s_i, a, s_j) in the current trajectory T, we transform it into a node–edge–node tuple $(v_i, e_{(i,j)}, v_j)$ in the graph. Then, we check if the edge $e_{(i,j)}$ exists in the memory graph $\mathcal{G}^{t-1}$, where the superscript $(t-1)$ denotes the result of iteration at time step $(t-1)$. If it exists, we update the corresponding node attribute x_i and edge attribute $w_{(i,j)}$ according to the state feature $\phi(s_i)$, $\phi(s_j)$ and reward $r(s_i, a)$

$$w_{(i,j)}^{(t)} \leftarrow w_{(i,j)}^{(t-1)} + \eta \left[r(s_i, a) + \gamma \max_{v_m \in \mathrm{ne}[v_j]} w_{(j,m)}^{(t-1)} - w_{(i,j)}^{(t-1)} \right] \tag{4}$$

where η denotes the learning rate, and $\text{ne}[v_j]$ denotes neighbors of v_j. Equation (4) is a transformation of (2) under the formulation of the graph-based memory. If the edge does not exist, we add a new edge $e_{(i,j)}$ to get $\mathcal{G}^t$ and transform $r(s_i, a)$ into $r(e_{(i,j)})$. Additionally, we overwrite the least recently used items when reaching the memory's maximum capacity.

2) Reading: To make a decision according to experience, the agent reads the node corresponding to the current state and the edges connected to that node from the memory graph. We use a function $f_{\text{read}}(\cdot)$ to represent the reading operation. Typically, it is almost impossible for agents to encounter exactly the same observation as in the past, especially in continuous state space. Therefore, the *Reading* operation resorts to retrieving a set of similar nodes, and weights their values according to the similarity when making decisions.

It takes the current state s_i and the current memory $\mathcal{G}^t$ as input and outputs candidate nodes $\mathcal{N}_{\text{cand}}$ and corresponding weights $\mathcal{P}_{\text{cand}}$

$$(\mathcal{N}_{\text{cand}}, \mathcal{P}_{\text{cand}}) = f_{\text{read}}\left(s_i, \mathcal{G}^t\right). \tag{5}$$

To read similar nodes from the graph, we use different similarity metrics in different tasks. Here, we take the cosine distance between the state feature $\phi(s_i)$ and the node attributes x_j as an example

$$d(i,j) = \cos(\phi(s_i), x_j) = \frac{\phi(s_i) \cdot x_j}{|\phi(s_i)||x_j|}. \tag{6}$$

We find the indices $j_1, \ldots, j_{\text{top}_k}$ corresponding to the top_k largest values of $d(i,j)$. top_k is a hyperparameter that is manually set in different tasks. For $j \in \{j_1, \ldots, j_{\text{top}_k}\}$, the more similar they are, the more weight they will have in later decisions. Thus, the weight vector $\boldsymbol{\omega}$ is computed by

$$\omega_{j_k} = \frac{\exp(d(i,j_k))}{\sum_{m=j_1,\ldots,j_{\text{top}_k}} \exp(d(i,m))}. \tag{7}$$

Candidate nodes and corresponding weights are two outputs of the reading function f_{read}

$$\mathcal{N}_{\text{cand}} = \left\{v_{j_1}, v_{j_2}, \ldots, v_{j_{\text{top}_k}}\right\} \tag{8}$$

$$\mathcal{P}_{\text{cand}} = \left\{\omega_{j_1}, \omega_{j_2}, \ldots, \omega_{j_{top_k}}\right\}. \tag{9}$$

3) Decision-Making: After reading out the nodes that are similar to the current state, the agent will evaluate the value of each action in the current state based on the weights of the edges connected to these nodes. For each action $a \in \mathcal{A}$, we read its edge attribute $w_{(v_{j_k}, v_{m_k})}$ (state–action value) in each candidate state v_{j_k} of the current state s_t, where v_{m_k} denotes the next state caused by the action a in the state v_{j_k}. We weigh them to get the value of a in the current state s_t

$$q(s_t, a) = \sum_{k=1}^{\text{top}_k} \omega_{j_k} w_{(v_{j_k}, v_{m_k})}. \tag{10}$$

The agent executes a ϵ-greedy policy to tradeoff exploration and exploitation. With probability ϵ, the agent picks an action uniformly at random. Otherwise, it picks the action $a_t = \arg\max_a q(s_t, a)$.

Although it is a simple greedy algorithm, the values used for the greedy policy are obtained from multiple similar states, which improves the stability of the agent's decision-making compared with the traditional greedy policy. Furthermore, our decisions relate not only to the current observation but also to the observations associated with it.

Graph-based memory simultaneously models the state and the relationship between them. The *Writing* and *Reading* operations help the agent make decisions when interacting with the environment. However, if we use it directly in the RL task, the policy represented by the edge attributes will only be updated by interacting with the environment, which is inefficient for sampling. Graph-based memory has provided us with a good foundation from the data structure perspective. The next problem is how to make full use of it to complete a more efficient RL process.

IV. Memory Reconstruction

Based on the graph-based memory introduced in the previous section, we further propose a GBMR framework to improve sample efficiency. GBMR imitates the creating process of reconstructive memory Bartlett discovered in psychology, which establishes a *Reflection* process based on interaction. The *Interaction* process proceeds via a memory graph, as mentioned in Section III while the *Reflection* process updates the policy on this graph. By updating the important knowledge (e.g., attribute weights) on the memory graph with existing information, rather than interacting with the environment, our method attempts to reduce unnecessary interactions with the environment and, thus, increase sample efficiency significantly. To this end, we first find critical scenes in the past memory and, then, reassemble existing memory fragments to construct events that are not previously experienced or require extensive exploration to encounter, achieving the inference of the events in the memory graph without interacting with the environment. Specifically, we first get an abstract graph containing important states based on the graph-based memory and, then, reconstruct potential paths between these states to update the original graph and, finally, get the optimal policy through continuous iteration.

A. Memory Abstraction

The agent maintains two graph structures to establish the GBMR mechanism. The first one is the abstract memory graph $\mathcal{H}$, and the other is the original memory graph $\mathcal{G}$. The abstract graph retains a memory blueprint consisting of key nodes. We develop two methods, the key-node-finding (KF) function and the path-finding (PF) function, to obtain the abstract memory graph $\mathcal{H}$.

The key node indexes are obtained by $V_{\text{idx}} = \text{KF}(\mathcal{G}, K)$, where K is the number of nodes we aim to get in the abstract memory $\mathcal{H}$. The node features in $\mathcal{H}$ come from the original memory $\mathcal{G}$, i.e., $\mathcal{X}$. When we find the key-node indexes, we do a mask on $\mathcal{X}_{\mathcal{G}}$, that is, $\mathcal{X}_{\mathcal{H}} = \mathcal{X}_{\mathcal{G}}[V_{\text{idx}}]$.

1) Candidate Methods for $KF(\cdot)$: Benefiting from the graph structure memory, we conveniently measure the characteristics of nodes in the graph. For example, the node's importance is

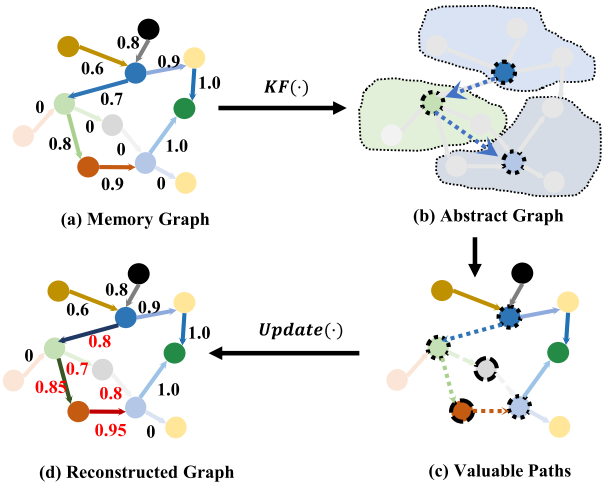


Fig. 2. Memory reconstruction process. The goal of memory reconstruction in graph-based RL is to update the original graph under the guidance of an abstract graph. From an original memory graph (a), we first use a KF function to get the key nodes, which are the basis of the abstract graph (b). From (b) to (c), we construct paths among these key nodes and apply the function $f_{\text{value}}(\cdot)$ to evaluate the value of the paths. After that, the edge attributes on the most valuable paths (c) (as shown in the dotted line in the figure) are updated to get a new graph (d) (the changed attributes are marked in red). By reconstructing the memory graph, we update the agent's policy without interacting with the environment [from (a) to (d)]. (a) Memory graph. (b) Abstract graph. (c) Valuable paths. (d) Reconstructed graph.

measured by its degree. We exploit the sum of in-degree $i(v)$ and out-degree $u(v)$ as the influence of a node v [33]. The more influential a state is, the more critical it should be. Here is the first definition of $\mathrm{KF}(\cdot)$

$$V_{\mathcal{H}} = \arg\max_{v}(i(v) + u(v)). \tag{11}$$

Another way to define $\mathrm{KF}(\cdot)$ is to evaluate the representative ability of the node attributes. We find the corresponding cluster centers by clustering the node attributes. The clustering method can be a simple one, such as K-means, CANOPY [34], or a complex graph clustering method [35]. Another definition of $\mathrm{KF}(\cdot)$ based on K-means is

$$V_{\mathcal{H}} = \text{K-means}\left(\mathcal{X}_{\mathcal{G}}, K\right). \tag{12}$$

There exist many other ways to find the key nodes, such as the nodes with more rewards or the cross-nodes of trajectories with more cumulative rewards. In the ablation experiment (see Section V-A), we compare the effects of different methods in various environments.

By $\mathrm{KF}(\cdot)$ detailed before, we get the most representative or valuable nodes in the agent's memory. So far, the edges in the abstract graph are virtually connected, as shown in Fig. 2(b), and we will give them more practical meanings in the path reconstructing process.

B. Path Reconstruction

The set of paths among key nodes v_i and v_j is noted as $p[v_i, v_j]$, where $v_i, v_j \in \mathcal{V}_{\mathcal{H}}$. We evaluate the most valuable path $p^*[v_i, v_j]$ by function

$$p^*\left[v_i, v_j\right] = \underset{p_i \in p[v_i, v_j]}{\arg\max}\ f_{\text{value}}(p_i). \tag{13}$$

The paths obtained here may not be actual trajectories, and most of them are fragments of trajectories or combinations of fragments. They are inferences about paths that may not occur, which allows the agent to improve the policy without interacting with the environment.

1) Candidate Methods for $f_{value}(\cdot)$: There are usually different methods to evaluate the value of a path $f_{\text{value}}(p)$ according to the tasks agent faced, such as the cumulative rewards, number of nodes, or the node entropy.

Cumulative rewards are the most straightforward way to evaluate the path in the RL problem

$$f_{\text{value}}(p) = \sum_{e_{(i,j)}) \in p} r\left(e_{(i,j)}\right). \tag{14}$$

In some environments with a fixed structure, the shortest path may be another more effective way of evaluating value

$$f_{\text{value}}(p) = 1/\operatorname{length}\left(p\left[v_i, v_j\right]\right). \tag{15}$$

The Dijkstra algorithm [36] can be used to find the shortest path between two key nodes quickly in graph-based memory.

The entropy of nodes may be helpful in an environment with sparse rewards. We can evaluate the value of the path by the entropy of nodes in this path

$$f_{\text{value}}(p) = -\sum_{v_i \in p} P(v_i) \log P(v_i) \tag{16}$$

where $P(v_i)$ denote the probability distribution of node v_i.

We use these most valuable paths as virtual graphs to update our memory without interacting with the environment. In other words, the attributes of the corresponding edge of the path are updated to affect the decision-making process in the next iteration while keeping the node and association unchanged. Because nodes are sampled from the graph memory $\mathcal{G}$, they are all valid states and actions. This result fits our intuition because people will never have episodic memory of things that have not happened.

We update edges directly by

$$w_{(i,j)}^{(t)} \leftarrow w_{(i,j)}^{(t-1)} + \eta\left[r(e_{(i,j)}) + \gamma \max_{v_m \in \text{ne}[v_j]} w_{(j,m)}^{(t-1)} - w_{(i,j)}^{(t-1)}\right]. \tag{17}$$

It is worth noting that the reward here is about an edge $e_{(i,j)}$ in memory instead of a new interaction state s_i. Compared to (4), the policy update in (17) does not consume any interaction resources but has a positive impact on decisions.

C. GBMR-Driven RL

Overall, the GBMR includes two parts, *Interaction* and *Reflection*, as shown in Algorithm 1. *Interaction* refers to the agent obtaining practical trajectory $T = \{s_1, \ldots, s_i, \ldots\}$ from the environment according to the old memory $\mathcal{G}^t$, and *Reflection* reconstructs the memory $\mathcal{G}^{t+1}$ in the agent's mind after obtaining the trajectory T.

Algorithm 1: Graph-Based Memory Reconstruction (GBMR).

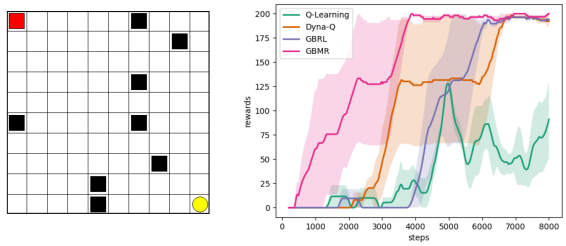
Initialize empty memory graph $\mathcal{G}^0$.
while $t < MaxEpisode$ **do**
  Initialize environment.
  while $i < MaxStep$ **do** ▷ *Interaction*
    Receive state s_i from environment with feature $\phi(s_i)$
    Read candidate states $(\mathcal{N}_{\text{cand}}, \mathcal{P}_{\text{cand}}) = f_{\text{read}}(s_i, \mathcal{G}^t)$
    Calculate $q(s_i, a)$ according to (10)
    $a_t \leftarrow \epsilon - greedy$ exploration by $q(s_i, a)$
    Take action a_t, receive reward r_{t+1}, and s_{i+1}
    $s_i = s_{i+1}$ and $i = i + 1$
  end while
  Write the current record T into $\mathcal{G}^t$. ▷ *Reflection*
  Get abstract graph $\mathcal{H}^t$ from $\mathcal{G}^t$ by $KF(\cdot)$.
  Get the valuable path $p^*[v_i, v_j]$ by $f_{\text{value}}(\cdot)$.
  Update the edges' attributes in $\mathcal{G}^t$ to get $\mathcal{G}^{t+1}$:
    $w_{(i,j)}^{(t)} \leftarrow w_{(i,j)}^{(t-1)} + \eta[r(e_{(i,j)}) + \gamma \max_{v_m \in \text{ne}[v_j]} w_{(j,m)}^{(t-1)} - w_{(i,j)}^{(t-1)}]$.
  $t = t + 1$
end while

1) Interaction: During the *Interaction* process, we first convert the state to a node by reading similar nodes of the current state s_i from the current memory graph $\mathcal{G}^t$ by $f_{\text{read}}(\cdot)$. Subsequently, we use the reading results $(\mathcal{N}_{\text{cand}}, \mathcal{P}_{\text{cand}})$ to calculate the state–action value $q(s_i, a)$ of each $a \in \mathcal{A}$ by (10). Finally, we take action by ϵ-greedy exploration based on these state–action values. The environment will return the next state s_{i+1} and reward r_{i+1}.

2) Reflection: After getting a trajectory from the environment, we write it into the graph memory $\mathcal{G}^t$. To get an abstract graph $\mathcal{H}^t$, we use $KF(\cdot)$ to get key nodes, and $f_{\text{value}}(\cdot)$ to get a valuable path from $\mathcal{G}^t$. Then, we reconstruct the memory graph $\mathcal{G}^t$ under the guidance of the abstract graph $\mathcal{H}^t$ by updating the attributes of corresponding edges. The new memory graph $\mathcal{G}^{t+1}$ will be used in the decision-making process in the subsequent *Interaction*.

A major advantage of memory reconstruction is that there are some reconstructed paths that may have occurred and some that have not actually occurred. Therefore, the policy is updated without interacting with the environment, which improves sample efficiency. Another advantage is that the reconstruction is carried out among key nodes, and the key nodes are representative of their neighbors, which improves the utilization of samples. With a graph-based data structure, the process of memory reconstruction is a depth-first propagation of information, which enables the information of key nodes to rapidly spread to the entire graph. Combined with the breadth-first search in the *Reading* process, the agent quickly updates the policy in memory.

Another important detail is that we use original nodes in $\mathcal{G}$, which ensures that all the reconstruction paths are possible in reality (but not necessarily already happening due to the low number of interactions). This detail ensures that the agent's reasoning is reliable and that the reasoning results can be directly applied to actual decision-making. We will further detail the GBMR framework through the experiment section and verify the effectiveness and efficiency of GBMR through the experiment results.



(a) (b)

Fig. 3. (a) Grid maze is a 10 × 10 maze with random holes and a reward in the bottom right corner. (b) Performance comparison of GBMR, graph-based RL (GBRL), Dyna-Q, and Q-learning.

V. Experimental Evaluations

In this part, we describe the experimental results of the GBMR in different games. First, ablation experiments illustrate the feasibility of graph-based memory and the roles of various candidates in the memory reconstruction process. Then, applications in video games verify that GBMR is a sample-efficient framework. At last, we explain why GBMR is effective by presenting and analyzing intermediate results.

A. Ablation Studies

We provide extensive analyses of the GBMR learning framework via ablation studies from two aspects. On the one hand, to show the effectiveness of graph-based memory in Section III and memory reconstruction in Section IV, we compare GBMR with the basic RL method and RL method with graph-based memory (GBRL). On the other hand, we test different attempts to get key nodes and construct the most valuable path in Section IV and perform ablation experiments on different combinations to illustrate their usability.

1) Environment Settings: We first test our model on the MiniMaze to verify GBMRs effectiveness and understand its different candidate functions. The structure of the environment is constructed by rending several impassable black holes, and all states in the environment have no reward except the yellow circle in the bottom-right corner. If the agent enters the yellow circle, then the Maze Environment will feedback a positive reward, e.g., +200, whereas entering any other state results in a reward of 0. When the agent hits a maze boundary or obstacle, it will stand still at the last state with no reward. The state space is regarded as a limited discrete one without reward. An agent starts from the beginning state in the top-left corner shown in Fig. 3(a). The action space is {"left", "right", "up", "down"} and is limited by the walls and holes. The maximum number of interactions is set as 200 to ensure that this is an episodic RL setting.

TABLE II
NOTATION OF CANDIDATE METHODS IN GBMR

	Notation	Meanings
Key-Node-Finding	K	Clustering algorithm K-means (Eqn. (12))
	D	Degree of nodes (Eqn. (11))
Path-Finding	C	Cumulative reward (Eqn. (14))
	S	Shortest path (Eqn. (15))
	E	Entropy value function (Eqn. (16))

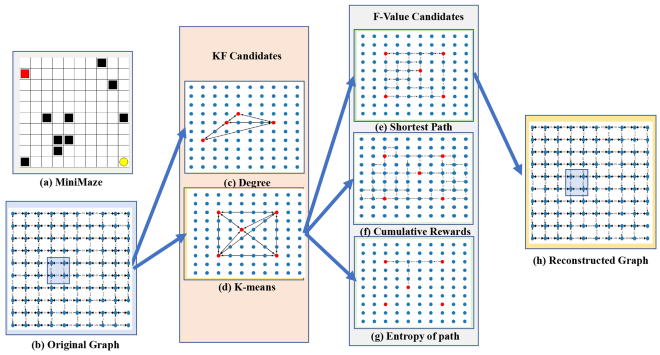


Fig. 4. (a) Grid maze. (b) Agent's memory graph before the reconstruction process. (c) and (d) Two different ways to find the key nodes. The red nodes are the nodes of the abstract graphs. (e)–(g) Three different methods of path reconstruction, and they get different reconstruction paths for the same set of key nodes. The weight on the reconstruction path will be updated without interaction. (h) Reconstructed graph after using one of the six combinations of key-nodes-finding methods and valuable path-finding methods.

Since we take the discrete location as the node feature $\phi(s)$, the read and write operations are precisely addressed without any approximation. In other words, we set top_k in (10) as $\text{top}_k = 1$. This setting allows us to focus more on the role of memory reconstruction, ignoring the effects of other details.

2) Effectiveness of the Graph-Based Memory and Memory Reconstruction: We compare the original Q-learning algorithm with a Dyna-Q [31], "Q-learning + graph memory" (GBRL) and "Q-learning + graph memory + memory reconstruction" (the whole GBMR framework) to verify the effectiveness of these two components. Fig. 3(b) shows their performances.

Compared to the Q-learning algorithm (the green curve), GBRL (the purple curve) gets slightly better experimental results within a similar average number of steps. This result shows that introducing the graph structure into the agent's memory does not influence the agent's policy learning, which validates further ablation study on the memory reconstruction.

Based on our graph structure, we further add the memory abstracting and path reconstruction modules to examine the whole GBMR framework (the pink curve). Compared with the model-based methods (e.g., Dyna-Q), which hold an estimation of the world model (or state transition probability), our GBMR framework reconstructs both the states' relationship in the environment and the policy of agent. In the experiment results, compared with the purple (GBRL) and orange (Dyna-Q) curves, the pink (GBMR) curve shows that the introduction of memory reconstruction dramatically reduces the interactions and improves the sample efficiency.

We use K-means (12) method to get representative key nodes and the shortest-path (15) method to get the reconstructive path in this experiment. The choices of these KF and PF algorithms are discussed in detail next.

3) Studies on KF and PF: There are many options for finding key nodes (11)–(12) and performing path evaluation (14)–(16). The following studies present their influence on different kinds of environments. We use abbreviations in Table II for clear representation. Taking GBMR-KS as an example, it means we use the clustering algorithm K-means (K) to get representative key nodes and the reconstructed path is the shortest path (S) among the paths.

GBMR allows different ways to update the weights in the memory graph. Fig. 4 shows some of the KF and PF candidates. This visualization of the GBMR learning process gives our framework an intuitive understanding. Fig. 4(c) and (d) shows abstract graphs found by two different KF functions from the same original memory graph in Fig. 4(b). The key nodes obtained by degree-of-nodes (D) usually appear at the intersection of many paths, whereas the key nodes obtained by the feature clustering method (K) are approximately uniformly distributed in the state space. Fig. 4(e)–(g) shows paths reconstructed by different $f_{\text{value}}(\cdot)$ from Fig. 4(d). As we can see from the three kinds of reconstruction paths, the F-value by cumulative rewards (C) usually allows more edges to have the opportunity to be updated. Fig. 4(h) shows the reconstructed graph from Fig. 4(e). The highlighted blue box in this reconstructed graph shows the updated edge attributes.

To study the different efficiency of these candidates, we give an example of grid mazes of three basic types. The first one is an empty environment with no structure (unrestricted movement of agent), and the second one is an environment with apparent structure (restricted movement of agent), both of which have no reward in the black hole. We can call them the sparse reward environment. The third one is a grid maze that has the same structure as the second one. The difference is that the black hole has negative rewards.

On the whole, GBMR converges to the optimal policy faster than Q-learning. Different ways to find the key nodes and reconstruct paths will lead to different performances. In the empty grid maze [see Fig. 5(a)], the performance of GBMR-KS and GBMR-KC is slightly better than other combinations, but the difference is not much [see Fig. 5(d)]. In the structured grid maze [see Fig. 5(b)], Q-learning cannot solve the problem in a limited number of steps. However, our GBMR-DS and GBMR-DC perform well [see Fig. 5(e)]. In the dense reward grid maze [see Fig. 5(c)], GBMR-KS and GBMR-DS perform better than others [see Fig. 5(f)].

By comparing the empty grid maze [see Fig. 5(a)] and the structured grid maze [see Fig. 5(b)], results show that for the environment with structure, the key nodes found by degree (GBMR-DS and GBMR-DC) are more conducive to the algorithm to play its advantages. Through the comparison of the sparse reward environment [see Fig. 5(b)] and the dense

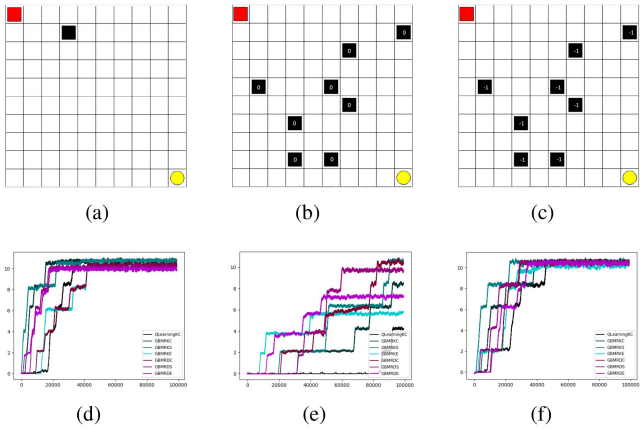


Fig. 5. (a)–(c) Three different grid maze with random holes. (a) Empty maze with only one positive reward at the bottom right corner. (b) Structured maze where the black squares cannot be crossed, so that the empty environment has structural information. However, black squares in both (a) and (b) have no rewards. They are both sparse reward environments. (c) Structured, dense reward environment where the black square has negative rewards. (d)–(f) Average rewards in each step are reported by five random seeds, and we limit steps to 10 000 steps. The performance of six combinations (GBMR-KC, GBMR-KS, GBMR-KE, GBMR-DC, GBMR-DS, GBMR-DE) is compared with the basic Q-learning algorithm. (a) Empty maze. (b) Structured maze. (c) Dense reward. (d) Rewards in (a). (e) Rewards in (b). (f) Rewards in (c).

reward grid maze [see Fig. 5(c)], reconstructing path by shortest path (GBMR-KS and GBMR-DS) and cumulative reward (GBMR-KC and GBMR-DC) both perform well, but in an environment with both dense reward and structured information [see Fig. 5(c)], reconstructing by the shortest path will be better.

B. Comparative Studies in Video Games

To further evaluate the sample efficiency of GBMR and applicability, we conduct experiments on Atari games from the arcade learning environment (ALE [37]), which provides various scenarios to test the RL algorithms under different settings. Some of them are shown in Fig. 7.

In Ms.Pac-Man, rewards are obtained by eating pellets while avoiding ghosts (contact with one causes Ms.Pac-Man to lose a life). Eating one of the unique power pellets turns the ghosts blue for a small duration, allowing them to be eaten for extra rewards. Bonus fruits can be eaten for further rewards, twice per level. When all pellets have been eaten, a new level is started. There are four different maps and seven different fruit types, each with a different reward value.

Here, we take another three examples, Alien, Amidar, and Bankheist. They both involve controlling a character, where there is an easy way to collect small rewards by collecting items of which there are plenty while avoiding enemies. On the other hand, the agent can pick up a particular item making enemies vulnerable, allowing the agent to attack them and get significantly larger rewards than collecting the small rewards.

We cannot take the agent's position as a feature in Atari games. There are two reasons: 1) The state in Atari games is a picture, where the position is not direct information. 2) The games we use have moving enemies. The relative relationships between enemies and agents play a crucial role in scores. It is impossible

TABLE III
HIGHEST SCORES EVER ACHIEVED IN ATARI GAMES (MAX FIVE MILLIONS STEPS)

Algorithm	Alien	MsPacman	Amidar	Bankheist
NEC	1528.0	2446.3	316.5	134.0
GBRL	**1562.0**	2431.0	334.9	**142.4**
GBMR	1336.0	**3180.3**	**379.5**	137.0

to accomplish the task with only the position of the agent as a feature. Therefore, we need a neural network to process video into embedding features for node attributes. The embedding network could be a random projection or parameter network trained with the process of *Interaction*. An episodic RL method, NEC [28], which contains slowly changing state representations and rapidly updated estimates of the value function, is suitable for our framework. We take its embedding network to get state features for our graph memory, which is shown in Fig. 6.

Excluding the way to get state features, the whole framework will be the same as GBMR mentioned before. To test the sample efficiency of our method, we compare GBMR with GBRL (GBMR without memory reconstruction) and NEC [28]. GBRL is used to verify the availability of graph-based memory, whereas NEC is used as a baseline for comparison. Since the embedding part is not the focus of this article, we essentially set up the same embedding network architectures and hyperparameter with NEC (also the same as DQN [1]).

Specifically, we store up to 5×10^5 nodes in the memory graph. While doing *Interaction*, the *Reading* process performs an approximate nearest neighbors algorithm based upon kd-trees [38], where the $\text{top}_k = 50$ in (7), and the writing process update edge weights with learning rate $\eta = 0.01$. In the *Reflection* part, we use cross nodes of trajectories with more cumulative rewards as key nodes and apply the shortest-path (15) to get the valuable path among these nodes.

Fig. 7 compares the training process and results in four video games. The left column is the image extracted from each video, and the right column is the corresponding experimental results. It can be seen from the mean score in the right column that 1) GBRL gets almost the same performance as NEC, and 2) GBMR works better than them.

From the analogous results obtained in GBRL and NEC, we draw similar conclusions as the ablation experiments, that is, the basic idea of introducing an attributed graph into the agent's memory to remember historical events and make decisions is practicable. Based on this result, we observe that GBMR has achieved further improvements. By comparing the red curve with the other two, GBMR gets an identical score in fewer interaction steps in each game. It reveals that we accelerate the convergence speed of the RL learning process by introducing the memory reconstruction process.

In addition to the comparison of the average scores and their ranges in Fig. 7, we compare the highest scores that each algorithm can get in five million steps, as shown in Table III, where the significance of boldface values is the best performance of different algorithms. The highlighted results in Table III show that the best policies ever obtained by GBMR are not necessarily

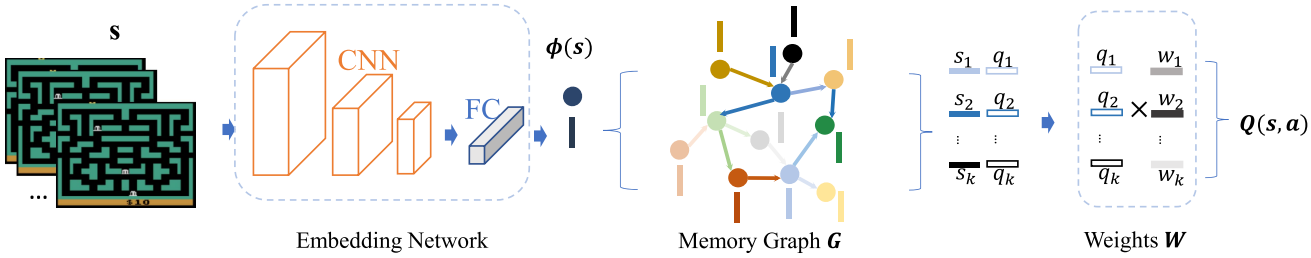


Fig. 6. Embedding network in Atari games. Pictures representing the current state s enter through an embedding network, which contains three convolutional neural networks and a fully connected neural network. After getting embedding $\phi(s)$, we read k most similar state $\{s_1, s_2, \ldots, s_k\}$ and its corresponding q-values $\{q_1, q_2, \ldots, q_k\}$. Then, a trainable weight w is used to obtain the final q-values $Q(s, a)$.

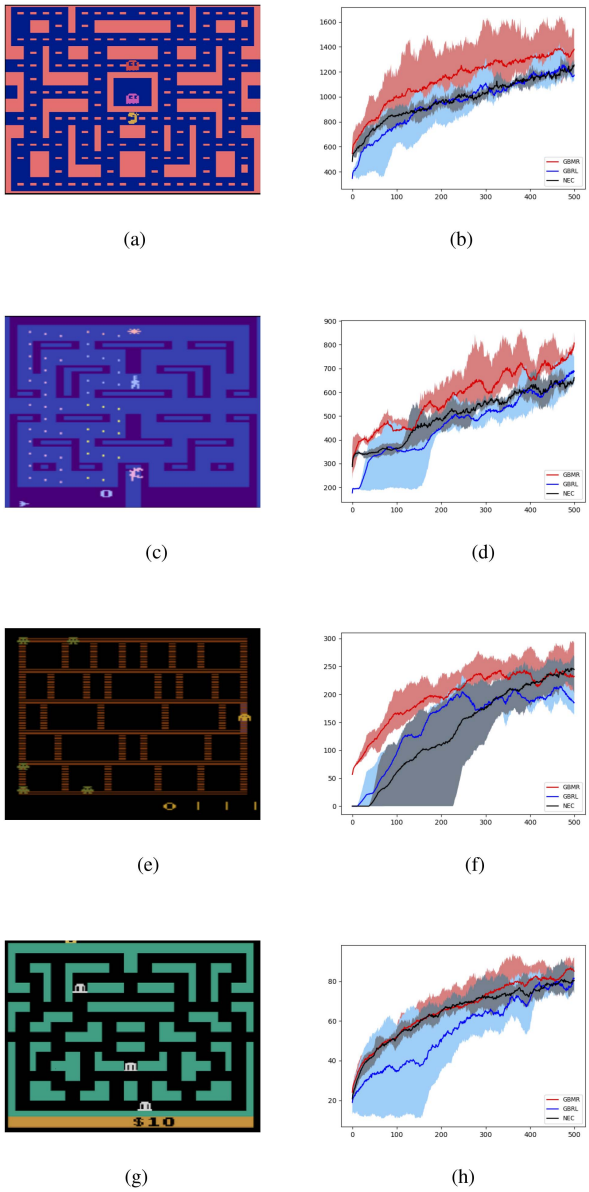


Fig. 7. Comparisons among NEC, GBRL, and GBMR. The x-axis is the number of interaction steps ($\times 1e4$) with the corresponding environment, and the y-axis is the average score in each game. The solid line in the result graph is the mean score of five random experiments for each algorithm, and the shaded area is the range formed by their maximum and minimum rewards. (a) Mspacman. (b) Mspacman. (c) Alien. (d) Alien. (e) Amidar. (f) Amidar. (g) BankHeist. (h) BankHeist.

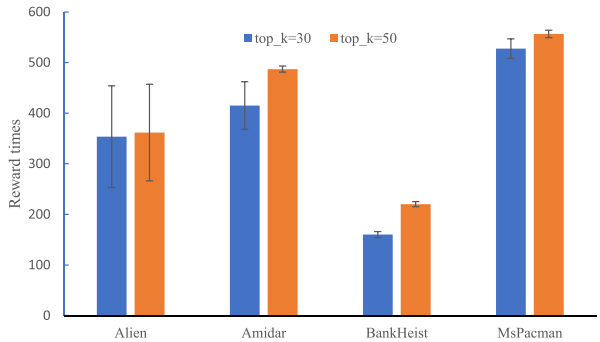


Fig. 8. Effect of the number of nearest neighbors, top_k, on the final results. To facilitate fair comparison among each game, we ignore the value of the reward. The y-axis represents the maximum reward times per episode that an agent get in 1 million steps under $\text{top}_k = 30$ and $\text{top}_k = 50$.

better than those by GBRL. However, combined with the results in Fig. 7, we can see that the average policy of GBMR is more stable than that of GBRL, which further indicates that GBMR stabilizes the distribution of policies through the process of reflection.

To differentiate our GBMR from model-based approaches, we pull the results of these four games mentioned above from SimPLe [25] (which is reported in four million steps), 616.9 (Alien), 1480.0 (MsPacman), 74.3 (Amidar), and 34.2 (Bankheist). The model-based method has some improvement over the traditional model-free method, but the progress of our approach is more pronounced. Another related work is TPG [39]. TPG reported their results in 50 million steps as 3382.7 (Alien), 5156.0 (MsPacman), 389.4 (Amidar), and 1051.0 (Bankheist). It achieved this with 50 million steps while we achieved comparable results using only five million training steps.

We also compare the effect of varying top_k on four games in Fig. 8. In the results of the experiments, we noticed that agents yield a higher score on these Atari games as the number of neighbors grows. At the same time, the variance of multiple results also decreases with the increase of the number of neighbors, indicating that more neighbors bring out more stable performance for GBMR. However, more neighbors will cost more computing resources. Considering the balance of computing resources and experimental effects, we chose $\text{top}_k = 50$ when reporting the results in Fig. 7.

Our framework is more suitable for tasks with obvious structure information and particular meanings of key nodes. However,

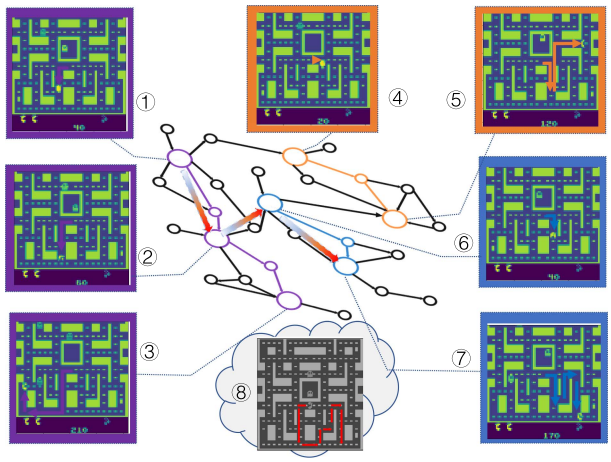


Fig. 9. Reconstruction process in Ms.Pac-Man. We first do key node finding to get key nodes shown in ①–⑦. When we reconstruct paths between the key nodes, ⑧ shows a path that may exist in the process of reflection but has not yet happened in actual interaction. By updating the edge weight on this kind of path, we are more likely to get a more reasonable policy with less interaction.

the extraction of key nodes and the path reconstruction between key nodes have no practical meaning in environments with no moving track, such as Pong and Bowling, so GBMR does not significantly improve the traditional algorithm's performances in those games.

C. Analysis of Intermediate Results

The demonstration of the intermediate results visualizes the reason why memory reconstruction can get a policy (or memory graph) updates with less interaction. The nodes visualized in the purple background (①,②,③ in Fig. 9), the orange background (④,⑤), and the blue background (⑥, ⑦) are key nodes in three trajectories, respectively. When we do reconstruction in the agent's memory, the path visualized in the red line (⑧ in Fig. 9) will be reconstructed, and the edges' attributes along the path will be updated. It is worth mentioning that this red path may require extensive exploration to encounter. However, GBMR updates this vital path in the graph memory without any interaction with the environment. The updating process is similar to people's reasoning about past events in their memory. People infer the results of various possible combinations of events, not limited to events that have occurred. In RL with GBMR, this process will optimize policy through fewer interactions and improve sample efficiency.

At the same time, the results in Fig. 9 show that graph-based RL combines the parametric method (networks for embedding) and the nonparametric method (graph-based policy updating). The nonparametric part makes the whole process of RL more explicable than just using the parametric part. We can intuitively see the mechanism of RL by visualizing the memory graph, which provides a good reference for the further study of explainable RL.

VI. Related Work and Discussion

Although there is no standard formal description of memory in previous RL, some existing algorithms, such as the ER and episodic RL, introduce historical information to address the horizon limitation of MDP modeling for the agent.

1) Model-Free RL and Model-Based RL: Model-free RL algorithms, such as DQN [1], PPO [40], have been hugely successful with Atari games. However, model-free RL algorithms often require more interactive data than human players. The reason is that human players can easily story their experiences and reuse their policies. World models use VAE to learn a latent representation of observation and use long-short term memory (LSTM) to model the transition process in latent space [25], [26]. They offer a straightforward way to represent an agent's knowledge about the world in a parametric model that makes predictions about the future [26]. These works improve policy by interacting with the world model and applying it to the actual environment. However, the model-based RL will face the problem of inaccurate prediction. Therefore, GBMR does not attempt to predict the environment's future. It only reorganizes experiences, including the states that have been observed and the policies that have been adopted. Then, we update the policy through memory reconstruction on the attributed graph, which covers the longer timescales.

2) ER and Episodic RL: Actor–critic RL algorithms such as A2C/A3C [2] and PPO [40] are known for their sampling inefficiency. The success of DQN [1] and its variations [41], [42] owe much to the usage of ER buffer. Prioritized ER (PER) [23] improves the sample efficiency by prioritizing experiences based on TD-errors. It is a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling, which gives different weights to transition tuples with TD-errors. The replay buffer can be regarded as some form of memory. In another way, Hindsight methods [24], [43], [44] encourage the agent to learn from the states it has encountered. Episodic RL methods, such as [27], [28], [29], store good experiences in a tabular-based nonparametric memory and rapidly latch onto past successful policies when encountering similar states [45], [46], [47], [48]. To address the continuous state problem, EMAC [45] proposed algorithm combines episodic memory with actor–critic architecture by modifying critic's objective and GEM [46] organizes the state–action values of episodic memory in a generalizable manner and supports implicit planning on memorized trajectories. When estimating Q-value, researchers propose an associative memory graph as guidance for a value network with a certain frequency [30]. However, graph memory should not be just an auxiliary value but be constructed for decision-making.

In GBMR, we store the events we have experienced and the policy we have adopted in the form of an attributed graph. The graph structure stores not only the attributes of each state but also stores association relationships, which provides a basis for efficient and diverse reconstruction.

3) Graph-Based Methods and Evolutionary Methods: Since the graph is an expressive data structure, many fields use it as the basis of algorithms. DRL+GNN [49] combines graph neural networks (GNNs) with RL and applies them to routing optimization problems. When faced with a high-dimensional state, it usually faces the problem of slow iteration. GNP [50] and TPG [39] try to solve the problem from the perspective

of an evolutionary algorithm and genetic programming. In this article, we model the RL process as an information propagation in the graph structure. And we hand over the high-dimensional encoding part to the trainable convolutional network. Then, the two form an end-to-end learning structure.

4) Other Perspectives on GBMR: The abstract and reconstruction process of GBMR is not only inspired by the mechanism of human brain memory reconstruction but also takes into account the particularity of RL. We treat the state space that has been explored as a memory graph and RL as a problem of searching the optimal path in the memory graph. The process of a similar node search on the memory graph is to conduct a range of breadth-first searches. The reconstruction of the key node path is essentially the propagation of information, but it is a depth-first propagation along the valuable path. Therefore, GBMR realizes the strategy combination of breadth-first search and depth-first search.

VII. Conclusion and Future Work

This article has argued that memory reconstruction in graph-based memory improves the sample efficiency of RL algorithms. The experiment analyses support the idea that the memory stored in the form of an attributed graph is well integrated with the RL process, and the memory reconstruction improves the sample efficiency while ensuring the effectiveness of the learning results. Humans have the ability to reason based on memory due to their unique brain structure. GBMR has designed a similar storage structure and basic reconstruction framework for agents. We have verified its feasibility and efficiency through experiments. These studies try to provide some understanding of the mechanism of humanlike artificial intelligence in solving hard-exploration problems.

This work permits further improvements in several directions. For example, if we use GNN in the abstract process, our GBMR can thus learn the abstract memory in a data-driven manner. Therefore, exploiting more powerful GNN models (e.g., k-GNN [51]) to improve performance will be interesting future directions.

References

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[3] D. Silver et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] D. Silver et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[5] D. Silver et al., "A general reinforcement learning algorithm that masters chess, Shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[6] O. Vinyals et al., "Grandmaster level in Starcraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[7] H. Eichenbaum et al., *Memory, Amnesia, and the Hippocampal System*. Cambridge, MA, USA: MIT Press, 1993.

[8] C. Wong, "Reconstructive memory for abstract selective recall," *Representative Final Project Archive for Stanford CS379C in Spring 2018*, 2018.

[9] J. Feldman, "The neural binding problem (s)," *Cogn. Neurodynamics*, vol. 7, no. 1, pp. 1–11, 2013.

[10] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," *Psychol. Rev.*, vol. 102, no. 3, 1995, Art. no. 419.

[11] D. L. Schacter, J. E. Eich, and E. Tulving, "Richard Semon's theory of memory," *J. Verbal Learn. Verbal Behav.*, vol. 17, no. 6, pp. 721–743, 1978.

[12] D. L. Schacter, "Stranger behind the engram: Theories of memory and the psychology of science," *Amer. J. Psychol.*, vol. 96, no. 3, 1984, Art. no. 427.

[13] L. A. Denardo et al., "Temporal evolution of cortical ensembles promoting remote memory retrieval," *Nature Neurosci.*, vol. 22, no. 3, pp. 460–469, 2019.

[14] G. Vetere et al., "Chemogenetic interrogation of a brain-wide fear memory network in mice," *Neuron*, vol. 94, no. 2, pp. 363–374, 2017.

[15] D. S. Roy et al., "Brain-wide mapping of contextual fear memory engram ensembles supports the dispersed engram complex hypothesis," 2019, *bioRxiv 668483*.

[16] S. A. Josselyn and S. Tonegawa, "Memory engrams: Recalling the past and imagining the future," *Science*, vol. 367, no. 6473, 2020, Art. no. eaaw4325.

[17] D. Hebb, *The Organization of Behavior*. New York, NY, USA: Psychology Press, 1949.

[18] S. Tonegawa, M. D. Morrissey, and T. Kitamura, "The role of engram cells in the systems consolidation of memory," *Nature Rev. Neurosci.*, vol. 19, no. 8, pp. 485–498, 2018.

[19] X. Zhang, J. Kim, and S. Tonegawa, "Amygdala reward neurons form and store fear extinction memory," *Neuron*, vol. 105, no. 6, pp. 1077–1093, 2020.

[20] F. C. Bartlett, *Remembering: A Study in Experimental and Social Psychology*. Cambridge, U.K.: Cambridge Univ. Press, 1932.

[21] B. Wagoner, *The Constructive Mind: Bartlett's Psychology in Reconstruction*. Cambridge, U.K.: Cambridge Univ. Press, 2017.

[22] B. Wagoner, "Bartlett's concept of schema in reconstruction," *Theory Psychol.*, vol. 23, no. 5, pp. 553–575, 2013.

[23] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–21.

[24] M. Andrychowicz et al., "Hindsight experience replay," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5055–5065.

[25] Ł. Kaiser et al., "Model based reinforcement learning for Atari," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–14.

[26] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Adv. Neural Inf. Process. Syst.*, pp. 1–13, 2018.

[27] C. Blundell, B. Uria, A. Pritzel, Y. Li, and D. Hassabis, "Model-free episodic control," 2016, *arXiv:1606.04460*.

[28] A. Pritzel et al., "Neural episodic control," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2827–2836.

[29] Z. Lin, T. Zhao, G. Yang, and L. Zhang, "Episodic memory deep Q-networks," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 2433–2439.

[30] G. Zhu, Z. Lin, G. Yang, and C. Zhang, "Episodic reinforcement learning with associative memory," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–10.

[31] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[32] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, U.K., 1989.

[33] Y. Kang, E. Zhao, K. Li, and J. Xing, "Exploration via state influence modeling," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 8047–8054.

[34] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proc. 6th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2000, pp. 169–178.

[35] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3734–3743.

[36] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[37] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013.

[38] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[39] S. Kelly and M. I. Heywood, "Emergent solutions to high-dimensional multitask reinforcement learning," *Evol. Comput.*, vol. 26, no. 3, pp. 347–380, 2018.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[41] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[42] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[43] R. Zhao and V. Tresp, "Energy-based hindsight experience prioritization," in *Proc. Conf. Robot Learn.*, 2018, pp. 113–122.
[44] Z. Ren, K. Dong, Y. Zhou, Q. Liu, and J. Peng, "Exploration via hindsight goal generation," in *Adv. Neural Inf. Process. Syst.*, 2019, pp. 1–12.
[45] I. Kuznetsov and A. Filchenkov, "Solving continuous control with episodic memory," in *Proc. Int. Joint Conf. Artif. Intell.*, 2021, pp. 2651–2657.
[46] H. Hu, J. Ye, G. Zhu, Z. Ren, and C. Zhang, "Generalizable episodic memory for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 4380–4390.
[47] L. Bärmann, F. Peller-Konrad, S. Constantin, T. Asfour, and A. Waibel, "Deep episodic memory for verbalization of robot experience," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5808–5815, Jul. 2021.
[48] X. Ma et al., "Offline reinforcement learning with value-based episodic memory," in *Proc. Int. Conf. Learn. Representations*, 2022, pp. 1–11.
[49] P. Almasan, J. Suárez-Varela, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case," *Comput. Commun.*, vol. 196, pp. 184–194, 2022.
[50] S. Mabu, K. Hirasawa, and J. Hu, "A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning," *Evol. Comput.*, vol. 15, no. 3, pp. 369–398, 2007.
[51] C. Morris et al., "Weisfeiler and Leman go neural: Higher-order graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4602–4609.

Yongxin Kang received the bachelor's and master's degrees in engineering from Harbin Engineering University, Harbin, China, in 2014 and 2017, respectively. He is currently working toward the Ph.D. degree in computer science and technology with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China.

His research interests include single agent learning and deep reinforcement learning.

Enmin Zhao received the bachelor's degree in engineering from Tsinghua University, Beijing, China, in 2018. He is currently working toward the Ph.D. degree in pattern recognition and intelligent systems with the Institute of Automation, Chinese Academy of Sciences, Beijing.

His research interests include computer poker and deep reinforcement learning.

Yifan Zang received the bachelor's degree in science from Jilin University, Changchun, China, in 2019. He is currently working toward the Ph.D. degree in pattern recognition and intelligent systems with the Institute of Automation, Chinese Academy of Sciences, Beijing, China.

His main research interests include multiagent system and reinforcement learning.

Lijuan Li received the Ph.D. degree in electronic science and technology from Tsinghua University, Beijing, China, in 2018.

She is currently an Associate Professor with the Institute of Automation, Chinese Academy of Sciences, Beijing. Her main research interest includes single- and multiple-agent learning.

Kai Li (Member, IEEE) received the Ph.D. degree in pattern recognition and intelligent system from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2018.

He is currently an Associate Professor with the Institute of Automation, Chinese Academy of Sciences, Beijing. His main research interests include large-scale imperfect-information games and deep multiagent reinforcement learning.

Pin Tao (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1997 and 2002, respectively.

He is currently an Associate Researcher with the Department of Computer Science and Technology, Tsinghua University. He has authored or coauthored more than 80 papers and more than 10 patents. His current research interests mainly include human–AI hybrid intelligence and multimedia-embedded processing.

Junliang Xing (Senior Member, IEEE) received the dual B.S. degree in computer science and mathematics from Xi'an Jiaotong University, Shaanxi, China, in 2007, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2012.

He is currently a Professor with the Department of Computer Science and Technology, Tsinghua University. He has authored or coauthored more than 100 peer-reviewed papers with more than 11 000 citations from Google Scholar. His current research interests mainly include computer gaming problems related to single/multiple-agent learning and human–computer interactive learning.