Highway Graph to Accelerate Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement Learning (RL) algorithms often suffer from low training efficiency. A strategy to mitigate this issue is to incorporate a model-based planning algorithm, such as Monte Carlo Tree Search (MCTS) or Value Iteration (VI), into the environmental model. The major limitation of VI is the need to iterate over a large tensor with the shape $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$, where $\mathcal{S}/\mathcal{A}$ denotes the state/action space. This process iteratively updates the value of the preceding state s_{t-1} based on the state s_t in one step via value propagation. These still lead to intensive computations. We focus on improving the training efficiency of RL algorithms by improving the efficiency of the value learning process. For the deterministic environments with discrete state and action spaces, on the sampled empirical state-transition graph, a non-branching sequence of transitions can directly bring the agent from s_0 to s_T without deviating from intermediate states, which we call a *highway*. On such non-branching highways, the value-updating process can be merged as a one-step process instead of iterating the value step-by-step. Based on this observation, we propose a novel graph structure, named *highway graph*, to model the state transition. Our highway graph compresses the transition model into a concise graph, where edges can represent multiple state transitions to support value propagation across multiple time steps in each iteration. We thus can obtain a more efficient value learning approach by facilitating the VI algorithm on highway graphs. By integrating the highway graph into RL (as a model-based off-policy RL method), the RL training can be remarkably accelerated in the early stages (within 1 million frames). Moreover, a deep neural network-based agent is trained using the highway graph, resulting in better generalization and lower storage costs. Comparison against various baselines on four categories of environments reveals that our method outperforms both representative and novel model-free and model-based RL algorithms, demonstrating 20 to 200 times more efficiency while maintaining an equal or superior expected return, as confirmed by carefully conducted analyses.

1 Introduction

Reinforcement learning (RL) agent training is usually time-consuming, due to the low training efficiency of the value updating process. From the view of the RL agents, the environment to interact can be represented as a state-transition graph, which uses graph structure to describe the transitions between different possible states, following a Markov decision process (MDP). Thus, the process of value updating of RL agents (refer to Figure 1 (1)) iteratively transits the value among known states on the sampled state-transition graph (empirical state-transitions graph), before the end of training. For environments with large state-transition graphs, updating the value of states until convergence is time-consuming since values from all the possible future states of any state can only be propagated back state-by-state. One approach to solve this issue is to incorporate a model-based planning algorithm to search the future state values, such as Monte Carlo Tree Search (MCTS) (Schrittwieser et al., 2020) or Value Iteration (VI) (Tamar et al., 2016). The action selection of MCTS is relatively slow since it plans at decision time which needs a huge number of sampling (Swiechowski et al., 2023). Hence, we are instead intrigued by the potential of the classical VI algorithm to enhance the training efficiency. The major limitation of VI is its necessity to iterate over a huge matrix with the shape $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ ($\mathcal{S}/\mathcal{A}$ denotes the state/action space), during which state s_t at time step t is only propagated back by one or several steps to update the value of the preceding state s_{t-1} per iteration.

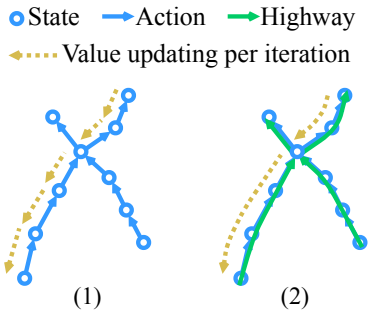


Figure 1: Comparison of the state-transition graph and its corresponding highway graph. An RL agent on the state-transition graph learns/propagates values state-by-state per learning step, and the highway graph propagates values for a stack of states in the highway per learning step. The highway graph is much smaller than the state-transition graph which leads to more efficient value learning. Take the Atari game Star Gunner as an example, the 234,521 states in the sampled empirical state-transition graph are represented by highway graph with only 1,084 (0.5% of its original size) states connected by highways.

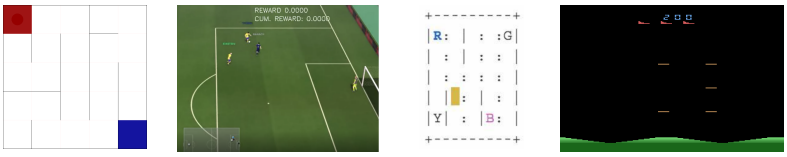
In this paper, we enhanced the VI algorithm via a novel graph structure of the environments to accelerate the value-updating process for RL methods. Our approach takes inspiration from real-world highway systems connecting significant and frequently visited destinations, where drivers need not concern themselves with constant turning decisions and travel swiftly from one location to another without encountering branches at high speeds. Given this, we consider a path without branching within the empirical state-transitions graph as a *highway*, where the value updating for states along the path can be consolidated and executed as a single operation. Such an operation is opposed to updating values incrementally state by state. If values can be efficiently propagated over a long range from any potential future states simultaneously, it significantly enhances the overall training efficiency.

We introduce the concept of a *highway graph* for modeling the environment, shown in Figure 1. For simplicity, we focused on deterministic environments with discrete state and action spaces. The highway graph is constructed from the empirical state-transitions graph (Figure 1 (1)). In the highway graph, nodes are intersection states, and edges are highways. A highway is formed using the sequence of transitions in a not-branching path that only bridges two states without adjacent (Figure 1 (2)). Decisions are only necessary at intersection states, such as when a driver must choose a direction at an intersection, and continue along the highway otherwise.

Due to the highways, the original empirical state-transition graph shrinks into a very small highway graph, with two advantages. First, the values of any state connected by the highways are updated with a single operation, achieving a very high training efficiency. Second, no experience sampling for the value updating is required, and the VI algorithm can be conducted on the entire graph to avoid sub-optimal value learning caused by the sampling. We theoretically demonstrate the convergence to the optimal state value function of the VI algorithm on a highway graph.

As a model-based RL method, the highway graph still needs to be stored in the memory. There is a chance that the highway graph will be too vast to fit into memory. To solve this, we further replace the highway graph with a neural network-based agent obtained by re-parameterizing the value information of the highway graph.

We compare our proposed methods with recent model-based (Schrittwieser et al., 2020) and model-free (Mnih et al., 2015; Kapturowski et al., 2019; Schulman et al., 2017; Espeholt et al., 2018; Mnih et al., 2016; Lin et al., 2018; Hu et al., 2021; Blundell et al., 2016; Pritzel et al., 2017a) methods on four different categories of tasks. Figure 2 shows the speedups of our highway graph approach compared with popular baselines on four different environments. We can see our highway graph achieved on-par or better expected return with a speedup ranging from about 10 to more than 150, which serves as compelling evidence of the feasibility and effectiveness of the conversion to the highway graph. Additionally, the re-parameterized neural network



		Simple maze (5 x 5)	Google research football	Toy text	Atari games
Baselines	Converged time (min)	14.86 ± 2.90 (A3C)	79.81 ± 0.42 (IMPALA)	41.67 ± 0.18 (DQN)	251.26 ± 17.64 (MFEC)
	Eval. return	0.76 ± 0.03	1.01 ± 0.09	-88.18 ± 0.73	764.46 ± 239.06
Highway graph	Converged time (min)	0.15 ± 0.02 (99.0 speedup)	7.33 ± 0.10 (10.9 speedup)	3.82 ± 0.23 (10.9 speedup)	1.51 ± 0.10 (166.4 speedup)
	Eval. return	0.83 ± 0.00	1.03 ± 0.62	5.99 ± 0.74	2201.62 ± 918.52

Figure 2: A comparison of converged time of training (within one million frames) and corresponding speedups by the highway graph compared to baselines. The first row of images are example states from each environment. The results demonstrate a 10 to more than 150 times faster RL agent training with higher evaluation return when adopting the highway graph. All the experiments were performed on the same machine with a 12-core CPU and 128 GB Memory.

as the agent of the highway graph achieved an extra 5% to 20% performance improvements because of the generalization ability of the neural network.

In summary, our paper makes the following key contributions:

- We introduce a graph structure that facilitates more efficient value learning, referred to as the highway graph, which enables long-range value propagation per learning iteration.
- We increase the training efficiency of the classic VI algorithm based on the highway graph. We also show theoretically the convergence to the optimal state value function of the improved VI algorithm on the highway graph.
- We re-parameterize the highway graph into a neural network as the policy with better generalization ability and lower storage cost.
- We empirically test the implemented framework on four categories of learning tasks and compare it to state-of-the-art RL baselines, showing that it drastically reduces training time, achieving a speedup ranging from about 10 to more than 150, while maintaining an equal or superior expected return.

2 Related work

In this section, we will discuss how model-based and model-free RL methods learn the state value, and show the difference between these methods and our highway graph method.

Generally, model-based RL agents utilize the environment model, i.e. the state-transition graph, by two different approaches: value search and graph value iteration. Both of them use the model to conduct value learning. The goal of value learning is to propagate the value from terminal states to the starting states in the state-transition graph.

Value search methods use tree search algorithms (Schrittwieser et al., 2020; Kaiser et al., 2020), such as MCTS, for both agent training and action selection by simulation, which is usually a time-consuming process. The searching for the state value for action selection, i.e. planning, is usually conducted within a range on the state-transition graph, and the values of the leaf states of the searching tree are usually given by the state value function. The planning uses sampling strategies to estimate the value of the current state, which decreases the efficiency of the training, see Figure 3 (3).

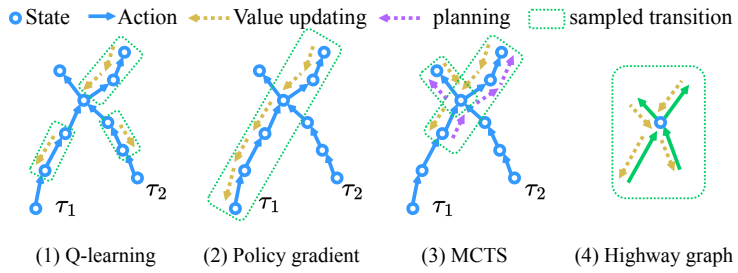


Figure 3: Types of value updating.

Value iteration methods obtain the state values by retrieving from a previously maintained memory of the state-transition model (Tamar et al., 2016). Also, episodic control algorithms (Blundell et al., 2016; Zhu et al., 2020; Pritzel et al., 2017a; Hu et al., 2021; Zhu et al., 2020) are in this category. The episodic control algorithms maintain a memory to make decisions that explicitly store the transition information as a memory. During value updating, the maximum value of either all the future states (Blundell et al., 2016) or states of related trajectories (Zhu et al., 2020) will be used to update the current state's value (Zhu et al., 2020). Such memory can also be parameterized to improve the generalization and performance (Lin et al., 2018; Pritzel et al., 2017b) which gives insight into the parameterized agent trained by memorized data.

Model-free RL algorithms do not explicitly learn the environment model but instead use sampled transitions from the replay to recognize the dynamics of the environment when learning the state value. The value-based and policy-based approaches are typical model-free RL algorithms.

Q-learning methods are typical value-based methods, like DQN (Mnih et al., 2015), which propagate the value of the next few states in the randomly sampled transitions to the current one in each update using n-step temporal difference error, see Figure 3 (1). However, not all valuable transitions are sampled for value updates. To solve this, a prioritized replay buffer was used to sample more useful transitions when training the agent Schaul et al. (2016). R2D2 improved DQN by using a recurrent encoder to obtain better state representations considering all past states in the episode (Kapturowski et al., 2019). The recurrent model considers the causal effect of the historical transition which gives us an insight into designing the encoder for the state.

Policy gradient methods update the value of states directly from the gradient of future value estimation from the sampled trajectories (Schulman et al., 2017; Che et al., 2023; Fatkhullin et al., 2023; Mnih et al., 2016), see Figure 3 (2). The training efficiency can be improved by using distributed systems (Mnih et al., 2016; Espeholt et al., 2018).

During the value update of the model-free agents, inconsistent value updates may occur. The value updating of these methods heavily depends on the sampled transitions or trajectories, and if transitions from s_i with the highest value are not sampled, the updated value of s_i will bias to the real value. Often more sampling is required to overcome this bias, which leads to low training efficiency. In addition, it also makes model-free RL training inefficient since the value of terminal states is propagated to the starting states state-by-state.

Consequently, there are major differences between the highway graph and the above methods, see Figure 3 (4). First, the highway graph itself acts as a memory to store the transiting information, and during value updating, the value will be propagated throughout the entire graph, leading to complete value propagation on each learning iteration. Second, the value updating operation on the highway graph is only for the states of intersection (much fewer than the original states) and highways, instead of all states and transitions, to reduce unnecessary computation and therefore increase training efficiency.

3 Methodology

In this section, we will introduce a novel graph structure to model the MDP of the environment, named highway graph. The highway graph is built upon the empirical state-transition graph to accelerate value

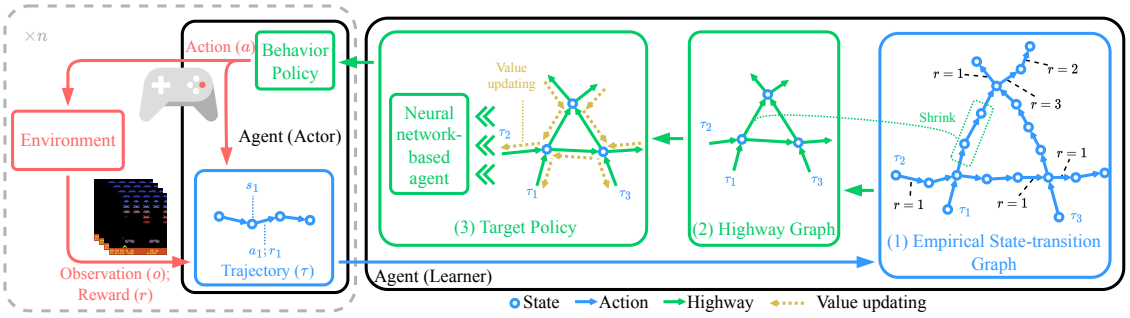


Figure 4: Overall data flow of our highway graph RL method. The actor (on the left) sends the sampled transitions by the behavior policy to the learner (on the right) which (1) constructs the empirical state-transition graph with rewards (in Section 3.1); (2) converts the empirical state-transition graph to the corresponding highway graph (Section 3.2); (3) updates the value of state-actions in the highway graph by an improved value iteration algorithm and re-parameterize the highway graph to a neural network-based agent as the new behavior policy (Section 3.3).

updating, as well as action selection for the environment, see Figure 4 for the overall data flow of our highway graph RL method.

3.1 Markov decision process and empirical state-transition graph

Generally, RL agents try to maximize the expected return when interacting with the environment (Kaelbling et al., 1996). The environment can be described as a Markov decision process (MDP) (Sutton & Barto, 2018) by a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the spaces for state and action, $\mathcal{T}(s', a, s) = p(s'|a, s)$ is the probability of transition into state s' given current state s and action a, $\mathcal{R}(s, a)$ is the immediate reward of action a in current state s, and $0 \leqslant \gamma < 1$ is the discount factor. With known transition and reward function, the state value function of the policy at learning step u can be obtained by value update as follows,

$$V^{(u)}(s) = \max_a \left[\mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{T}(s', a, s) V^{(u-1)}(s') \right], \tag{1}$$

where $V^{(0)}(s)$ can be initialized as 0. For deterministic environments, $\mathcal{T}(s', a, s)$ is constant and $\sum_{s'} \mathcal{T}(s', a, s) V^{(u-1)}(s')$ will be $\mathcal{T}(s', a, s) V^{(u-1)}(s')$.

Empirical transition and reward functions For general RL methods, obtaining the transition and reward functions of the entire MDP is merely impossible. Only empirical state transitions and rewards, sampled from the environments, are used by the RL algorithms. Therefore, we provide a method that efficiently updates state values for the empirical transition and reward functions. The empirical transition and reward functions are defined as

Definition 1 (Empirical Transition and Reward Functions)**.** *Given an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, assume that we sampled k samples (s'_i, a_i, s_i, r_i) with some policy π, then the empirical transition function $\hat{\mathcal{T}}$ is*

$$\hat{\mathcal{T}}(s', a, s) = \frac{\sum_{i=1}^{k} \mathbb{1}(s' = s'_i, a = a_i, s_i = s)}{\sum_{i=1}^{k} \mathbb{1}(a = a_i, s_i = s)}, \tag{2}$$

and many consecutive transitions in the k samples form the trajectory $l \in \mathcal{L}$ which is

$$l = \bigcup_{i=m}^{n} \{(s'_i, a_i, s_i, r_i)\}, \tag{3}$$

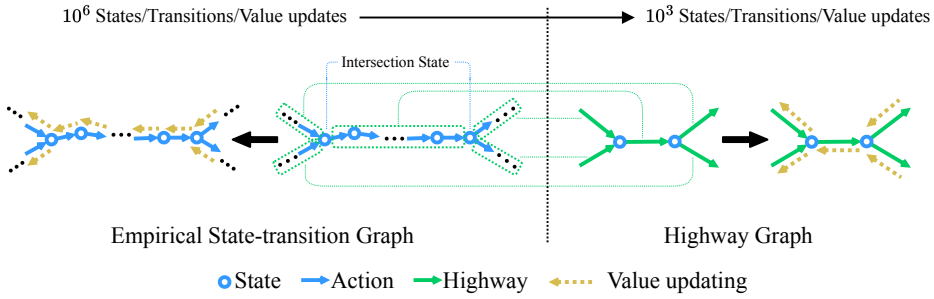


Figure 5: Intuitive idea of the highway graphs. The path without branching in the empirical state-transition graph can be merged as a highway. Value updating on the highway graph will be much less computationally extensive due to the dramatic reduction of the nodes and edges.

where m and n is the beginning and terminal time step of l. The empirical reward function $\hat{\mathcal{R}}$ is

$$\hat{\mathcal{R}}(s,a) = \begin{cases} r_i, & \text{if } \exists i \text{ s.t. } s = s_i, a = a_i, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Empirical State-transition graph The empirical state-transition graph is from the empirical transition and reward functions which can be defined as

Definition 2 (Empirical State-Transition Graph)**.** *Given an empirical transition function $\hat{\mathcal{T}}$ and an empirical reward function $\hat{\mathcal{R}}$, the corresponding empirical state-transition graph is given by $\mathcal{G}_{\hat{\mathcal{T}}} = \left\{\mathcal{S}, \mathcal{E}_{\hat{\mathcal{T}}}\right\}$, where $\mathcal{S}$ is the node set (i.e. each node corresponds to a state), and the edge $(s', a, s, \hat{\mathcal{R}}(s,a)) \in \mathcal{E}_{\hat{\mathcal{T}}}$ if and only if $\hat{\mathcal{T}}(s', a, s) > 0$.*

In a deterministic environment, an action a_i given current state s_i uniquely determines the consequent state s_j. Therefore, $\hat{\mathcal{T}}(s', a, s) \in \{0, 1\}$ in deterministic environments.

3.2 Highway graph

In our work, we convert the empirical state-transition graph into the highway graph with reduced states and transitions, shown in Figure 4 (2). What is a highway graph and how to obtain it are discussed in Section 3.2.1 and Section 3.2.2 respectively.

3.2.1 Highway and highway graph

Naively, the value iteration algorithm requires iterating over a space of the whole empirical state-transition graph ($\mathcal{S} \times \mathcal{A} \times \mathcal{S}$) to reach the converged state value function, leading to a very high computational cost (van den Berg et al., 2012). To tackle this problem, we propose a new graph structure, named highway graph, which is obtained from the original empirical state-transition graph.

Instead of naively iterating over the whole empirical state-transition graph, we seek to merge various value updates to reduce the computations. The intuitive idea of the highway graph is to merge the non-intersection states in the empirical state-transition graph, which converts a path in between intersection states, into a highway (an edge of the highway graph) to reduce the overall graph size, see Figure 5. The intersection states (nodes in the highway graph), which fork, merge, or loop state transitions, remain the same in the highway graph. Within this smaller, yet equivalent, highway graph, we can define a new value update that effectively merges a few value updates under the original empirical state transition graph. This leads to better efficiency since the highway graph is smaller, which means fewer value updates are required.

The edge from the merging, i.e. highway, denoted as $h \in \mathcal{H}$, is defined in Definition 3.

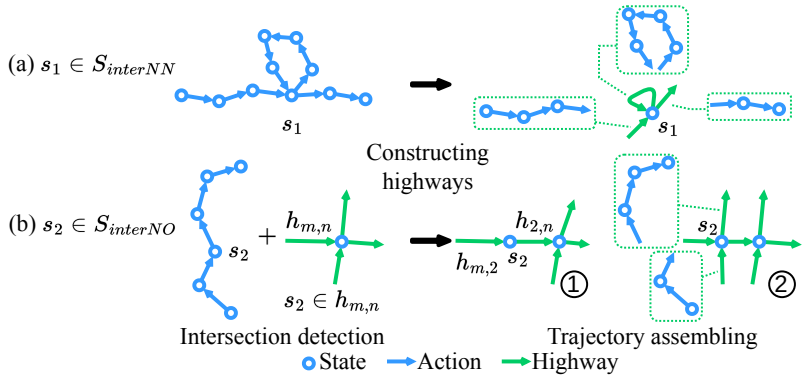


Figure 6: The highway graph construction.

Definition 3 (Highway)**.** *Given an empirical state-transition graph* $\mathcal{G}_{\hat{\mathcal{T}}} = \left\{\mathcal{S}, \mathcal{E}_{\hat{\mathcal{T}}}\right\}$ *that are constructed from a series of samples* (s'_i, a_i, s_i, r_i)*, if for some* $s, s' \in \mathcal{S}$*, there is only one trajectory* l *from* s *to* s' *in the empirical state-transition graph, then* l *between* s *and* s' *will be considered as a highway, denoted as* $h_{s,s'} \in \mathcal{H}$*:*

$$h_{s,s'} = l_{s,s'}. \tag{5}$$

Based on highways, the highway graph with shrunk graph size is defined in Definition 4. The merging to make the highways occurs for all possible nodes between non-branching paths, and only intersection nodes will be excluded. Thus, the highway graph is the minimal graph structure to fulfill identical value updating on the original empirical state-transition graph. The value propagation of state transitions will be replaced by a single operation on highways as much as possible, and the value updating process will consequently speed up.

Definition 4 (The highway graph)**.** *Given an empirical state-transition graph* $\mathcal{G}_{\hat{\mathcal{T}}} = \left\{\mathcal{S}, \mathcal{E}_{\hat{\mathcal{T}}}\right\}$ *that are constructed from a series of samples* (s'_i, a_i, s_i, r_i)*, its corresponding highway graph* $\mathcal{G}_{\mathcal{H}} = \{\mathcal{S}_{inter}, \mathcal{H}\}$ *can be constructed as follows:*

(1) $\mathcal{S}_{inter} = \{s | \exists s'_i, s'_j \in \mathcal{S}, s'_i \neq s'_j \neq s$*, such that there are transitions from* s *to* s'_i *(*$\hat{\mathcal{T}}(s'_i, a, s) > 0$*) and from* s *to* s'_j *(*$\hat{\mathcal{T}}(s'_j, a, s) > 0$*) in* $\mathcal{E}_{\hat{\mathcal{T}}}$*, or there are transitions from* s'_i *to* s *(*$\hat{\mathcal{T}}(s, a, s'_i) > 0$*) and from* s'_j *to* s *(*$\hat{\mathcal{T}}(s, a, s'_j) > 0$*) in* $\mathcal{E}_{\hat{\mathcal{T}}}\}$*;*

(2) $\forall s, s' \in \mathcal{S}_{inter}$*,* $h_{s,s'} \in \mathcal{H}$ *if and only if there exist only one trajectory of* $\mathcal{G}_{\hat{\mathcal{T}}}$ *starts from* s *to* s'*.*

3.2.2 Highway graph construction

Based on this idea, we next give the incremental algorithm to continually construct the highway graph from the sampled trajectory l. The entire algorithm can be divided into two phases: intersection detection and trajectory assembling, shown in Figure 6.

Intersection detection It aims to find all the intersection states and store them into $\mathcal{S}_{inter}$, which is initially an empty set, for the nodes of the highway graph. There are mainly two sources of intersection states. (1) it happens among new states of the newly sampled trajectory by revisiting previous states in the same sampled trajectory, denoted as $\mathcal{S}_{interNN}$; (2) or during the merge between the new trajectories and the existing old highway graph, denoted as $\mathcal{S}_{interNO}$.

The first type of the intersection state is given by Equation 6.

$$\begin{aligned}\mathcal{S}_{interNN} = &\left\{s_t : s_t, s_{t+1} \in l, s_t \in \mathcal{S}_{visited}, s_{t+1} \notin \mathcal{S}_{visited}\right\} \cup \\ &\left\{s_{t+1} : s_t, s_{t+1} \in l, s_t \notin \mathcal{S}_{visited}, s_{t+1} \in \mathcal{S}_{visited}\right\} \cup \\ &\left\{s_t, s_{t+1} : s_t, s_{t+1} \in l, s_t, s_{t+1} \in \mathcal{S}_{visited}, (s_{t+1}, a_t, s_t) \notin \mathcal{T}_{visited}\right\},\end{aligned} \tag{6}$$

Algorithm 1 Highway graph incremental construction

Input : New trajectories $\mathcal{L}$ and existing $\mathcal{G}_{\mathcal{H}}$
Output: The updated highway graph $\mathcal{G}_{\mathcal{H}}$

foreach $l \in \mathcal{L}$ **do**
  Let source state s_i be $s_0 \in l_0 = (s'_0, a_0, s_0, r_0)$
  Find all intersection states $\mathcal{S}_{interNN} \cup \mathcal{S}_{interNO}$ of l using Equation 6 and Equation 7
  foreach *current intersection state* $s_j \in sort\left(\left\{s_j : s_j \in \mathcal{S}_{interNN} \cup \mathcal{S}_{interNO}\right\}\right)$ **do**
    Search sub-trajectory $l_{t:t'}$ that bridge s_i and s_j
    Convert $l_{t:t'}$ into highway $h_{i,j}$ according to Definition 3
    if $s_j \in \mathcal{S}_{inter}$ **then**
      if $h_{i,j} \notin \mathcal{H}$ **then**
        Let $\mathcal{H}$ be $\mathcal{H} \cup h_{i,j}$
    else if $s_j \in h_{m,n}$ *where* $h_{m,n} \in \mathcal{H}$ **then**
      Let $\mathcal{S}_{inter}$ be $\mathcal{S}_{inter} \cup s_j$
      Split $h_{m,n}$ into $h_{m,j}$ and $h_{j,n}$
      Let $\mathcal{H}$ be $\mathcal{H} \cup \left\{h_{m,j}, h_{j,n}\right\} \setminus h_{m,n}$
    else
      Let $\mathcal{S}_{inter}$ be $\mathcal{S}_{inter} \cup s_j$
      Let $\mathcal{H}$ be $\mathcal{H} \cup h_{i,j}$
    Let source state s_i be s_j
  Let $\mathcal{H}$ be $\mathcal{H} \cup h_{j,T}$ where $|l| = T$
return $\mathcal{G}_{\mathcal{H}} = \{\mathcal{S}_{inter}, \mathcal{H}\}$

where $\mathcal{S}_{visited} = \cup_{k=0}^{t-1} s_k$ and $\mathcal{T}_{visited} = \cup_{k=0}^{t-1}(s_{k+1}, a_k, s_k)$ are the visited states and transitions of the current step t in the trajectory l. Sets in different lines of Equation 6 are for the forking, merging, and crossing of the current trajectory respectively.

Another type of the intersection state is given by Equation 7.

$$\begin{aligned}\mathcal{S}_{interNO} = &\left\{s_t : s_t, s_{t+1} \in l, s_t \in \mathcal{G}_{\mathcal{H}}, s_{t+1} \notin \mathcal{G}_{\mathcal{H}}\right\} \cup \\ &\left\{s_{t+1} : s_t, s_{t+1} \in l, s_t \notin \mathcal{G}_{\mathcal{H}}, s_{t+1} \in \mathcal{G}_{\mathcal{H}}\right\} \cup \\ &\left\{s_t, s_{t+1} : s_t, s_{t+1} \in l; s_t, s_{t+1} \in \mathcal{G}_{\mathcal{H}}, \hat{\mathcal{T}}(t+1, a, t) = 0\right\},\end{aligned} \tag{7}$$

where $s_t \in \mathcal{G}_{\mathcal{H}}$ means either $s_t \in \mathcal{S}_{inter}$ or $s_t \in h (h \in \mathcal{H})$, and $\hat{\mathcal{T}}(t+1, a, t)$ is the transition function for any action a from state s_t to s_{t+1} of $\mathcal{G}_{\mathcal{H}}$. The first two sets of Equation 7 indicate whether s_t or s_{t+1} of current trajectory l is in the existing highway graph $\mathcal{G}_{\mathcal{H}}$ (either is the existing intersection or within a highway $h \in \mathcal{H}$) that should be considered as possible new intersection nodes. The last set of Equation 7 is for the new intersections in an existing highway or is the existing intersections in the $\mathcal{G}_{\mathcal{H}}$. Thus, the intersection states will be in Equation 8.

$$\mathcal{S}_{inter} = \mathcal{S}_{inter} \cup \{s_i : s_i \in \mathcal{S}_{interNN} \cup \mathcal{S}_{interNO}\} \tag{8}$$

Trajectory assembling This phase will split the new trajectory l, by the newly found intersection states, and convert them into highways. If the intersection state is on the existing highway, this highway will be split into two highways by this state. After that, the intersection states and highways will be used to assemble the highway graph. Algorithm 1 shows the process of highway graph construction. The function *sort* () sorts the input set of states in ascending order according to the time step of the trajectory.

After the construction of the highway graph, the original MDP can be converted to a new MDP, which we call the highway MDP, defined in Definition 5. The transition and reward function of the highway MDP is

different from the ones in the original MDP. The transition function of highway MDP is only for intersection states in the highway graph, and the reward function is for the aggregated reward of highways instead of individual states. The action in the highway graph for s_i, which transits the agent to $h_{i,j}$, will bring the agent from s_i to s_j.

Definition 5 (Highway MDP)**.** *The highway graph $\mathcal{G}_{\mathcal{H}}$ convert the classic MDP $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ into a highway MDP, denoted as $MDP_{\mathcal{H}} = \left(\mathcal{S}_{inter}, \mathcal{A}, \mathcal{T}_{\mathcal{H}}, \mathcal{R}_{\mathcal{H}}, \gamma\right)$. $\mathcal{R}_{\mathcal{H}} = \mathcal{R}\left(\mathcal{S}_{inter}, \mathcal{A}\right)$ and*

$$\mathcal{T}_{\mathcal{H}}(s_i, a, s_j)_{s_i, s_j \in \mathcal{S}_{inter}} = \begin{cases} 1, & h_{i,j} \in \mathcal{H} \text{ and } a \text{ is the first action of } h_{i,j}, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

3.3 Value updating and target policy

During value updating, we introduce the graph Bellman operator as the planning module to iterate values on the highway graph. After that, we discuss how highway graphs help to speed up value updating and act as the policy afterward to tackle the task in the environment, as shown in Figure 4 (3).

3.3.1 Value updating on the highway graph

Once the highway graph and its highway MDP are obtained, a more efficient yet accurate value update of all states can be performed based on value iteration.

Highway MDP shares the same properties as the classic MDP, and the state values during updating are still in a complete metric space shown by Lemma 1, and RL algorithms suitable for MDP can be applied to Highway MDP as well. Lemma 1, associated with Lemma 2, will be used to show the convergence of value updating on the highway, shown in Proposition 1.

Lemma 1. *Denote the state value vector $v = \left[\hat{V}(s_1), \hat{V}(s_2), \cdots, \hat{V}(s_i), \cdots\right]^{\intercal} \in \mathcal{V}$, where $\mathcal{V} \in \mathbb{R}^{|\mathcal{S}_{inter}|}$, $\hat{V}$ is an estimate of state value function V, and $s_i \in \mathcal{S}_{inter}$ is from Highway MDP, and define a metric $d(w, v) = \|w - v\|_{\infty}$ $(w, v \in \mathcal{V})$, then $(\mathcal{V}, d)$ is a complete metric space.*

Bellman equation is widely used by temporal-difference learning (Sutton & Barto, 2018). Given the currently sampled empirical transitions from the environment, states' values can be propagated within several time steps. However, this update process only considers the future value along with one trajectory per iteration and leads to non-optimal value updates if the best transitions are not sampled for the training. To avoid this issue, we give the graph Bellman operator to update the value of states in a value iteration scheme on the highway graph, considering all the known future states for accurate value updating, defined in Definition 6.

Definition 6 (Graph Bellman operator)**.** *Denote Γ_i^1 is the 1-order neighbors of s_i on the highway graph $\mathcal{G}_{\mathcal{H}}$, i.e. $h_{i,j} \in \mathcal{H}$ for $s_j \in \Gamma_i^1$, and a max pooling function Φ to retrieve the effective value propagated from the neighbor. Given the complete metric space $(\mathcal{V}, d)$, the graph-based Bellman operator is $G(V(\mathcal{S}_{inter})) : \mathcal{V} \to \mathcal{V}$, and G_i is*

$$G_i(V(s_i)) = \Phi_{s_j \in \Gamma_i^1}\left(\mathcal{R}(s_i, a) + \gamma V(s_j)\right), \tag{10}$$

where $s_i \in \mathcal{S}_{inter}$.

Graph Bellman operator is still a contracting mapping given in Lemma 2.

Lemma 2. *Graph Bellman operator G is a max-norm γ-contraction mapping.*

According to the graph Bellman operator, the u-th value updating on the highway graph to obtain the state value and state-action value functions will correspondingly be

$$V_{\mathcal{H}}^{(u)}(s_i) = \Phi_{s_j \in \Gamma_i^1}(V_{\mathcal{H}}^{(u-1)}(s_j) \cdot \gamma^{|h_{i,j}|} + r_{h_{i,j}}), \tag{11}$$

$$Q_{\mathcal{H}}^{(u)}(s_i, a_j) = V_{\mathcal{H}}^{(u-1)}(s_j) \cdot \gamma^{|h_{i,j}|} + r_{h_{i,j}}, \tag{12}$$

Algorithm 2 Value updating on highway graph

Input : Highway graph $\mathcal{G}_h = \{\mathcal{S}_{inter}, \mathcal{H}\}$, max learning iteration U, threshold of changes δ
Output: Highway graph with estimated state values $\mathcal{G}'_{\mathcal{H}}$

Let u be 0
while $u < U$ **do**
  Update state value function $V_{\mathcal{H}}^{(u)}(s_i)$ using Equation 11
  Update state-action value function $Q_{\mathcal{H}}^{(u)}(s_i, a_j)$ using Equation 12
  Set u be $u + 1$
  if $\sum_{s_i \in \mathcal{S}_{inter}} \sum_{a_j \in \mathcal{A}} \Delta Q_{\mathcal{H}}^{(u)}(s_i, a_j) < \delta$ **then**
    break

where $s_i, s_j \in \mathcal{S}_{inter}$, and a_j is the action to s_j from s_i. $V_{\mathcal{H}}^{(0)}(s_i)$ and $Q_{\mathcal{H}}^{(0)}(s_i, a_j)$ are initialized with zeros. The reward of the highway $r_{h_{i,j}}$ is

$$r_{h_{i,j}} = \sum_{t=1}^{|h_{i,j}|} r_t \cdot \gamma^t, \tag{13}$$

where r_t is the reward of the t-th state s_t in the highway. The value updating iterations of the empirical state-transition graph in Figure 4 (1) are shown in Figure 7, and the value of states are obtained by Algorithm 2. The process calculates the value of both intersection states and highways for U (less than ∞) rounds until the state value function reaches optimal.

Next, we analyze the convergence of highway graph value updating in Proposition 1, and we further give the speed advantages of value updating using the highway graph.

Proposition 1 (Convergence of value updating on highway graphs.)**.** *Let $(\mathcal{V}, d)$ be a complete metric space of the $MDP_{\mathcal{H}} = \left(\mathcal{S}_{inter}, \mathcal{A}_{\mathcal{H}}, \mathcal{T}_{\mathcal{H}}, \mathcal{R}_{\mathcal{H}}, \gamma\right)$, sufficient application of the graph Bellman operator G for value updating on highway graph, according to Equation 11, converges to a unique optimal state value function $V_{\mathcal{H}}^*$.*

Remark 1. *The advantage of the highway graph is that it uses a much smaller highway graph to replace the original empirical state-transition graph. This replacement leads to high computational speedup. The time complexity of value iteration on the original empirical state-transition graph is $\mathcal{O}\left(|\mathcal{S}| \cdot |\mathcal{A}| \cdot |\mathcal{S}|\right)$ where $|\mathcal{S}|$ and $|\mathcal{A}|$ are the numbers of states and actions of the empirical state-transition graph respectively. Highways merge trajectories without branching, and the number of states is reduced to $z = \frac{|\mathcal{S}_{inter}|}{|\mathcal{S}|}$. Consequently, the time complexity of value updating on the highway graph is $\mathcal{O}\left(|\mathcal{S}_{inter}| \cdot |\mathcal{A}| \cdot |\mathcal{S}_{inter}|\right)$, which is $\frac{1}{z^2}$ times smaller than that of vanilla value iteration algorithm.*

3.3.2 Highway graph as the policy

Once the values of states are obtained, the highway graph will be used as the policy. An example of action selection by the highway graph is also shown in Figure 7. The selected action for the state may change during the value updating until the highway graph with optimal state values is obtained. The action will be accordingly generated by the policy in Equation 14 to interact with the environment.

$$a_{s_i} = \begin{cases} \arg\max_a Q(s_i, a) & s_i \in \mathcal{S}_{inter} \\ a_{h_{i,j}} & s_i \in h_{i,j} \\ \text{random}(s_i) & s_i \notin \mathcal{G}_{\mathcal{H}} \end{cases}, \tag{14}$$

where $a_{h_{i,j}}$ is the action recorded in the highway $h_{i,j}$, and random(s_*) generate the action a for s_* by uniformed random sampling from the valid actions $\mathcal{A}$.

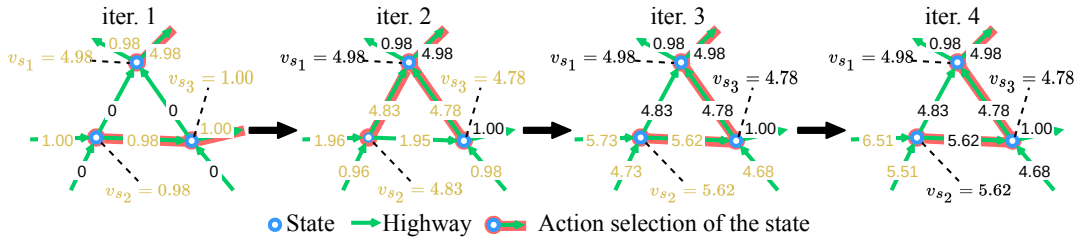


Figure 7: Value updating and action selection of corresponding highway graph of the empirical state-transition graph in Figure 4 (1). The discount factor $\gamma = 0.99$. The updated values of states and highways are in yellow in each learning iteration. The selected actions of intersection states are changing according to the latest learned value.

3.3.3 Highway graph re-parameterization

Despite the compact nature of value updating, an important property of the highway graph is the increasing graph size during construction from newly sampled empirical transitions. Thus, it is required to use a neural network-based agent, derived from the highway graph, to interact with the environment. Ideally, a neural network-based agent can inherit useful state-action value information from the highway graph, and maintain the generalization ability of neural networks. To this end, we re-parameterize the state-action value of the highway graph to obtain a neural network-based agent to (1) reduce the storage cost for the growing highway graph; and (2) generalize the highway graph to more unknown states. By doing this the highway graph can be considered as a jump-starter for both model-free and model-based RL agent training at the early training stage, by providing the state transitions and its values.

Take the model-free Q-learning algorithm as an example, the highway graph can be re-parameterized as the initial state-action value function $Q_\theta(s_i, a)$ before continuing improvement using the Q-learning. A simple MSE loss between $Q_{\mathcal{H}}(s, a)$ and $Q_\theta(s, a)$ can be used during re-parameterization. Denote Q_θ as the trained neural network-based agent utilizing the highway graph. The action a_{s_i} generated by Q_θ for the s_i will be

$$a_{s_i} = \arg\max_a Q_\theta(s_i, a). \tag{15}$$

4 Experiments

In this section, we first evaluate the expected return (accumulated rewards with discount) versus frames and computed time respectively on different categories of environments to answer the following questions: (1) is our proposed highway graph RL method suitable for both simple and complex environments? (2) can our method converge with less sampled experience/compute time to a policy yielding higher returns compared to other baselines? Next, we analyze the functionality and behavior of our method by investigating: (3) evidence of the structural advantage of the highway graph; (4) effectiveness and efficiency details of value updating on the highway graph; (5) the performance of the re-parameterized neural network-based agent.

Environments To better show the training efficiency advantages of our highway graph RL method, we only use one million frames of interaction from different types of Environments. Different random seeds are used to guarantee the deterministic assumption of the environments. Types of environments are listed below.

- Simple Maze [1]: a simple maze environment with customizable sizes.
- Toy Text (Towers et al., 2023): a tiny and simple game set, with small discrete state and action spaces, including FrozenLake, Taxi, CliffWalking, and Blackjack.

[1] https://github.com/MattChanTK/gym-maze

Table 1: The HG behavior on The Simple Maze. We evaluate the convergence to the optimal state value function of value updating on the highway graph on three maze environments of varying sizes. We show the total reward, average discount return, frames, and time taken until convergence, as well as the learning iterations for the convergence.

	Total Reward	Discounted avg. return	Frames used (M)	Wall clock time (min)	Converged learning iteration(s)
3x3	0.97±0.01	0.93±0.01	0.07±0.01	0.03±0.01	1
5x5	0.95±0.01	0.82±0.02	0.09±0.01	0.12±0.02	1
15x15	0.96±0.01	0.37±0.05	0.43±0.02	1.01±0.04	1

- Google Research Football (GRF) (Kurach et al., 2020): a physical-based football simulator.
- Atari learning environment (Bellemare et al., 2013): a simulator for Atari 2600 console games. These games are different, interesting, and designed to be a challenge for reinforcement learning.

Baselines We compare our highway graph RL method (HG) with different baselines for different environments based on difficulty and complexity. For Simple Maze and Toy Text, our method is compared with model-free Q-learning methods DQN (Mnih et al., 2015) and policy gradient method PPO (Schulman et al., 2017), as well as model-based method MuZero (Schrittwieser et al., 2020) to show the way different categories of RL methods update the state value (as shown in Figure 3) and the advantages of the highway graph. For more difficult and computationally demanding GRF environments, our method is compared with the high throughput Q-learning method R2D2 (Kapturowski et al., 2019) and policy gradient method IMPALA (Espeholt et al., 2018) to show the efficiency of value updating on the highway graph. The Atari games are important image-based environments for RL algorithm evaluation due to their versatility, so we take both classic methods including DQN, PPO, A3C (Mnih et al., 2016), and state-of-the-art episodic-control-based methods including EMDQN (Lin et al., 2018), GEM (Hu et al., 2021), MFEC (Blundell et al., 2016), NEC (Pritzel et al., 2017a) as the baselines for a more comprehensive comparison. The reason we include the episodic-control-based methods is that these methods also use a memory module of transition to speed up the value iteration which is similar to the highway graph RL method. In addition, we use RLlib (Liang et al., 2018) implementations for DQN, PPO, A3C, R2D2, IMPALA, and MuZero. Other baselines including NEC, MFEC, EMDQN, and GEM are obtained from its official repository. The network architectures and hyperparameters of different baselines are given in the Appendix.

Experiment setup Our method also adopts an actor-learner training scheme similar to SEED RL (Espeholt et al., 2020). The learner constructs the highway graph from the state transitions obtained by training actors interacting with the environment in parallel. Next, the evaluation actor uses the constructed highway graph to interact with the environment for evaluation. All the experiments were running in the Docker container with identical system resources including 8 CPU cores with 128 GB RAM, and an NVIDIA RTX 3090Ti GPU with 24 GB VRAM. The reported result of each experiment is from the averaged results among 10 different runs. To review and reproduce the code of our highway graph RL method, please refer to the anonymized repository at https://anonymous.4open.science/r/highwayRL-51D8.

4.1 Comparison with baselines

4.1.1 Simple Maze

The Simple Maze environment generates a grid-like maze whose width and length can be specified. A sequence of adjacent grids not separated by a line constitutes a path, and the lines between grids are the walls that the agent cannot get through. The state of this environment at each step will be the current agent location, i.e. 2D coordinates, in the maze. The agent can move within the maze with up, down, left, and right actions. The RL agent needs to find a path from the blue/start point in the upper left to the red/terminate point in the lower right. Once the path is found, a reward of +1.0 can be obtained. A very

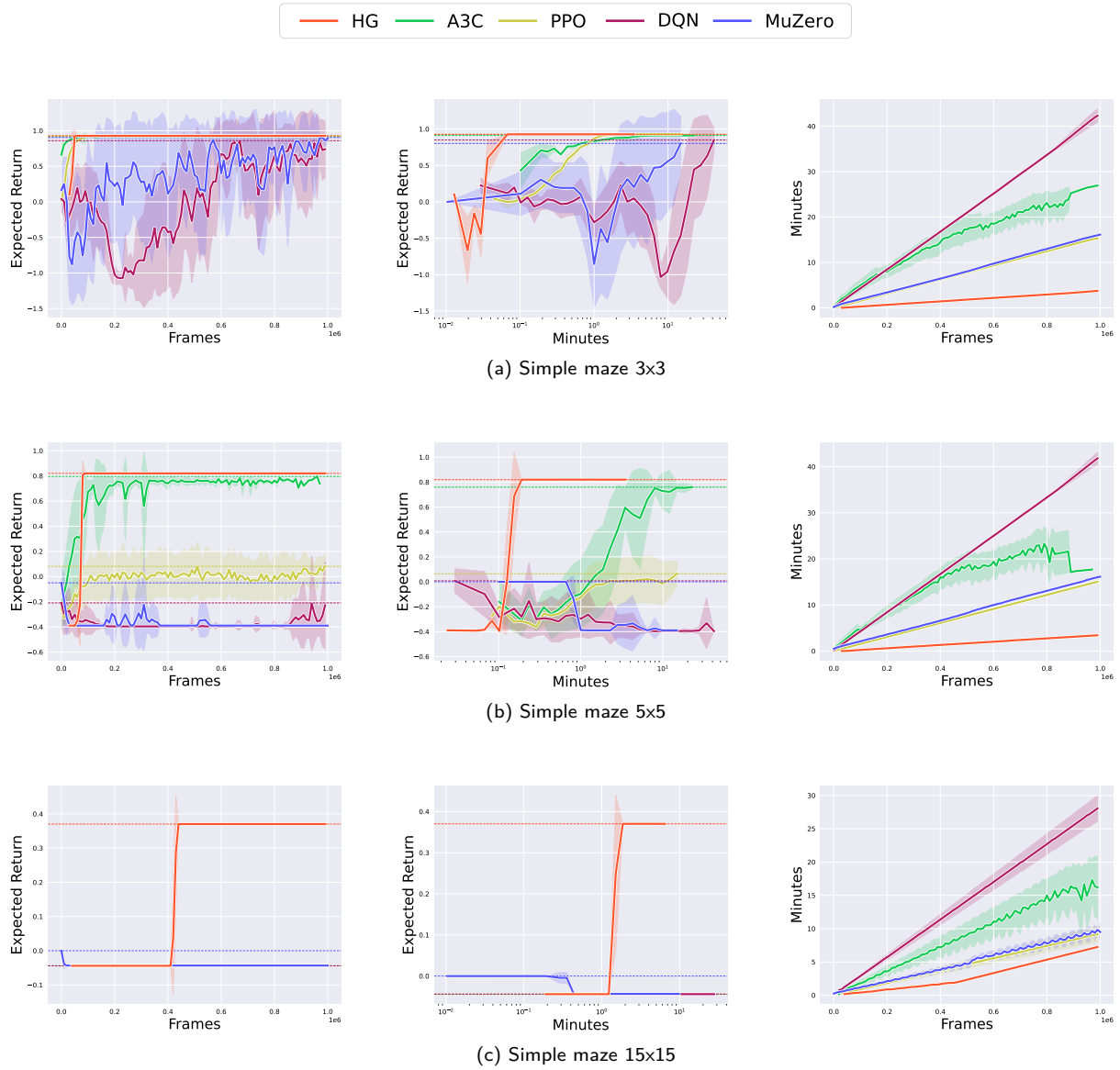


Figure 8: Expected return and efficiency comparison on 3x3, 5x5, and 15x15 Simple Maze environments. The lines and shaded areas are for the average values and the corresponding range ($\pm$) of standard deviations, and dashed horizontal lines show the asymptotic performance. Every row records the training curves over minutes and frames of the same experiments. The first and rest columns show the improved returns and training efficiency respectively.

small penalty ($1/numberofgrids$) will be applied for each move. The maze environments are configured in different sizes: 3x3, 5x5, 15x15.

The behavior of HG on different mazes is shown in Table 1. Although the time used varies, the optimal path to the desired location was found in all environments by the agent after a single iteration of value updating, and the training converged to the optimal policy within a minute. The state space of this kind of environment is relatively tiny (i.e. less than hundreds of states), so the highway can be constructed with no more than half a million frames and fulfill value iteration entirely on the GPU in parallel. The value iteration on the whole graph, instead of sampling part of the graph, makes sure the optimal value will be propagated accurately to all states after each value learning iteration.

Table 2: The HG behavior on Toy Text. We evaluate the convergence of value updating on the highway graph on three Toy Text environments to show the total reward, average discount return, frames, and time taken until convergence, as well as the learning iterations for the convergence.

	Total Reward	Discounted avg. return	Frames used (M)	Wall clock time (min)	Converged learning iteration(s)
Taxi	7.54±6.33	5.12±5.99	0.11±0.01	0.24±0.01	1
CliffWalking	-13.00±0.01	-12.88±0.01	0.10±0.01	0.30±0.02	1

Next, we compare the expected return of our method and other classic RL baselines, shown in Figure 8. The highway graph RL method is 20 to more than 1000 times faster than other baselines to solve the Simple Maze 3x3, achieves the best expected return on Simple Maze 5x5, and is the only method to solve the Simple Maze 15x15 within one million frames. Both the model-free Q-learning-based DQN and model-based value searching method MuZero exhibit instability and slow value learning, which cannot find a path to the desired location in Simple Maze 5x5 and 15x15 by using such a small amount of frames. This is because the sampled transitions may not always be in the optimal path when updating state values, preventing the optimal values from being propagated. This can be correct with much more sampling, but if the amount of sampled transitions is small, it is hard to update by the optimal value for states. Model-free policy gradient methods PPO and A3C outperformed DQN and MuZero in Simple Maze 3x3 and 5x5 with much fewer frames and time. However, PPO and A3C did not find the optimal path of Simple Maze 5x5 and 15x15 within one million frames. The reason is that the trajectories, used to update the state values, to the desired location sampled by PPO and A3C are not enough, and there is still improving space to obtain more precise state values. See Section 4.3 for more detailed analyses.

Consequently, our HG can update state values with values from the optimal future states to avoid inaccurate and redundant value updates for every learning iteration.

4.1.2 Toy Text

Toy Text is more complex than Simple Maze since there will be many more paths to the desired location (Towers et al., 2023). We choose the deterministic CliffWalking and Taxi environments from Toy Text to evaluate the RL methods.

HG acts similarly when it is on Simple Maze, shown in Table 2. HG agent converged to the optimal policy and the optimal solution was found after one learning iteration within a minute.

The execution time and expected return of all methods are shown in Figure 9. HG achieved the best expected returns for all environments, and the growth rate of time over frames is the smallest among all methods, according to the third column in Figure 9. For CliffWalking results in Figure 9 (a), MuZero firstly converted to a locally optimal policy which is worse than the optimal policy due to the lack of sampling and training. DQN and PPO gradually increased the ability to get higher scores until the global optimal was reached. A3C did not effectively find a solution within one million frames of training. CliffWalking is different from Simple Maze since most grid has paths to the desired location. Thus, most sampled transitions of DQN can propagate optimal values to the starting state. However, randomly sampled trajectories will fade and slow down the optimal value propagation of A3C for each state. For Taxi-v3, all baselines were unable to find the solution within one million frames, see Figure 9 (b). MuZero, A3C, and DQN converged to a local optimal policy, and more training is required for better behavior.

Thus, these environments are again examples to show our HG can update state values without inaccuracy and redundancy.

4.1.3 Google Research Football

Next, we test our algorithm on the more difficult and computationally demanding Google Research Football (GRF) (Kurach et al., 2020) environment to show the efficiency of value updating. Agents are evaluated

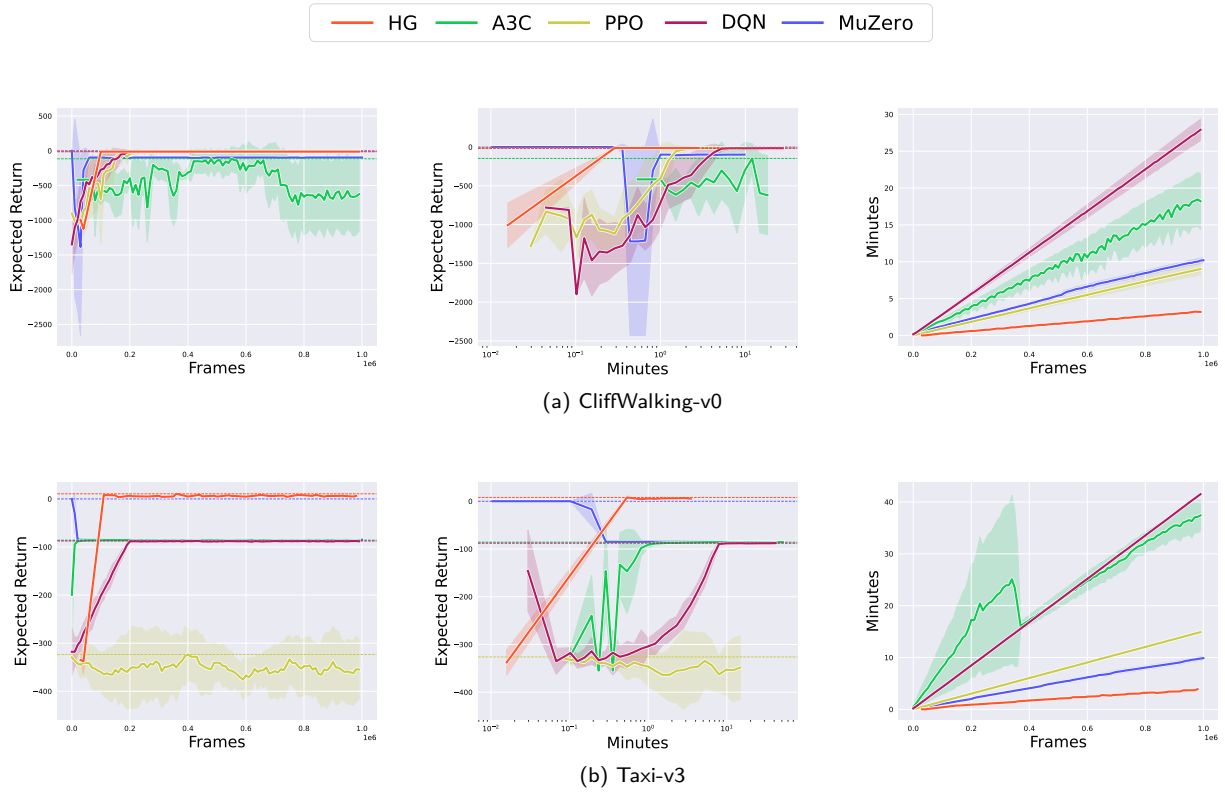


Figure 9: Episodic reward and efficiency comparison on Toy Text environments. The lines and shaded areas are for the average values and the corresponding range (±) of standard deviations, and dashed horizontal lines show the asymptotic performance. Every row records the training curves over minutes and frames of the same experiments. The first and rest columns show the improved returns and training efficiency respectively.

on three customized environments of 3 vs 2, 5 vs 5, and 11 vs 11 with increasing difficulty. The 3v2 is the attacking scenario with three defenders and two offenders including one goalkeeper for both teams. The tasks of 5v5/11v11 are confrontation scenarios with 5/11 players on each side. We use the single RL agent to control all the players in the team on the left-hand side according to the player's specific state at each time step, and the state is the SMM (Super Mini Map) feature (72 x 96 x 4) direct output from the GRF environment. Random seeds are used for environments to meet the deterministic assumption. There are two different rewards: goal reward and distance reward. If a team scores a goal, the simulator returns a +1.0 reward. The distance reward is determined by the minimum distance to the opponent's goal any team player ever made in this episode (the closer, the higher, with max +1.0).

The expected return and execution time of different RL agents are shown in Figure 10. The HG achieved the best expected return and scored for all scenarios within one million frames. Both IMPALA and R2D2 cannot score goals in all scenarios because the input of these environments is more complex and requires more computation to update the value of states. IMPALA made more progress on all scenarios compared with R2D2. This is partially because, in a large state space, the states on the path to the goal have a higher chance of being chosen to update value by a policy gradient method than the Q-learning method. With the number of possible states becoming larger from 3 vs 2 to 11 vs 11, the learning efficiency of R2D2 correspondingly decreased.

From the result on GRF environments, we can see the efficiency advantage of our highway graph RL agent. See Section 4.3 for more speed advantage of value updating.

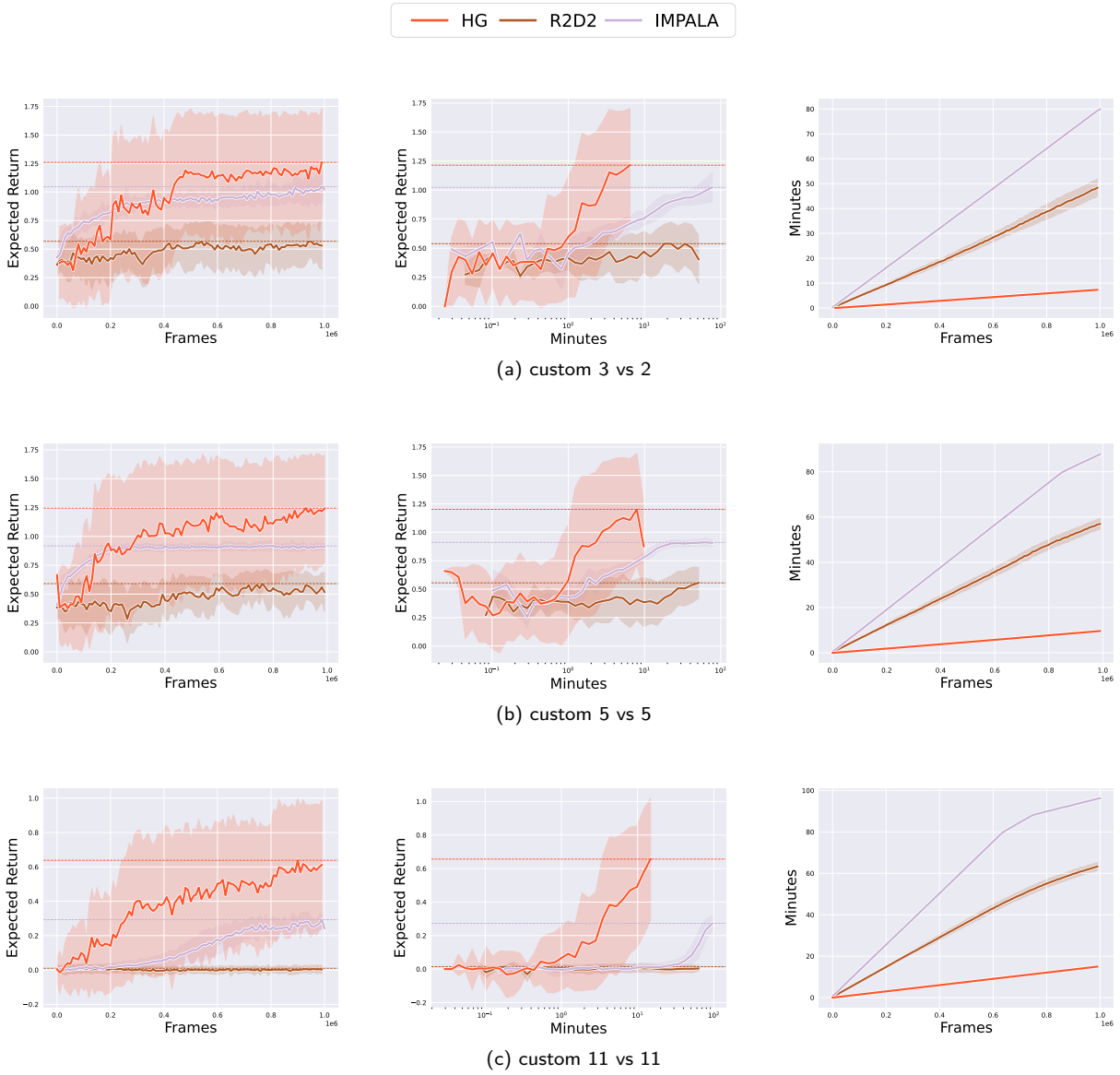


Figure 10: Expected return and efficiency comparison on 3v2, 5v5, 11v11 scenarios. The lines and shaded areas are for the average values and the corresponding range (±) of standard deviations, and dashed horizontal lines show the asymptotic performance. Every row records the training curves over minutes and frames of the same experiments. The first and rest columns show the improved returns and training efficiency respectively.

4.1.4 Atari learning environment

The Atari games using the Atari Learning Environment (Bellemare et al., 2013) are also used to evaluate our proposed method. We randomly select 3 games with mainstream genera of Atari to train the different RL agents for one million frames. An environment seed is also used during initialization for all games to guarantee the deterministic property.

The state of Atari games may not be directly accessible, instead we got an observation from the environment. To solve this, we use a recurrent random linear projector on all the past observations of the episode, to obtain a more compact state representation for the highway graph. This projection does not affect the deterministic property of the environments. More details of the random linear projector are in the Appendix.

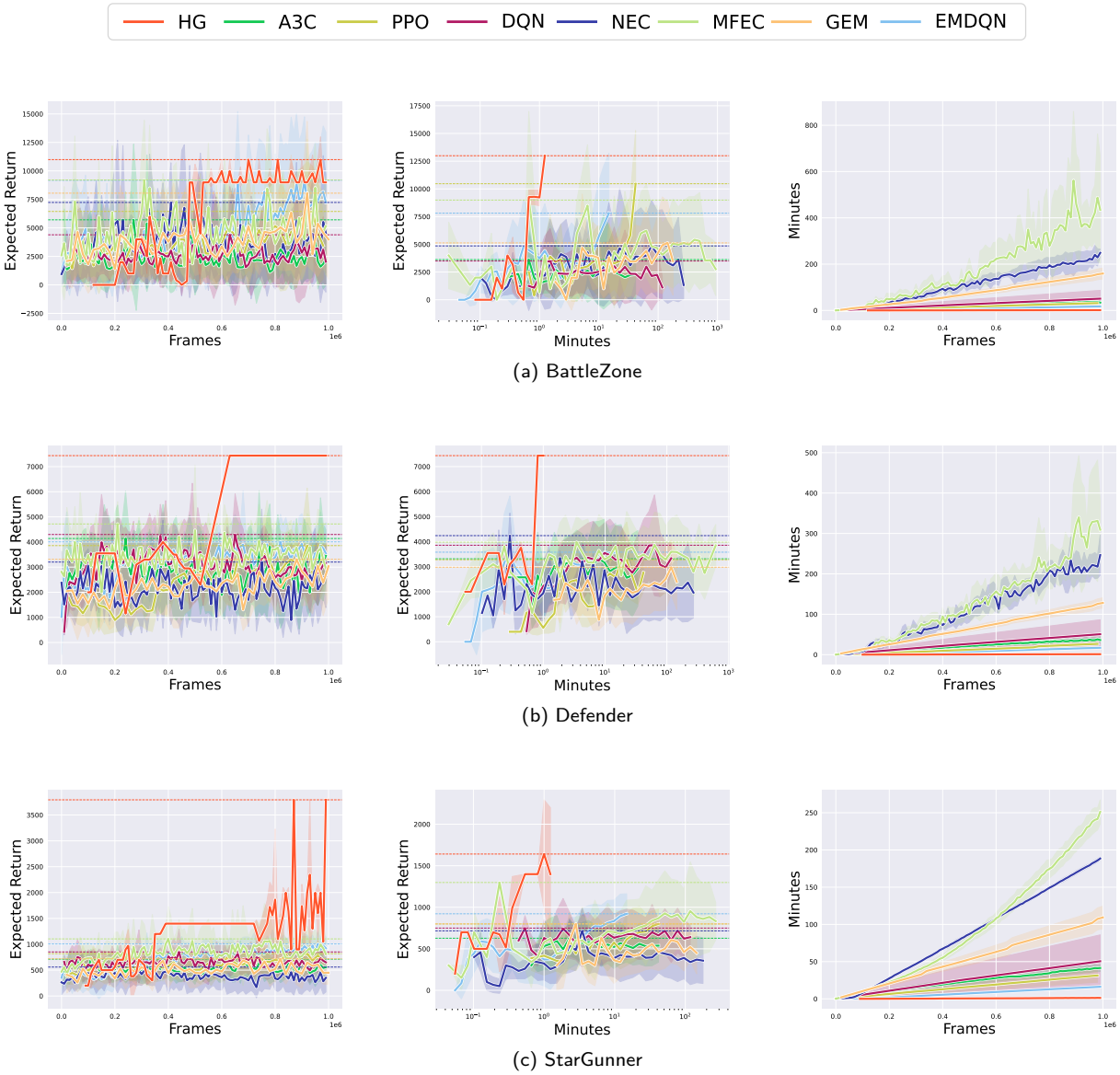


Figure 11: Expected return and efficiency comparison on Atari games. The lines and shaded areas are for the average values and the corresponding range ($\pm$) of standard deviations, and dashed horizontal lines show the asymptotic performance. Every row records the training curves over minutes and frames of the same experiments. The first and rest columns show the improved returns and training efficiency respectively.

Next, we compare the total scores of different agents on different games, shown in Figure 11. We can see the expected return curve of HG is above all the baselines for every environment. It shows the value of states of a trained HG agent will be more accurate than other methods. Another advantage of HG is drawn on the efficiency. The growth rate of time over frames of HG is the smallest. Additionally, HG only used about 1/10 to less than 1/150 period of time compared to other baselines to get converged and finish the training.

Many episodic-control-based methods including EMDQN and MFEC show a high sample efficiency by achieving relatively high scores, but the training efficiency of these methods is low.

Consequently, the highway of HG improves the value iteration efficiency and value updating accuracy simultaneously on Atari games.

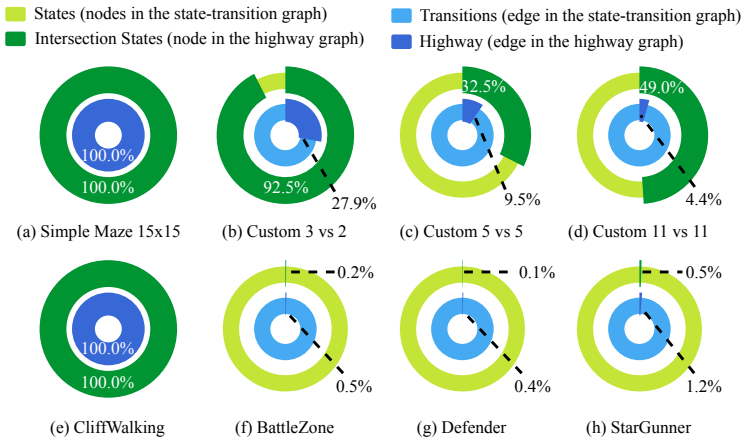


Figure 12: Percentage of nodes and edges in the highway graph compared with its corresponding state-transition graph. The conversion into highway graphs dramatically reduces the size of both nodes and edges.

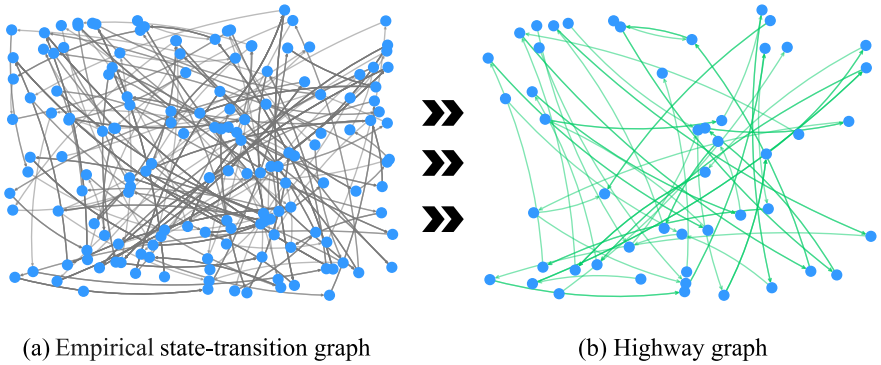


Figure 13: The state-transition graph (a) and its corresponding highway graph (b) of Custom 5 vs 5. The states in both (a) and (b) are blue cycles, highways in (b) are green arrows, and transitions are in grey. The nodes and edges in the highway graph are considerably reduced.

4.2 Structural advantage of highway graph

To investigate the advantage of our highway graph compared with the original empirical state-transition graph, we compare the size of nodes and edges for different environments, see Figure 12. The size of the nodes and edges of the highway graph is smaller than the original empirical state-transition graph. For example, the empirical state-transition graph of StarGunner has 234,521 nodes and 236,266 edges, whereas its highway graph only has 1,084 nodes (0.5%) and 2,829 edges (1.2%). According to Remark 1, $z = 0.005 \ll 1$ means the computation required by the value iteration will be theoretically reduced to $\frac{1}{2.5\times10^5}$. Many environments, including all evaluated Atari games, have very few intersection states (nodes), which makes the highway graph small enough to be loaded entirely into GPU to update all known state-action values in parallel, and hence achieve a promising acceleration during training.

Simple tasks, such as Simple Maze and Toy Text environments, have a densely connected state-transition graph with a small state space of the empirical state-transition graph, whereas tough tasks observe a nearly tree-like state-transition graph structure, where states on different branches are not connected, with a large state space. This trend becomes more obvious as the complexity of the environment increases, since from a state, in a complex environment, it is not easy to move to a previously visited state. After converting to highways, many paths can be shrunk as highways to reduce the number of states and edges. Thus, the use of the highway graph to model and update the value can be less computationally extensive. A comparison of a highway graph and its original empirical state-transition graph for the Custom 5 vs 5 is shown in Figure 13.

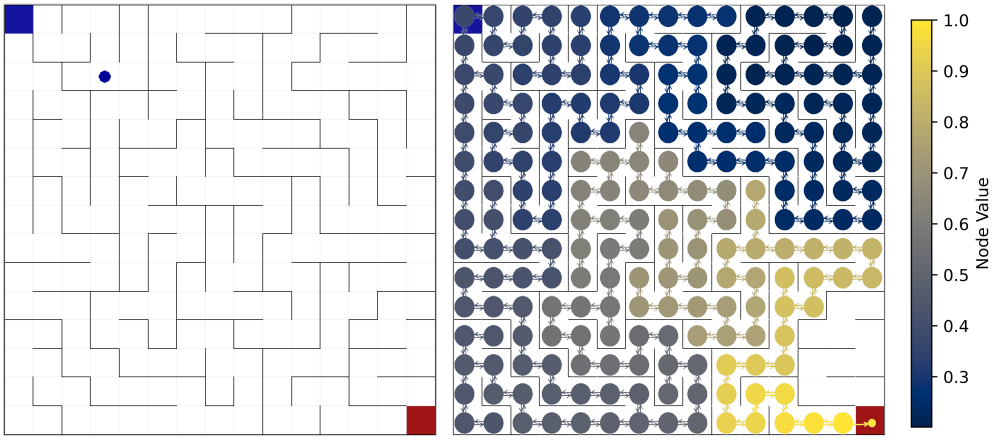


Figure 14: The 15x15 maze environment (left) and its state values on the corresponding highway graph (right, self-loops are not shown).

Table 3: Value updating completeness. It shows the distance to the ground truth of the learned value after one million frames of training. Completeness indicates the percentage of state value that is identical to the ground truth.

Method	Min Distance	Max Distance	Avg. Distance	Completeness (%)
PPO	0.24	0.99	0.53	1.88%
DQN	0.13	0.92	0.41	6.57%
HG (Ours)	0.00	0.00	0.00	100.00%

4.3 Value updating advantage of highway graph

Value updating completeness This experiment investigates the distance between the values output by trained RL agents and the ground truth values of each state on Simple Maze 15x15 (225 states in total). Completeness indicates the percentage of state values that are identical to the ground truth. The distances and completeness of value updating of different methods are shown in Table 3. From the results, our method performs optimal value updating of this task, and all states' values are updated ideally. The visualization of the learned HG model for the Simple Maze with a size of 15x15 is shown in Figure 14. The comparison methods cannot completely update the values of all states and many states have incorrect values.

In the case of Simple Maze 15x15, the DQN cannot successfully find a path to the target location. We find this is caused by the design of Q-learning. The effective training of Q-learning-based algorithms relies on a considerable amount of high-rewarded samples in the replay buffer (the ratio of the high-rewarded samples is task-dependent). The training of the state-action value function of this kind of method is influenced by the sampled transition and its built-in target state-action value function. Using the high-rewarded samples, the state-action value function will be more close to the real situation. Later, the target state-action value function can be updated more reliably, and the agent training enters a virtuous cycle. However, if the high-rewarded sample is not enough, the target state-action value function will take more effect. The sampling in the Simple Maze environment is not necessary to always find high-rewarded samples, and more exploration may find samples with lower rewards due to moving penalties. That leads to an unusable target state-action value function at an early stage so that the entire Q-learning will fail.

Value updating speed advantage We further show the value updating speed (the value propagation steps among the original empirical state-transition graph per training iteration). We use the Simple Maze with the size of 15 x 15 (225 states in total) and StarGunner to show the value updating speed. The results are in Figure 15. The speed of value updating on the highway graph is converted to value updating

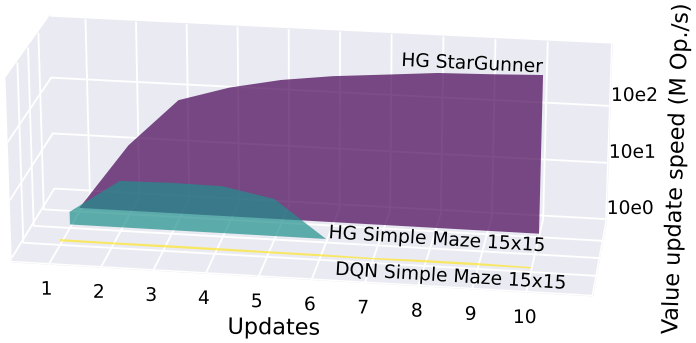


Figure 15: The million step/operation (Op.) per second of different value update iterations. The value update speed is the value propagation steps among the original empirical state-transition graph per training iteration. In each iteration, the highway graph is converted to an empirical transition graph to calculate the value update speed.

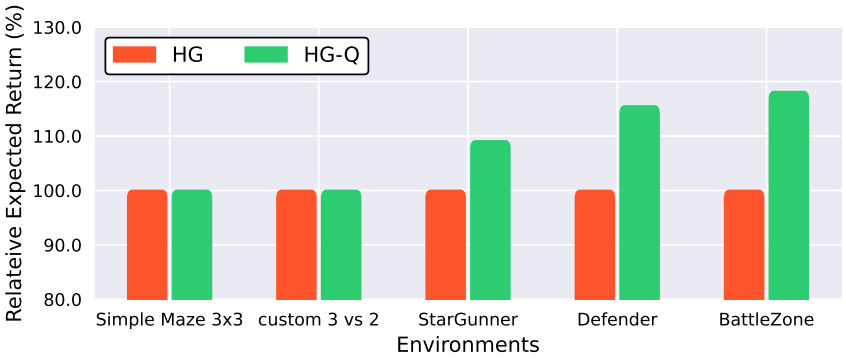


Figure 16: The relative expected return of HG-Q compared with HG.

steps in the empirical state-transition graph, by counting the value updating operation for each transition. The results show the value updating speed of DQN is the slowest which is due to the N-step TD learning algorithm, where only parts of the states are sampled for the value learning. On the contrary, our HG always updates the entire graph, and HG reaches the 1159 $\pm$ 31 million Op./s value updating speed during the 10th value update in the StarGunner. HG's value update speed gives evidence for faster convergence on previously evaluated environments.

4.4 Performance of re-parameterized neural network-based agent

Graph re-parameterization Atari games require very large state-transition graphs to store the environmental information, and graph re-parameterization can help to reduce storage costs. To re-parameterize the information in the highway graph, we use a very simple 2-layered MLP with 512 units in each layer to store the state-action value of the highway graph. The re-parameterized neural network-based agent is HG-Q. The relative performance of the expected return of the neural network-based agent HG-Q compared with the HG is shown in Figure 16.

From the result, we can see the neural network-based agent achieved an on-par performance compared with the agent built from the highway graph. Additionally, on Atari games, the neural network-based agent outperformed the highway graph by about 5% to 20%, which shows the generalization ability of the neural networks. These experiments indicate that the highway graph can be a jump-starter for other RL agents during the early stages of training.

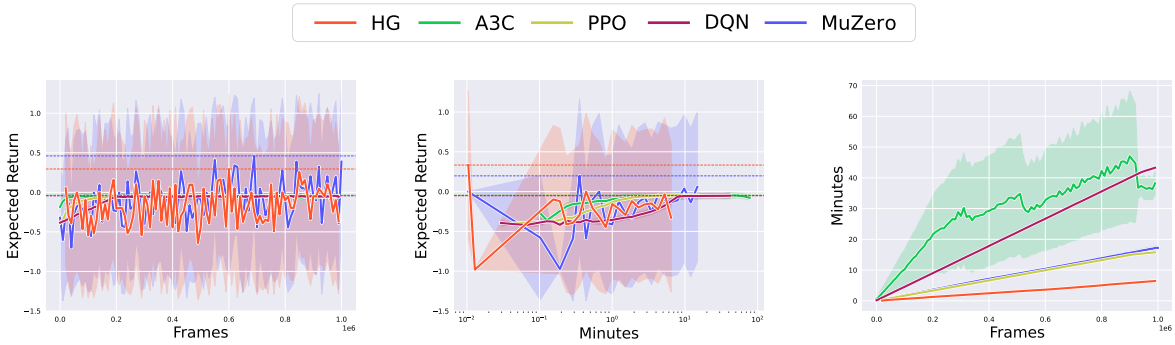


Figure 17: Highway graph on stochastic environments Blackjack.

5 Limitations

One of the important assumptions when designing the highway graph is that the state can be restored by observation. Thus, the highway graph does not suit partially observable and stochastic environments. (1) In a stochastic environment, from a specific state, the same action does not always lead to the same future state. In this case, the value updating of the highway graph will not be correct since the value will not always propagate in the same way. (2) In a partially observable environment, the state in the state-transition graph will not always be reliable, one state in the graph might link to two different real states, i.e. state conflict. When state conflict exists, value updating of the highway graph will be incorrectly caused by the same issue in the stochastic environment.

Experimental results on Blackjack support this idea. The goal for players of Blackjack in Figure 17 is to obtain 2 cards that sum to closer to 21 (without going over 21) than the dealer's cards. The card the player gets every time is randomly selected, which makes the environment stochastic. We evaluate the expected return of the highway graph of stochastic environment Blackjack from Toy text and the expected return will be unpredictable, resulting in a failed highway graph, see Figure 17. The found result is mainly caused by the state conflict, which means two different states will have the same representation and use the same node in the highway graph. The different nodes in the highway graph should be reacted differently, the state conflict prevents the agent from doing so and produces bad results.

6 Conclusions

The high cost of training RL agents has limited their research and applications. We explored an approach to significantly reduce this cost by applying the concept of highways in real-world transportation systems to the value-updating process of RL algorithms.

We introduced the concept of the highway graph structure, which is derived from the empirical state-transition graph and offers a significantly reduced state space. Furthermore, we demonstrated how this highway graph structure can be learned from sampled transitions and seamlessly integrated with a value iteration procedure to achieve an efficient value-updating process. Additionally, we provided theoretical evidence of the convergence to the optimal state value function of the value iteration on the highway graph.

In our experiment, We empirically evaluated our highway graph RL method on four categories of learning tasks and compared it with the state-of-the-art RL baselines, showing that it drastically reduces training time, achieving a speedup of about 10 to more than 150, while maintaining an equal or even superior expected return. We also re-parameterized the highway graph into a simple neural network-based agent that achieves on-par or better performance with a meager storage cost.

To make the highway graph RL method more versatile, future work can extend the highway graph to partially observable environments with stochastic features in nature. In addition, the highway graph's aggregated and

precise value updating can be integrated into different RL algorithms during value learning.By doing this, the value learning process of more RL algorithms can speed up using the highway graph.

References

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z. Leibo, Jack W. Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Fengdi Che, Gautham Vasan, and A. Rupam Mahmood. Correcting discount-factor mismatch in on-policy policy gradient methods. In *International Conference on Machine Learning*, volume 202, pp. 4218–4240. PMLR, 2023.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, volume 80, pp. 1406–1415. PMLR, 2018.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. SEED RL: scalable and efficient deep-rl with accelerated central inference. In *International Conference on Learning Representations*, 2020.

Ilyas Fatkhullin, Anas Barakat, Anastasia Kireeva, and Niao He. Stochastic policy gradient methods: Improved sample complexity for fisher-non-degenerate policies. In *International Conference on Machine Learning*, pp. 9827–9869. PMLR, 2023.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, volume 9, pp. 249–256, 2010.

Hao Hu, Jianing Ye, Guangxiang Zhu, Zhizhou Ren, and Chongjie Zhang. Generalizable episodic memory for deep reinforcement learning. In *International Conference on Machine Learning*, volume 139, pp. 4380–4390. PMLR, 2021.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.

Steven Kapturowski, Georg Ostrovski, John Quan, Rémi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

Mohamed A Khamsi and William A Kirk. *An introduction to metric spaces and fixed point theory*. John Wiley & Sons, 2011.

Karol Kurach, Anton Raichuk, Piotr Stanczyk, Michal Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4501–4510, 2020.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, volume 80, pp. 3059–3068. PMLR, 2018.

Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks. In *International Joint Conference on Artificial Intelligence*, pp. 2433–2439, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, volume 48, pp. 1928–1937. PMLR, 2016.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International Conference on Machine Learning*, volume 70, pp. 2827–2836. PMLR, 2017a.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International conference on machine learning*, pp. 2827–2836. PMLR, 2017b.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Maciej Swiechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mandziuk. Monte carlo tree search: a review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu, and Garrett Thomas. Value iteration networks. *Advances in Neural Information Processing Systems*, 29:2146–2154, 2016.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023. URL https://zenodo.org/record/8127025.

Jur van den Berg, Sachin Patil, and Ron Alterovitz. Efficient approximate value iteration for continuous gaussian pomdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pp. 1832–1838, 2012.

Guangxiang Zhu, Zichuan Lin, Guangwen Yang, and Chongjie Zhang. Episodic reinforcement learning with associative memory. In *International Conference on Learning Representations*, 2020.