A-MCTS: Adaptive Monte Carlo Tree Search for Temporal Path Discovery

Pengfei Ding, Guanfeng Liu, *Member, IEEE*, Yan Wang, *Senior Member, IEEE*, Kai Zheng, *Senior Member, IEEE*, and Xiaofang Zhou, *Fellow, IEEE*

Abstract—An attributed dynamic graph contains multiple dynamic attributes associated with each edge in the graph. In attributed dynamic graph based applications, people usually can specify multiple constraints in the attributes to illustrate their requirements, such as the total cost, the total travel time and the stopover interval of a flight between two cities. This inspires the multi-constrained temporal path discovery in attributed dynamic graphs, which is a challenging NP-complete problem. To solve the problem, the existing methods adopt reinforcement learning with Monte Carlo tree search. However, these reinforcement learning based methods require a certain degree of "discovery experience" to obtain better results, which can lead to the expensive cost of query time and storage space, and thus are not applicable in real-time applications, e.g., route recommendations in a self-driving car. This motivates us to develop a new Adaptive Monte Carlo Tree Search algorithm, namely A-MCTS. A-MCTS dynamically adjusts the priority of historical records that are used in Monte Carlo tree search to improve the performance and reduce the size of required "discovery experience". In addition, A-MCTS proposes a new adaptive replay memory structure that can store important historical records based on graph properties and optimize the consumption of storage space. The experimental results on ten real-world dynamic graphs demonstrate that the proposed A-MCTS outperforms the state-of-the-art methods in terms of both efficiency and effectiveness.

Index Terms—Temporal path, monte carlo tree, path finding

1 Introduction

Path discovery is popular in many applications, like trip planning in transportation networks [1] and route planning on the Internet [2]. Path queries can be modelled as the path discovery problem in graphs, which has been well studied in the literature. Many real applications can be encoded as *dynamic graphs*, in each of which a node is associated with another node at particular time instances [3]. For example, the connections between two people via telephone calls start at a certain time point of a day [4], and the connections between cities via courier vehicles based on the schedules of transportation [1].

There can be many attributes associated with the edges in dynamic graphs, forming *attributed dynamic graphs* (ADGs). These attributes can be like, the travel cost and the travel distance between two cities by flights. Users would like to specify constraints on these attributes in a path query to illustrate their requirements on the target dynamic graph. For example, in a traffic network, if a user plans to drive to a restaurant after 5pm, what is the shortest route that the user can arrive at the restaurant under certain constraints on the toll fee and the driving time? *Example 1* below discusses a scenario with an attributed dynamic graph structure.

Example 1. Fig. 1 shows an attribute dynamic graph. Assume the ADG is a road network, where each node denotes a bus station. The tuple on each edge contains a departure time point and an arrival time point, e.g., edge $\langle a \rightarrow c \rangle$ represents the bus leaves a at 2pm and arrives at c at 3pm. And the cost of taking the bus from a to c is \$3. For the sake of simplicity, the distance between any two connected stations is set to 1km. For the shortest temporal path from a to e, if there are no other constraints, the shortest temporal path is $\langle a \rightarrow c, c \rightarrow e \rangle$ and its length is 2km. But when the travel cost is constrained to be smaller than \$6, $\langle a \rightarrow c, c \rightarrow e \rangle$ becomes an invalid temporal path because the total cost reaches \$7. Instead, the shortest temporal path becomes $\langle a \rightarrow d, d \rightarrow f, f \rightarrow e \rangle$ with a length of 3km and a total cost of \$4.

The above discussion leads to the *multi-constrained temporal path* (MCTP) discovery problem, which has been proved to be NP-complete [5]. Targeting this challenging problem, the existing methods for the temporal path discovery [6], [7] have to enumerate all the possible temporal paths that satisfy each of the constraints, or they have to traverse all possible graph structures at all time points, leading to expensive overhead in query processing time [8]. Therefore, the existing methods are not applicable when dealing with large-scale attribute dynamic graphs.

In contrast, machine learning methods have the potential to be applicable across many optimization tasks by

- *Pengfei Ding, Guanfeng Liu, and Yan Wang are with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia. E-mail: pengfei.ding2@students.mq.edu.au, {guanfeng.liu, yan.wang}@mq.edu.au.*
- *Kai Zheng is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Chengdu 610054, China. E-mail: zhengkai@uestc.edu.cn.*
- *Xiaofang Zhou is with the The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: zxf@cse.ust.hk.*

Manuscript received 28 January 2021; revised 12 September 2021; accepted 1 October 2021. Date of publication 19 October 2021; date of current version 3 February 2023.
This work was supported by Australian Research Council Discovery Project No. DP200101441, and National Natural Foundation of China under Grants 62072125, 61972069, 61836007 and 61832017.
(Corresponding author: Guanfeng Liu.)
Recommended for acceptance by Y. Xia.
Digital Object Identifier no. 10.1109/TKDE.2021.3120993

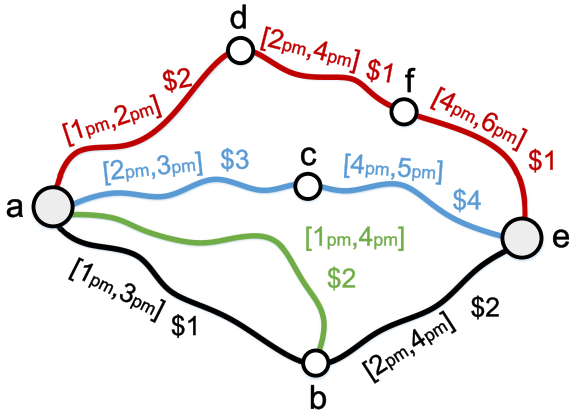

Fig. 1. An illustrative example of an attribute dynamic graph.

automatically discovering their own heuristics based on the training data, thus requiring less hand-engineering than solutions that are optimized for one task only. The power of generalization makes machine learning methods possible to predict the results of unseen data, thus benefiting real-time search in large datasets. While supervised learning is widely used in most successful machine learning based methods, it is not applicable to the MCTP problem because one does not have access to optimal labels. However, the *reinforcement learning* (RL) paradigm [9] can be used to tackle the optimal path discovery in dynamic graphs. Recently, RL has achieved success in many graph-based problems, such as graph generation [10], genetic network programming [11], knowledge graph reasoning [12] and some optimization problems on graphs, e.g., *traveling salesman problem* (TSP) [13] and graph coloring [14]. Among them, the most relevant ones to the MCTP problem are to use RL to solve knowledge graph reasoning and TSP. However, both of them aim at solving queries on statistic graphs and consider none/single constraint. Thus, they cannot be applied to solve the MCTP problem. Moreover, as they require a certain amount of training data to train their models, they are not suitable for real-time path searching. In order to apply RL to the MCTP problem, a verifier can be designed to compare the quality of temporal paths, and provide some reward feedbacks to a learning algorithm. In addition, the idea of *Monte Carlo tree search* (MCTS) [15] can be adopted, which is a family of directed search algorithm that can explore search spaces quickly. Thus, combining MCTS with RL can be an effective way of using machine learning to solve the MCTP problem. In the previous work [16], a *reinforcement learning based Monte Carlo tree search* (RL-MCTS) method is proposed, which formulates the MCTP problem as a *Markov decision process* (MDP) [17], then follows the RL and adopts MCTS to discover the required MCTP. The key idea of RL-MCTS is to construct a search tree of states evaluated by the rewards of MDPs. The search tree is maintained to guide the selection of states, for which bandit algorithms can be employed to balance exploration and exploitation [18].

Nevertheless, in the MDP of the MCTP problem, each temporal path is formulated as a state[16]. Because an ADG may contain millions of edges, the state space of the ADG is large, and thus it is unrealistic to visit all states and get their rewards for the state selection. In order to solve the problem, the existing method, RL-MCTS [16] sets a *replay memory* to store the features of visited states. Then, the rewards of unvisited states are estimated by comparing the similarity of their features with those features that are stored in the replay memory. Since it is very time-consuming to calculate the similarity for all the states in the replay memory, RL-MCTS randomly samples states from the replay memory for comparison.

However, the replay memory mechanism has some disadvantages.

- *In the replay memory, the reward distribution of states is not uniform.* In some conditions, sampling states indiscriminately can mislead building the search tree and degrade the performance of the algorithm. Thus, the accuracy of the reward estimation can not be effectively guaranteed. For example, when the constraints are strict (i.e., less travel cost and earlier arrival time of the search path) and the exploration resources are limited (i.e., limited storage space and search time), positive rewards can be received at only a few temporal paths, and most of the states in the replay memory may have negative or zero rewards.
- *In the replay memory, some states' features are not useful for the reward estimation.* e.g., a shorter path starting from the query node, which may include 2-3 edges only, can appear repeatedly as a sub-path in the search process. As these shorter paths have limited information, i.e., nodes and edges that are close to the query node will appear repeatedly in these shorter paths and reduce the variance of these paths, there is a limited difference between their features. However, their rewards can be totally different as many different states may start from these shorter paths, which makes the reward estimation inaccurate.

Based on the above analysis, specifically, there are two problems that need to be solved.

- ***Problem 1.*** How to sample states in the replay memory efficiently to overcome sparse reward?
- ***Problem 2.*** How to adjust replay memory to store useful states with appropriate path length?

In order to solve the above problems, in this paper, we propose a new *Adaptive Monte Carlo Tree Search* (A-MCTS) algorithm which contains a new *priority mechanism* and a new *adaptive replay memory structure* for different search conditions. First, A-MCTS sets a "priority value" for each state, which will be updated during the search process. The priority value is used to sample states with more influence and authority, leading to more accurateness of the reward estimation. In addition, exploring low-priority states can ensure our algorithm discovers more unvisited states and MCTP results. Furthermore, A-MCTS does not store states with fewer edges. When states are visited for the first time, each state has a probability to be saved in the replay memory. The probability is set based on the number of edges of the state. Thus A-MCTS can reduce storage space and cut off the correlation between adjacent states.

Our contributions are summarized as follows.

1) We propose a new priority mechanism for each state that can address the inaccuracy reward estimation;

TABLE 1
Summary of the Related Temporal Path Discovery Algorithms

Algorithm	Model Type	Constraints	Complexity
Orda *et al*. [19]	Continuous	Single	$O(n^2)$
Ding *et al*. [20]	Continuous	Single	$O((m+n)\alpha(T))$
Wang *et al*. [25]	Continuous	Single	$O(l_{max}mn \cdot log(l_{max}n))$
Yang *et al*. [24]	Continuous	Single	$O(knlogn + mk)$
Wu *et al*. [26]	Discrete	Single	$O(m\pi(log\pi + logn))$
Xuan *et al*. [8]	Discrete	Single	$O(mnlog\pi)$
Wang *et al*. [27]	Discrete	Single	$O(nm + n^2logn)$
TPA [30]	Discrete	Multiple	$O(n + mlog(d_{max}))$
RL-MCTS [16]	Discrete	Multiple	$O((U+F)I \cdot N_D)$

2) We propose a new adaptive replay memory structure to store more important features into the replay memory and optimize storage space;
3) We propose a new Adaptive Monte Carlo Tree Search algorithm that combines the priority mechanism and the adaptive replay memory structure. A-MCTS adopts a new tree search policy to explore low-priority states and get more MCTP results;
4) We conduct extensive experiments on ten real-world attribute dynamic graphs. The experimental results illustrate that on average, our algorithm can greatly save query processing time by 84.20%, and the average path length of MCTP delivered by A-MCTS is 26.71% shorter than the state-of-the-art algorithms.

2 Related Work

In the literature, there are many studies of the temporal path discovery in dynamic graphs, which can be classified into two groups: the *continuous time model* and the *discrete time model*. Table 1 summarises the features of the related works, where m and n denote the number of nodes and edges in a dynamic graph, respectively.

In the *continuous time model*, a continuous edge-delay function is associated with each edge in a dynamic graph. Orda *et al.* [19], propose a Bellman-Ford based algorithm that calculates the earliest time to reach any node in the graph. Ding *et al*., [20], apply a more precise refinement approach that expands the time interval step by step, which can avoid the unnecessary computations and achieve better performance. To improve the efficiency of the path queries, some studies build different kinds of indices, such as time-dependent contraction hierarchies [21] and time-dependent hop labeling [22]. Yu *et al*., [23], study the dynamic shortest path problem in distributed environments. Furthermore, by considering the waiting time at the starting point, Yang *et al*., [24], propose an algorithm to find a cost-optimal path with a time constraint by using the iteration method. These methods can be used to solve the path problem in the continuous time model, but suffer from high computational complexity when adopted to the scenario where the connection between two nodes exists at specific time points [25].

In the *discrete time model*, the departure time points of an edge in a dynamic graph are discrete. In such a model, Xuan *et al*., [8] propose the algorithm for searching the shortest, the fastest, and the earliest arrival path, respectively, in dynamic graphs. Based on this work, in addition to the three paths, Wu *et al*., [26], propose an one-pass algorithm for the latest departure path. After that, some methods have been proposed to improve the efficiency of path discovery by using indexing techniques [27] and shared computation [28]. Moreover, some studies consider the waiting time in path discovery. Dean *et al*., [29], investigate different waiting policies and demonstrate how to accelerate the dynamic programming to efficiently address the minimum cost path problem under these policies.

However, these methods in the above two groups do not consider the multiple constraints on the attributes of edges in an attributed dynamic graph. Such a constrained path is popular and fundamental in many dynamic graph-based applications. Therefore, these methods cannot support the NP-complete MCTP problem that exists in many real applications.

Recently, based on the one-pass algorithm for the discrete time model, [30] proposes a *two-pass approximation algorithm* (TPA) for the MCTP problem, which uses a greedy optimization strategy that cannot guarantee the algorithm performance. TPA has to traverse the graph more than once, which costs a lot of time when the dynamic graph is large. Therefore, it's not a good solution to the MCTP problem. [16] first proposes a reinforcement learning based algorithm for the MCTP problem, i.e., RL-MCTS. RL-MCTS stores discovery experience to evaluate unvisited paths for selection. In order to achieve better performance, a certain amount of visited temporal paths that have positive rewards are needed. Therefore, when the storage space is limited and the rewards of visited temporal paths are sparse, RL-MCTS can hardly deliver good performance.

In Table 1, the time complexity of RL-MCTS [16] is independent of the size of the dynamic graph, since U and F are dependent on the size of the search tree of RL-MCTS, I and N_D are user-defined parameters. As another algorithm for multiple constraints, TPA [30] adopts greedy optimization strategy to achieve a time complexity similar to that of single constraint algorithms. The time complexity of single constraint algorithms is proportional to the scale of the dynamic graph. Therefore, transforming them to multiple constraint algorithms will lead to more expensive overhead in query processing time, which is inapplicable to deal with large-scale attribute dynamic graphs.

3 Preliminaries

This paper focuses on the shortest temporal path discovery with multiple constraints in attributed dynamic graphs. The notations used in this paper are shown in Table 2.

3.1 Attributed Dynamic Graph

An attributed dynamic graph is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, F)$, where:

- $\mathcal{V}$ is the set of nodes of $\mathcal{G}$;
- $\mathcal{E}$ is the set of edges of $\mathcal{G}$, an edge $e \in \mathcal{E}$ is a quintuple $(u, v, l, t^u_{dep}, t^v_{arr})$, where $u, v \in \mathcal{V}$, l is the length of e, t^u_{dep} and t^v_{arr} are the departure time point from u and the arrival time point at v respectively, hence, $t^u_{dep} \leq t^v_{arr}$;

TABLE 2
Notations and Explanations

Notation	Explanation
$\mathcal{G}$	attributed dynamic graph
$\mathcal{V}$	set of nodes
$\mathcal{E}$	set of edges
$F = \{f_1, f_2, \ldots, f_K\}$	set of K attribute functions
$P_{u,v}(t^v_{arr})$	temporal path starts from u, arrives v at t^v_{arr}
$[t_\phi, t_\varphi]$	constraint of time interval
$\mathcal{X} = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$	set of K constraints
$\mathcal{S}$	set of states
$\mathcal{A}$	set of actions
$\mathcal{R}$	reward function
D	replay memory
δ, μ	discount factors
$V(P)$	state-value function
$Q(P, e)$	action-value function
$Q_U(P, e)$	updated action-value function
$\mathcal{R}_{avg}(P)$	average reward value of P
$\mathcal{R}_{mem}(P)$	approximate reward of P
$N(P)$	number of times that P has been visited
$\mathbf{M}(e)$	embedding of e
$\mathbf{H}(P)$	attribute embedding of P
$\mathbf{T}(P)$	trajectory embedding of P
$d(P_i, P_j)$	distance function
$PR(P)$	priority value of P
SN_{max}	maximum sample number
$SN(P)$	sample number of P
E_{lim}	Limitation of edge number

- $F = \{f_1, f_2, \ldots, f_K\}$ is a set of K attribute functions that assign every edge in $\mathcal{E}$ a non-negative value $f_j(e)$, i.e., $f_j : (u, v, l, t^u_{dep}, t^v_{arr}) \in \mathcal{E} \rightarrow \mathbb{R}^+, j \in \{1, 2, \ldots, K\}$.

3.2 Temporal Path

In an ADG $\mathcal{G}$, a temporal path P is a sequence of edges $P_{v_1,v_n}(t^n_{arr}) = \langle e_1, e_2, \ldots, e_{n-1}\rangle$, where:

- $e_i = (v_i, v_{i+1}, l_i, t^{v_i}_{dep}, t^{v_{i+1}}_{arr})$, $i \in \{1, 2, \ldots, n-1\}$ is the ith edge in P. $t^{v_i}_{arr}$ and $t^{v_i}_{dep}$ can be seen as the arrival time point and the departure time point of v_i in e_{i-1} and e_i respectively, thus $t^{v_i}_{arr} \leq t^{v_i}_{dep}$;
- the attribute functions on temporal paths are defined as the aggregated attribute value of edges in the temporal path, i.e., $f_j(P_{v_1,v_n}(t^{v_n}_{arr})) = \sum_{i=1}^{n-1} f_j(e_i)$, thus $F(P) = \{f_1(P), f_2(P), \ldots, f_K(P)\}$;
- $|P_{v_1,v_n}(t^{v_n}_{arr})| = \sum_{i=1}^{n-1} l_n$ is the path length of $P_{v_1,v_n}(t^{v_n}_{arr})$.

3.3 Feasible Temporal Path

Given an ADG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, F)$, $s, d \in \mathcal{V}$ denote the source node and the destination node respectively, $\mathcal{X} = \{\lambda_1, \lambda_2, \ldots, \lambda_K\}$ is the set of K constraints on F, $[t_\phi, t_\varphi]$ is the constraint of the time interval. A temporal path $P_{s,d}(t^d_{arr})$ is a *feasible temporal path*, where:

- $t^s_{dep} \geq t_\phi, t^d_{arr} \leq t_\varphi$;
- $\forall j \in \{1, 2, \ldots, K\}$, $f_j(P_{s,d}(t^d_{arr})) \leq \lambda_j$.

Example 2. Recall the ADG shown in Fig. 1, the temporal path from a to e (i.e., the blue path) can be denoted as $P_{a,e}(5) = \langle (a, c, 1, 2, 3), (c, e, 1, 4, 5)\rangle$. And the total cost of $P_{a,e}(5)$ is $f(P_{a,e}(5)) = \$7$. If the source node is set as a, the destination node is set as e, and the constraints of the total cost and the time interval are set as $\lambda = \$5$ and $[1, 6]$, respectively, there is only one *feasible temporal path*, i.e., the red path $P_{a,e}(6) = \langle (a, d, 1, 1, 2), (d, f, 1, 2, 4), (f, e, 1, 4, 6)\rangle$, and the total cost of $P_{a,e}(6)$ is $\$4$.

Definition 1. Multi-Constrained Shortest Temporal Path. Given an ADG $\mathcal{G}$, a source node v_s and a destination node v_d in $\mathcal{G}$, a constraint set $\mathcal{X}$ and a time interval $[t_\phi, t_\varphi]$. Let $\mathbf{P}$ denote the set of feasible temporal paths. *Multi-constrained shortest temporal path* (MC-STP) is to find the temporal path P that meets $|P| = \min\{|P'| : P' \in \mathbf{P}\}$.

3.4 Markov Decision Process for Multi-Constrained Shortest Temporal Path Discovery

The MC-STP problem can be formulated as a Markov decision process, which is defined by $(\mathcal{S}, \mathcal{A}, \mathcal{R})$. $\mathcal{S}$ is the set of states. Each temporal path denotes a state, i.e., $P \in \mathcal{S}$. $\mathcal{A}$ is the set of actions. For a state $P_{u,v}(t^v_{arr})$, $\mathcal{A}(P_{u,v}(t^v_{arr}))$ is the union of all edges that $P_{u,v}(t^v_{arr})$ can select after reaching v at t^v_{arr}, i.e., all edges in $\mathcal{A}(P_{u,v}(t^v_{arr}))$ departure from v after t^v_{arr}. After taking an action of $\mathcal{A}(P)$, the corresponding edge of the action will be added at the end of P, thus forming a new temporal path. $\mathcal{R}$ is the reward function. In the MDP for MC-STP discovery, only *terminal states* get rewards. Terminal states are defined as follows: if (1) the current state is a feasible temporal path; or (2) the aggregated attribute values of the current state exceed the constraints, i.e., $\exists j \in \{1, 2, \ldots, K\}$, $f_j(P) > \lambda_j$; or (3) the current state has no actions, i.e., $\mathcal{A}(P) = \emptyset$. Then, the current MDP reaches a terminal state and will receive a reward. Since the path length of the MC-STP is unknown, the reward of the first visited feasible temporal path P_F is set as $+0.5$, and its length is used to evaluate the rewards of the following feasible temporal paths P:

$$\mathcal{R}(P) = \frac{\max(|P_F|, |P|) + |P_F| - |P|}{2\max(|P_F|, |P|)}. \quad (1)$$

The range of $\mathcal{R}(\cdot)$ is (0,1), and shorter feasible temporal paths have larger rewards. For terminal states that are not feasible temporal paths, the rewards will be 0.

Example 3. Fig. 2 shows an example of an MDP, which starts from source node a, takes action $(a, b, 1, 1, 3)$, and reaches state $P_{a,b}(3)$. For $P_{a,b}(3)$, $\mathcal{A}(P_{a,b}(3)) = \{(b, d, 1, 4, 7), (b, e, 1, 5, 9)\}$. $(b, c, 1, 2, 3)$ is an invalid action because the departure time point of it (i.e., 2) is earlier than the arrival time point of $P_{a,b}(3)$ (i.e., 3). Then, $P_{a,b}(3)$ selects $(b, d, 1, 4, 7)$ and is transformed into $P_{a,d}(7)$. Finally, the MDP reaches a terminal state and gets reward $\mathcal{R}(\cdot)$.

4 Adaptive Monte Carlo Tree Search Algorithm

In this section, we propose a new adaptive Monte Carlo tree search algorithm, namely A-MCTS, by adopting reinforcement learning with Monte Carlo tree search. Because existing RL and MCTS methods are not suitable for path discovery and have limitations in solving MC-STP problem, A-MCTS focuses on how to combine the characteristics of

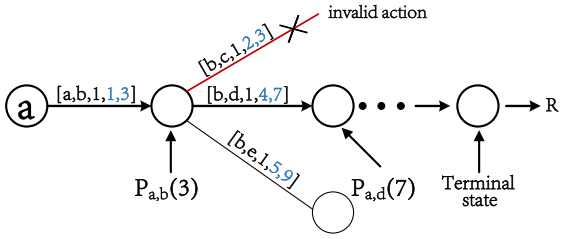

Fig. 2. A markov decision process.

dynamic graphs and temporal paths to overcome these limitations. Specifically, these difficulties can be solved from the perspective of RL and MCTS.

For RL, evaluating states and actions is an important step in RL. Classic RL methods require a certain number of explorations to have accurate evaluations. In order to speed up the exploration, A-MCTS utilizes the characteristics of graphs. Specifically, A-MCTS adopts a state-value function to evaluate states (i.e., temporal paths) based on their attributes, and adopts an action-value function to evaluate actions (i.e., edges). State-value and action-value functions evaluate states and actions before visiting them, respectively, Therefore, our model can reduce the cost of explorations.

For MCTS, the key idea of MCTS is to construct a search tree consisting of states, which is maintained to guide the selection of actions. The rewards are sparse because positive rewards can be received only at several temporal paths. As a result, evaluations of states and actions is likely to have high variance, which can mislead building the search tree and severely degrade the performance of the MCTS. In order to solve the problem, (1) for the state-value function, A-MCTS sets different discount factors to ensure states receive larger update magnitude when rewards are positive; (2) for the action-value function, A-MCTS sets a "priority value" for each state. The priority value is used to sample states with more influence and authority, leading to more accurateness of action-value.

In this section, A-MCTS first introduces the state-value function $V(\cdot)$ and the action-value function $Q(\cdot)$. As $Q(\cdot)$ is calculated under ideal conditions, $Q(\cdot)$ has limitations in some situations. Then, A-MCTS proposes the adaptive replay memory structure to overcome these limitations and introduce the updated action-value function $Q_U(\cdot)$. Lastly, A-MCTS builds a tree structure and model the MC-STP discovery as a tree search process, then introduce how to combine $Q_U(\cdot)$ and $V(\cdot)$ to guide the Monte Carlo tree search process.

4.1 State-Value Function

This section introduces the state-value function $V(\cdot)$ and explains how to update $V(\cdot)$. Given an ADG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, F)$ and a set of constraints $\mathcal{X}$, when a state P is visited for the first time, the initial state-value for P is defined as in Eq. (2).

$$V(P) = 1 - \max\left\{\frac{f_j(P)}{\lambda_j} : j \in \{1, 2, \ldots, K\}\right\}, \tag{2}$$

where $f_j(P) \in F(P)$ and $\lambda_j \in \mathcal{X}$ are the value and the constraint of jth attribute, respectively. The lower values of attribute functions, the higher initial state-value $V(\cdot)$. Note that if $V(P) < 0$, it means there exists at least an attribute function that exceeds the constraint, i.e., $\exists j \in \{1, 2, \ldots, K\}$, $f_j(P) > \lambda_j$, thus P is not a feasible temporal path.

When MDP $= \langle P_1, P_2, \ldots, P_d \rangle$ reaches terminal state P_d and gets reward $\mathcal{R}(P_d)$, the intermediate state $P_i (i \in \{1, 2, \ldots, d-1\})$ in the MDP is updated as in Eqs. (3) and (4).

- If P_d is a feasible temporal path, $V(\cdot)$ of each intermediate state P_i in the MDP will be updated by Eq. (3).

$$V(P_i) \leftarrow V(P_i) + \delta^{(d-i)} \cdot \mathcal{R}(P_d). \tag{3}$$

  By setting a discount factor $\delta \in (0, 1)$, intermediate states P_i that are closer to P_d will get larger discount rewards;
- If P_d is not a feasible temporal path, though $\mathcal{R}(P_d) = 0$, A-MCTS sets a small penalty to the intermediate states P_i and update $V(P_i)$ as in Eq. (4).

$$V(P_i) \leftarrow V(P_i) - \mu^{(d-i)} \cdot \min(|V(P_d)|, 1). \tag{4}$$

  Here $\min(|V(P_d)|, 1)$ is used to ensure the range of $\min(|V(P_d)|, 1)$ is the same as $\mathcal{R}(\cdot)$, i.e., $[0, 1]$. Because of the sparse reward, i.e., a small part of terminal states are feasible temporal paths, intermediate states will get penalties more often than positive rewards. Then, in Eq. (4), a smaller discount factor $\mu \in (0, \delta)$ is set to deal with the imbalanced distribution of terminal state rewards. The different settings of δ and μ will only influence the update magnitude of $V(\cdot)$, i.e., the larger the value of δ, the more the increase of $V(\cdot)$, and the larger the value of μ, the more the decrease of $V(\cdot)$. Therefore, by setting appropriate values of δ and μ, $V(\cdot)$ can converge to a stable value smoothly. Otherwise, it needs to take more searches and updates to converge $V(\cdot)$.

In Eqs. (3) and (4), $\delta^{(d-i)} \cdot \mathcal{R}(P_d)$ and $-\mu^{(d-i)} \cdot \min(|V(P_d)|, 1)$ can be regarded as the rewards of intermediate state P_i in the MDP. Therefore, in addition to the previously defined reward functions for terminal states, the reward function of intermediate states $P_i (i \in \{1, 2, \ldots, d-1\})$ can be complemented as follows:

$$\mathcal{R}(P_i) = \begin{cases} \delta^{(d-i)} \cdot \mathcal{R}(P_d) & \text{if } \mathcal{R}(P_d) > 0 \\ -\mu^{(d-i)} \cdot \min(|V(P_d)|, 1) & \text{otherwise,} \end{cases} \tag{5}$$

where P_d is the terminal state of the current MDP.

Since state-value $V(\cdot)$ is influenced by the attribute functions $F(\cdot)$, $V(\cdot)$ cannot represent the *long-term reward* of states, i.e., the average rewards that states can obtain in the search process. Thus, $\mathcal{R}_{avg}(\cdot)$ is proposed to characterize the long-term reward of states as in Eq. (6).

$$\mathcal{R}_{avg}(P) = \frac{\sum_{j=1}^{N(P)} \mathcal{R}_j(P)}{N(P)}, \tag{6}$$

where $N(P)$ is the number of times that P has been visited in the search process, $\mathcal{R}_j(P)$ is the reward of P when P is visited for the jth time. As $N(P) \to \infty$, $\mathcal{R}_{avg}(P)$ will be close to the *true reward* of P, i.e., $\mathbb{E}(\mathcal{R}(P))$.

4.2 Action-Value Function

This section introduces an action-value function $Q(\cdot)$. $Q(\cdot)$ represents the long-term reward of taking actions. If P_i takes e and is transformed into P_j, the action-value $Q(P_i, e)$ can be defined intuitively as the difference between the true rewards of P_i and P_j shown in Eq. (7).

$$Q(P_i, e) = \mathbb{E}(\mathcal{R}(P_j)) - \mathbb{E}(\mathcal{R}(P_i)), P_i \xrightarrow{e} P_j \tag{7}$$

If $Q(P_i, e)$ is positive, it means that selecting e will transform P_i into a state that is approaching to feasible temporal paths. From Eq. (6), we can see $\mathcal{R}_{avg}(P) \approx \mathbb{E}(\mathcal{R}(P))$ when $N(P)$ is large. Therefore, $\mathcal{R}_{avg}(\cdot)$ can replace $\mathbb{E}(\mathcal{R}(\cdot))$ to calculate $Q(\cdot)$. However, getting accurate $\mathcal{R}_{avg}(P)$ requires traversing most of MDPs that contain P, which is unrealistic under large state space and limited search time. In addition, if P has not been visited, $\mathcal{R}_{avg}(P)$ is unknown. In order to overcome the limitation of $Q(\cdot)$, A-MCTS proposes the *adaptive replay memory structure* to approximate $\mathbb{E}(P)$ when $N(P)$ is small or zero, which is denoted as the *approximate reward* $\mathcal{R}_{mem}(P)$.

Next, we first introduce the adaptive replay memory structure and how to use it to get $\mathcal{R}_{mem}(\cdot)$; then propose the updated version of action value function, i.e., $Q_U(\cdot)$, which uses $\mathcal{R}_{mem}(\cdot)$ to overcome the limitation of $Q(\cdot)$.

4.3 Adaptive Replay Memory Structure

This section introduces the adaptive replay memory structure D and explains how it works. The replay memory D adopts the discovery experience, i.e., visited temporal paths, to get the approximate rewards $\mathcal{R}_{mem}(\cdot)$ of states. It mainly has the following three operations: *store*, *query* and *update*. *D stores* information of states that are visited for the first time in the search process, *updates* their information when these states are visited again, returns $\mathcal{R}_{mem}(P)$ for the *query* of P.

4.3.1 Store

Each entry of D corresponds to the information of a state. It contains the tuple $\{\mathbf{H}(P), \mathbf{T}(P), \mathcal{R}_{avg}(P), N(P), PR(P)\}$ of P. $|D|$ is the size of the replay memory, i.e., the number of entries that are stored in D. As D does not store the same state repeatedly, the replay memory stores the information of $|D|$ different states. $\mathbf{H}(\cdot)$ and $\mathbf{T}(\cdot)$ are feature representations of states. D samples a part of entries for *query*, and sets the "priority value" (denoted as $PR(\cdot)$) for each state, which is an indicator that records sampling priority of states. Next, we introduce how to encode features of states into embeddings $\mathbf{H}(\cdot)$ and $\mathbf{T}(\cdot)$.

In ADGs, temporal paths contain two characteristics, attribute features and trajectory features. Therefore, A-MCTS encodes them into attribute embeddings $\mathbf{H}(\cdot)$ and trajectory embeddings $\mathbf{T}(\cdot)$, respectively. For $\mathbf{H}(\cdot)$, since each attribute function is independent with others, A-MCTS normalizes values of attribute functions and takes each attribute as a dimension of $\mathbf{H}(\cdot)$. Then $\mathbf{H}(\cdot)$ can be defined as in Eq. (8).

$$\mathbf{H}(P) = B([|P|, f_1(P), f_2(P), \ldots, f_K(P)]), \tag{8}$$

where $B(\cdot)$ is a normalizing function.

In order to get trajectory embeddings $\mathbf{T}(\cdot)$, A-MCTS uses *DeepWalk* [31], one of graph embedding techniques, to convert temporal paths into a low dimensional space in which the structure information is maximally preserved. DeepWalk has two main subcomponents: random walk and Skip-gram model. DeepWalk first generates a corpus by sampling a number of random walks starting at each node. Then, the Skip-gram uses these random walks to learn the representation vector for each node. As DeepWalk is used to learn node embeddings in static graphs, it needs to be transferred to learn edge embeddings in dynamic graphs. Specifically, A-MCTS modifies the random walk subcomponent to make sampled random walks contain structural and temporal information of edges. That is, A-MCTS samples random walks consisting of edges instead of nodes. Edges in a random walk follow the chronological order. Formally, A-MCTS first sets $E_N(e_v) \subset \mathcal{E}$ as a *temporal neighborhood* of edge $e_v = (p, q, l_v, t^p_{dep}, t^q_{arr})$, i.e., every edge $e_i = (m, n, l_i, t^m_{dep}, t^n_{arr}) \in E_N(e_v)$ satisfies $m = q$ and $t^m_{dep} \geq t^q_{arr}$. Then, given a source edge e_u, a random walk of fixed length l_w is simulated. Let c_i denote the ith edge in the walk, starting with $c_0 = e_u$. Edges c_i are generated by the following distribution:

$$\mathcal{P}_{walk}(c_i = e_x | c_{i-1} = e_v) = \begin{cases} \frac{\beta(e_x)}{Z(E_N(e_v))} & \text{if } e_x \in E_N(e_v) \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where $\beta(e_x)$ is the unnormalized transition probability between edges e_v and e_x. A-MCTS samples the next edge based on its length, i.e., $\beta(e_x) = 1/l_x$. $Z(\cdot)$ is used to normalize transition probabilities, i.e., $Z(E_N(e_v)) = \sum_{e_j \in E_N(e_v)} \beta(e_j)$.

Then, a Skip-gram uses these random walks to learn the embedding for each edge, which is denoted as $\mathbf{M}(\cdot)$. Thus, edges that appear and are connected within similar time intervals will have similar embeddings. The trajectory embedding $\mathbf{T}(P_c)$ is defined as a sequence consisting of edge embeddings:

$$\mathbf{T}(P_c) = [\mathbf{M}(e_1), \mathbf{M}(e_2), \ldots, \mathbf{M}(e_{n-1})], P_c = \langle e_1, e_2, \ldots, e_{n-1} \rangle. \tag{10}$$

When D *stores* P, if D is not full, the tuple of P (i.e., $\{\mathbf{H}(P), \mathbf{T}(P), \mathcal{R}_{avg}(P), N(P), PR(P)\}$) will be initialized and stored in D. Otherwise, the tuple of P will replace the state in D that has not been visited for the longest time. Next, we will introduce how to calculate approximate reward $\mathcal{R}_{mem}(\cdot)$ and how to set $PR(\cdot)$ based on $\mathcal{R}_{mem}(\cdot)$.

4.3.2 Query

When $N(P)$ is zero or small, $\mathcal{R}_{avg}(P)$ will be unknown or inaccurate. In this condition, *query* operation can be used to approximate stable reward of P, i.e., $\mathcal{R}_{mem}(P)$. Specifically, the replay memory first adopts the distance function $d(\cdot)$ to calculate the *similarity* between states, then uses similar states to calculate $\mathcal{R}_{mem}(\cdot)$. After that, the replay memory proposes two strategies to improve the accuracy of $\mathcal{R}_{mem}(\cdot)$, i.e., priority mechanism that samples states biasedly, and *storage mechanism* that improves the quality of states saved in the replay memory.

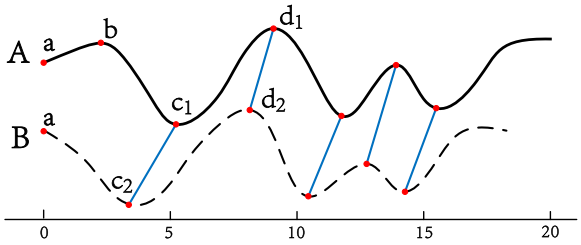

Fig. 3. Two similar trajectory embeddings.

Distance Function.

The similarity between two states can be considered from two aspects: the similarity of attribute embeddings $\mathbf{H}(\cdot)$ and the similarity of trajectory embeddings $\mathbf{T}(\cdot)$. For the first part, A-MCTS adopts the *cosine distance* to approximate the difference of $\mathbf{H}(\cdot)$ between two states as follows:

$$d_{att}(\mathbf{H}(P_i), \mathbf{H}(P_j)) = \cos(\mathbf{H}(P_i), \mathbf{H}(P_j)). \tag{11}$$

For the similarity of trajectory embeddings, it is necessary to identify similar trajectories when they have different trajectory lengths. Fig. 3 shows this with a simple example. $\mathbf{T}(A) = [\mathbf{a}, \ldots, \mathbf{b}, \ldots, \mathbf{c_1}, \ldots, \mathbf{d_1}, \ldots]$ and $\mathbf{T}(B) = [\mathbf{a}, \ldots, \mathbf{c_2}, \ldots, \mathbf{d_2}, \ldots]$ are trajectory embeddings of states A and B, $(\mathbf{c_1}, \mathbf{c_2})$, $(\mathbf{d_1}, \mathbf{d_2})$ are two pairs of similar edge embeddings. Although edge b is selected at the beginning of the search process of A, $\mathbf{T}(A)$ and $\mathbf{T}(B)$ have an overall similar shape, which means the selection of b has little effect on the following search process. Hence, $\mathbf{T}(A)$ and $\mathbf{T}(B)$ should have a high degree of similarity. However, in this case, a distance measure that assumes the ith edge embedding on $\mathbf{T}(A)$ is aligned with ith edge embedding on $\mathbf{T}(B)$ will produce a pessimistic dissimilarity, e.g., cosine similarity. It should be noted that if other simple path embedding methods are adopted, e.g., sum/mean/max of edge embeddings, it is easy to calculate trajectory similarity but the structural hierarchy information of trajectories will be lost. For example, if the path embedding is set as mean/max of edge embeddings, for two similar path embeddings, whether the two paths contain similar number of edges cannot be judged. Setting path embedding as sum of edge embeddings can overcome the problem, but it is not easy to determine whether the two paths have similar substructure, i.e., some edges in the two paths have similar embeddings. In order to solve the problem, A-MCTS adopts *dynamic time warping* (DTW) [32] to calculate the similarity of trajectory embeddings. DTW is an effective method to measure the similarity between two sequences that have different number of elements. The differences between two trajectory embeddings are approximated by using DTW as in Eq. (12).

$$d_{traj}(\mathbf{T}(P_i), \mathbf{T}(P_j)) = \min_{\mathbf{W}(\mathbf{T}(P_i), \mathbf{T}(P_j))} \frac{\sqrt{\sum_{l=1}^{L} W_l(\mathbf{T}(P_i), \mathbf{T}(P_j))}}{L}, \tag{12}$$

where $\mathbf{W}(\mathbf{T}(P_i), \mathbf{T}(P_j))$ is the *warping path* [33] of $\mathbf{T}(P_i)$ and $\mathbf{T}(P_j)$, which is a contiguous set of elements that defines a mapping between two trajectory embeddings. Each element $W_l(\mathbf{T}(P_i), \mathbf{T}(P_j))$ in $\mathbf{W}(\mathbf{T}(P_i), \mathbf{T}(P_j))$ is the euclidean distance between two edge embeddings in $\mathbf{T}(P_i)$ and $\mathbf{T}(P_j)$ respectively.

A-MCTS adopts the variant of DTW, fast-DTW [34], which utilizes some acceleration methods and can reach the $O(n)$ time complexity, n is the number of elements in the shorter sequence.

Then the distance function can be defined as in Eq. (13).

$$\begin{aligned} d(P_i, P_j) = {} & \alpha \times d_{att}(\mathbf{H}(P_i), \mathbf{H}(P_j)) \\ & + \gamma(1-\alpha) \times d_{traj}(\mathbf{T}(P_i), \mathbf{T}(P_j)), \end{aligned} \tag{13}$$

where α is a constant to balance two functions, γ is used to keep the range of the two distance functions consistent.

Approximate Reward Calculation.

In order to calculate $\mathcal{R}_{mem}(\cdot)$ for a *query* of P_c, A-MCTS first samples entries from D. Then, in the sampled entries, A-MCTS finds the top m similar states based on the distance function $d(\cdot)$. The approximate reward of P_c is then computed by Eq. (14).

$$\mathcal{R}_{mem}(P_c) = \sum_{i=1}^{m} \omega_i(P_c) \mathcal{R}_{avg}(P_i), \tag{14}$$

where P_i is the ith similar state of P_c, and $\omega_i(P_c)$ is the weight of P_i that is calculated by Eq. (15).

$$\omega_i(P_c) = \frac{N(P_i)\exp(-d(P_i, P_c))}{\sum_{j=1}^{m} N(P_j)\exp(-d(P_j, P_c))}. \tag{15}$$

In Eq. (15), A-MCTS gives higher weights to states that have more visited times and higher similarities.

Priority Mechanism.

The priority mechanism sets $PR(\cdot)$ to sample states in D effectively. As mentioned in Problem 1 in Section 1, if randomly sampling entries in D to calculate $\mathcal{R}_{mem}(\cdot)$, $\mathcal{R}_{mem}(\cdot)$ is inaccurate under some conditions. This is because when constraints are strict and search time is limited, there are only a few "positive states" (i.e., states that are close to feasible temporal paths) can be searched and stored in D. As a result, in the replay memory, most states are "negative states" (i.e., states are close to infeasible temporal paths) and have low values of $\mathcal{R}_{avg}(\cdot)$. As $\mathcal{R}_{mem}(\cdot)$ is calculated based on the $\mathcal{R}_{avg}(\cdot)$ of sampled entries, random sampling makes $\mathcal{R}_{mem}(\cdot)$ smaller than true reward, i.e., $\mathbb{E}(\mathcal{R}(\cdot))$. In order to solve the problem, A-MCTS sets a "priority value" (denoted as $PR(\cdot)$) for each state in D. $PR(\cdot)$ is set in the range of [0, 1]. The larger the $PR(P)$, the higher the sampling priority of P. When a new state P_c is saved in D, the initial value of $PR(P_c)$ is set to 0. The priority value $PR(P_c)$ will be updated after visiting P_c again.

$$PR(P_c) = 1 - \frac{|\mathcal{R}_{avg}(P_c) - \mathcal{R}'(P_c)|}{\max(\mathcal{R}_{avg}(P_c), \mathcal{R}'(P_c))}, \tag{16}$$

where $\mathcal{R}'(P_c)$ is the reward of P_c in the current MDP. If $\mathcal{R}'(P_c)$ and $\mathcal{R}_{avg}(P_c)$ are similar, it means that $\mathcal{R}_{avg}(P_c)$ is relatively stable and $PR(P_c)$ will be larger. As positive states usually have high values of $\mathcal{R}_{avg}(\cdot)$, they tend to have larger $PR(\cdot)$. Thus, when constraints are strict, A-MCTS is able to ensure that a certain part of positive states can be sampled. A-MCTS can also filter out negative states with low $\mathcal{R}_{avg}(\cdot)$ when sampling. In addition, for the same state P, as $\mathcal{R}'(P)$ changes in different MDPs, the priority value of the same

state is not fixed. This ensures A-MCTS will not always sample the same states when computing $\mathcal{R}_{mem}(\cdot)$. The sampling probability of state in D are defined as below:

$$\mathcal{P}_{sample}(P_c) = \frac{PR(P_c) \cdot \ln(|\mathcal{A}(P_c)|)}{\sum_{j=1}^{|D|} PR(P_j) \cdot \ln(|\mathcal{A}(P_j)|)}, \tag{17}$$

here $\ln(|\mathcal{A}(P_c)|)$ is used to ensure states that have stable $\mathcal{R}_{avg}(\cdot)$ under more actions tend to have larger sampling probabilities.

Storage Mechanism.

The storage mechanism improves the quality of states stored in D to increase the accuracy of $\mathcal{R}_{mem}(\cdot)$. As mentioned in Problem 2 in Section 1, saving states with fewer edges may not be useful, because these states have limited attribute information, and their simple structures can be found in a lot of states. Thus, rewards of fewer-edge state have a large variance and it is not easy to get stable $\mathcal{R}_{avg}(\cdot)$. If a large number of these states are stored in D, the accuracy of $\mathcal{R}_{mem}(\cdot)$ will be influenced. By contrast, the state with more edges contains more attribute information and a complex structure, which only exists in some certain structures. So saving states with more edges can improve the accuracy of $\mathcal{R}_{mem}(\cdot)$.

In order to solve the problem, the replay memory needs to *store* states with more edges. Hence, A-MCTS sets the "limitation of edge number" which is denoted as E_{lim}. The replay memory is used for states whose number of edges is more than E_{lim}, which means that information of shorter paths is not *stored* in the replay memory. In contrast, the replay memory will try to *store* the states with more edges. For states whose edge numbers are more than E_{lim} and are visited for the first time, each state has a probability to be *stored* in D, which is defined as follows:

$$\mathcal{P}_{store}(P) = 1 - \frac{\sqrt{E_{lim}}}{|P|_{edge}}, \tag{18}$$

where $|P|_{edge}$ is the number of edges of P. The more the edges, the higher the preservation probability. This random storage method also cuts off the correlation between consecutive states that are in the same MDP. In addition, a threshold θ is set to ensure all entries in D meet $|\mathcal{R}_{avg}(\cdot)| \geq \theta$. This is because the state has little effect on calculating $\mathcal{R}_{mem}(\cdot)$ when the average reward of the state is close to 0.

As states with more edges usually have complex information, i.e., longer trajectories, when perform *query* for these states, they require sampling more entries in D to get accurate $\mathcal{R}_{mem}(\cdot)$. Therefore, A-MCTS sets the sampling number for *query* states, i.e., $SN(\cdot)$, which is proportional to the edge number of query state:

$$SN(P) = SN_{max}\left(1 - \frac{\sqrt{E_{lim}}}{|P|_{edge}}\right), \tag{19}$$

where SN_{max} is the maximum sample number.

4.3.3 Update

The *update* operation will be performed when the current MDP reaches a terminal state. Each state P_c in the MDP will be *updated* in two ways: (1) if P_c is not saved in D, the replay memory performs the *store* operation and store $\{\mathbf{H}(P_c), \mathbf{T}(P_c), \mathcal{R}_{avg}(P_c), N(P_c), PR(P_c)\}$ of P_c into D; (2) if P_c has already been stored in D, the replay memory updates $\mathcal{R}_{avg}(P_c)$, $N(P_c)$ and $PR(P_c)$ at the corresponding entry.

4.4 Updated Action-value function

As introduced in 4.2, the ideal action value function is $Q(P_i, e) = \mathbb{E}(\mathcal{R}(P_j)) - \mathbb{E}(\mathcal{R}(P_i)), P_i \xrightarrow{e} P_j$. And $\mathcal{R}_{avg}(P)$ can replace $\mathbb{E}(\mathcal{R}(P))$ when $N(P)$ is large enough, but it is not realistic under large search space and limited search time. As the replay memory can be used to approximate the long-term reward (i.e., $\mathcal{R}_{mem}(\cdot)$) of a state even the state hasn't been visited, A-MCTS proposes an updated action-value function $Q_U(\cdot)$ that combines $\mathcal{R}_{mem}(\cdot)$ and $\mathcal{R}_{avg}(\cdot)$ to overcome the limitation of $Q(\cdot)$.

$Q_U(\cdot)$ assigns different weights to $\mathcal{R}_{mem}(\cdot)$ and $\mathcal{R}_{avg}(\cdot)$. As the visited time of P that makes $\mathbb{E}(\mathcal{R}(P))$ and $\mathcal{R}_{avg}(P)$ closer (denoted as $N_{\mathbb{E}}(P)$) is proportional to the number of MDPs that contain P, i.e., $N_{\mathbb{E}}(P) \propto |\mathcal{A}(P)|$, the weight of $\mathcal{R}_{mem}(P)$ should become smaller when $N(P)$ is close to $|\mathcal{A}(P)|$. Therefore, the weight $\tau(P)$ of $\mathcal{R}_{mem}(P)$ can be defined as in Eq. (20).

$$\tau(P) = \frac{1}{\log_{|\mathcal{A}(P)|}|\mathcal{A}(P)|(1 + N(P))}. \tag{20}$$

When $N(P) \approx |\mathcal{A}(P)|$, $\tau(P) \approx 0.5$, which means $\mathcal{R}_{mem}(\cdot)$ and $\mathcal{R}_{avg}(\cdot)$ are equally credible at this time. As there may exist repeated action selections in the search process, $N_{\mathbb{E}}(P)$ is much larger than $|\mathcal{A}(P)|$. The log function is used to ensure $\tau(P)$ becomes small when $N(P)$ is large enough. Then, $Q_U(\cdot)$ can be defined as in the below Eq. (21).

$$\begin{aligned} Q_U(P_i, e) = {} & \tau(P_j)\mathcal{R}_{mem}(P_j) + (1 - \tau(P_j))\mathcal{R}_{avg}(P_j) \\ & - (\tau(P_i)\mathcal{R}_{mem}(P_i) + (1 - \tau(P_i))\mathcal{R}_{avg}(P_i)), P_i \xrightarrow{e} P_j \end{aligned} \tag{21}$$

At first, P_j has not been visited and $N(P_j) = 0$, then $\tau(P_j) = 1$ and A-MCTS only uses $\mathcal{R}_{mem}(P_j)$ to calculate $Q_U(P_i, e)$. The weight of $\mathcal{R}_{avg}(P_j)$ becomes larger when $N(P_j)$ increases.

4.5 Monte Carlo Tree Search

This section introduces how to combine $Q_U(\cdot)$ and $V(\cdot)$ to search feasible temporal paths. A-MCTS first builds a tree structure and models the temporal path discovery as a tree search process. Then, A-MCTS proposes a new tree search policy that combines $Q_U(\cdot)$ and $V(\cdot)$ to guide the search process, which explores low-priority states to obtain more feasible temporal paths. Lastly, A-MCTS performs Monte Carlo tree search on the tree structure.

4.5.1 Tree Structure

In the tree structure of A-MCTS, each node in the tree represents a state, while the tree's edges correspond to actions. A-MCTS grows the tree structure iteratively. With each iteration, the tree structure is traversed and expanded. The root node in the tree represents the starting state, i.e., the source node v_s; a child node represents the state that is transformed from its ancestor's state after tacking an action from $\mathcal{A}(\cdot)$, the edge between them indicates the corresponding action; a leaf node represents a terminal state or a newly expanded

state. Each node in the tree structure stores the set of statistics: $\{V(\cdot), N(\cdot), \mathcal{R}_{avg}(\cdot), \mathcal{R}_{mem}(\cdot)\, PR(\cdot)\}$. In this way, an MDP can be regarded as an iteration of A-MCTS, which starts from the root node, selects child nodes recursively, and finally reaches a leaf node.

4.5.2 Upper Confidence bounds applied to Tree

Upper Confidence bounds applied to Tree (UCT) was originally proposed by [18] to balance the exploration versus exploitation in Monte Carlo tree search. In A-MCTS, the exploration approach promotes the exploration of unexplored areas (i.e., leaf nodes that are unvisited states) in the tree structure. It means that the exploration will expand the tree's breath more than its depth. Exploitation tends to stick to actions that have the greatest estimated values (i.e., the maximal action value $Q_U(\cdot)$), this approach will extend the tree's depth more than its breadth. The balance of exploration and exploitation can ensure our algorithm does not overlook any potential feasible temporal paths, thus avoiding the ineffectiveness of the search under a large state space.

Algorithm 1. A-MCTS

Input: ADG $\mathcal{G}$; Source node v_s, Destination node v_d; Time interval $[t_\phi, t_\varphi]$; Constraint set $\mathcal{X}$; Number of iterations I; Replay memory D;
for iteration it in $[1..I]$ **do**
  Set v_s as the root node of the tree structure;
  Set the current state $P_{v_s,v_c}(t^c_{arr})$; $v_c \leftarrow v_s, t^c_{arr} \leftarrow t_\phi$;
  while $t^c_{arr} \le t_\varphi$ **do**
    if $P_{v_s,v_c}(t^c_{arr})$ is unvisited **then**
      Mark $P_{v_s,v_c}(t^c_{arr})$ visited;
      if $\mathcal{A}(P_{v_s,v_c}(t^c_{arr}))$ is $\emptyset$ **then**
        Break;
      for each edge
      $e_i = (v_c, v_i, l_c, t^c_{dep}, t^i_{arr}) \in \mathcal{A}(P_{v_s,v_c}(t^c_{arr}))$
      do
        Add $P_{v_s,v_i}(t^i_{arr})$ as a child of $P_{v_s,v_c}(t^c_{arr})$;
        Initial $\{V, N, \mathcal{R}_{avg}, \mathcal{R}_{mem}, PR\}$ of $P_{v_s,v_i}(t^i_{arr})$;
        Mark $P_{v_s,v_i}(t^i_{arr})$ unvisited;
    else
      Put $P_{v_s,v_c}(t^c_{arr})$ into the *UCT function* and get the action $e_a = (v_c, v_a, l_c, t^c_{dep}, t^a_{arr})$;
      Add the transformed state $P_{v_s,v_a}(t^a_{arr})$ into the state list;
      Set current state $P_{v_s,v_c}(t^c_{arr}) \leftarrow P_{v_s,v_a}(t^a_{arr})$;
      if $P_{v_s,v_c}(t^a_{arr})$ is a terminal state **then**
        Break;
  Compute $\mathcal{R}(P_{v_s,v_c}(t^c_{arr}))$;
  for each state $P_{v_s,v_i}(t^i_{arr})$ in the state list **do**
    Update $\{V, N, \mathcal{R}_{avg}, \mathcal{R}_{mem}, PR\}$ of $P_{v_s,v_i}(t^i_{arr})$ in the tree structure;
    Perform the *update* operation for $P_{v_s,v_i}(t^i_{arr})$ in D.

Specifically, in each iteration of A-MCTS, an MDP is rolled out by selecting actions according to the proposed variant of UCT algorithm [18] from the root node (defined in Eq. (22)).

$$e_a = \arg\max_{e_j} \left\{ c\left(\frac{V(P_j)}{\sum_{k=1}^{|\mathcal{A}(P_i)|} V(P_k)} + \epsilon(1 - PR(P_j)) \right) \times \sqrt{\frac{\ln(N(P_i))}{1 + N(P_j)}} + Q_U(P_i, e_j) \right\}, e_j \in \mathcal{A}(P_i), P_i \xrightarrow{e_j} P_j \tag{22}$$

where c is the constant that controls the level of exploration, P_i is the current state that needs to select an action from $\mathcal{A}(P_i)$, P_j is transformed from P_i after taking the action e_j, P_k is transformed from P_i after taking the kth action from $\mathcal{A}(P_i)$, i.e., $P_i \xrightarrow{e_k} P_k$. In this UCT function, in order to explore the low-priority states, priority values of states are also considered. A low-priority state means (1) the state is just added into D and initialized, without performing two or more visits; (2) starting from the low-priority state, there are multiple different states, and the search results obtained from these states may have a big difference. Therefore, visiting and updating these low-priority states can ensure A-MCTS will not overlook any potential feasible temporal paths. So A-MCTS considers the priority value in the exploration part of the UCT function, and ϵ is a constant that controls the weight of the priority value.

Overall, UCT function initially prefers actions that can transform the current state into the states with high state values $V(\cdot)$, low visit counts $N(\cdot)$ and low priority values $PR(\cdot)$, but then asymptotically prefers the actions with high action values $Q_U(\cdot)$.

4.5.3 Iteration Details of Adaptive Monte Carlo Tree Search

Each iteration of A-MCTS can be divided into four steps: *selection*, *expansion*, *simulation* and *backpropagation*. These four steps are iteratively applied until the maximum number of iterations is reached. The four steps are discussed below and the corresponding pseudo-code is shown in Algorithm 4.5.2.

Step 1: Selection. In the selection step, starting from the root node, A-MCTS traverses the current tree structure using a tree policy that prioritizes the actions with the greatest values of UCT function. When the traversal reaches a leaf node, if the leaf node is a terminal state, A-MCTS transfers to the backpropagation step; otherwise, it means the leaf node is a newly expanded state that has children left to be added, then A-MCTS transfers to the expansion step. In Fig. 4, starting from the root node A, the tree policy needs to make a decision among B, C and D. Suppose A-MCTS chooses $(A \rightarrow B)$ and reaches B. Since B is a leaf node with children yet to be added, A-MCTS transfers to the expansion step.

Step 2: Expansion. In the expansion step, the search process reaches an expanded node. A-MCTS selects all actions of the expanded node, adds the transformed states as child nodes of the expanded node into the tree structure, and initializes the statistics of them, i.e., $\{V(\cdot), N(\cdot) = 0, \mathcal{R}_{avg}(\cdot) = 0, \mathcal{R}_{mem}(\cdot), PR(\cdot) = 0\}$. After that, A-MCTS moves to the simulation step. If there is no actions for the expanded node, it means the expanded node is a terminal state, then A-MCTS transfers to the backpropagation step. In Fig. 4, the algorithm is currently at the expanded node B. B has two actions, i.e., $\mathcal{A}(B) = \{B \rightarrow E, B \rightarrow F\}$, so there are two child nodes added into B and indicated by E and F, then A-MCTS transfers to the simulation step.

Step 3: Simulation. In this step, A-MCTS performs selection and expansion repeatedly until reaching a terminal state and getting a reward, then transfers to the backpropagation step. In Fig. 4, the child nodes of B have been added into the tree structure. A-MCTS performs the above two

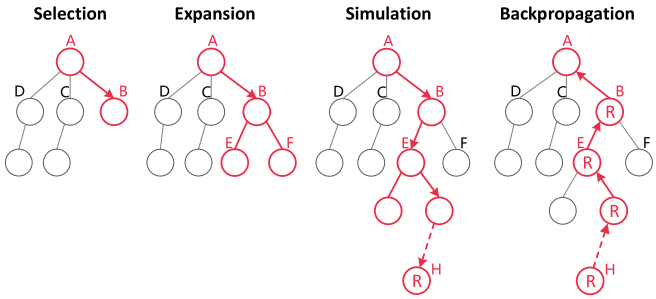

Fig. 4. A brief illustration of A-MCTS process.

steps repeatedly until reaching the terminal state H and gets reward R. Then, A-MCTS moves to the backpropagation step.

Step 4: Backpropagation. Now that the A-MCTS has reached a terminal state and got the corresponding reward, the rest of the tree structure must be updated. Starting at the node of the terminal state, A-MCTS traverses back to the root node. During the traversal, the statistics stored in each node of the traversal (i.e., $\{V(\cdot), N(\cdot), \mathcal{R}_{avg}(\cdot), \mathcal{R}_{mem}(\cdot), PR(\cdot)\}$) are updated, $N(\cdot)$ and $\mathcal{R}_{avg}(\cdot)$ are updated as follows:

$$N(P) \leftarrow N(P) + 1,$$

$$\mathcal{R}_{avg}(P) \leftarrow \mathcal{R}_{avg}(P) + \frac{\mathcal{R}'(P) - \mathcal{R}_{avg}(P)}{N(P)},$$

where $\mathcal{R}'(P)$ is the reward of P in the current MDP. After that, the replay memory D will perform the *update* operation to update statistics saved in D.

A-MCTS performs the above four steps for I iterations. So its time complexity is $O(I \times (\textit{Selection} + \textit{Expansion} + \textit{Simulation} + \textit{Backpropagation}))$. Let F denote the depth of the tree structure, U denote the average number of child nodes at each layer of the tree structure, SN_{avg} denote the average number of entries sampled in the replay memory. From Eq. (19) we can see $SN_{avg} < SN_{max}$. The *Selection* step has the time complexity of $O(U)$, the *Expansion* step has the time complexity of $O(U \cdot SN_{avg})$, the *Simulation* step has the time complexity of $O(U \cdot SN_{avg})$, and the *Backpropagation* step has the time complexity of $O(F \cdot SN_{avg})$. Therefore, the time complexity of A-MCTS is $O((U+F)I \cdot SN_{avg})$.

The time complexity of RL-MCTS [16] is $O((U+F)I \cdot N_D)$, where N_D is a fixed number of entries that are sampled in the replay memory. In A-MCTS, $SN_{max} = N_D$, thus $N_{avg} < N_D$ and the complexity of A-MCTS is better than RL-MCTS. In addition, in RL-MCTS, all states in the search process will use replay memory to calculate their approximate rewards. However, in A-MCTS, the replay memory is not used for the state whose number of edges is less than E_{lim}. Therefore, the time complexity of A-MCTS is better than RL-MCTS.

5 Experiments

In this section, we conduct experiments on ten real-world graphs to investigate: (1) Whether A-MCTS can effectively find the shortest temporal paths that meet multiple constraints; (2) Whether A-MCTS can greatly reduce the overall query processing time.

TABLE 3
Datasets

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert d_{avg}\rvert$	$\lvert d_{max}\rvert$	$\lvert m_{edge}\rvert$	$\lvert T\rvert$
Arxiv	28K	4597K	327.3	11K	262	2K
Elec	7K	104K	29.1	1K	15	101K
Enron	87K	1148K	26.3	38K	1074	213K
Slash	51K	141K	5.5	3K	17	90K
Digg	280K	1732K	12.4	12K	25	83K
Conflict	118K	2918K	49.4	136K	562	274K
Youtube	3223K	9375K	9.9	5K	327	921K
Opsahl	2K	60K	63.1	1K	98	58K
Openflights	12K	67K	11.15	1K	20	200
Tasmania	25K	9783K	793.68	3K	3070	1.5K

5.1 Datasets

We first conducted experiments on eight large-scale real-world attribute dynamic graphs that are available at *konect.uni-koblenz.de*. These datasets have been widely used in the literature for the studies of dynamic graphs[30]. In addition, we conducted experiments on another two datasets, i.e., an airports dataset Openflights[1] that contains 12,000 airports, and 67,000 routes, and a transportation dataset Tasmania[2] that contains the metro information of Tasmania in Australia. Details of datasets are shown in Table 3, including the number of nodes and edges, the average degree (d_{avg}) and the maximal degree (d_{max}), the maximal number of the edges between two nodes (m_{edge}), and the number of time points (T) of each attribute dynamic graphs.

5.2 Parameter settings

- For each ADG, we selected 10 pairs of nodes which have the highest temporal in-degree and out-degree as a source node and a destination node, respectively.
- The discount factors in state-value function (i.e., Eq. (5)) are set as $\delta = 0.95$ and $\mu = 0.9$. In Eq. (22), c is set to 0.45 and ϵ is set to 0.3. Table 4 is the variation of main parameters. The rest parameters of TPA and RL-MCTS are consistent with the settings in [30] and [16], respectively. A-MCTS uses the same parameters as RL-MCTS.
- The length of each edge is normalized to the range of $[0-100]$. In general, temporal paths consisting of fewer edges have shorter path lengths and smaller aggregated attribute values, however, these paths typically contain less information and therefore cannot extract valid features. In order to ensure that temporal paths contain sufficient features, in the experiment, we limited the minimum number of edges that each temporal path contains (denoted as N_m). N_m is set as $\{10, 15, 20, 25, 30\}$, to evaluate the effectiveness of using the memory replay. Accordingly, for the constraint set $\mathcal{X}$, the constraint on jth attribute is set as $\lambda_j = \{\frac{1}{5}N_m f_j^a, \frac{2}{5}N_m f_j^a, \frac{3}{5}N_m f_j^a, \frac{4}{5}$

1. openflights.org/data
2. code.google.com/archive/p/googletransitdatafeed/wikis/PublicFeeds.wiki

TABLE 4
Parameter Settings

Parameters	Settings
Number of iterations	1000, 2000, 3000, 4000, 5000
Number of attributes	5, 10, 15, 20, 25
Maximum size of replay memory ($\|D\|$)	500, 1000, 1500, 2000, 3000
Maximum sample number (SN_{max})	100, 200, 300, 400, 500
Limitation of edge number (E_{lim})	1, 2, 4, 6, 8
Time interval	$[0, \frac{1}{5}\|T\|]$, $[0, \frac{2}{5}\|T\|]$, $[0, \frac{3}{5}\|T\|]$, $[0, \frac{4}{5}\|T\|]$, $[0, \|T\|]$

TABLE 5
Parameter Settings of Replay Memory Under 4 Common Query Situations

Situation	Number of attributes	Minimum number of edges (N_m)	Constraint of attributes	Constraint of time interval
Insuff-Relax	10	5	$N_m f_j^a$	$[0, \|T\|]$
Insuff-Strict	10	5	$2/5 N_m f_j^a$	$[0, 2/5\|T\|]$
Suff-Relax	20	15	$N_m f_j^a$	$[0, \|T\|]$
Suff-Strict	20	15	$2/5 N_m f_j^a$	$[0, 2/5\|T\|]$

$N_m f_j^a, N_m f_j^a\}$. f_j^a is the average value of jth attribute function.

- The replay memory has three main parameters: the size of replay memory $|D|$, the limitation of edge number E_{lim}, and the maximum sample number SN_{max}. A-MCTS may achieve better performance with larger $|D|$ and SN_{max}. However, it also leads to larger resource cost, i.e., more query time and more storage space. In practice, it is necessary to trade off performance improvement with resource cost. Therefore, we construct four common query situations and discuss the appropriate parameter settings of replay memory in each query situation through experiments. Specifically, we consider the query situation from two aspects: (1) Whether the features of temporal paths in the ADG are sufficient. Sufficient features of temporal paths mean a larger number of attributes in the ADG and a larger number of edges in temporal paths; (2) Whether the constraints are strict. Detailed settings can be seen at Table 5, where "Insuff-Strict" stands for the query situation that features are limited and the constraints are strict.

5.3 Comparison Methods

In the following experiments, we will compare our A-MCTS with one state-of-the-art algorithm for MCTS, i.e., reinforcement learning based Monte Carlo tree search algorithm, RL-MCTS [16], and the state-of-the-art temporal path discovery method, TPA [30]. The performance is indictated by the execution time, and the length of the delivered temporal path.

All TPA, RL-MCTS and A-MCTS algorithms are implemented using Python running on a server with 44 CPU kernels, 900GB RAM, Linux operating system and MySql 5.7 database.

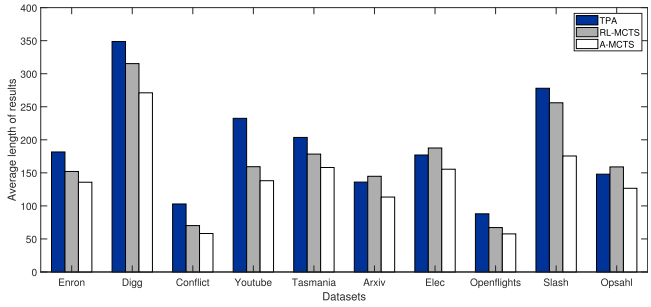

Fig. 5. The average path length based on different datasets.

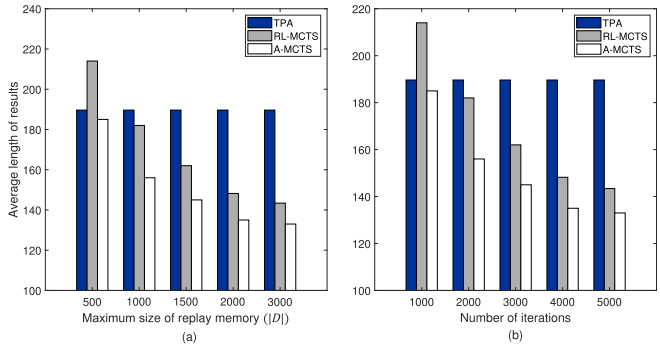

Fig. 6. The average path length based on different iterations and size of memory.

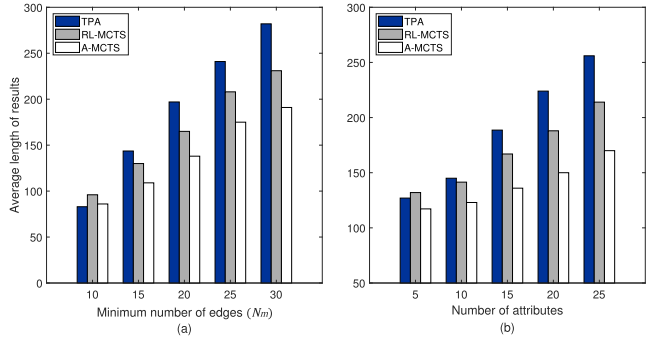

Fig. 7. The average path length based on different attributes and minimum number of edges.

5.4 Experimental Results and Analysis

Exp-1: Effectiveness. In order to investigate the effectiveness of A-MCTS, we compare the lengths of the identified paths based on different datasets and parameters. In addition, we investigate the effectiveness of adaptive replay memory structure by comparing the search results based on the different numbers of E_{lim} and SN_{max}.

Results. Figs. 5, 6, 7, and 8 depict the average path length delivered by TPA, RL-MCTS and A-MCTS, respectively, on different datasets. From these figures, we can see that (1) in all the cases, A-MCTS can obtain shorter paths than RL-MCTS and TPA. Overall, the average path length delivered by A-MCTS is *17.74%* shorter than that of RL-MCTS and *26.71%* shorter than that of TPA, respectively; (2) A-MCTS can achieve promising performance under limited exploration resources, i.e., around 1000 iterations, 500 entries of replay memory size and 100 of maximum sample number SN_{max}. By contrast, RL-MCTS requires nearly twice the exploration resources to achieve the same performance as

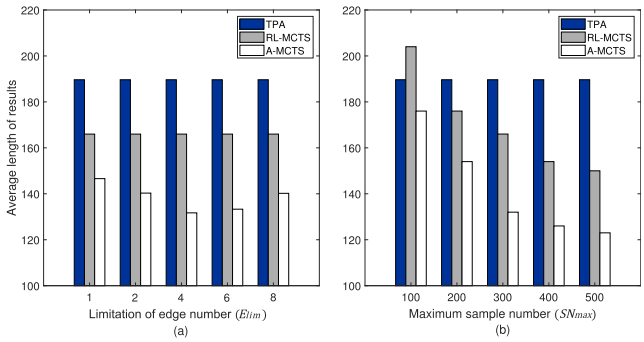

Fig. 8. The average path length based on different maximum sample number (SN_{max}) and limitation of edge number (E_{lim}).

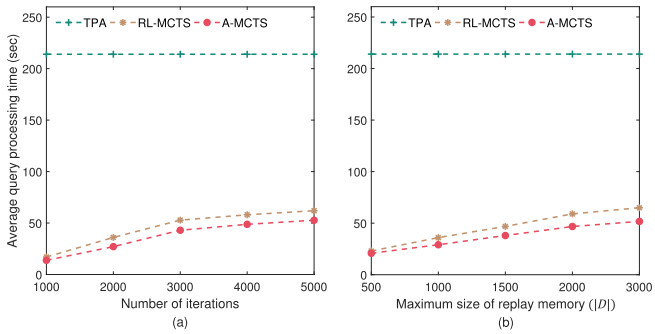

Fig. 10. The average query processing time based on different iterations and size of memory.

A-MCTS; (3) when the features of temporal paths become sufficient, i.e., larger N_m and more attributes, the performance of TPA and RL-MCTS degrades a lot. By contrast, A-MCTS is more stable with small fluctuations; (4) when the number of iterations and the size of replay memory increases, the performance difference between A-MCTS and RL-MCTS is gradually reduced; (5) setting appropriate E_{lim} can improve the performance of A-MCTS, E_{lim} around 4 is ideal in A-MCTS.

Analysis. The experimental results illustrate that (1) A-MCTS can avoid the tree search via a sub path that has larger attribute values and longer path length to satisfy the constraints, rather than greedily optimizing the objectives in TPA, thus A-MCTS has a higher probability to obtain shorter paths than TPA; (2) A-MCTS can obtain more accurate approximate rewards by updating the states with low priority values and sampling the states with higher priority values. In addition, when processing longer path queries, the storage mechanism ensures A-MCTS will not be affected by the limited features of shorter paths. Therefore, A-MCTS requires fewer exploration resources (i.e., fewer iterations, the smaller size of replay memory and smaller SN_{max}) than RL-MCTS to achieve better performance; (3) when features of temporal paths become sufficient, i.e., larger N_m and more attributes, A-MCTS uses the adaptive replay memory structure to contract more important features of states and gets more accurate action values, leading to more stable results than TPA and RL-MCTS; (4) when the number of iterations is small, most of the visited states are invalid because of the sparse reward. A-MCTS uses the storage mechanism to avoid storing these invalid states, and uses the priority mechanism to capture important states and guide the search process. By contrast, RL-MCTS does not filter visited states and randomly samples states of them, and thus the search process of RL-MCTS may be misled by these invalid states. With the increase of iterations and the size of replay memory, the number of valid states that are saved in the replay memory increases. Thus, the performance difference between the two methods becomes narrow gradually; (5) setting smaller E_{lim} cannot filter invalid states, whereas a larger E_{lim} may ignore important states. Thus, it is necessary to select an appropriate E_{lim}.

Exp-2: Efficiency. This experiment is to investigate the efficiency of A-MCTS by comparing the average query processing time of TPA, RL-MCTS and A-MCTS based on the different datasets and parameters. In addition, we investigate the efficiency of adaptive replay memory structure by comparing the search results based on the different numbers of E_{lim} and SN_{max}.

Results. Figs. 9, 10, 11, and 12 depict the average query processing time of TPA, RL-MCTS and A-MCTS, respectively, on different datasets. From these figures, we can see that (1) in all the cases, the query processing time of A-MCTS is less than that of TPA and RL-MCTS; (2) on average, A-MCTS can save *84.20%* of the query processing time compared with TPA and *22.57%* of the query processing time compared with RL-MCTS, respectively. The statistics are shown in Table 6; (3) TPA suffers from large datasets, e.g., Conflict and Youtube. By contrast, A-MCTS and RL-MCTS save query processing time on these datasets to a large extent; (4) when the exploration resources become sufficient, i.e., more iterations, the larger size of replay memory and larger maximum sample number SN_{max}, A-MCTS saves more query processing time than RL-MCTS; (5) when the features of temporal paths become sufficient, i.e., larger N_m

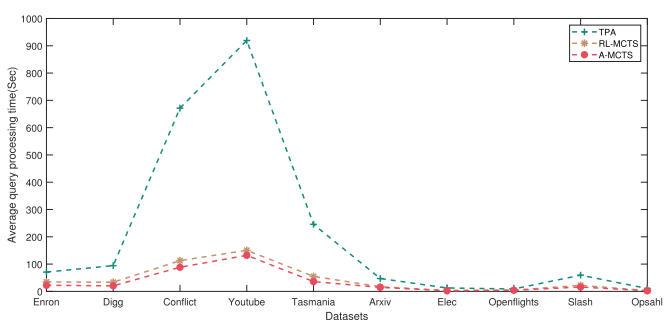

Fig. 9. The average query processing time based on different datasets.

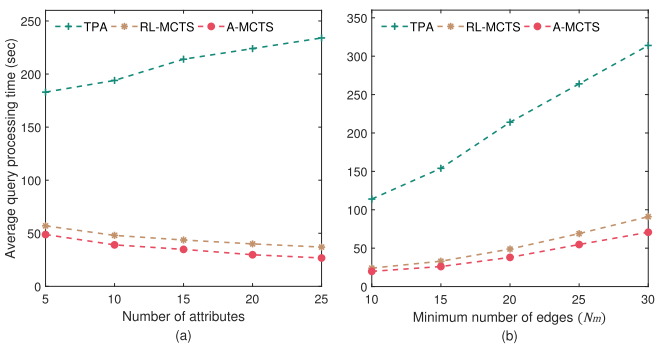

Fig. 11. The average query processing time based on different attributes and minimum number of edges.

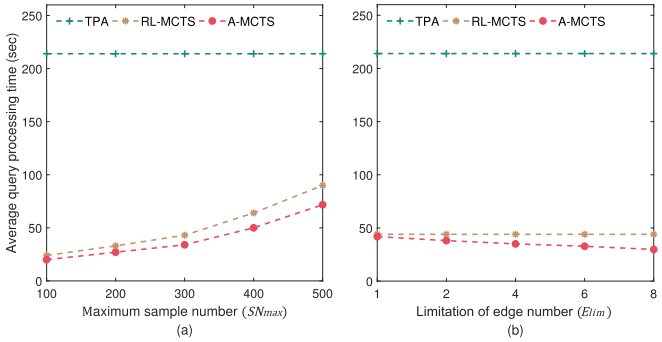

Fig. 12. The average query processing time based on different maximum sample number (SN_{max}) and limitation of edge number (E_{lim}).

TABLE 6
The Comparison of the Performance of TPA, RL-MCTS and A-MCTS

Comparison	TPA	RL-MCTS	A-MCTS
The average path length	189.6431	168.9137	138.9972
The query processing time (Sec)/Path	213.9152	43.6527	33.8056

and more attributes, A-MCTS consumes less query processing time than RL-MCTS and TPA.

Analysis. The experimental results illustrate that (1) A-MCTS and RL-MCTS access only part of nodes that are connected with the destination node, while TPA needs to visit all nodes connected with the destination node, leading to too much query processing time, especially on large datasets; (2) in A-MCTS, the operations of replay memory (i.e., *store*, *update* and *query*) need to be performed for visited states whose number of edges is more than E_{lim}. By contrast, RL-MCTS has to perform the three operations for all visited states. Therefore, A-MCTS saves more query processing time than RL-MCTS; (3) with the increase of iterations, A-MCTS and RL-MCTS traverse more unvisited states and consume more time on expanding the tree structure. Since A-MCTS uses the UCT function to select important states only, A-MCTS builds a smaller tree structure than RL-MCTS, which can save more query processing time; (4) with the increase of the replay memory size and SN_{max}, A-MCTS and RL-MCTS take more time on the operations of replay memory D, i.e., *store* more states in D and use more states to *query*. A-MCTS adopts the storage mechanism to avoid storing invalid states. By contrast, RL-MCTS does not filter candidate states and *stores* a lot of invalid states, and thus consumes more query processing time; (5) with the increase of N_m, A-MCTS and RL-MCTS need to process states that contain more complex information, i.e., long trajectories with more nodes. The storage mechanism ensures A-MCTS will not process some shorter paths that contain limited features. Therefore, A-MCTS saves more query processing time than RL-MCTS; (6) with the increase of the number of attributes, the constraints become strict, and the number of feasible temporal paths decreases. A-MCTS utilizes the UCT function to avoid tree search via potential unfeasible temporal paths, thus saves more query processing time than RL-MCTS and TPA.

Exp-3: Statistical Test.

In order to test the significance of our method's results, we perform a statistical test on the average experimental results of 10 query pairs (query node and target node) of each dataset. As the distribution of results of 10 query pairs cannot be assumed to be normally distributed, we adopt Wilcoxon Signed Ranks Test [35] to compute p-value of A-MCTS compared with two baselines. Wilcoxon Signed Ranks Test is a common statistical test used to compare two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ. Tables 7 and 8 show that the average p-value of A-MCTS is smaller than 0.03, which illustrates the significance of our method's results.

Exp-4: Parameter Analysis.

Discount Factors. We compare the performance under different discount factor settings, i.e., $\delta = \mu$, $\delta > \mu$ and $\delta < \mu$. From Table 9, we can see (1) when $\delta \leq \mu$, the path length delivered by A-MCTS and RL-MCTS increases, and the performance getting worse greatly when $\delta < \mu$; (2) when $\delta > \mu$, different settings of δ and μ have slight influence on the performance. Therefore, a good strategy to choose right values of δ and μ in Eq. (5) is to ensure $\delta > \mu$ for avoiding imbalanced distribution of terminal state rewards.

Replay Memory. Fig. 13a depicts the influence of size of replay memory $|D|$ on different query situations.

TABLE 7
Statistical Test on Average Length of Results Based on Different Datasets

p-value	Arxiv	Elec	Enron	Slash	Digg	Conflict	Youtube	Opsahl	Openflights	Tasmania	AVERAGE
TPA	0.0253	0.0588	0.0091	0.0020	0.0082	0.0027	0.0020	0.0104	0.0082	0.0273	0.0154
RL-MCTS	0.0143	0.0455	0.0253	0.0103	0.0253	0.0339	0.0143	0.0082	0.0455	0.0588	0.0281

TABLE 8
Statistical Test on Average Query Processing Time Based on Different Datasets

p-value	Arxiv	Elec	Enron	Slash	Digg	Conflict	Youtube	Opsahl	Openflights	Tasmania	AVERAGE
TPA	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020
RL-MCTS	0.0020	0.0020	0.0020	0.0020	0.0273	0.0020	0.0273	0.0020	0.0020	0.0020	0.0071

TABLE 9
The Average Path Length Based on Different Discount Factors

Parameters	The average path length	
	RL-MCTS	A-MCTS
$\delta = 0.95$ $\mu = 0.9$	168.9137	138.9972
$\delta = 0.95$ $\mu = 0.85$	170.3248	141.1257
$\delta = 0.9$ $\mu = 0.9$	179.2467	151.5245
$\delta = 0.85$ $\mu = 0.9$	204.2376	179.4642

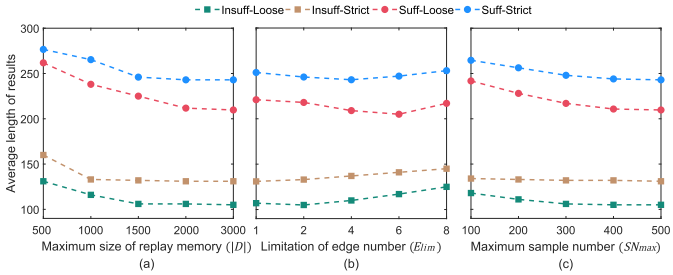

Fig. 13. The average path length based on different query situations.

Specifically, (1) when the features of temporal paths are insufficient, it is better to set $|D|$ as no more than 1500. When the constraints are strict, setting $|D|$ in the range of 500-1000 is appropriate. Size settings of $|D|$ can increase with the relaxation of constraints. (2) when the features are sufficient, setting $|D|$ as more than 2000 does not seem to achieve a good trade-off between performance and resource cost. In this condition, $|D|$ can be set around 1500 when constraints are strict, and the size can be set to increase slightly as the constraints become loose.

Fig. 13b shows the influence of the limitation of edge number (E_{lim}) on different query situations. Specifically, (1) when features of temporal paths are insufficient, it is better to set E_{lim} around 2. (2) when the features are sufficient, setting E_{lim} in 4-6 is appropriate.

Fig. 13c displays the influence of maximum sample number on different query situations. Specifically, (1) when features of temporal paths are insufficient, it is better to set SN_{max} as no more than 300. When the constraints are strict, setting SN_{max} to be smaller than 100 is appropriate. SN_{max} can slightly increase with the relaxation of constraints. (2) when the features are sufficient, setting SN_{max} in 300-400 can achieve a good trade-off between performance and resource cost. In this situation, SN_{max} can be set around 300 when constraints are strict, and increase SN_{max} slightly as the constraints become loose.

6 Conclusion and Future Work

In this paper, we have proposed a new Adaptive Monte Carlo Tree Search algorithm, A-MCTS, to overcome sparse reward and improve the performance under the small size of replay memory. A-MCTS achieves the time complexity of $O((U+F)I \cdot SN_{avg})$. The experiments conducted on ten real-world large-scale dynamic graphs have demonstrated the superiority of our proposed approaches in terms of effectiveness and efficiency.

A-MCTS has two potential applications. First, it can be applied to real-time path searching under multiple constraints, e.g., self-driving. Our method can automatically choose the safest routes according to road conditions, surrounding environment, weather, traffic flow, etc. Second, our method can be applied to discover specific nodes in dynamic graphs that have rich features, e.g., social networks and protein-protein interaction networks. In social networks, users have multiple features, such as hobbies, followed users, participated social groups, activities, etc., and these features change over time. Our method can be used to help users find potential friends that have specific interests and features based on users' social circles. Similarly, in protein-protein interaction networks, our method can also help find proteins with specific interactions and change patterns.

In future work, we intend to improve A-MCTS by dealing with some limitations. First, A-MCTS cannot apply to multiple optimization problems, e.g., searching for the shortest and fastest temporal path simultaneously, as A-MCTS has to redesign specific reward functions for multiple optimizations; In addition, if the constraints change, a certain number of searches need to be performed to recalculate state-value and action-value of states and actions, respectively.

References

[1] Y. Liu, Y. Li, and Y. Zhang, "Study of the logistics transportation vehicle terminal path optimization and algorithm based on GIS," *Appl. Mechanics Materials*, vol. 644/650, pp. 2249–2252, 2014.
[2] R. Milano, R. Baggio, and R. Piattelli, "The effects of online social media on tourism websites," in *Proc. Inf. Commun. Technol. Tourism*, 2011, pp. 471–483.
[3] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.
[4] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos "TimeCrunch: Interpretable dynamic graph summarization," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2015, pp. 1055–1064.
[5] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proc. IEEE INFOCOM Conf. Comput. Commun. 20th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, 2001, pp. 834–843.
[6] G. Kossinets, J. Kleinberg, and D. Watts "The structure of information pathways in a social communication network," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2008, pp. 435–443.
[7] R. Pan and J. Saramaki, "Path lengths, correlations, and centrality in temporal networks," *Phys. Rev. E*, vol. 84, no. 2, pp. 1577–1589, 2011.
[8] B. Xuan, A. Ferreira, and A. Jarry, "Computing shortest, fastest, and foremost journeys in dynamic networks," *Int. J. Found. Comput. Sci.*, vol. 14, no. 2, pp. 267–285, 2003.
[9] R. Sutton and A. Barto, *Reinforcement Learning:An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[10] J. You, B. Liu, Z. Ying, and V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6412–6422.
[11] 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.
[12] W. Xiong, T. Hoang, and W. Y. Wang, "DeepPath: A reinforcement learning method for knowledge graph reasoning," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2017, pp. 564–573.
[13] I. Bello, H. Pham, and Q. V. Le, "Neural combinatorial optimization with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Representations*, 2017, pp. 1–5.
[14] Y. Zhou, J. Hao, and B. Duval, "Reinforcement learning based local search for grouping problems: A case study on graph coloring," *Expert Syst. Appl.*, vol. 64, pp. 412–422, 2016.
[15] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Proc. Int. Conf. Comput. Games*, 2006, pp. 72–83.

[16] P. Ding, G. Liu, P. Zhao, A. Liu, Z. Li, and K. Zheng "Reinforcement learning based monte carlo tree search for temporal path discovery," in *Proc. IEEE Int. Conf. Data Mining*, 2019, pp. 140–149.
[17] R. Bellman, "A markovian decision process," *J. Math. Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.
[18] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proc. Eur. Conf. Mach. Learn.*, 2006, pp. 282–293.
[19] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *J. ACM*, vol. 37, no. 3, pp. 607–625, 1990.
[20] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *Proc. 11th Int. Conf. Extending Database Technol.: Adv. Database Technol.*, 2008, pp. 205–216.
[21] D. Ouyang, L. Yuan, and L. Qin, "Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees," *Proc. VLDB Endowment*, vol. 13, no. 5, pp. 602–615, 2020.
[22] L. Li, S. Wang, and X. Zhou, "Time-dependent hop labeling on road network," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 902–913.
[23] Z. Yu *et al.*, "Distributed processing of k shortest path queries over dynamic road networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 665–679.
[24] Y. Yang, H. Gao, J. X. Yu, and J. Li, "Finding the cost-optimal path with time constraint over time-dependent graphs," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 673–684, 2014.
[25] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," *Proc. VLDB Endowment*, vol. 10, no. 2, pp. 61–72, 2016.
[26] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu, "Efficient algorithms for temporal path computation," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2927–2942, Nov. 2016.
[27] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 967–982.
[28] L. Li, M. Zhang, W. Hua, and X. Zhou, "Fast query decomposition for batch shortest path processing in road networks," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1189–1200.
[29] B. C. Dean, "Algorithms for minimum-cost paths in time-dependent networks with waiting policies," *Networks*, vol. 44, no. 1, pp. 41–46, 2004.
[30] A. Zhao, G. Liu, B. Zheng, Y. Zhao, and K. Zheng, "Temporal paths discovery with multiple constraints in attributed dynamic graphs," *World Wide Web*, vol. 23, no. 1, pp. 313–336, 2020.
[31] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710.
[32] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoustics Speech Signal Process.*, vol. 26, no. 1, pp. 43–49, Feb. 1978.
[33] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," in *Proc. SIAM Int. Conf. Data Mining*, 2001, pp. 1–11.
[34] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intell. Data Anal.*, vol. 11, no. 5, pp. 561–580, 2007.
[35] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bull.*, vol. 1, no. 6, pp. 80–83, 1945.

Pengfei Ding received the BS and MS degrees from Soochow University, P.R. China, in 2017 and 2020, respectively. He is currently working toward the PhD degree at the Department of Computing, Macquarie University, Sydney, Australia. His research interests include graph mining and social computing.

Guanfeng Liu (Member, IEEE) received the PhD degree in computer science from Macquarie University, in 2013. He is currently an assistant professor with the Department of Computing, Macquarie University, Sydney, Australia. His research interests include graph database, trust computing, and social computing. He has published more than 80 papers on international journals and conferences including *IEEE Transactions on Knowledge and Data Engineering*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Emerging Topics in Computing*, *Transactions on the Web*, *ACM Transactions on Data Scienc*, ICDE, WWW, IJCAI, AAAI and CIKM.

Yan Wang (Senior Member, IEEE) received the BEng, MEng, and the DEng degrees in computer science and technology from the Harbin Institute of Technology (HIT), P.R. China, in 1988, 1991, and 1996, respectively. He is currently a professor with the Department of Computing, Macquarie University, Sydney, Australia. His research interests include trust computing, social computing, service computing and recommender systems.

Kai Zheng (Senior Member, IEEE) received the PhD degree in computer science from the University of Queensland, in 2012. He is currently a professor of computer science with the University of Electronic Science and Technology of China. He has been working in the area of spatial-temporal databases, uncertain databases, social-media analysis, in-memory computing and blockchain technologies. He has published more than 100 papers in prestigious journals and conferences in data management field such as SIGMOD, ICDE, the *VLDB Journal*, *ACM Transactions and IEEE Transactions*.

Xiaofang Zhou (Fellow, IEEE) received the BSc and MSc degrees in computer science from Nanjing University, China, and the PhD degree in computer science from the University of Queensland, Australia, in 1984, 1987, and 1994, respectively. He is currently a professor of computer science with the Hong Kong University of Science and Technology. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, bioinformatics, and e-research.