Reinforcement Learning Based Monte Carlo Tree Search for Temporal Path Discovery

Pengfei Ding[1], Guanfeng Liu[2], Pengpeng Zhao[1], An Liu[1*], Zhixu Li[1], and Kai Zheng[3*]
[1]*School of Computer Science and Technology, Soochow University, Suzhou, China*
[2]*Department of Computing, Macquarie University, NSW, Australia*
[3]*University of Electronic Science and Technology of China, Chengdu, China*
guanfeng.liu@mq.edu.au; {ppzhao, anliu, zhixuli}@suda.edu.cn; zhengkai@uestc.edu.cn

***Abstract*—An Attributed Dynamic Graph (ADG) contains multiple dynamic attributes associated with each edge. In ADG based applications, people usually can specify multiple constrains 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 a type of Multi-Constrained Temporal Path (MCTP) discovery in ADGs, which is a challenging NP-Complete problem. In order to deliver an efficient and effective temporal path discovery method to be used in real-time environment, we propose a Reinforcement Learning (RL) based, Monte Carlo Tree Search algorithm (RL-MCTS). RL-MCTS uses a newly designed memory structure to address the challenges of Monte Carlo Tree Search (MCTS) in MCTP discovery. To the best of our knowledge, RL-MCTS is the first RL algorithm that supports path discovery in ADGs. The experimental results on ten real dynamic graphs demonstrate that our algorithm outperforms the state-of-the-art methods in terms of both efficiency and effectiveness.**

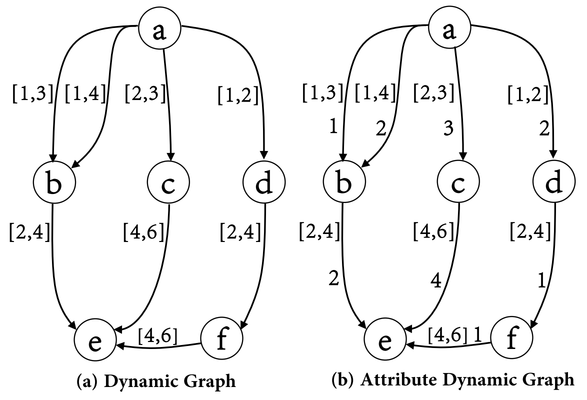

Fig. 1. Dynamic graph G and its attribute dynamic graph G_A

I. Introduction

Path queries, like trip planning in transportation networks [1], can be modelled as the path discovery problem in graphs [2]–[4], which has been well studied in the literature. Many real applications can be encoded as *dynamic graphs*, in which a node is associated with another node at particular time instances [5]. For example, the connections between two people via telephone calls start at a certain time point of a day [6], and the connections between cities via courier vehicles based on the schedules of the vehicles [1].

In addition, 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 some constraints on these attributes in a path query to illustrate their requirements on the targeted dynamic graph. For example, in a traffic network, if a user plans to drive to a restaurant after 5:00 PM, what is the shortest route that the user can arrive at the restaurant under certain constraints on the toll fee and the drive time? *Example 1* below discusses a scenario with an attributed dynamic graph structure.

Example 1: Fig. 1(a) shows a dynamic graph G, assuming that G is a subway-transport network, where each node represents a subway station, the tuple on each edge contains a departure time point and an arrival time point, e.g., for the edge $\langle a \rightarrow c \rangle$, the subway starts from a at $2:00$ and arrives at c at $3:00$. For simplicity, we assume that the distance between two connected stations is 1 km. Fig. 1(b) shows the attribute dynamic graph G_A of G, the number added on each edge is the cost of taking the subway between two connected stations.

Now consider the shortest temporal path from a to e in the two graphs. In Fig. 1(a), the shortest temporal path is $\langle a \rightarrow c, c \rightarrow e \rangle$ with distance 2km. But when we constrain the travel cost smaller than 6, in Fig. 1(b), $\langle a \rightarrow c, c \rightarrow e \rangle$ is an invalid temporal path because the cost of it is 7. Instead, the shortest temporal path is $\langle a \rightarrow d, d \rightarrow f, f \rightarrow e \rangle$ with distance 3km, and the cost of it is 4.

The discussion above leads to a new type of *Multi-Constrained Temporal Path* (MCTP) discovery problem, which is NP-Complete [7]. The existing methods for the temporal path discovery [5], [8]–[11] need 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 time cost [12], [13]. Therefore, the existing methods are not applicable when dealing with large-scale attribute dynamic graphs.

In order to devise an effective and efficient method to solve the MCTP problem, inspired by Reinforcement Learning (RL), we can compare the quality of temporal paths by using a verifier, and provide the reward feedback to a learning algorithm. Hence, we formulate the MCTP problem as a Markov Decision Process (MDP), follow the RL paradigm to tackle the problem [14], and adopt the idea of Monte Carlo Tree Search (MCTS) to discover the required MCTP, which is an effective way of solving NP-Complete problems [15], [16].

MCTS is a general-purpose planning algorithm that has found great success in a number of seemingly unrelated applications, ranging from Bayesian reinforcement learning [17] to General-Game Playing [18]. It is often applied to domains where it is difficult to incorporate expert knowledge. The key idea of MCTS is to construct a search tree of states evaluated by fast Monte Carlo simulations [19]. Then, the state value is estimated as the mean outcome of the simulations. Meanwhile, a search tree is maintained to guide the direction of simulation, for which bandit algorithms can be employed to balance exploration and exploitation [20].

However, in the MDP of the MCTP problem, if we consider each temporal path as a state, as an ADG of a social network may contain millions of edges, the state space of the ADG will be large. Therefore there are two challenges of using MCTS to solve the MCTP problem. (1) Since the mean state estimation is likely to have high variance under large states and relatively limited search time, the accuracy of state estimation can not be effectively guaranteed, which can mislead building the search tree and severely degrade performance of the algorithm. (2) The rewards are sparse because positive rewards can be received only at several temporal paths, for instance, when the temporal path is an MCTP.

In this paper, we utilize the structure of ADGs and history experience (i.e., traversed temporal paths) to effectively address these two challenges. First, RL-MCTS uses the *state-value* function that modelled by properties of ADGs to make evaluations of states. Integrating the state-value function with MCTS, RL-MCTS can avoid the search of invalid states (i.e., temporal paths that can not bring the MCTP result), then performs more searches on other states and improves the accuracy of state estimation. Second, to address the sparse reward, RL-MCTS utilizes the ADG structure to encode history experience into vector representations, and saves them into the *replay memory*, which is further used to model the *action-value* function. Then RL-MCTS combines MCTS with the action-value function to generate temporal paths that have significantly more positive rewards than using the MCTS alone.

Our contributions are summarized as follows.

1) We formulate the MCTP problem as a Markov Decision Process (MDP), and propose a Reinforce Learning based Monte Carlo Tree Search algorithm (RL-MCTS), which searches a tree with a designed memory structure that saves history searching results;
2) We design the state-value function and action-value function to solve the challenges of the inaccuracy state estimation and the sparse reward;
3) We conduct extensive experiments on ten real-world attribute dynamic graphs. The experimental results illustrate that on average, our algorithm can save 79.17% execution time, and the average performance of MCTP delivered by RL-MCTS is 8.92% better than the state-of-the-art algorithms.

II. Preliminaries

In this paper, we focus on the shortest temporal path discovery with multiple constraints in ADGs. The notations used in this paper are shown in Table I.

A. Attributed Dynamic Graph

An Attributed Dynamic Graph (ADG) is a graph $G_A = (V_N, E, \mathbf{f})$, where:

- V_N is the set of nodes of G_A;
- E is the set of edges of G_A, an edge $e \in E$ is a quintuple (u, v, l, tp, t), where $u, v \in V_N$, l is the length of e, tp and t are the departure time point from u and the arrival time point at v respectively, hence $tp \leq t$;
- f_j, $j \in [1, K]$ is a series of K attribute functions that assign every edge in E a non-negative value $f_j(e)$, i.e., $f_j : (u, v, l, tp, t) \in E \to \mathbb{R}^+$, let $\mathbf{f}$ represent the attribute vector, $\mathbf{f} = [f_1, f_2, ..., f_K]$.

B. Temporal Path

A temporal path P in an ADG $G_A = (V_N, E, \mathbf{f})$ is a sequence of edges $P_{v_1,v_n}(t_n) = \langle e_1, e_2, ..., e_{n-1}\rangle$, where:

- t_n is the arrival time point from v_1 to v_n;
- $(v_i, v_{i+1}, l_i, tp_i, t_{i+1}) \in E$ is the i-th edge on $P_{v_1,v_n}(t_n)$, $i \in [1, n-1]$;
- $\forall i \in [2, n-1]$, $t_i \leq tp_i$;
- $\Gamma_j(P_{v_1,v_n}(t_n)) = \sum_{i=1}^{n-1} f_j(e_i)$ is the j-th aggregated attribute value of $P_{v_1,v_n}(t_n)$;
- $|P_{v_1,v_n}(t_n)| = \sum_{i=1}^{n-1} l_n$ is the length of $P_{v_1,v_n}(t_n)$.

C. Feasible Temporal Path

Given an ADG $G_A = (V_N, E, \mathbf{f})$, $v_s, v_d \in V_N$ denote the source node and the destination node respectively, $\mathcal{X} = [\lambda_1, \lambda_2, ..., \lambda_K]$ denotes the constraint vector on attribute vector $\mathbf{f}$, $[t_\phi, t_\varphi]$ is the constraint of time interval. A temporal path $P_{v_s,v_d}(t_d)$ is the feasible temporal path, where:

- $tp_s \geq t_\phi, t_d \leq t_\varphi$;
- $\forall j \in [1, K]$, $\Gamma_j(P_{v_s,v_d}(t_d)) \leq \lambda_j$.

Definition 1. Multi-Constrained Shortest Temporal Path (MC-STP) Given an ADG G_A, a source node v_s and a destination node v_d in G_A, a constraint vector $\mathcal{X}$ 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 $|P| = min\{|P'| : P' \in \mathbf{P}\}$.

D. Markov Decision Process for MCTP Discovery

Now we formulate MCTP discovery problem as a Markov Decision Process (MDP), which is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$, where $\mathcal{S}$ is the set of states, in our algorithm, let each temporal path $P \in \mathcal{S}$ denote a state. $\mathcal{A}$ is the set of actions, for a state P, the action space of P (i.e., $\mathcal{A}(P)$) is the union of all edges that connected to the last edge of P, and the departure time points of edges in $\mathcal{A}(P)$ are later than the arrival time point of P. Therefore, after tacking an action from $\mathcal{A}(P)$, the corresponding edge of the action will be added to the end of P, then forming a new temporal path P'. $\mathcal{R}$ is the reward function, in the MDP for MCTP discovery,

TABLE I
TABLE OF NOTATION

Notation	Meaning
G_A	Attributed dynamic graph
V_N	Set of nodes
E	Set of edges
$\mathbf{f} = [f_1, f_2, ..., f_K]$	Series of K attribute functions
$P_{u,v}(t)$	Temporal path starts from u, arrives v at t
$\Gamma_j(P_{u,v}(t))$	j-th aggregated attribute value of $P_{u,v}(t)$
$\|P_{u,v}(t)\|$	Length of $P_{u,v}(t)$
$[t_\phi, t_\varphi]$	Constraint of time interval
$\mathcal{X} = [\lambda_1, \lambda_2, ..., \lambda_K]$	Constraint vector on $\mathbf{f}$
$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions
$\mathcal{R}$	Reward function
$\mathcal{P}$	State transition probability
D	Replay memory
δ, μ	Discount factors
$V(P)$	State-value function
$Q(P, e)$	Action-value function
$\mathcal{R}_V(P)$	Average reward value of P
$\hat{\mathcal{R}}_D(P)$	Approximate reward of P
$N(P)$	Number of traversal times of P
$\mathcal{E}(e)$	Vector representation of e
$\mathbf{H}(P)$	Attribute vector representation of P
$\mathbf{T}(P)$	Trajectory vector representation of P
$d(P_i, P_j)$	Distance function

only *terminal states* get rewards. We define terminal states 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., $\exists j \in [1, K]$, $\Gamma_j(P) > \lambda_j$; or (3) the current state meets none of above two conditions and has no actions to select (i.e., $\mathcal{A}(P) = \emptyset$), then the current MDP reaches a terminal state and will receive a reward. Since the length of MC-STP is unknown, we give $+0.5$ reward to the first traversed feasible temporal path and set it's length as a *base*, then the rewards of the following feasible temporal paths are set as E.q. 1.

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

We can see the range of $\mathcal{R}(\cdot)$ is $(0, 1)$, and the feasible temporal paths with shorter lengths have larger rewards. For other terminal states which are not feasible temporal paths, the rewards will be 0. The rewards are sparse because only feasible temporal paths can receive positive rewards, and these temporal paths are rare in large-scale ADGs. $\mathcal{P}$ is the state transition probability, since G_A is known, the MDP transition probability is deterministic, e.g., for a state $P_{v_s,v_c}(t_c)$, once an action of $\mathcal{A}(P_{v_s,v_c}(t_c))$ is selected, i.e., an edge $(v_c, v_n, l_c, tp_c, t_n)$, the next state $P_{v_s,v_n}(t_n)$ and its associated $\mathcal{A}(P_{v_s,v_n}(t_n))$ are known.

For an MDP starting from v_1 at t_1, after taking a series of actions, states of the MDP are transitioned, assuming the MDP finally reaches the terminal state $P_{v_1,v_n}(t_n)$ and gets a reward $\mathcal{R}(P_{v_1,v_n}(t_n))$. Then the MDP ends and can be denoted as:

$\langle(v_1, t_1), (v_1, v_2, l_1, tp_1, t_2), P_{v_1,v_2}(t_2), (v_2, v_3, l_2, tp_2, t_3), P_{v_1,v_3}(t_3), \ldots, P_{v_1,v_n}(t_n), \mathcal{R}(P_{v_1,v_n}(t_n))\rangle$.

Example 2: Fig. 2 shows the example of an MDP, starting from the source node a, taking the action of edge

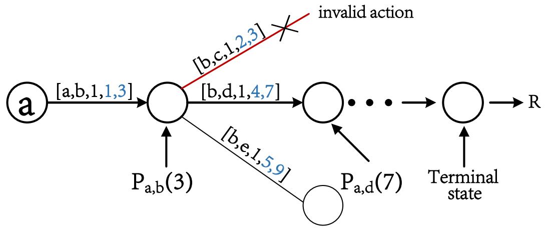

Fig. 2. A markov decision process

$(a, b, 1, 1, 3)$ and reaching the state $P_{a,b}(3)$. There are three edges connected to b. Since the departure time point of $(b, c, 1, 2, 3)$ (i.e., 2) is earlier than the arrival time point of $P_{a,b}(3)$, $(b, c, 1, 2, 3)$ is an invalid action. Then $\mathcal{A}(P_{a,b}(3)) = \{(b, d, 1, 4, 7), (b, e, 1, 5, 9)\}$, $P_{a,b}(3)$ takes the action of the edge $(b, d, 1, 4, 7)$ and transitions to the state $P_{a,d}(7)$. Finally, it reaches a terminal state and gets a reward R.

III. Reinforce Learning Based Monte Carlo Tree Search Algorithm (RL-MCTS)

In this section, we first define $V(P)$ and $Q(P, e)$ as the state-value function and the action-value function respectively. The state-value function $V(P)$ uses properties and searching results of states to evaluate P. The action-value function $Q(P, e)$ uses the replay memory that stores traversed temporal paths to approximate the long-term reward of taking action e at state P. Then we introduce the tree structure of RL-MCTS, and discuss the details of RL-MCTS combined with the state-value function and the action-value function.

A. State-Value Function

Now we introduce the state-value function $V(P)$ and how to update it. The initial state value for a state $P_{v_s,v_c}(t_c)$ is set as E.q. 2.

$$V(P_{v_s,v_c}(t_c)) = 1 - \max\{\frac{\Gamma_j(P_{v_s,v_c}(t_c))}{\lambda_j} : j \in [1, K]\} \quad (2)$$

Note that if $V(P_{v_s,v_c}(t_c)) < 0$, $P_{v_s,v_c}(t_c)$ is not a feasible temporal path. At the beginning of the searching process, RL-MCTS prefers states with high state values (see Section III-D for details), since the initial values of states with high aggregated attribute values are close to 0, at first, most of these states will be ignored.

For a completed MDP, only the terminal state will receive a reward $\mathcal{R}(\cdot)$. Similar to an RL method, TD(λ) [14], we use a discount reward to update state values of the intermediate states. When the MDP reaches a terminal state $P_{v_s,v_d}(t_d)$ and gets the reward $\mathcal{R}(P_{v_s,v_d}(t_d))$, then:

(1) If $P_{v_s,v_d}(t_d)$ is a feasible temporal path, each intermediate state $P_{v_s,v_i}(t_i)$ in the MDP will be updated by E.q. 3.

$$V(P_{v_s,v_i}(t_i)) \leftarrow V(P_{v_s,v_i}(t_i)) + \delta^h \mathcal{R}(P_{v_s,v_d}(t_d)) \quad (3)$$

where $\delta \in (0, 1)$ is a discount factor, h is the number of times from $P_{v_s,v_i}(t_i)$ transitions to the terminal state $P_{v_s,v_d}(t_d)$. For example, in the MDP, if $P_{v_s,v_i}(t_i)$ takes an action $(v_i, v_d, l_i, tp_i, t_d)$ and transitions to $P_{v_s,v_d}(t_d)$, then $h = 1$. From E.q. 3 we can see the intermediate states that near feasible temporal paths will get greater discount rewards.

(2) If the terminal state $P_{v_s,v_d}(t_d)$ is not a feasible temporal path, even $\mathcal{R}(P_{v_s,v_d}(t_d)) = 0$, we give a small penalty to all intermediate states in the MDP and update them as E.q. 4.

$$V(P_{v_s,v_i}(t_i)) \leftarrow V(P_{v_s,v_i}(t_i)) - \mu^h |V(P_{v_s,v_d}(t_d))| \quad (4)$$

Because of the sparse reward, lots of terminal states' rewards will be 0. Then we set a different discount factor μ that smaller than δ. For a terminal state P_t that is not a feasible temporal path, if $\mathcal{A}(P_t) = \emptyset$ and the aggregated attribute values of P_t do not exceed the constraints, $V(P_t)$ will be a non-negative value and smaller than 1, otherwise, $V(P_t)$ will be a small negative value. Therefore, the range of $|V(P_t)|$ is approximately the same with $\mathcal{R}(\cdot)$.

In E.q. 3-E.q. 4, when the MDP is completed, $\delta^h \mathcal{R}(P_{v_s,v_d}(t_d))$ and $-\mu^h|V(P_{v_s,v_d}(t_d))|$ can be regarded as the reward of the intermediate state $P_{v_s,v_i}(t_i)$ in the MDP. Therefore, in addition to define the rewards of terminal states (i.e., E.q. 1), we also define the reward function of intermediate states as E.q. 5.

$$\mathcal{R}(P_{v_s,v_i}(t_i)) = \begin{cases} \delta^h \mathcal{R}(P_{v_s,v_d}(t_d)) & \text{if } \mathcal{R}(P_{v_s,v_d}(t_d)) > 0 \\ -\mu^h |V(P_{v_s,v_d}(t_d))| & \text{else} \end{cases} \quad (5)$$

where $P_{v_s,v_d}(t_d)$ is the terminal state of the current MDP.

Since the initial value of $V(P)$ is determined by the aggregated attribute values of P, $V(P)$ can not represent the long-term reward of P. Then we use $\mathcal{R}_V(P)$ to characterize the long-term reward of P and define $\mathcal{R}_V(P)$ as E.q. 6.

$$\mathcal{R}_V(P_{v_s,v_i}(t_i)) = \frac{\sum_{j=1}^{N(P_{v_s,v_i}(t_i))} \mathcal{R}_j(P_{v_s,v_i}(t_i))}{N(P_{v_s,v_i}(t_i))} \quad (6)$$

Here, $N(P_{v_s,v_i}(t_i))$ is the number of times that $P_{v_s,v_i}(t_i)$ has been traversed. We can see as $N(P) \to \infty$, $\mathcal{R}_V(P)$ will be close to the *true reward* of P (i.e., $\mathbb{E}(\mathcal{R}(P))$).

B. Action-Value Function

As $\mathcal{R}_V(P)$ characterizes the long-term reward of P, similarly, we define action-value function $Q(P, e)$ to represent the long-term reward of action e at P. For a state $P_{v_s,v_c}(t_c)$ with an action $e = (v_c, v_n, l_c, tp_c, t_n)$, as $\mathcal{P}$ is deterministic, the next state will be $P_{v_s,v_n}(t_n)$. Intuitively, the action-value can be defined as the difference between the true reward values of the two states:

$$Q(P_{v_s,v_c}(t_c), e) = \mathbb{E}(\mathcal{R}(P_{v_s,v_n}(t_n))) - \mathbb{E}(\mathcal{R}(P_{v_s,v_c}(t_c))) \quad (7)$$

If $Q(P_{v_s,v_c}(t_c), e)$ is a positive value, it means that selecting the action e will transition the current state $P_{v_s,v_c}(t_c)$ to the state that is approaching to the feasible shortest temporal path. Conversely, if the value of $Q(P_{v_s,v_c}(t_c), e)$ is negative, it indicates that the action selection of e may lead the current state to a worse result.

From E.q. 6, we can use $\mathcal{R}_V(P)$ to replace $\mathbb{E}(\mathcal{R}(P))$. However, getting the stable value of $\mathcal{R}_V(P)$ requires traversing most MDPs that contain P, which is unrealistic under large states and limited search time. In addition, if P has not been traversed, we cannot get its $\mathcal{R}_V(P)$. Then we use the history experience, i.e., traversed temporal paths, to approximate long-term rewards of states (denoted as *approximate rewards*) and get the action value. In RL-MCTS, we encode states into vector representations, use the *replay memory* to save the vector representations of history experience. Then the approximate reward of a state is calculated by its similarity with vector representations that in the replay memory.

In ADGs, temporal paths contain two characteristics, attribute features and trajectory features. Therefore, we encode them into attribute vector representations and trajectory vector representations respectively.

Attribute Vector Representation: For the attribute feature of a state, since each attribute of the temporal path is independent with others, we normalize aggregated attribute values and take each attribute as a dimension of the attribute vector representation. Then the attribute vector representation of a state P can be defined as below:

$$\mathbf{H}(P) = B([|P|, \Gamma_1(P), \Gamma_2(P), \cdots, \Gamma_K(P)]) \quad (8)$$

where $B(\cdot)$ is a function to normalize all elements in the attribute vector representation.

Trajectory Vector Representation: For the trajectory feature of a state, graph embedding techniques [21] such as *deepwalk* [22], can be used to convert temporal paths into a low dimensional space in which the structure information is maximally preserved. Deepwalk optimizes embeddings to encode the statics of random walks, however, using deepwalk to encode states into vector representations requires a large amount of temporal paths, and costs a lot of training time because of the recurrent neural network (RNN) structure. Since the ADG is known, we encode edges of the ADG into vector representations as a pre-work. In the searching process, a state trajectory vector representation is composed of its edge vector representations.

Another problem is that deepwalk uses simple unbiased random walks over the graph, which fails to leverage the temporal and attribute information that is informative with respect to edges' positions in the ADG. In our algorithm, we propose a strategy that biases random walks with the time order and the edge length. Specifically, we label each edge of an ADG with β:

$$\beta(v_c, v_i, l_c, tp_c, t_i) = \frac{1}{l_c} \quad (9)$$

In the random walk process, when the walk arrives v_c at t_c, we define the probability of the walk taking $e = (v_c, v_i, l_c, tp_c, t_i)$ as the next step as:

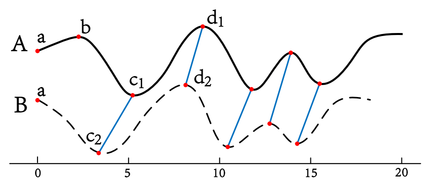

Fig. 3. Two similar trajectory vector representations

$$Pr(e) = \frac{\beta(e)}{\sum_{e_j \in C(v_c, t_c)} \beta(e_j)} \tag{10}$$

Here, $C(v_c, t_c)$ is the edge set from v_c to its neighbor nodes which departure time points later than t_c. With the improved strategy, the walk prefers edges with short lengths and obeys the chronological order.

Then we use the improved deepwalk to encode edges of the ADG into vector representations, which denoted as $\mathcal{E}(e)$. Then the trajectory feature of a state $P_{v_1,v_n}(t_n) = \langle e_1, e_2, \ldots, e_{n-1} \rangle$ can be represented as below:

$$\mathbf{T}(P_{v_1,v_n}(t_n)) = [\mathcal{E}(e_1), \mathcal{E}(e_2), \ldots, \mathcal{E}(e_{n-1})] \tag{11}$$

Distance Function: The similarity between two states contains two parts, i.e., the similarity of attribute features and the similarity of trajectory features respectively. For the sake of simplicity, in the following discussion, let P_i and P_j denote two different states $P_{v_s,v_i}(t_i)$ and $P_{v_s,v_j}(t_j)$ respectively. For the first part, we use cosine distance to approximate the difference between the two attribute vector representations of P_i and P_j by E.q. 12.

$$d_{att}(\mathbf{H}(P_i), \mathbf{H}(P_j)) = \cos(\mathbf{H}(P_i), \mathbf{H}(P_j)) \tag{12}$$

For the second part, since the trajectory vector representations of different states usually have different lengths, it is often the case that two vector representations have the approximately the same overall component shapes. But these shapes do not line up in the sequence dimension. Fig. 3 shows this with a simple example. For two trajectory vector representations $\mathbf{A} = [\mathbf{a}, \ldots, \mathbf{b}, \ldots, \mathbf{c_1}, \ldots, \mathbf{d_1}, \ldots]$ and $\mathbf{B} = [\mathbf{a}, \ldots, \mathbf{c_2}, \ldots, \mathbf{d_2}, \ldots]$, $\mathbf{c_1}$ and $\mathbf{c_2}$, $\mathbf{d_1}$ and $\mathbf{d_2}$ are similar edge vector representations. Although $\mathbf{A}$ selects the action of $\mathbf{b}$ at the beginning of the searching, $\mathbf{A}$ and $\mathbf{B}$ have an overall similar shape, which means the selection of $\mathbf{b}$ has little effect on the searching result. Hence, $\mathbf{A}$ and $\mathbf{B}$ should have a high degree of similarity. However, in this case, a distance measure that assumes the i-th edge vector representation on $\mathbf{A}$ is aligned with i-th edge vector representation on $\mathbf{B}$ will produce a pessimistic dissimilarity. Dynamic Time Warping (DTW) [23] is an effective way to measure the similarity between two sequences under such a situation. We approximate the difference between two trajectory vector representations by DTW as E.q. 13.

$$d_{traj}(\mathbf{T}(P_i), \mathbf{T}(P_j)) = \min_{\mathbf{W}(\mathbf{T}(P_i), \mathbf{T}(P_j))} \frac{\sqrt{\sum_{l=1}^{L} W_l(\mathbf{T}(P_i), \mathbf{T}(P_j))}}{L} \tag{13}$$

$\mathbf{W}(\mathbf{T}(P_i), \mathbf{T}(P_j))$ is the *warping path* [24] of $\mathbf{T}(P_i)$ and $\mathbf{T}(P_j)$, which is a contiguous set of elements that defines a mapping between two trajectory vector representations. 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 elements in $\mathbf{T}(P_i)$ and $\mathbf{T}(P_j)$ respectively. $\max(p, q) \leq L < p+q-1$, p and q are the length of $\mathbf{T}(P_i)$ and $\mathbf{T}(P_j)$ respectively. There may exist more than 1 warping paths, and we choose the warping path which can calculate the minimal function value.

The time complexity of general DTW [23] is $O(n^2)$, n is the length of the shorter vector representation between two vector representations. In RL-MCTS, we use the variant of DTW, fast-DTW [25], which adopts some acceleration methods and has the $O(n)$ time complexity.

Then the distance function can be defined as E.q. 14.

$$\begin{aligned} d(P_i, P_j) = &\alpha \times d_{att}(\mathbf{H}(P_i), \mathbf{H}(P_j)) \\ &+ \gamma(1-\alpha) \times d_{traj}(\mathbf{T}(P_i), \mathbf{T}(P_j)) \end{aligned} \tag{14}$$

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

Replay Memory: Now we introduce the replay memory D and how it works. Each entry of D corresponds to a state $P_{v_s,v_c}(t_c)$ (here we use P_c to represent $P_{v_s,v_c}(t_c)$), it contains the tuple $\{\mathbf{H}(P_c), \mathbf{T}(P_c), \mathcal{R}_V(P_c), N(P_c)\}$ of P_c. Specifically, RL-MCTS only stores tuples of states that in the last M MDPs, the constant M can be seen as the size of D. As different MDPs may contain the same states, if P_c has already been stored in the memory, we only update $\mathcal{R}_V(P_c)$ and $N(P_c)$ at the corresponding entry. In addition, we set a threshold θ and ensure $|\mathcal{R}_V(\cdot)| \geq \theta$ for all tuples in D because the state that the average reward is close to 0 has little effect on the approximate result.

However, directly using continuous states of an MDP to approximate the long-term reward of a state is inefficient, due to the strong correlation between these consecutive states. Therefore, every evaluation RL-MCTS randomly picks N_D tuples from D to break these correlations, which also increases the randomness of our algorithm as the approximate rewards of a specific state are not always consistent. Then, given a state P_c, after sampling N_D tuples of states from D randomly, RL-MCTS finds the top m similar states based on the distance function as E.q. 14. The approximate reward of P_c is then computed by E.q. 15.

$$\hat{\mathcal{R}}_D(P_c) = \sum_{i=1}^{m} \omega_i(P_c)\mathcal{R}_V(P_i) \tag{15}$$

where P_i is the i-th similar state of P_c, and $\omega_i(P_c)$ is the weight of P_i that is calculated by E.q. 16.

$$\omega_i(P_c) = \frac{N(P_i)\exp(-d(P_i, P_c))}{\sum_{j=1}^{m} N(P_j)\exp(-d(P_j, P_c))} \tag{16}$$

From E.q. 6, we can see if state P_j has a larger $N(P_j)$ value, $\mathcal{R}_V(P_j)$ is closer to the true reward value of P_j. In addition, it is known that $\exp(x)$ is close to $x+1$ for a small

x. Then RL-MCTS gives higher weights to the similar states that have more traversal times and higher similarities.

Weight Function: RL-MCTS combines the approximate reward value $\hat{\mathcal{R}}_D(\cdot)$ with the average reward value $\mathcal{R}_V(\cdot)$ to calculate the action-value, and assigns different weights to $\hat{\mathcal{R}}_D(\cdot)$ and $\mathcal{R}_V(\cdot)$ respectively, as we mentioned before, the traversal time of getting the true reward value of a state P_i (denoted as $N_t(P_i)$) is proportional to the number of MDPs that contain P_i, in other words, $N_t(P_i) \propto |\mathcal{A}(P_i)|$. Then we define the weight function $\tau(P_i)$ of the approximate reward $\hat{\mathcal{R}}_D(P_i)$ as E.q. 17.

$$\tau(P_i) = \frac{1}{\log_{|\mathcal{A}(P_i)|} |\mathcal{A}(P_i)|(1 + N(P_i))} \tag{17}$$

Then combining with the replay memory and E.q. 17, the action-value function can be defined as E.q. 18.

$$\begin{aligned} Q(P_i, e) =& \tau(P_j)\hat{\mathcal{R}}_D(P_j) + (1 - \tau(P_j))\mathcal{R}_V(P_j) \\ &- (\tau(P_i)\hat{\mathcal{R}}_D(P_i) + (1 - \tau(P_i))\mathcal{R}_V(P_i)) \end{aligned} \tag{18}$$

Here P_j is transitioned from P_i after tacking the action e. At first, P_j has not been traversed and $N(P_j) = 0$, then $\tau(P_j) = 1$, RL-MCTS uses $\hat{\mathcal{R}}_D(P_j)$ to calculate the action value. When $N(P_j) \approx |\mathcal{A}(P_j)|$, $\tau(P_j) \approx 0.5$. As there may exist duplicate action selections and $N_t(P)$ is much larger than $|\mathcal{A}(P)|$, in E.q. 17 we use the log function to ensure that $\tau(P)$ becomes small when $N(P)$ is large enough.

By applying these approximations, our action-value function becomes a special case of kernel based methods, such as Locally Weighted Regression and Kernel Regression [26], where the kernel function can be defined by $k_i(P_c) = \exp(-d(P_i, P_c))/\sum_{j=1}^{m} \exp(-d(P_j, P_c))$.

C. Tree Structure

Now we introduce the tree structure of RL-MCTS. Each node in the tree represents a state, while the tree's edges correspond to actions. MCTS grows the tree structure iteratively. With each iteration, the tree structure is traversed and expanded. In a tree structure, a root node represents the starting state, i.e., the source node v_s with the constraint of departure time point t_ϕ; a child node represents the state transitioned from its ancestor's state after tacking an action from $\mathcal{A}$, the edge between them indicates the corresponding action; a leaf node represents a terminal state or a new expanded state that has not been traversed. Each node of the tree structure stores the set of statistics of a state P_i: $\{V(P_i), N(P_i), \mathcal{R}_V(P_i), \hat{\mathcal{R}}_D(P_i)\}$. In this way, an MDP can be regarded as an iteration of MCTS, which starts from the root node, selects child nodes recursively, and finally reaches a leaf node that represents a terminal state.

D. UCT Function

We now discuss how to combine $V(\cdot)$ and $Q(\cdot)$ to select actions in the tree search of RL-MCTS. An important problem of monte carlo tree search is to balance the exploration versus exploitation. The exploration approach promotes the exploration of unexplored areas (i.e., leaf nodes that represent untraversed states) in the tree structure, this 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 value (i.e., the maximal action value of E.q. 18), this approach is greedy and will extend the tree's depth more than its breath. 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 large states.

Specifically, on each MCTS iteration, an MDP is rolled out by selecting actions according to the proposed variant of UCT [27] from the root node (defined in E.q. 19).

$$e_a = \arg\max_{e_j}\{c \times \frac{V(P_j)}{\sum_{k=1}^{|\mathcal{A}(P_i)|} V(P_k)} \sqrt{\frac{\ln(N(P_i))}{1 + N(P_j)}} + Q(P_i, e_j)\} \tag{19}$$

where c is the constant that control the level of exploration, P_i is the current state that needs to select an action from $\mathcal{A}(P_i)$, P_j is the transitioned state after taking e_j from P_i, $e_j \in \mathcal{A}(P_i)$, P_k is the state transitioned from P_i that carries out the k-th action from $\mathcal{A}(P_i)$. Overall, UCT initially prefers actions that can transition to the states with high values of $V(P_j)$ and low visit counts $N(P_j)$, but then asymptotically prefers actions with high values of $Q(P_i, e_j)$.

E. Details of MCTS

Now we elaborate on how MCTS is used to guide our searching process. Each iteration of MCTS can be divided into four steps: *selection*, *expansion*, *simulation* and *backpropagation*. These four steps are iteratively applied until the maximum number of iterations (i.e., I) is reached. The four steps are discussed below and the corresponding pseudo-code is shown in Algorithm 1.

Step 1: Selection. In the selection step, starting from the root node, MCTS traverses the current tree structure using a tree policy. A tree policy uses an evaluation function (i.e., UCT function) that prioritize actions with the greatest estimated values. When the traversal reaches a leaf node, if the leaf node is a terminal state, then MCTS transitions to the backpropagation step, otherwise, it means the leaf node is a new expanded state that has children left to be added, then MCTS transitions to the expansion step. In Fig. 4, starting from the root node A, the tree policy must make a decision among B, C and D, suppose the edge $(A \to B)$ has the maximum UCT value, MCTS will then choose $(A \to B)$ and reach B. Since B is a leaf node with children yet to be added, so now MCTS will transition into the expansion step.

Step 2: Expansion. In the expansion step, for the expanded node reached in the selection step, MCTS selects all valid actions of the expanded node, and adds states transitioned from these actions into the tree structure as child nodes of the expanded node, then initializes $\{V(\cdot), N(\cdot) = 0, \mathcal{R}_V(\cdot) = 0, \hat{\mathcal{R}}_D(\cdot)\}$ of these child nodes, here $V(\cdot)$ and $\hat{\mathcal{R}}_D(\cdot)$ are calculated by E.q. 2 and E.q. 15 respectively, then MCTS

Algorithm 1: RL-MCTS

```
Input: ADG $G_A$; Source node $v_s$, Destination node $v_d$;
       Time interval $[t_\phi, t_\varphi]$; Constraint vector $\mathcal{X}$;
       Number of iterations $I$; Replay memory $D$;
for iteration $it$ in $[1..I]$ do
    Set $v_s$ as the root node of tree structure; set the
      current state $P_{v_s,v_c}(t_c)$; $v_c \leftarrow v_s, t_c \leftarrow t_\phi$;
    while $t_c \leq t_\varphi$ do
        if $P_{v_s,v_c}(t_c)$ is unexplored then
            Mark $P_{v_s,v_c}(t_c)$ explored;
            if $\mathcal{A}(P_{v_s,v_c}(t_c))$ is $\emptyset$ then
                Break;
            for each edge
              $e_i = (v_c, v_i, l_c, tp_c, t_i) \in \mathcal{A}(P_{v_s,v_c}(t_c))$ do
                Add $P_{v_s,v_i}(t_i)$ as the child of $P_{v_s,v_c}(t_c)$;
                Initial $\{V, N, \mathcal{R}_V, \hat{\mathcal{R}}_D\}$ of $P_{v_s,v_i}(t_i)$;
                Mark $P_{v_s,v_i}(t_i)$ unexplored;
        else
            Select the action $e_a = (v_c, v_a, l_c, tp_c, t_a)$ with
              the maximum UCT value;
            Add the transitioned state $P_{v_s,v_a}(t_a)$ into a
              state list;
            Update $P_{v_s,v_c}(t_c) \leftarrow P_{v_s,v_a}(t_a)$;
            if $P_{v_s,v_c}(t_c)$ is a terminal state then
                Break;
    Compute the reward value $\mathcal{R}(P_{v_s,v_c}(t_c))$;
    Delete tuples of earliest iteration in $D$;
    for each state $P_{v_s,v_i}(t_i)$ in the state list do
        Update $\{V, N, \mathcal{R}_V, \hat{\mathcal{R}}_D\}$ of $P_{v_s,v_i}(t_i)$ in the tree
          structure;
        Add $\{\mathbf{H}, \mathbf{T}, \mathcal{R}_V, N\}$ of $P_{v_s,v_i}(t_i)$ into $D$;
```

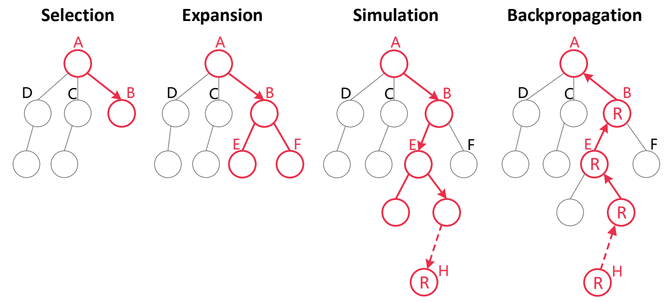

Fig. 4. A brief illustration of MCTS process

TABLE II
DATASET

Dataset	$\lvert V_N \rvert$	$\lvert E \rvert$	$\lvert d_{avg} \rvert$	$\lvert d_{max} \rvert$	π	$\lvert T \rvert$
Arxiv	28K	4597K	327.3	11K	262	2K
Elec	7K	104K	29.1	1K	15	101K
Enron	87K	1148K	26.3	38K	1074	213K
Facebook	47K	877K	37.3	2K	21	737K
Epinions	132K	841K	12.8	3K	13	0.9K
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

moves to the simulation step. Otherwise, if there is no valid actions of the expanded node, it means the expanded node is a terminal state, then MCTS transitions to the backpropagation step. In Fig. 4, the algorithm is currently at the expanded node B, and has two valid actions, $\mathcal{A}(B) = \{B \to E, B \to F\}$, so there are two child nodes added onto B indicated by E and F, then MCTS transitions to the simulation step.

Step 3: Simulation. Since the child nodes of the current expanded node have been added to the tree structure, MCTS can continue to traverse the tree structure. Therefore, In this step, MCTS performs selection and expansion repeatedly until reaches a terminal state and gets a reward, then transitions to the backpropagation step. In Fig. 4, the child nodes of B have been added into the tree structure, then MCTS performs above two steps repeatedly until reaches the terminal state H, and gets the reward R, then transitions to the backpropagation step.

Step 4: Backpropagation. Now that the MCTS has reached the leaf node of 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, the algorithm traverses back to the root node. During the traversal, the statistics stored in each node of the traversal (i.e., $\{V(P_i), N(P_i), \mathcal{R}_V(P_i), \hat{\mathcal{R}}_D(P_i)\}$) are incremented (except the root node), $V(P_i)$ is updated by E.q. 3 or E.q. 4, $\hat{\mathcal{R}}_D(P_i)$ is updated by E.q. 15, $N(P_i)$ and $\mathcal{R}_V(P_i)$ are updated as below:

$$N(P_i) \leftarrow N(P_i) + 1$$

$$\mathcal{R}_V(P_i) \leftarrow \mathcal{R}_V(P_i) + \frac{\mathcal{R}(P_i) - \mathcal{R}_V(P_i)}{N(P_i)}$$

Here $\mathcal{R}(P_i)$ is calculated by E.q. 5, then we use $\mathcal{R}(P_i)$ to update the average reward value of P_i.

In addition, since the replay memory D only stores last M MDPs, the algorithm deletes all tuples of the earliest MDP that stored in D, and puts $\{\mathbf{H}(\cdot), \mathbf{T}(\cdot), \mathcal{R}_V(\cdot), N(\cdot)\}$ of all states that in the traversal into D. In Fig. 4 we can see only nodes of the MDP are updated, i.e., $\{B, E, \cdots, H\}$. This step ensures that the statistics of each node accurately reflect searching results performed in the tree structure.

RL-MCTS performs the above four steps for I iterations. So its time complexity is $O(I \times (selection + expansion + simulation + backpropagation))$. Let F denote the depth of the tree structure, U denote the average number of child nodes at each layer of the tree structure. N_D is the number of entries in D that is used to evaluate the action value. The *Selection* step has the time complexity of $O(U)$, the *Expansion* step has the time complexity of $O(UN_D)$, and the *Simulation* step has the time complexity of $O(UN_D)$, and the *Backpropagation* step has the time complexity of $O(FN_D)$. Therefore, the time complexity of RL-MCTS is $O((U + F)IN_D)$.

TABLE III
PARAMETER SETTINGS

Parameters	Settings
Number of iterations	1000, 2000, 3000, 4000, 5000
Number of attributes	5, 10, 15, 20, 25
Number of entries to approximate (N_D)	100, 500, 1000

TABLE IV
THE COMPARISON OF THE PERFORMANCE OF TPA, MCTS, AND RL-MCTS

Comparison	TPA	MCTS	RL-MCTS
The average path length	163.0400	318.9970	148.4890
The query processing time (Sec)	200.8863	21.7056	41.8482

IV. EXPERIMENTS

A. Experiment Settings

- We conducted experiments on ten large-scale real-world attribute dynamic graphs available at *konect.uni-koblenz.de*. These datasets have been widely used in the literature for the studies of dynamic graphs. The details of these datasets are shown in Table II, 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 (π), and the number of time points ($|T|$) included in each of the attribute dynamic graphs.
- The time interval is set as $\{[0, \frac{1}{5}|T|], [0, \frac{2}{5}|T|], [0, \frac{3}{5}|T|], [0, \frac{4}{5}|T|], [0, |T|]\}$, to investigate the algorithm performance under different $[t_\phi, t_\varphi]$.
- 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 (E.q. 3-E.q. 4) are set to $\delta = 0.95$ and $\mu = 0.9$. When encoding edges of ADGs, the parameters of deepwalk are consistent with the settings in [22]. The parameter of distance function (E.q. 14) is set to $\alpha = 0.5$ and the constant of UCT function (E.q. 19) is set to $c = 0.5$. In the configuration of replay memory, the number of MDP stored in the memory is set to $M = 300$, the threshold is set to $\theta = 0.001$, the number of similar states that is used to make approximation (E.q. 15) is set to $m = 50$. Table III is the variation of mainly parameters.
- The length of each edge is set randomly within the range $[0 - 100]$. 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 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 $\{5, 10, 15, 20, 25\}$, to evaluate the effectiveness of using the memory replay. Accordingly, for the constraint vector $\mathcal{X}$, the constraint on j-th attribute is set as $\lambda_j = \{\frac{1}{5}N_m f_j^a, \frac{2}{5}N_m f_j^a, \frac{3}{5}N_m f_j^a, \frac{4}{5}N_m f_j^a, N_m f_j^a\}$. f_j^a is the average value of j-th attribute function in an ADG.

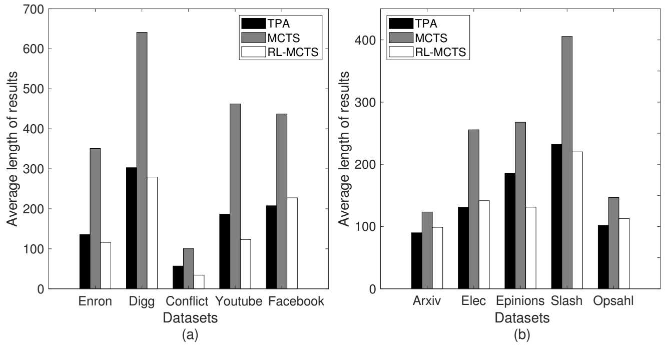

Fig. 5. The average path length based on different datasets

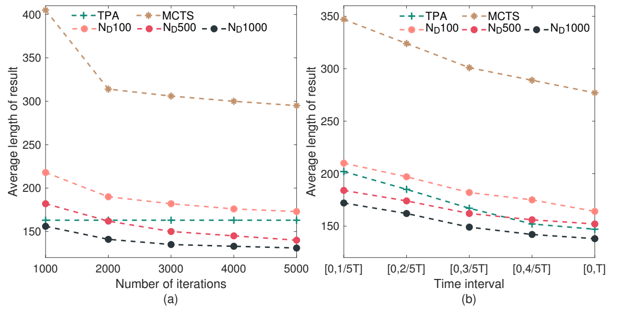

Fig. 6. The average path length based on different iterations and time intervals

B. Implementation

In the following experiments, we will compare our RL-MCTS with the basic Monte Carlo Tree Search algorithm (MCTS), and the state-of-the-art temporal path discovery method, TPA [13]. The performance was investigated by the execution time, and the length of the delivered temporal path.

All TPA, MCTS, and RL-MCTS algorithms are implemented using Matlab R2019a running on a PC with Intel Core i7-7700K 4.2GHz CPU, 16GB RAM, Windows 10 operating system and MySql 5.7 database, all the experimental results are averaged based on five independent runs, and each run the settings of different combinations of all parameters.

C. Experimental Results and Analysis

Exp-1: Effectiveness. In order to investigate the effectiveness of RL-MCTS, we compare the lengths of the searching results based on different datasets and parameters. In addition, we investigate the effectiveness of our replay memory by comparing the searching results based on different number of N_D.

Results: Fig. 5-7 depict the average path length delivered by TPA, MCTS and RL-MCTS in different datasets. From these figures, we can see that (1) in those datasets with more than 100K nodes, RL-MCTS can return shorter average path lengths than those of TPA. Overall, in all ADGs, the average path length delivered by RL-MCTS is *8.92%* less than that of TPA; (2) in all cases, RL-MCTS can always deliver shorter average path lengths than those of MCTS. The average path

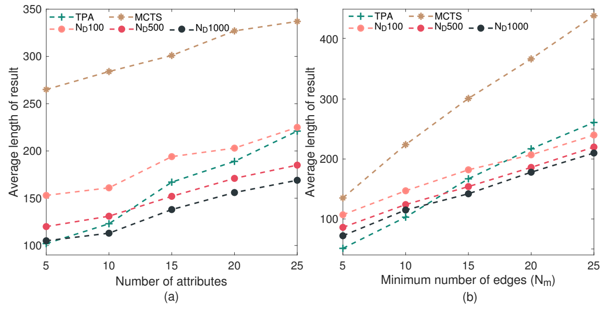

Fig. 7. The average path length based on different attributes and the minimum number of edges

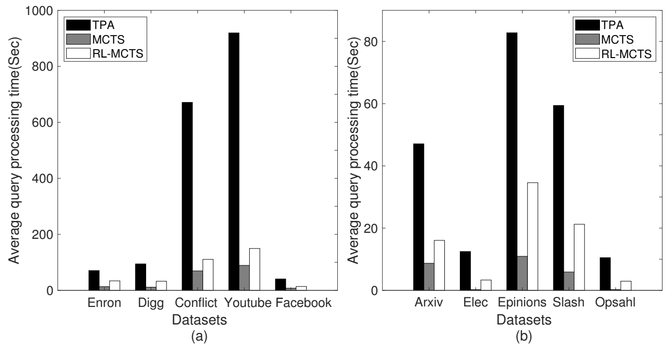

Fig. 8. The average query processing time based on different datasets

length delivered by RL-MCTS is *53.45%* less than that of MCTS; (3) with the increase of iterations and the range of time interval, the path length delivered by all three methods decreases, and RL-MCTS can get shorter path lengths than TPA over more iterations or shorter range of time intervals; (4) with the increase of N_m and number of attributes, the path length delivered by all methods increases, and RL-MCTS can get shorter path lengths than TPA; (5) with the increase of N_D, RL-MCTS with a larger N_D can get better results.

Analysis: The experimental results illustrate that (1) RL-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 greedy optimizing the objectives in TPA, thus RL-MCTS has a higher probability to deliver a better result than TPA; (2) RL-MCTS can use history experience to speed up the searching process by ignoring a large number of states that are unlikely to bring feasible temporal paths, while MCTS needs to take more iterations to get the stable state estimations, and MCTS may converge to the near optimal results, then RL-MCTS can get better results than MCTS; (3) in short range of time intervals, RL-MCTS can discover feasible temporal paths that TPA lost due to the greedy strategy, then RL-MCTS can get shorter paths than TPA; (4) with the increase of N_m and the number of attributes, the states in the searching process will contain more information, then RL-MCTS can use replay memory to contract valid features of states, make more accurate estimations of state values and state action values; (5) with the increase of N_D, RL-MCTS can have more states to be used in the action value estimation, which can improve its performance in the shortest temporal path discovery.

Exp-2: Efficiency. This experiment is to investigate the efficiency of our RL-MCTS by comparing the average query processing time of TPA, MCTS and RL-MCTS based on the different datasets.

Results: Fig. 8 depicts the average query processing time of TPA, MCTS and RL-MCTS on different datasets. From the figure, we can see that (1) in all the cases, TPA costs more query processing time than others, especially on larger scale ADGs; (2) in all cases, the query processing time of RL-MCTS is less than that of TPA and more than that of MCTS; (3) on average, RL-MCTS can save *79.17%* query processing time compared with TPA, and cost *48.13%* more time compared with MCTS. The statistics are shown in Table IV.

Analysis: The experimental results illustrate that (1) RL-MCTS and MCTS access only part of nodes that connected with the destination node and have higher probabilities to possessing feasible temporal paths, while TPA needs to visit all nodes connected with the destination node, leading too much query processing time; (2) due to the replay memory, RL-MCTS has to perform updates of states and make estimations of actions in the searching process, therefore every iteration of RL-MCTS costs more time than MCTS.

V. Related Work

In the literature, there are many studies of the path problem in dynamic graphs, which can be categorised into either the path problem in the *continuous time model* or the path problem in the *discrete time model*.

In the *continuous time model*, an edge between two nodes always exists, and a continuous edge-delay function is associated with each edge in a dynamic graph. Ding et al., [28] apply a more precise refinement approach that expands the time interval step by step. To improve the efficiency of the path queries, some studies build different kinds of indices, such as time-dependent CH [29] and time-dependent SHARC [30]. Furthermore, by considering the waiting time at the starting point, Yang et al., [31] propose an algorithm to find a cost-optimal path with a time constraint by using the iteration method, and Li et al., [32] further consider the waiting time at each of the intermediate vertices in a path query. 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 some specific time points [33].

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., [12] propose the algorithm for searching the shortest, the fastest, and the earliest arrival path in dynamic graphs. Based on this work, in addition to the three paths, Wu et al., [5], [11], propose a one-pass algorithm for the latest departure path [12]. After that, some methods have been proposed to improve the efficiency of path discovery by using indexing techniques [34], [35]. Moreover, some studies consider the waiting time in path discovery. Dean et al., [36] 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 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 that exists in many real applications.

Recently, based on the one-pass algorithm for the discrete time model, Zhao et al., [13] proposes a two-pass approximation algorithm (TPA) for MCTP, which uses a greedy optimization strategy that cannot guarantee the algorithm performance, and TPA has to traverse the graph more than once, which costs much time when the dynamic graph is large.

VI. Conclusion and Remarks

In this paper, we have proposed a new Reinforcement Learning Based Monte Carlo Tree Search algorithm, RL-MCTS, to support Multi-Constrained Temporal Path (MCTP) discovery that is a cornerstone for many dynamic graph-based applications. The experiments conducted on ten real-world large-scale social graphs have demonstrated the superiority of our proposed approaches in terms of effectiveness and efficiency.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (NSFC) (Grant No. 61972069, 61836007, 61832017, 61532018, 61572336), Natural Science Research Project of Jiangsu Higher Education Institution (No. 18KJA520010), and a project funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).

References

[1] Y. Liu, Y. Li, and Y. Zhang, "Study of the logistics transportation vehicle terminal path optimization and algorithm based on gis," in *Applied Mechanics and Materials*, 2014, pp. 2249–2252.

[2] G. Liu, Y. Wang, M. A. Orgun, and E.-P. Lim, "Finding the optimal social trust path for the selection of trustworthy service providers in complex social networks," *IEEE Transactions on Services Computing (TSC)*, 2011.

[3] G. Liu, K. Zheng, Y. Wang, M. A. Orgun, A. Liu, L. Zhao, and X. Zhou, "Multi-constrained graph pattern matching in large-scale contextual social graphs," in *ICDE*, 2015, pp. 351–362.

[4] J. Li, X. Wang, K. Deng, T. Sellis, J. X. Yu, and X. Yang, "Most influential community search over large social networks," in *ICDE*, 2017, pp. 871–882.

[5] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *VLDB*, vol. 7, no. 9, pp. 721–732, 2014.

[6] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos, "Timecrunch: Interpretable dynamic graph summarization," in *KDD*, 2015, pp. 1055–1064.

[7] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *INFOCOM*, 2001, pp. 834–843.

[8] G. Kossinets, J. Kleinberg, and D. Watts, "The structure of information pathways in a social communication network," in *KDD*, 2008, pp. 435–443.

[9] E. Sdol, "Temporal graphs," *Physica A Statistical Mechanics and Its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[10] R. Pan and J. Saramaki, "Path lengths, correlations, and centrality in temporal networks," *Physical Review E*, vol. 84, no. 2, pp. 1577–1589, 2011.

[11] H. Wu, J. Cheng, Y. Ke, S. Huang, Y. Huang, and H. Wu, "Efficient algorithms for temporal path computation," *IEEE TKDE*, vol. 28, no. 11, pp. 2927–2942, 2016.

[12] B. Xuan, A. Ferreira, and A. Jarry, "Computing shortest, fastest, and foremost journeys in dynamic networks," *International Journal of Foundations of Computer Science*, vol. 14, no. 02, pp. 267–285, 2003.

[13] A. Zhao, G. Liu, B. Zheng, Y. Zhao, and K. Zheng, "Temporal paths discovery with multiple constraints in attributed dynamic graphs," *World Wide Web*, 2019.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[15] B. Kartal, E. Nunes, J. Godoy, and M. Gini, "Monte carlo tree search for multi-robot task allocation," in *AAAI*, 2016, pp. 4222–4223.

[16] G. Liu, Y. Wang, and M. A. Orgun, "Optimal social trust path selection in complex social networks," in *AAAI*, 2010, pp. 1391–1398.

[17] A. Guez, D. Silver, and P. Dayan, *Scalable and Efficient Bayes-Adaptive Reinforcement Learning Based on Monte-Carlo Tree Search*, 2013.

[18] H. Finnsson and Y. Bjrnsson, "Simulation-based approach to general game playing," in *National Conference on Artificial Intelligence*, 2008.

[19] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *Proc of the International Conference on Computer and Games*, 2006.

[20] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, 2006, pp. 282–293.

[21] W. L. Hamilton and R. Ying, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[22] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 2014.

[23] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[24] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping," 2001, pp. 1–11.

[25] 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.

[26] T. Hastie, R. Tibshirani, J. H. Friedman, and J. Franklin, *The Elements of Statistical Learning*. Springer New York, 2009.

[27] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[28] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT*, 2008, pp. 205–216.

[29] G. V. Batz, D. Delling, P. Sanders, and C. Vetter, "Effective indexing for approximate constrained shortest path queries on large road networks," in *Proceedings of the Meeting on Algorithm Engineering and Experiments*, 2009, pp. 97–105.

[30] D. Delling, "Time-dependent sharc-routing," *Algorithmica*, vol. 60, no. 1, pp. 60–94, 2011.

[31] Y. Yang, H. Gao, J. X. Yu, and J. Li, "Finding the cost-optimal path with time constraint over time-dependent graphs," *VLDB*, vol. 7, no. 9, pp. 673–684, 2014.

[32] L. Li, W. Hua, X. Du, and X. Zhou, "Minimal on road time route scheduling on time-dependent graph," in *VLDB*, 2017, pp. 1274–1285.

[33] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," in *VLDB*, 2015, pp. 61–72.

[34] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *SIGMOD*, 2015, pp. 967–982.

[35] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in *ICDE*, 2016, pp. 145–156.

[36] B. C. Dean, "Algorithms for minimum-cost paths in time-dependent networks with waiting policies," *Networks*, vol. 44, no. 1, pp. 41–46, 2004.