Finding the Cost-Optimal Path with Time Constraint over Time-Dependent Graphs

Yajun Yang [1,2], Hong Gao [3], Jeffrey Xu Yu [4], Jianzhong Li [3]
[1]School of Computer Science and Technology, Tianjin University, China
[2]Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
[3]School of Computer Science and Technology, Harbin Institute of Technology, China
[4]The Chinese University of Hong Kong, China
yjyang@tju.edu.cn, honggao@hit.edu.cn, yu@se.cuhk.edu.hk, lijzh@hit.edu.cn

ABSTRACT

Shortest path query is an important problem and has been well studied in static graphs. However, in practice, the costs of edges in graphs always change over time. We call such graphs as time-dependent graphs. In this paper, we study how to find a cost-optimal path with time constraint in time-dependent graphs. Most existing works regarding the Time-Dependent Shortest Path (TDSP) problem focus on finding a shortest path with the minimum travel time. All these works are based on the following fact: the earliest arrival time at a vertex v can be derived from the earliest arrival time at v's neighbors. Unfortunately, this fact does not hold for our problem. In this paper, we propose a novel algorithm to compute a cost-optimal path with time constraint in time-dependent graphs. We show that the time and space complexities of our algorithm are $O(kn \log n + mk)$ and $O((n + m)k)$ respectively. We confirm the effectiveness and efficiency of our algorithm through conducting experiments on real datasets with synthetic cost.

1. INTRODUCTION

Shortest path query is an important problem in graphs and has been well studied in static graphs. However, graphs often evolve over time. For example, the Vehicle Information and Communication System (VICS) and the European Traffic Message Channel (TMC) are two transportation systems, which can provide real-time traffic information to users. Such transportation networks are time-dependent graphs, i.e., the travel time for a road varies with time taking "rush hour" into account. Meanwhile, the toll fee of a road is also time-dependent. For example, there are "London congestion charge" and "Road pricing policy" to reduce traffic congestion and control traffic pollution in the United Kingdom [14]. The vehicles are charged if they pass through the major roads in rush hours. The similar policies are also applied in Singapore. This results in the variation of the toll fee of a road in different hours of a day and different days of a week, which shows the toll fee is time-dependent. Moreover, there are several works that study the pricing mechanism for the time-dependent toll fee [15, 12].

Proceedings of the VLDB Endowment, Vol. 7, No. 9

Consider an application in a road network. Someone has an appointment with her friends. The earliest departure time for her is t_1 and she has to arrive at rendezvous before time t_2. In this road network, there are two kinds of costs for every road, travel time and toll fee, which are both time-dependent. According to the travel time, it can be verified whether a path p satisfies time constraint or not, i.e., whether one can arrive before time t_2 along path p. It is worth noting that there may be several paths from the source to the destination satisfying the time constraint. Therefore, it is very important to find an optimal path with the minimum cost from all the paths satisfying the time constraint.

In the above example, the road network can be considered as a large graph G with time information. Every edge (v_i, v_j) in G has two kinds of costs: $w_{i,j}(t)$ and $f_{i,j}(t)$. $w_{i,j}(t)$ is the time cost to specify how long it takes to travel through an edge (v_i, v_j), and $f_{i,j}(t)$ is the toll fee for traveling through an edge (v_i, v_j). Both $w_{i,j}(t)$ and $f_{i,j}(t)$ are the functions that are dependent on the departure time t at the starting endpoint v_i of the edge (v_i, v_j). We call such graphs time-dependent graphs. The query of a cost-optimal path with time constraint in time-dependent graphs can be defined as follows. Given a source v_s, a destination v_e, the earliest departure time t_d and the latest arrival time t_a, find an optimal path p from v_s to v_e, satisfying the following two conditions: (1) departing from v_s after time t_d, one can arrive at v_e before time t_a along path p; and (2) path p has the minimum cost (toll fee) among all the paths satisfying the condition (1).

There are many works on the shortest path problem in time-dependent graphs [13, 7]. Most of them are to find an optimal path with the minimum travel time from the source to the destination, when the departure time from the source can be selected from a user-given starting-time interval. These works assume there is only one time function $w_{i,j}(t)$ on every edge in a time-dependent graph. Let λ_i denote the earliest arrival time at vertex v_i. λ_i can be calculated by the following equation:

$$\lambda_i = \min\{(\lambda_j + \omega(v_j)) + w_{j,i}(\lambda_j + \omega(v_j)) | v_j \in N^-(v_i)\}$$

where $v_j \in N^-(v_i)$ represents that v_j is an incoming neighbor of v_i and $\omega(v_j)$ is the waiting time at v_j. This equation indicates that the earliest arrival time at a vertex can be obtained based on the earliest arrival time at this vertex's incoming neighbors. All these works on the TDSP problem utilize this property to compute the shortest paths with the minimum travel time. Unfortunately, this property does not hold for our problem (detailed in Section 2.2). Thus, the existing works on the TDSP problem cannot solve our problem proposed in this paper.

In this paper, we study the problem of identifying a cost-optimal path with time constraint in time-dependent graphs. Different to

the TDSP problem, we consider two kinds of costs for every edge in this paper. To the best of our knowledge, our work is the first one regarding this problem under the continuous time model. The main contributions are summarized below. First, we propose a novel algorithm to find a cost-optimal path with time constraint in time-dependent graphs. Our algorithm can handle both undirected and directed time-dependent graphs. Second, we show that the time and space complexities of our algorithm are $O(kn \log n + mk)$ and $O((n+m)k)$ respectively, where n is the number of vertices, and m is the number of edges, and k is the average number of piecewise constant values of the atmc-function (detailed in Section 3.1). Third, we confirm the effectiveness and efficiency of our algorithm through conducting experiments on real datasets. Compared with the state of the art algorithm used for discrete time model, our algorithm not only can find the optimal path but also makes at least two orders of magnitude improvement in time and space overhead.

The rest of the paper is organized as follows. Section 2 defines the problem. Section 3 proposes a two-step algorithm to find a cost-optimal path with time constraint. The experiment results are presented in Section 4. The related works are introduced in Section 5. We conclude this paper in Section 6.

2. PROBLEM STATEMENT

2.1 Time-Dependent Graph and Cost-Optimal Path with Time Constraint

Definition 2.1: (**Time-Dependent Graph**) A time-dependent graph is a simple graph, denoted as $G_T(V, E, W, F)$ (or G_T for short), where $V = \{v_i\}$ is the set of vertices; $E \subseteq V \times V$ is the set of edges; W and F are two sets of non-negative value functions. For every edge $(v_i, v_j) \in E$, there are two functions: time-function $w_{i,j}(t) \in W$ and cost-function $f_{i,j}(t) \in F$, where t is a time variable. A time function $w_{i,j}(t)$ specifies how much time it takes to travel from v_i to v_j, if departing from v_i at time t. A cost function $f_{i,j}(t)$ specifies how much cost (e.g., toll fee) it takes to travel from v_i to v_j, if departing from v_i at time t. □

In this paper, we assume that $w_{i,j}(t) \geq 0$ and $f_{i,j}(t) \geq 0$. The assumption is reasonable, because the travel time and travel cost cannot be less than zero in real applications. Our work can be easily extended to handle undirected graphs, in which an undirected edge (v_i, v_j) is equivalent to two directed edges (v_i, v_j) and (v_j, v_i), where $w_{i,j}(t) = w_{j,i}(t)$ and $f_{i,j}(t) = f_{j,i}(t)$. For simplicity, we only consider directed graphs in the rest of this paper.

We assume that cost-function $f_{i,j}(t)$ is a piecewise constant function, which can be formalized as follows:

$$f_{i,j}(t) = \begin{cases} c_1, & t^0 \leq t < t^1 \\ c_2, & t^1 \leq t < t^2 \\ \dots & \\ c_p, & t^{\sigma-1} \leq t \leq t^{\sigma} \end{cases}$$

Here, $[t^0, t^\sigma]$ is the time domain of function $f_{i,j}(t)$. c_x $(1 \leq x \leq \sigma)$ is a constant value, which represents the value of $f_{i,j}(t)$ when $t \in [t^{x-1}, t^x]$. The assumption is reasonable. In real applications, the cost functions are always piecewise constant. For example, in road networks, the toll fees for traveling through a road are distinct constant values during day and night. It means the cost-function of this road is a piecewise constant function.

Given a path p, the cost of p is also time dependent. For any edge $(v_i, v_j) \in p$, if the departure time from v_i is different, the travel cost for edge (v_i, v_j) is different too. In order to find a cost-optimal path, some waiting time is allowed, denoted as $\omega(v_i)$, at each vertex v_i. That is, when arriving at vertex v_i, one can wait a time period $\omega(v_i)$ if the cost of path p can be minimized. We use $\mathsf{arrive}(v_i)$ and $\mathsf{depart}(v_i)$ to denote the arrival time at v_i and departure time from v_i, respectively. For each vertex v_i, we have

$$\mathsf{depart}(v_i) = \mathsf{arrive}(v_i) + \omega(v_i)$$

Let $p = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_h$ be a given path with the earliest departure time t_d and the waiting time $\omega(v_i)$ for each vertex v_i, then we have

$$\begin{aligned} &\mathsf{arrive}(v_1) = t_d \\ &\mathsf{arrive}(v_2) = \mathsf{depart}(v_1) + w_{1,2}(\mathsf{depart}(v_1)) \\ &\quad \dots \\ &\mathsf{arrive}(v_h) = \mathsf{depart}(v_{h-1}) + w_{h-1,h}(\mathsf{depart}(v_{h-1})) \end{aligned}$$

For any vertex $v_i \in p$, we use $\mathsf{cost}_p(v_i)$ to denote the cost from v_1 to v_i by path p. $\mathsf{cost}_p(v_i)$ can be calculated recursively as follows:

$$\begin{aligned} &\mathsf{cost}_p(v_1) = 0 \\ &\mathsf{cost}_p(v_2) = \mathsf{cost}_p(v_1) + f_{1,2}(\mathsf{depart}(v_1)) \\ &\quad \dots \\ &\mathsf{cost}_p(v_h) = \mathsf{cost}_p(v_{h-1}) + f_{h-1,h}(\mathsf{depart}(v_{h-1})) \end{aligned}$$

The cost of path p is defined as $\mathsf{cost}(p) = \mathsf{cost}_p(v_h)$. Next, we give the definition of the problem of the cost-optimal path with time constraint in time-dependent graphs.

Definition 2.2: (**Cost-Optimal Path with Time Constraint**) Given a time-dependent graph G_T, a source vertex v_s, a destination vertex v_e, the earliest departure time t_d and the latest arrival time t_a, the problem of the cost-optimal path with time constraint is to find an optimal path p^* and the optimal waiting time $\omega^*(v_i)$ $(\omega^*(v_i) \geq 0)$ for every vertex $v_i \in p^*$, such that (1) $\mathsf{depart}(v_s) \geq t_d \wedge \mathsf{arrive}(v_e) \leq t_a$; and (2) $\mathsf{cost}(p^*)$ is the minimum among all the paths from v_s to v_e that satisfy the condition (1). □

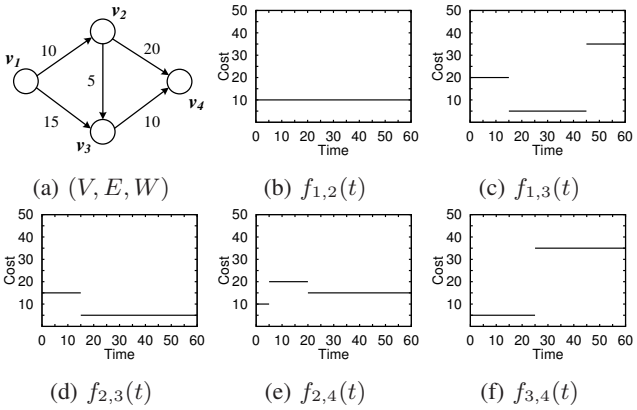


Figure 1: Time-dependent graph

Fig. 1 illustrates an example of time-dependent graph G_T. Here, Fig. 1(a) presents the structure of G_T and the travel time $w_{i,j}(t)$ for every edge $(v_i, v_j) \in G_T$. In this example, the travel time for every edge is a constant value. The cost-functions $f_{i,j}(t)$ for five edges, (v_1, v_2), (v_1, v_3), (v_2, v_3), (v_2, v_4) and (v_3, v_4) are shown in Fig. 1(b), (c), (d), (e) and (f), respectively.

Given a query of the cost-optimal path with time constraint: $v_s = v_1$, $v_e = v_4$, $t_d = 0$ and $t_a = 60$, a cost-optimal path is $p^* = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$. The optimal waiting time are $\omega^*(v_1) = 0$, $\omega^*(v_2) = 5$, and $\omega^*(v_3) = 0$, respectively. The cost of the optimal path p^* is $\mathsf{cost}(p^*) = 20$.

2.2 Existing Solutions for the TDSP Problem

Most existing works for the TDSP problem are to find an optimal path with the minimum travel time. We discuss two most recently published efficient algorithms for the TDSP problem and give the reasons that why they cannot be used to solve our problem.

A* Algorithm: Kanoulas et al. in [13] propose an extension to A* algorithm for the TDSP problem. The main idea is to estimate a lower bound of travel time from source to destination and expand path utilizing this lower bound. The main problems of this A*-extended algorithm are summarized below. (1) This algorithm needs to compute the Euclidean distance between any two vertices and the maximum speed in a road network to estimate the lower bound by the equation "$\frac{\text{distance}}{\text{speed}}$". However, for our problem, travel cost cannot be estimated by this equation. Therefore, this algorithm cannot be used to our problem. (2) The efficiency of this algorithm is dependent on the pruning power of the estimation of travel time. This algorithm is efficient when source and destination are close to each other in a graph. It is difficult to figure out such an estimation in a large graph. When a graph is large or source is far away from destination, the algorithm is inefficient. (3) In the worst case, all paths from source to destination are enumerated and maintained, and then the time and space complexities of this algorithm is exponential w.r.t. the size of G_T.

2S Algorithm: Ding et al. in [7] propose a more efficient algorithm to address the TDSP problem. This algorithm includes two phases: (1) *time-refinement phase*; and (2) *path-selection phase*. In the first phase, the algorithm refines the earliest arrival time for every vertex v_i by the following equation:

$$g_i(t) = \min_{v_j \in N^-(v_i), \omega(v_j)} \{(g_j(t) + \omega(v_j)) + w_{j,i}(g_j(t) + \omega(v_j))\} \quad (1)$$

Here, $g_i(t)$ is the earliest arrival time for v_i, if departing from source v_s at starting time t. $N^-(v_i)$ is v_i's incoming neighbor set, i.e., $N^-(v_i) = \{v_j | (v_j, v_i) \in E\}$. The algorithm utilizes a priority queue Q to maintain the earliest arrival time function $g_i(t)$ and a time interval $[t_s, \tau_i]$ for all the vertices in G_T. The value of $g_i(t)$ for $t \in [t_s, \tau_i]$ is corrected. In each iteration, a vertex v_i with the minimum $g_i(\tau_i)$ is dequeued from Q. The algorithm refines $g_i(t)$ and $[t_s, \tau_i]$ by Eq. (1). Let I denote the user-given starting time interval. If $[t_s, \tau_i] \neq I$, then v_i is inserted into Q again. The algorithm terminates when the earliest arrival time function $g_e(t)$ of destination v_e has been refined in the whole time interval I.

The main problem of the 2S algorithm is that this algorithm needs to compute the earliest arrival time function by Eq. (1). However, the rationale Eq. (1) based on does not hold for the cost-optimal path problem proposed in this paper. We clarify this point using the example in Fig. 1. Suppose our objective is to find a cost-optimal path from v_1 to v_4. For an incoming neighbor v_3 of vertex v_4, we find the minimum cost from v_1 to v_3 is $g_3(15) = 5$, i.e., The cost of path $v_1 \rightarrow v_3$ is the minimum when departing from source v_1 at time $t = 15$. In this case, the arrival time at v_3 is 30. In the other words, the earliest departure time from v_3 is 30 if one arrives at v_3 with cost $g_3(15) = 5$. By Eq. (1), the minimum cost from v_1 to v_4 by edge (v_3, v_4) is $g_3(15) + w_{3,4}(30) = 5 + 35 = 40$. However, the cost-optimal path from v_1 to v_4 via v_3 is $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$, and its cost is 20. In this path, the cost from v_1 to v_3 is 15, which is larger than the minimum cost $g_3(15) = 5$. This example shows that a sub-path of the cost-optimal path may not be a cost-optimal path. In the example, the cost-optimal path from v_1 to v_4 is $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$, but its sub-path $v_1 \rightarrow v_2 \rightarrow v_3$ is not a cost-optimal path from v_1 to v_3. Let $p_{s \rightsquigarrow j}$ denote the cost-optimal path from source v_s to v_j and t_j denote the arrival time at v_j along path $p_{s \rightsquigarrow j}$. Note that there may exist another path $p'_{s \rightsquigarrow j}$ from v_s to v_j.

Algorithm 1 TWO-STEP-SEARCH (G_T, v_s, v_e, t_d, t_a)

Input: G_T, v_s and v_e, t_d and t_a.
Output: an optimal path p^* and $\omega^*(v_i)$ for every vertex $v_i \in p^*$.

```
1: COMPUTE-MINIMUM-COST (G_T, v_s, v_e, t_d, t_a);
2: if g_e(t_e) ≠ ∞ then
3:     PATH-SELECTION (g_e(t_e), G_T, v_s, v_e, t_d, t_a);
4:     return p* and ω*(v_i) for every v_i ∈ p*;
5: else
6:     return ∅;
```

Similarly, t'_j is the arrival time at v_j along path p'_j. The cost of path $p'_{s \rightsquigarrow j}$ is slightly larger than that of $p_{s \rightsquigarrow j}$. However, the minimum cost to travel through edge (v_j, v_i) after time t_j is far more than after time t'_j, i.e., $\min\{w_{j,i}(t) | t \geq t_j\} \gg \min\{w_{j,i}(t) | t \geq t'_j\}$. Thus, $p_{s \rightsquigarrow j \rightarrow i}$ is not a cost-optimal path from v_s to v_i. It means the minimum arrival cost at a vertex v_i cannot be computed based on the minimum arrival costs at v's incoming neighbors, which is indicated by Eq. (1). It is because that a sub-path of a cost-optimal path may not be the cost-optimal path.

Several studies in the field of operation research consider the optimal path problem under the discrete time model [3, 2]. In the discrete time model, a whole time interval is discretized to a set of time points, $\{t_1, t_2, \cdots, t_l\}$. For any edge (v_i, v_j), only one specified time point t_x to depart from v_i can be selected. The main disadvantages of these works are as follows: (1) An optimal path may not be found under the discrete time model. Suppose one arrives at vertex v_i at time t, $t_{i-1} < t < t_i$, where t_{i-1} and t_i are two consecutive time points in the given set of discrete time points. The earliest departure time from v_i is t_i. However, the best departure time is t', because the cost to travel through edge (v_i, v_j) is the minimum if one departs from v_i at time t'. Here, $t < t' < t_i$. (2) These works need to compute the arriving cost for every vertex at every time point. The time and space costs are expensive.

3. TWO-STEP ALGORITHM

We propose an efficient TWO-STEP-SEARCH algorithm. We first introduce what is the "arrival-time and the minimum-cost function" (or atmc-function for simplicity) and how to compute atmc-function for a vertex v_i in G_T. Second, we introduce the first step of the TWO-STEP-SEARCH algorithm, i.e., how to compute the minimum cost from source v_s to destination v_e . Third, we introduce the second step of the TWO-STEP-SEARCH algorithm, i.e., how to find a cost-optimal path and the optimal waiting time for every vertex in the optimal path. Finally, we discuss the time complexity and space complexity. The TWO-STEP-SEARCH algorithm is shown in Algorithm 1.

In the following, we first focus on the case where the travel time for every edge (v_i, v_j) in G_T is a constant value $w_{i,j}$, i.e., $w_{i,j}(t) = w_{i,j}$. We will discuss how to deal with the general case where the travel time for every edge is a function $w_{i,j}(t)$ in Section 3.5. $w_{i,j}(t)$ is related to the departure time t from v_i.

3.1 Arrival-Time and Min-Cost Function

Given a time-dependent graph G_T, for any vertex $v_i \in G_T$, there may exist several paths that depart from source v_s after time t_d and arrive at v_i at time point t. Let $P_i(t)$ denote the set of all such paths and $g_i(t)$ denote the minimum cost among all the paths in $P_i(t)$, that is,

$$g_i(t) = \min\{\mathsf{cost}(p) | p \in P_i(t)\}$$

Note that some waiting time is allowed, as long as a path p satisfies the time constraint: (1) $\mathsf{depart}_p(v_s) \geq t_d$, it means that one departs

Table 1: Important notations

Notation	Description
G_T, v_s, v_e	time-dependent graph, source, destination
t_d, t_a	earliest departure time, latest arrival time
$f_{i,j}(t)$, $w_{i,j}(t)$	cost-function, time-function
$g_i(t)$	atmc-function of vertex v_i
$g_{j\to i}(t)$	atmc-function by edge (v_j, v_i)
λ_j	earliest arrival time at v_j

from source v_s after time t_d by path p; and (2) $\mathsf{arrive}_p(v_i) = t$, it means that one arrives at v_i at time t by path p. $g_i(t)$ is a function related to arrival time t for v_i. We call $g_i(t)$ the atmc-function of vertex v_i. $g_i(t)$ represents the minimum cost that one can arrive at v_i at time t from source v_s.

Given a time-dependent graph G_T, if the cost function $f_{i,j}(t)$ is a piecewise constant function for every edge (v_i, v_j) in G_T, then for any vertex $v_i \in V$, it is obvious that $g_i(t)$ of v_i is also a piecewise constant function.

Based on the atmc-function, the minimum cost from v_s to v_e can be defined as $g_e(t_e) = \min\{g_e(t)|t \in [\lambda_e, t_a]\}$, where t_e is a time point that minimizes $g_e(t)$ for $t \in [\lambda_e, t_a]$ and λ_e represents the earliest arrival time for v_e if departing from v_s after the earliest departure time t_d.

The main idea in the first step of the TWO-STEP-SEARCH algorithm is to update atmc-function $g_i(t)$ iteratively for every vertex $v_i \in G_T$ until the minimum cost from source v_s to destination v_e is derived. Note that in each iteration, the current $g_i(t)$ of v_i may not be an optimal (or correct) atmc-function. We need to update $g_i(t)$ using its incoming neighbors' atmc-functions such that $g_i(t)$ is closer to the optimal atmc-function.

Next, we discuss how to update $g_i(t)$ for a vertex v_i. Let v_j be an incoming neighbor of v_i, i.e., $v_j \in N^-(v_i)$. Suppose $g_j(t)$ is the current atmc-function of v_j. The updating process includes two phases: (1) compute $g_{j\to i}(t)$, $g_{j\to i}(t)$ is a function as similar as atmc-function, which represents the minimum cost that one departing from v_s can arrive at v_i at time point t by edge (v_j, v_i); and (2) update $g_i(t)$ using $g_{j\to i}(t)$.

If the waiting time is not allowed, i.e., $\omega(v_j) = 0$, we have:

$$g_{j\to i}(t) = g_j(t - w_{j,i}) + f_{j,i}(t - w_{j,i}) \tag{2}$$

The meaning of Eq. (2) is that: if the arrival time for vertex v_i is t, then one need to depart from v_j at time $t - w_{j,i}$. Because there is no waiting time, the arrival time for v_j is $t - w_{j,i}$. $g_j(t - w_{j,i})$ is the minimum cost from v_s to v_j for arrival time $t - w_{j,i}$ and $f_{j,i}(t - w_{j,i}(t))$ is the cost to travel edge (v_j, v_i), then $g_{j\to i}(t)$ is the sum of $g_j(t - w_{j,i})$ and $f_{j,i}(t - w_{j,i}(t))$.

In this paper, some waiting time is allowed, then we have

$$\begin{aligned} g_{j\to i}(t) &= \min_{t', \omega(v_j)} \{g_j(t') + f_{j,i}(t' + \omega(v_j))\} \\ t &= t' + \omega(v_j) + w_{j,i} \end{aligned} \tag{3}$$

Here, $t' + \omega(v_j)$ is the departure time from v_j and t' is the arrival time for v_j. It means after arriving at v_j, one needs to wait for $\omega(v_j)$ time before departure from v_j. To guarantee that the arrival time for v_i is t, an appropriate waiting time is necessary to satisfy that $t = t' + \omega(v_j) + w_{j,i}$. Therefore, to compute $g_{j\to i}(t)$ is equivalent to find the optimal arrival time t' and the waiting time $\omega(v_j)$, which satisfies $t = t' + \omega(v_j) + w_{j,i}$, such that $g_{j\to i}(t)$ is minimized in Eq. (3).

Given the arrival time t for vertex v_i, the departure time from v_j is fixed to $t - w_{j,i}$. Then, the cost to travel the edge (v_j, v_i) is $f_{j,i}(t - w_{j,i})$. By the definition of the function $g_{j\to i}(t)$, we have the following equation:

$$\begin{aligned} g_{j\to i}(t) &= \min_{t' \le t - w_{j,i}} \{g_j(t') + f_{j,i}(t - w_{j,i})\} \\ &= \min_{t' \le t - w_{j,i}} \{g_j(t')\} + f_{j,i}(t - w_{j,i}) \end{aligned} \tag{4}$$

By Eq. (4), we find that computing $g_{j\to i}(t)$ is equivalent to find the optimal time t' ($t' \le t - w_{j,i}$), such that $g_j(t')$ is the minimum. In the other words, for any departure time point $t - w_{j,i}$, we only need to find the minimum $g_j(t')$ for $t' \le t - w_{j,i}$ to minimize the sum of $g_j(t')$ and $f_{j,i}(t - w_{j,i})$.

We use λ_j to denote the earliest arrival time at vertex v_j if departing from v_s at the earliest departure time t_d. It indicates that one cannot arrive at v_j before λ_j if departing from v_s at (or after) time t_d. Thus, the time domain of $g_j(t)$ is $[\lambda_j, t_a]$. Similarly, the time domain of $g_{j\to i}(t)$ is $[\lambda_{j\to i}, t_a]$, where $\lambda_{j\to i} = \lambda_j + w_{j,i}$. $\lambda_{j\to i}$ is the earliest time that one can arrive at v_i by edge (v_j, v_i) if departing from v_s at time t_d.

The procedure to compute $g_{j\to i}(t)$ is as follows. We find the minimum $g_j(t')$ iteratively to minimize $g_{j\to i}(t)$ in Eq. (4) until $g_{j\to i}(t)$ is computed for $t \in [\lambda_{j\to i}, t_a]$. To compute $g_{j\to i}(t)$, the whole time interval of $g_j(t)$ is set as $T_{j,i} = [\lambda_j, t_a - w_{j,i}]$, because one cannot arrive at v_i before time t_a if departing from v_j after $t_a - w_{j,i}$. We use $S_{j,i}$ to denote the processed time interval for $g_j(t)$ in each iteration. It means that the minimum $g_j(t')$ in Eq. (4) has been found to minimize $g_{j\to i}(t)$ for $t \in S_{j,i} \oplus w_{j,i}$. Here, $S_{j,i} \oplus w_{j,i}$ represents the time interval derived by adding $w_{j,i}$ to all the time points in $S_{j,i}$. For example, if $S_{j,i} = [t^a, t^b]$, then $S_{j,i} \oplus w_{j,i} = [t^a + w_{j,i}, t^b + w_{j,i}]$. $S_{j,i}$ is initialized as $\emptyset$. We use τ_j to denote the minimum value of $g_j(t)$ for $t \in T_{j,i} - S_{j,i}$, i.e.,

$$\tau_j = \min\{g_j(t)|t \in T_{j,i} - S_{j,i}\}$$

We use t_j to denote the minimum time point t such that $g_j(t)$ equals to τ_j for $t \in T_{j,i} - S_{j,i}$, i.e.,

$$t_j = \min\{t|g_j(t) = \tau_j, t \in T_{j,i} - S_{j,i}\}$$

We use $R_{j,i}$ to denote time interval $[t_j, t_a - w_{j,i}]$. Then, τ_j is the minimum value of $g_j(t)$, i.e., $g_j(t')$ in Eq. (4), which minimizes $g_{j\to i}(t)$ for $t \in (R_{j,i} - S_{j,i}) \oplus w_{j,i}$. $g_{j\to i}(t)$ for $t \in (R_{j,i} - S_{j,i}) \oplus w_{j,i}$ can be computed by the following equation:

$$g_{j\to i}(t) = f_{j,i}(t - w_{j,i}) + \tau_j$$

After computing $g_{j\to i}(t)$ for $t \in (R_{j,i} - S_{j,i}) \oplus w_{j,i}$, $S_{j,i}$ is updated as $S_{j,i} \leftarrow R_{j,i}$ and the procedure to compute $g_{j\to i}(t)$ is repeated. This procedure terminates when $S_{j,i} = T_{j,i}$. At this moment, $g_{j\to i}(t)$ is computed for any time point $t \in [\lambda_{j\to i}, t_a]$ because $T_{j,i} \oplus w_{j,i} = [\lambda_{j\to i}, t_a]$.

Note that it cannot find the minimum time point t_j such that $g_j(t) = \tau_j$ if $g_j(t) = \tau_j$ for an open time interval, i.e., $g_j(t) = \tau_j$ for $t \in (t^a, t^b]$. In this case, t_j is set as the left endpoint t^a and $R_{j,i}$ is also an open time interval $(t_j, t_a - w_{j,i}]$.

We illustrate the procedure to compute $g_{j\to i}(t)$ by the example in Fig. 2. In Fig. 2, the solid line represents the cost-function $f_{j,i}(t)$ and the dash line represents the atmc-function $g_j(t)$. Initially, $S_{j,i} = \emptyset$ and $T_{j,i} - S_{j,i} = [\lambda_j, t_a - w_{j,i}]$. We find the minimum value of $g_j(t)$ is 15 for $t \in [\lambda_j, t_a - w_{j,i}]$, i.e., $\tau_j = 15$. t_j is the earliest time point such that $g_j(t) = \tau_j$, and thus $R_{j,i} = [t_j, t_a - w_{j,i}]$. Then, we have $g_{j\to i}(t) = f_{j,i}(t - w_{j,i}) + \tau_j = 10{+}15 = 25$ for $t \in [t_j + w_{j,i}, t_a]$. Next, $S_{j,i} \leftarrow R_{j,i}$. We find the minimum value of $g_j(t)$ is 20 for $t \in T_{j,i} - S_{j,i}$ and λ_j is the earliest time point such that $g_j(t) = 20$. We have $R_{j,i} = [\lambda_j, t_a - w_{j,i}]$ and $R_{j,i} - S_{j,i} = [\lambda_j, t_j)$. Thus, $g_{j\to i}(t) = 20 + 10 = 30$ for

$t \in (R_{j,i} - S_{j,i}) \oplus w_{j,i} = [\lambda_{j\to i}, t_j + w_{j,i})$. At this moment, $g_{j\to i}(t)$ has been computed for the whole time interval $[\lambda_{j\to i}, t_a]$.

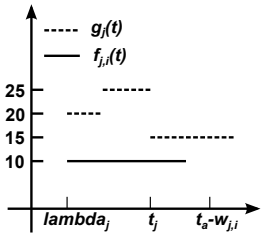


Figure 2: Computing $g_{j\to i}(t)$

The following theorem guarantees the correctness of the procedure to compute $g_{j\to i}(t)$.

Theorem 3.1: *Given a time-dependent graph G_T, $v_i, v_j \in G_T$ and $v_j \in N^-(v_i)$. Let $g_j(t)$ be the* atmc*-function of v_j. The $g_{j\to i}(t)$ computed in the above procedure is correct, i.e., for $\forall t \in [\lambda_{j\to i}, t_a]$, $g_{j\to i}(t)$ is the minimum cost that one can arrive at v_i at time point t by edge (v_j, v_i) if departing from v_s after (or at) the earliest departure time t_d.* □

PROOF. For $\forall t \in [\lambda_{j\to i}, t_a]$, we use $g^*_{j\to i}(t)$ to denote the minimum cost that one can arrive at v_i via v_j at time point t by edge (v_j, v_i) if departing from v_s after (or at) time point t_d. We only need to prove $g_{j\to i}(t) = g^*_{j\to i}(t)$. Because $g^*_{j\to i}(t)$ is the minimum cost, then we have $g^*_{j\to i}(t) \leq g_{j\to i}(t)$. Next, we prove $g_{j\to i}(t) \leq g_{j\to i}(t)^*$. By the definitions of $g_{j\to i}(t)$ and $g^*_{j\to i}(t)$, we have the equation:

$$g^*_{j\to i}(t) = g_j(t'^*) + f_{j,i}(t - w_{j,i})$$

and the equation:

$$g_{j\to i}(t) = g_j(t') + f_{j,i}(t - w_{j,i})$$

where $g_j(t'^*)$ and $g_j(t')$ are the costs that one arrives at v_j for $g^*_{j\to i}(t)$ and $g_{j\to i}(t)$ respectively. It is obvious that $t'^* \leq t - w_{j,i}$ and $t' \leq t - w_{j,i}$. To compute $g_{j\to i}(t)$, $g_j(t')$ is selected as the minimum value of $g_j(t)$ for $t' \leq t - w_{j,i}$. Then we have $g_j(t') \leq g_j(t'^*)$ and $g_{j\to i}(t) \leq g^*_{j\to i}(t)$. □

After computing $g_{j\to i}(t)$, we can utilize $g_{j\to i}(t)$ to update $g_i(t)$ by the following equation:

$$g_i(t) = \min\{g_{j\to i}(t), g_i(t) | t \in [\lambda_{j\to i}, t_a]\} \quad (5)$$

In Section 3.2.1, we show that a vertex v_j is dequeued iteratively from queue Q according to τ_j in the first step of the TWO-STEP-SEARCH algorithm. Therefore, in each iteration, we only need to compute $g_{j\to i}(t)$ and update $g_i(t)$ for $t \in (R_{j,i} - S_{j,i}) \oplus w_{j,i}$, for every outgoing neighbor v_i of v_j.

3.2 Compute the Minimum Cost

In this section, we introduce the first step of the TWO-STEP-SEARCH algorithm, i.e., how to compute the minimum cost from source v_s to destination v_e.

3.2.1 Main idea

We pre-compute the earliest arrival time λ_i for every vertex $v_i \in V$. The minimum travel time to every vertex v_i from v_s can be computed by executing the single-source shortest path algorithm on G_T according to the time cost $w_{i,j}$ on every edge (v_i, v_j). The earliest arrival time λ_i of v_i is the sum of the earliest departure time t_d and the minimum travel time to v_i. The time complexity of the single-source shortest path algorithm is $O(n \log n + m)$.

Algorithm 2 COMPUTE-MINIMUM-COST $(G_T, v_s, v_e, t_d, t_a)$

Input: G_T, v_s and v_e, t_d and t_a.
Output: minimum cost $g_e(t_e)$ from v_s to v_e.

```
1: g_s(t) ← 0; τ_s ← 0; S_s ← T_s; v_i ← v_s;
2: Let Q be the priority queue initially containing V;
3: while v_i ≠ v_e do
4:    Let t_i be the earliest time point such that g_i(t_i) = τ_i;
5:    S_i ← [t_i, t_a];
6:    for each v_j ∈ N^+(v_i) do
7:       if t_i ≤ t_a − w_{i,j} then
8:          R_{i,j} ← [t_i, t_a − w_{i,j}];
9:          g_{i→j}(t) ← f_{i,j}(t−w_{i,j}) + τ_i for t ∈ (R_{i,j} − S_{i,j}) ⊕ w_{i,j};
10:         S_{i,j} ← R_{i,j};
11:         g_j(t) ← min{g_j(t), g_{i→j}(t) | t ∈ (R_{i,j} − S_{i,j}) ⊕ w_{i,j}};
12:         τ_j ← min{g_j(t) | t ∈ T_j − S_j};
13:   if S_i ≠ T_i then
14:      τ_i ← min{g_i(t) | t ∈ T_i − S_i};
15:      enqueue(Q, v_i);
16:   v_i ← dequeue(Q);
17: g_e(t_e) ← τ_e;
18: return g_e(t_e), t_e
```

We use $g_e(t_e)$ to denote the minimum value of $g_e(t)$ of destination v_e, where t_e is any time point such that $g_e(t)$ is minimized. Obviously, $g_e(t_e)$ is the minimum cost from v_s to v_e with time constraint. Our objective in this section is to compute $g_e(t_e)$.

The algorithm to compute $g_e(t_e)$ is shown in Algorithm 2. For every vertex $v_i \in G_T$, we use T_i to denote the time interval between the earliest arrival time of v_i and the latest arrival time t_a, i.e., $T_i = [\lambda_i, t_a]$. Algorithm 2 updates $g_i(t)$ iteratively for every vertex $v_i \in G_T$. We use S_i to represent the processed time interval for $g_i(t)$ in which the minimum value of $g_i(t)$ has been found to update $g_j(t)$ for every outgoing neighbor v_j of v_i. Different from $S_{i,j}$ discussed in Section 3.1, S_i is updated as $[t_i, t_a]$ in every iteration, but $S_{i,j}$ is updated as $R_{i,j} = [t_i, t_a - w_{i,j}]$ for every outgoing neighbor v_j of v_i. Let τ_i denote the minimum value of the current $g_i(t)$ for $t \in T_i - S_i$. It is obvious that τ_i is also the minimum value of the current $g_i(t)$ for $t \in T_{i,j} - S_{i,j}$ when $t_i \leq t_a - w_{i,j}$. Note that the current $g_i(t)$ may not be an optimal (or correct) atmc-function. Algorithm 2 updates $g_i(t)$ iteratively such that $g_i(t)$ approaches to its correct value.

For source v_s, $g_s(t)$, S_s and τ_s are initialized as $g_s(t) \leftarrow 0$, $S_s \leftarrow T_s$ and $\tau_s \leftarrow 0$ respectively. Obviously, the cost from v_s to v_s is zero for any arrival time t. It means the atmc-function $g_s(t)$ of v_s equals to zero for any $t \in [t_d, t_a]$. For any other vertex $v_i \neq v_s$, $g_i(t)$, S_i and τ_i are initialized as $g_i(t) \leftarrow \infty$, $S_i \leftarrow \emptyset$ and $\tau_i \leftarrow \infty$ respectively.

Algorithm 2 utilizes a priority queue Q to maintain the vertices in time-dependent graph G_T. All vertices $v_i \in G_T$ are sorted in Q according to τ_i. Algorithm 2 repeatedly dequeues the top vertex in Q, which has the minimum τ_i. Initially, the top vertex in Q is v_s because $\tau_s = 0$. The algorithm terminates when destination v_e is dequeued from Q for the first time. It means that the minimum cost from source to destination has been computed.

In every iteration, Algorithm 2 first dequeues the top vertex from Q, denoted as v_i. τ_i is the minimum value of current $g_i(t)$ for $t \in T_i - S_i$. Let t_i be the earliest time point such that $g_i(t) = \tau_i$. Here, $g_i(t)$ is the current atmc-function of v_i. We update S_i to $S_i \leftarrow [t_i, t_a]$. For each outgoing neighbor v_j of v_i, if $t_i \leq t_a - w_{i,j}$, we update $g_j(t)$ for $t \in (R_{i,j} - S_{i,j}) \oplus w_{i,j}$. τ_j is also updated at the same time. Note that it cannot find the earliest time point t_i if $g_i(t) = \tau_i$ for an open interval, i.e., $g_i(t) = \tau_i$ for $t \in (t^a, t^b]$. In this case, t_i is set as t^a and S_i is updated to $(t_i, t_a]$. Let $[t^a, t^b]$

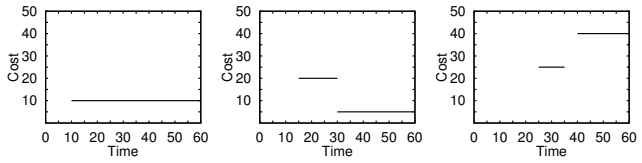
(or $(t^a, t^b]$) denote the time interval such that $g_i(t) = \tau_i$, then τ_i is the value of the optimal (or correct) atmc-function $g_i(t)$ for $t \in [t^a, t^b]$. We will prove it in section 3.2.3.

After updating $g_j(t)$ for all the outgoing neighbors $v_j \in N^+(v_i)$, Algorithm 2 checks whether the processed time interval equals to the whole time interval, i.e., $S_i = T_i$. If $S_i = T_i$, v_i is not necessary to update $g_j(t)$ for every outgoing neighbor v_j of v_i. Then v_i can be removed safely from Q. If $S_i \neq T_i$, Algorithm 2 computes the minimum value τ_i of current $g_i(t)$ for $t \in T_i - S_i$ and then enqueues v_i back into Q for further process.

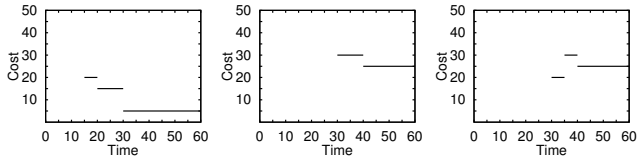
When the destination v_e is dequeued from the queue Q for the first time, τ_e is the minimum value of $g_e(t)$ for $t \in [\lambda_e, t_a]$. We will prove it in Section 3.2.3. Therefore, Algorithm 2 terminates because τ_e is the minimum cost from the source v_s to the destination v_e with time constraint.

3.2.2 Running example

We use the example in Fig. 1 to illustrate the process to compute $g_e(t_e)$. In this example, $v_s = v_1$, $v_e = v_4$, $t_d = 0$ and $t_a = 60$. We first utilize the single-source shortest path algorithm to compute the earliest arrival time for all vertices v_1, v_2, v_3 and v_4 in G_T. We have $\lambda_1 = 0$, $\lambda_2 = 10$, $\lambda_3 = 15$ and $\lambda_4 = 25$. Then the time domain of $g_1(t)$, $g_2(t)$, $g_3(t)$ and $g_4(t)$ are $[0, 60]$, $[10, 60]$, $[15, 60]$ and $[25, 60]$ respectively.



(a) $g_2(t)$(1st iteration) (b) $g_3(t)$(1st iteration) (c) $g_4(t)$(2nd iteration)



(d) $g_3(t)$(3rd iteration) (e) $g_4(t)$(3rd iteration) (f) $g_4(t)$(4th iteration)

Figure 3: Running Example

Fig. 3 illustrates how to compute $g_e(t_e)$. Initially, the priority queue Q contains all vertices v_1, v_2, v_3 and v_4. For source v_1, $\tau_1 = 0$, $S_1 = T_1$ and $g_1(t) = 0$. For other vertices $v_i (i = 2, 3, 4)$, $g_i(t) = \infty$, $\tau_i = \infty$ and $S_i = \emptyset$. It implies $g_i(t)$ is unknown.

In the first iteration, source v_1 is dequeued from the priority Q, because $\tau_1 = 0$ is the minimum in Q. Algorithm 2 computes $g_{1\to2}(t)$ and $g_{1\to3}(t)$ and updates $g_2(t)$ and $g_3(t)$ for v_1's two outgoing neighbors v_2 and v_3. The resulting atmc-functions $g_2(t)$ and $g_3(t)$ are shown in Fig. 3(a) and Fig. 3(b). Because $S_1 = T_1 = [0, 60]$, v_1 is removed from queue Q in this iteration.

In the second iteration, v_3 is the top vertex dequeued from Q, because $\tau_3 = 5$ is the minimum in Q. As shown in Fig. 3(b), $t_3 = 30$ is the earliest time such that $g_3(t) = \tau_3$. S_3 is updated as $S_3 \leftarrow [30, 60]$. v_4 is the outgoing neighbor of v_3 and $t_3 < t_a - w_{3,4} = 50$, then Algorithm 2 computes $g_{3\to4}(t)$ and updates $g_4(t)$. The resulting $g_4(t)$ is shown in Fig. 3(c). Because $S_3 \neq T_3$, v_3 is enqueued into Q again. Here, the minimum value of current $g_3(t)$ for $t \in T_3 - S_3 = [15, 30)$ is 20, then τ_3 is updated to 20.

In the third iteration, the top vertex dequeued from Q is v_2, because $\tau_2 = 10$ is the minimum in Q. As shown in Fig. 3(a), $g_2(t)$ equals to 10 for the whole time interval $t \in [10, 60]$, then S_2 is updated as $S_2 \leftarrow T_2$. For v_2's two outgoing neighbors v_3 and v_4, Algorithm 2 computes $g_{2\to3}(t)$ and $g_{2\to4}(t)$ and updates $g_3(t)$ and $g_4(t)$. The resulting atmc-functions $g_3(t)$ and $g_4(t)$ are shown in Fig. 3(d) and Fig. 3(e) respectively. Because $S_2 = T_2$, v_2 is removed from Q in this iteration.

In a similar way, v_3 is dequeued from Q in the fourth iteration and v_4 is dequeued from Q in the fifth iteration. Because v_4 is the destination, Algorithm 2 terminates. The minimum cost from source v_1 to destination v_4 is 20. As shown in Fig. 3(f), $g_4(t)$ equals to 20 for $t \in [30, 35)$. The arrival time t_e at v_4 can be any time point in $[30, 35)$.

3.2.3 Correctness

Next, we prove the correctness of Algorithm 2. Let $[t^a, t^b]$ denote the time interval during which the current $g_i(t)$ equals to τ_i, when v_i is dequeued from queue Q. The following theorem guarantees that the value of the optimal (or correct) atmc-function $g_i(t)$ is τ_i for $t \in [t^a, t^b]$.

Theorem 3.2: *Given a time-dependent graph G_T, let v_i be the vertex dequeued from Q with τ_i in the k-th iteration and $[t^a, t^b]$ be the time interval such that $g_i(t) = \tau_i$. Here, $g_i(t)$ is the current* atmc*-function of v_i. Then the value of the optimal (or correct)* atmc*-function $g_i(t)$ is τ_i for $t \in [t^a, t^b]$.* □

PROOF. For any $t^0 \in [t^a, t^b]$, we only need to prove $g_i(t^0) = \tau_i$, where $g_i(t)$ is the optimal (or correct) atmc-function of v_i. By the definition of the atmc-function, we have $g_i(t^0) \leq \tau_i$. Next, we need to prove $\tau_i \leq g_i(t^0)$. Without loss of generality, let p denote the path along which one can arrive at vertex v_i at time point t^0 with the cost $\mathsf{cost}(p) = g_i(t^0)$:

$$p : v_s \to v_1 \to v_2 \to \cdots \to v_h \to v_i$$

We use t_x to denote the arrival time at $v_x (1 \leq x \leq h)$ in path p and $g_x(t_x)$ to denote the cost that one arrives at v_x from v_s along path p. To distinguish, we use $g_x^k(t)$ to denote the current atmc-function of v_x in the k-th iteration. It is worth noting that $g_x^k(t)$ may not be the optimal (or correct) atmc-function $g_x(t)$. In the k-th iteration, the value of $g_h(t)$ for $t = t_h$ is $g_h(t_h)$. Consider two cases: (1) $g_h^k(t_h) = g_h(t_h)$; and (2) $g_h^k(t_h) > g_h(t_h)$. Note that $g_h(t_h)$ is the value of the optimal (or correct) atmc-function of v_h at time point t_h, then there does not exist the case of $g_h^k(t_h) < g_h(t_h)$. In case (1), there are also two cases for $f_{h,i}(t^0 - w_{h,i})$: (a) $f_{h,i}(t^0 - w_{h,i}) = 0$, and (b) $f_{h,i}(t^0 - w_{h,i}) \neq 0$. In case (a), if $t_h \in S_h$, then v_h has been dequeued from Q with $\tau_h \leq g_h^k(t_h)$. Thus $g_i(t)$ has been updated as $\tau_h + f_{h,i}(t - w_{h,i})$. By Theorem 3.1 and $f_{h,i}(t^0 - w_{h,i}) = 0$, we have $\tau_i \leq \tau_h + f_{h,i}(t^0 - w_{h,i}) \leq g_h^k(t_h) = g_h(t_h)$. If $t_h \notin S_h$, then $\tau_i = g_h^k(t_h) = g_h(t_h)$. Otherwise, $g_h^k(t_h) < \tau_i$, v_h should be dequeued from Q in the k-th iteration, which is in contradiction with that v_i is dequeued from Q in the k-th iteration. Note that $g_h^k(t_h)$ cannot be larger than τ_i. Because $g_h^k(t_h) = g_h(t_h)$, $g_h(t_h) \leq g_i(t^0)$ and $g_i(t^0) \leq \tau_i$, then $g_h^k(t_h) \leq \tau_i$. Then for case (a), we have $\tau_i \leq g_h(t_h) = g_h(t_h) + f_{h,i}(t^0 - w_{h,i}) = g_i(t^0)$. In case (b), $f_{h,i}(t^0 - w_{h,i}) \neq 0$, then $g_h^k(t_h) = g_h(t_h) < g_i(t^0)$. It indicates that v_h has been dequeued from Q with $\tau_h \leq g_h^k(t_h)$ and $g_i(t)$ has been updated as $\tau_h + f_{h,i}(t - w_{h,i})$. By Theorem 3.1, we have

$$\begin{aligned}\tau_i &\leq \tau_h + f_{h,i}(t^0 - w_{h,i}) \\ &\leq g_h^k(t_h) + f_{h,i}(t^0 - w_{h,i}) \\ &= g_h(t_h) + f_{h,i}(t^0 - w_{h,i}) = g_i(t^0)\end{aligned}$$

Thus for case (1), we have $\tau_i \leq g_i(t^0)$. Next, we prove that case (2) does not exist. Consider v_{h-1}. There are also two cases for

$g^k_{h-1}(t_{h-1})$ in the k-th iteration: $g^k_{h-1}(t_{h-1}) = g_{h-1}(t_{h-1})$ and $g^k_{h-1}(t_{h-1}) > g_{h-1}(t_{h-1})$. In a similar way as proving $\tau_i = g_i(t^0)$ for case (1), we can prove $g^k_h(t_h) = g_h(t_h)$ if $g^k_{h-1} = g_{h-1}(t_{h-1})$. Then we only need to prove that there does not exist the case of $g^k_{h-1}(t_{h-1}) > g_{h-1}(t_{h-1})$. Similarly, we only need to prove that there does not exist the case of $g^k_{h-2}(t_{h-2}) > g_{h-2}(t_{h-2})$. Recursively, we only need to prove that there does not exist the case of $g^k_1(t_1) > g_1(t_1)$. Because $g_1(t_1)$ has been computed when source v_s is dequeued from Q in the first iteration, then $g^k_1(t_1) = g_1(t_1)$. This is a contradiction. Then case (2) does not exist and we prove $\tau_i \leq g_i(t^0)$. Theorem 3.2 is proved. □

Next, we give Theorem 3.3 to guarantee that τ_i is the minimum value of the optimal (or correct) atmc-function $g_i(t)$ for $t \in T_i - S_i$ when v_i is dequeued from Q with τ_i.

Theorem 3.3: *Given a time-dependent graph G_T, let v_i be the vertex dequeued from Q with τ_i in the k-th iteration, then τ_i is the minimum value of the optimal (or correct)* atmc*-function $g_i(t)$ of v_i for $t \in T_i - S_i$.* □

PROOF. We prove it by contradiction and assume that there exist $t^0 \in T_i - S_i$ such that $g_i(t^0) < \tau_i$. Without loss of generality, let p denote the path along which one can arrive at vertex v_i at time point t^0 with the cost $\mathsf{cost}(p) = g_i(t^0)$:

$$p : v_s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_h \rightarrow v_i$$

We use t_x to denote the arrival time at v_x ($1 \leq x \leq h$) in path p and $g_x(t_x)$ to denote the cost that one arrives at v_x from v_s along path p. We also use $g^k_x(t_x)$ to denote the current atmc-function of v_x in the k-th iteration. Similar to that in Theorem 3.2, there are only two cases for $g^k_i(t^0)$: (1) $g^k_i(t^0) = g_i(t^0)$; and (2) $g^k_i(t^0) > g_i(t^0)$. For case (1), $g^k_i(t^0) = g_i(t^0) < \tau_i$, then v_i should be dequeued from Q with $g^k_i(t^0)$ in the k-th iteration, which is in contradiction with that v_i is dequeued from Q with τ_i. Then we only need to prove that case (2) does not exist. We can prove that in a similar way as proving that case (2) does not exist in Theorem 3.2. Because there is contradiction for case (1) and case (2) does not exist, then the assumption does not hold. Theorem 3.3 is proved. □

Theorem 3.3 indicates that τ_i is the minimum value of the optimal (or correct) atmc-function $g_i(t)$ for $t \in T_i - S_i$ when v_i is dequeued from Q. Therefore, when destination v_e is dequeued from Q for the first time, τ_e is the minimum value of $g_e(t)$ for $t \in T_e$, i.e., τ_e is the minimum cost from source v_s to destination v_e with time constraint.

Corollary 3.1: *Given a time-dependent graph G_T, for any vertex v_i in G_T, let τ^p_i and τ^q_i be the τ_i when v_i is dequeued from Q for the p-th and q-th time respectively. If $p < q$, then $\tau^p_i < \tau^q_i$.* □

PROOF. Let S^p_i and S^q_i be the S_i when v_i is dequeued from Q for the p-th and q-th time respectively. Because $p < q$, then $S^p_i \subset S^q_i$ and $T_i - S^p_i \supset T_i - S^q_i$. By Theorem 3.3, τ^p_i and τ^q_i are the minimum values of the optimal (or correct) $g_i(t)$ for $t \in T_i - S^p_i$ and $t \in T_i - S^q_i$, then $\tau^p_i < \tau^q_i$. □

3.3 Finding the Optimal Path and the Optimal Waiting Time

In this section, we introduce the second step of the TWO-STEP-SEARCH algorithm, i.e., how to find the optimal path p^* from source v_s to destination v_e and the optimal waiting time $\omega^*(v_i)$ for every vertex $v_i \in p^*$ such that $\mathsf{cost}(p^*) = g_e(t_e)$.

Algorithm 3 shows the algorithm to compute p^* and $\omega^*(v_i)$ for every $v_i \in p^*$. The main idea is to find the predecessor iteratively

Algorithm 3 PATH-SELECTION ($G_T, v_s, v_e, t_d, t_a, g_e(t_e)$)

Input: G_T, $g_e(t_e)$, v_s and v_e, t_d and t_a.
Output: the optimal path p^* and $\omega^*(v_i)$ for every vertex $v_i \in p^*$.

```
1: v_i(t) ← v_e; p* ← ∅; g_i(t_i) ← g_e(t_e); t_i ← t*;
2: while v_i ≠ v_s do
3:    for each v_j ∈ N⁻(v_i) do
4:       if ∃t_j ≤ t_i − w_{j,i}, g_i(t_i) = g_j(t_j) + f_{j,i}(t_i − w_{j,i}) then
5:          p* ← p* + v_j; ω*(v_i) = t_i − w_{j,i} − t_j;
6:          v_i ← v_j; t_i ← t_j;
7:          break;
8: return p* and ω*(v_i) for each v_i ∈ p*.
```

for every vertex $v_i \in P^*$ backward from destination v_e to source v_s. Initially, $v_i \leftarrow v_e$.

In every iteration, we find the predecessor v_j of v_i in the optimal path p^*. Let t_i be the arrival time at v_i in path p^* and $g_i(t_i)$ be the cost that one arrives at v_i along path p^*. We initialize $g_i(t_i)$ and t_i as $g_e(t_e)$ and t_e respectively. For each $v_j \in N^-(v_i)$, if there exists a time point t_j, $t_j \leq t_i - w_{j,i}$, such that

$$g_i(t_i) = g_j(t_j) + f_{j,i}(t_i - w_{j,i}) \tag{6}$$

then v_j is the predecessor of v_i in path p^* and t_j is the arrival time at v_j in path p^*. Such a predecessor v_j must exist, because $g_i(t_i)$ is computed by Algorithm 2 using $g_j(t_j)$. Then the optimal waiting time at v_j is:

$$\omega^*(v_j) = t_i - w_{j,i} - t_j \tag{7}$$

Algorithm 3 terminates when source v_s is found as a predecessor, i.e., $v_i = v_s$. Here, all the vertices in p^* are found and the optimal waiting time $\omega^*(v_i)$ for every vertex $v_i \in p^*$ is computed.

We use the example in Fig. 3 to illustrate the process to compute p^* and $\omega^*(v_i)$ for every vertex $v_i \in p^*$. From Fig. 3, we find that the minimum cost from v_1 to v_4 is $g_4(t_4) = 20$. Here, $g_4(t) = 20$ for $t \in [30, 35)$, then t_4 can be any time point in $[30, 35)$, e.g., $t_4 = 30$. For a v_4's incoming neighbor v_3, the departure time from v_3 is $t_4 - w_{3,4} = 30 - 10 = 20$. We find that $g_3(t_3) = 15$ when $t_3 = 20$ and then we have:

$$g_3(t_3) + f_{3,4}(t_4 - w_{3,4}) = g_3(20) + f_{3,4}(30 - 10) = 20 = g_4(t_4)$$

Thus v_3 is the predecessor of v_4 in the optimal path p^*. The optimal waiting time at v_3 is $\omega^*(v_3) = t_4 - w_{3,4} - t_3 = 0$.

In the second iteration, for a v_3's incoming neighbor v_2, the departure time from v_2 is $t_3 - w_{2,3} = 20 - 5 = 15$. $g_2(t_2) = 10$ when $t_2 = 10$ and then we have:

$$g_2(t_2) + f_{2,3}(t_3 - w_{2,3}) = 10 + 5 = 15 = g_3(t_3)$$

Thus v_2 is the predecessor of v_3 in the optimal path p^* and the optimal waiting time at v_2 is $\omega^*(v_2) = t_3 - w_{2,3} - t_2 = 5$.

In the similar way, v_1 is found as the predecessor of v_2 in path p^*. Because v_1 is the source vertex, Algorithm 3 terminates. Then the optimal path p^* is $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$. The optimal waiting time for all the vertices in the optimal path p^* are $\omega^*(v_1) = 0$, $\omega^*(v_2) = 5$ and $\omega^*(v_3) = 0$ respectively.

3.4 Time and Space Complexities

We first analyze the time complexities of Algorithm 2 and Algorithm 3 and then give the time complexity of the TWO-STEP-SEARCH algorithm (Algorithm 1). Let n and m be the number of vertices and edges in G_T respectively, and k be the average number of piecewise constant values of atmc-function $g_i(t)$ in G_T.

Lemma 3.1: *The time complexity of* COMPUTE-MINIMUM-COST *(Algorithm 2) is $O(kn \log n + mk)$.* □

PROOF. In every iteration of Algorithm 2, there are at most n vertices maintained in Q at the same time. Using Fibonacci Heap [6], dequeue(Q) and enqueue(Q) require $O(\log n)$ and $O(1)$ amortized time respectively. When a vertex v_i is dequeued from Q, Algorithm 2 needs to compute $g_{i\to j}(t)$ and update $g_j(t)$ for every outgoing neighbor v_j of v_i. To compute $g_{i\to j}(t)$, Algorithm 2 needs to add τ_i to every constant value of $f_{i,j}(t)$ for $t \in R_{i,j} - S_{i,j}$. We use $k_{i,j}[I^p]$ to denote the number of piecewise constant values of $f_{i,j}(t)$ for $t \in R_{i,j} - S_{i,j}$ when v_i is dequeued from Q for the p-th time. Here $I^p = R_{i,j} - S_{i,j}$. Then the time complexity is $O(k_{i,j}[I^p])$ for computing $g_{i\to j}(t)$ and updating $g_j(t)$. When v_i is dequeued from Q for the p-th time, the time complexity for this iteration is $O(\sum_{x=1}^{d^+(v_i)} k_{i,j_x}[I^p] + \log n)$, where v_{j_x} is the x-th outgoing neighbor v_j of v_i. We call t_i^p as an "connection point" of I_p and I_{p+1}, where t_i^p is t_i when v_i is dequeued from Q for the p-th time. For a constant value c_x of $f_{i,j}(t)$, assume c_x's corresponding time interval is $[t^{x-1}, t^x]$, i.e., $f_{i,j}(t) = c_x$ for $t \in [t^{x-1}, t^x]$. Note that c_x is counted in both $k[I^p]$ and $k[I^{p+1}]$ if the connection point $t_i^p \in (t^{x-1}, t^x)$. Here, c_x is the first constant value of $f_{i,j}(t)$ for $t \in I^p$ and the last one of $f_{i,j}(t)$ for $t \in I^{p+1}$. Let l_i be the number of times that v_i is dequeued from Q in total. There are at most (l_i-1) connection points, then $\sum_{p=1}^{l_i} k_{i,j}[I^p] \le k_{i,j}[T_i]+(l_i-1)$. Thus for v_i, the time complexity is:

$$O(\sum_{p=1}^{l_i}(\sum_{x=1}^{d^+(v_i)} k_{i,j_x}[I^p] + \log n))$$
$$\le O(\sum_{x=1}^{d^+(v_i)} (k_{i,j_x}[T_i] + (l_i - 1)) + \sum_{p=1}^{l_i} \log n)$$

and the total time complexity is:

$$O(\sum_{v_i\in V}(\sum_{x=1}^{d^+(v_i)} (k_{i,j_x}[T_i] + (l_i - 1)) + \sum_{p=1}^{l_i} \log n))$$
$$= O(\sum_{v_i\in V}\sum_{x=1}^{d^+(v_i)} (k_{i,j_x}[T_i] + (l_i - 1)) + \sum_{v_i\in V}\sum_{p=1}^{l_i} \log n)$$

It is obvious that $\sum_{v_i\in V}\sum_{x=1}^{d^+(v_i)} k_{i,j_x}[T_i]$ equals to the total number of constant values of all the cost functions on all the edges in G_t. Let k' be the average number of constant values of cost function $f_{i,j}(t)$ in G_T, then $mk' = \sum_{v_i\in V}\sum_{x=1}^{d^+(v_i)} k_{i,j_x}[T_i]$. Let k_i denote the number of constant values of atmc-function $g_i(t)$. By Theorem 3.3 and Corollary 3.1, we know that v_i is dequeued from Q repeatedly according to the different constant values (i.e., τ_i) of $g_i(t)$ in Algorithm 2. Thus v_i is dequeued from Q at most k_i times, then $l_i \le k_i$ and $\sum_{v_i\in V}\sum_{x=1}^{d^+(v_i)}(l_i - 1) < \sum_{x=1}^{d^+(v_i)}\sum_{v_i\in V} k_i$. Here, $\sum_{v_i\in V} k_i$ equals to the total number of constant values of all the atmc-functions in G_T. Thus we have $\sum_{v_i\in V} k_i = nk$ and $\sum_{x=1}^{d^+(v_i)}\sum_{v_i\in V} k_i = mk$. Similarly, $\sum_{v_i\in V}\sum_{p=1}^{l_i} \log n = nk \log n$. Then the total complexity is $O(m(k' + k) + nk \log n)$. Because every $g_i(t)$ are computed by different $f_{i,j}(t)$, then $k' \le k$. Thus the total time complexity is $O(nk \log n + mk)$. □

Lemma 3.2: *The time complexity of* PATH-SELECTION *(Algorithm 3) is* $O(mk)$. □

PROOF. In every iteration of Algorithm 3, for a vertex v_i in the optimal path p^*, all v_i's incoming neighbors need to be examined. For every incoming neighbor v_j of v_i, Algorithm 3 needs to check whether there exists a time t_j ($t_j \le t_i - w_{j,i}$), such that Eq. (6) holds. This operation requires $O(k)$ time. Thus the time complexity of computing predecessor for v_i is $d^-(v_i)k$ in every iteration. Because there is no loop in the optimal path p^*, then Algorithm 3 needs to compute predecessor for v_i at most once. Thus, the time complexity of Algorithm 1 is $O(\sum_{v_i\in V}(d^-(v_i)k)) = O(mk)$. □

By Lemma 3.1 and Lemma 3.2, then we have the time complexity of Algorithm 2.

Theorem 3.4: *The time complexity of* TWO-STEP-SEARCH *(Algorithm 1) is* $O(kn \log n + mk)$. □

PROOF. Algorithm 2 and Algorithm 3 are two steps of Algorithm 1. By Lemma 3.1 and Lemma 3.2, the time complexity of Algorithm 1 (TWO-STEP-SEARCH) is $O(kn \log n + mk)$. □

Next,we analyze the space complexity of Algorithm 1.

Theorem 3.5: *The space complexity of* TWO-STEP-SEARCH *(Algorithm 1) is* $O((n + m)k)$. □

PROOF. Algorithm 2 and Algorithm 3 are two steps of Algorithm 1. Algorithm 2 needs to maintain at most n vertices in Q. For every vertex v_i, Algorithm 2 and Algorithm 3 need to maintain its atmc-function $g_i(t)$. For every edge (v_i, v_j) in G_T, Algorithm 2 and Algorithm 3 need to maintain its cost function $f_{i,j}(t)$. Therefore, the space complexity of Algorithm 1 is $O((n + m)k)$. □

3.5 Discussion about Time Function

In the above discussion, we assume that travel time for every edge (v_i, v_j) is a constant value $w_{i,j}$. Next, we discuss how to handle the case where travel time is a function $w_{i,j}(t)$ which is related to the departure time t from v_i. In this paper, we assume that G_T has the FIFO property. FIFO property for an edge (v_i, v_j) implies that if departing earlier from v_i, one arrives earlier at v_j.

Definition 3.1: (**FIFO**) Given a time-dependent graph G_T, we say G_T is a FIFO graph if and only if the time function $w_{i,j}(t)$ of every edge (v_i, v_j) has the FIFO property, i.e., $w_{i,j}(t_0) < \Delta t + w_{i,j}(t_0 + \Delta t)$ for $\Delta t > 0$, or $t_1 + w_{i,j}(t_1) < t_2 + w_{i,j}(t_2)$ for $t_1 < t_2$. □

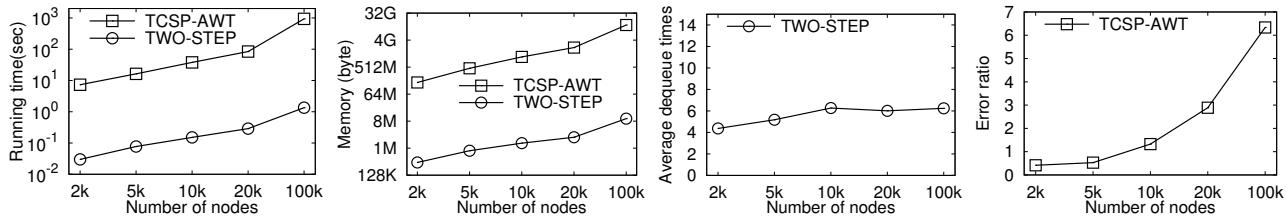
The restriction of the FIFO property is reasonable and many previous works also make this assumption [16, 13]. Consider a road network, for two cars towards the same road segment, the first one reaching the starting point should leave the end point first.

We use $\mathsf{arrive}_{i,j}(t)$ to denote the arrival-time function of edge (v_i, v_j), $\mathsf{arrive}_{i,j}(t) = t + w_{i,j}(t)$. $\mathsf{arrive}_{i,j}(t)$ indicates the arrival time at v_j if one departs from v_i at time point t. It is obvious that $\mathsf{arrive}_{i,j}(t)$ is a one-one mapping function. Given an arrival time t at v_j, we can compute the departure time from v_i by $\mathsf{arrive}_{i,j}^{-1}(t)$, where $\mathsf{arrive}_{i,j}^{-1}(t)$ is the inverse function of $\mathsf{arrive}_{i,j}(t)$.

As similar as Eq. (4) in Section 3.1, for every edge (v_j, v_i), given the arrival time t for v_i, the departure time from v_j is fixed as $\mathsf{arrive}_{j,i}^{-1}(t)$ and the cost to travel edge (v_j, v_i) is $f_{j,i}(\mathsf{arrive}_{j,i}^{-1}(t))$. Therefore, $g_{j\to i}(t)$ can be computed by the following equation:

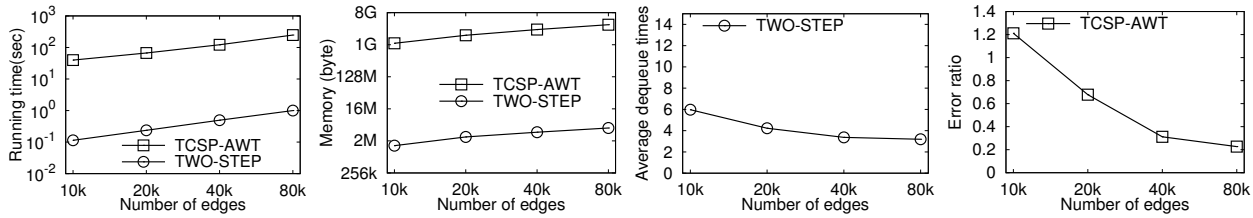
$$g_{j\to i}(t) = \min_{t' \le \mathsf{arrive}_{j,i}^{-1}(t)} \{g_j(t')\} + f_{j,i}(\mathsf{arrive}_{j,i}^{-1}(t)) \quad (8)$$

There is a special case where $w_{j,i}(t)$ is a non-continuous function. For example, $w_{j,i}(t)$ is a function $w_{j,i}^1(t)$ for $t \le t^\dagger$ and another function $w_{j,i}^2(t)$ for $t > t^\dagger$. Here $w_{j,i}^1(t)$ is always smaller than $w_{j,i}^2(t)$. Obviously, $w_{j,i}(t)$ is broken at $t^\dagger$. Then the arrival-time function for edge (v_j, v_i) is non-continuous, because one cannot arrive at v_i between time $t^\dagger + w_{j,i}^1(t^\dagger)$ and $t^\dagger + w_{j,i}^2(t^\dagger)$. We call $t^\dagger$ as a "break point" of $w_{j,i}(t)$. Suppose there exists a "break point" $t^\dagger$ in time interval $[t^a, t^b]$ during which $f_{j,i}(t) = c$, then $g_{j\to i}(t)$ equals to $c + \tau_j$ for $t \in [\mathsf{arrive}_{j,i}^1(t^a), \mathsf{arrive}_{j,i}^1(t^\dagger)]$ and



(a) querying time (b) memory overhead (c) average dequeue times (d) relative error

Figure 4: Impact of vertex size



(a) querying time (b) memory overhead (c) average dequeue times (d) relative error

Figure 5: Impact of edge size

$t \in (\mathsf{arrive}^2_{j,i}(t^\dagger), \mathsf{arrive}^2_{j,i}(t^b)]$, respectively. Here $\mathsf{arrive}^1_{j,i}(t) = t + w^1_{j,i}(t)$ and $\mathsf{arrive}^2_{j,i}(t) = t + w^2_{j,i}(t)$. In addition, $g_{j\to i}(t) = \infty$ for $t \in (\mathsf{arrive}^1_{j,i}(t^\dagger), \mathsf{arrive}^2_{j,i}(t^\dagger)]$. Note that $\mathsf{arrive}^1_{j,i}(t^\dagger) < \mathsf{arrive}^2_{j,i}(t^\dagger)$. It indicates one cannot arrive at v_i by edge (v_j, v_i) between time $\mathsf{arrive}^1_{j,i}(t^\dagger)$ and $\mathsf{arrive}^2_{j,i}(t^\dagger)$.

As similar as Eq. (6), Algorithm 3 can compute the predecessor v_j for a vertex v_i in the optimal path p^* by the following equation if travel time is a function $w_{i,j}(t)$ for every edge $(v_i, v_j) \in G_T$.

$$g_i(t_i) = g_j(t_j) + f_{j,i}(\mathsf{arrive}^{-1}_{j,i}(t_i)) \tag{9}$$

The optimal waiting time for v_j can be computed as follows:

$$\omega^*(v_j) = t_i - w_{j,i}(\mathsf{arrive}^{-1}_{j,i}(t_i)) - t_j \tag{10}$$

4. PERFORMANCE EVALUATION

We compare the TWO-STEP-SEARCH algorithm with the TCSP-AWT (Time Constraint Shortest Path Allow Waiting Time) algorithm [2] based on two real datasets. TCSP-AWT is the state of the art algorithm to compute the cost-optimal path with time constraint in time-dependent graphs. However, TCSP-AWT can be only used for the discrete time model. All the experiments are conducted on a 3.0GHz Intel Core i5 CPU PC with the 32GB main memory, running on Windows 7.

4.1 Dataset and Experiment Setup

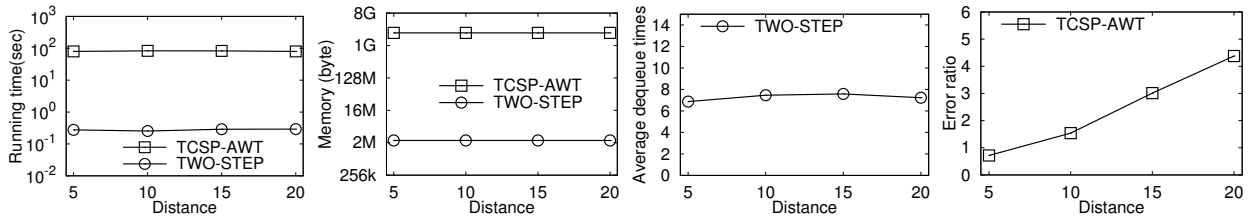
We employ the following two real road networks:

CARN: This dataset is California road network including 23,718 vertices and 33,561 edges. A vertex represents an intersection or a road endpoint and an edge represents a road segment.

EURN: This network describes Eastern USA road network and it includes 3,598,623 vertices and 8,778,114 edges. As the same as the CARN, a vertex represents an intersection or a road endpoint and an edge represents a road segment.

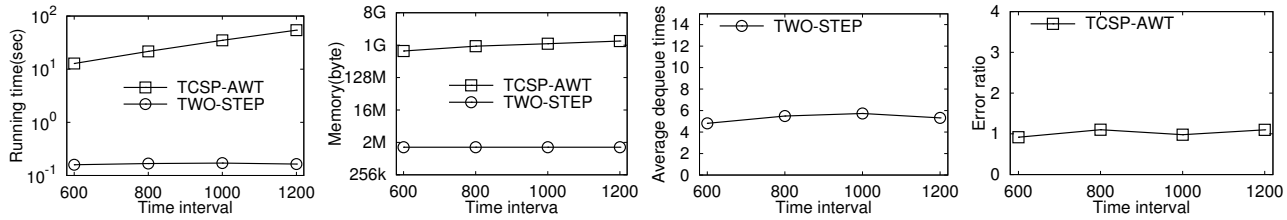
We generate four time-dependent graphs with different sizes using the CARN dataset and five time-dependent graphs with different sizes using the EURN datasets. For graphs of CARN dataset, the number of vertices ranges from 2k to 20k. For graphs of EURN dataset, the number of vertices ranges from one million to three millions. We generate travel time according to the road length. The travel time for an edge (v_i, v_j) is more if the road represented by (v_i, v_j) is longer. To simulate the real traffic case, we compute the betweenness centrality for every edge in G_t and sort all the edges in descending order of betweenness. We select the top 20% edges as the traffic hubs in a road network. We assign the more expensive travel cost and the more travel time on these edges. The time domain is set as $\mathcal{T} = [0, 2000]$, i.e., the departure time t can be selected from $[0, 2000]$ for any vertex in a graph. Here, 2000 means 2000 time units. For every $f_{i,j}(t)$, we split the time domain $\mathcal{T}$ to k subintervals and assign a constant value randomly for every subinterval and then it is a piecewise constant function. For every $w_{i,j}(t)$, the time domain T is also randomly divided to k subintervals $([t_0, t_1], [t_1, t_2], \cdots, [t_{k-1}, t_k])$, where t_0 and t_k are the start and the end of the time domain T respectively. The value of $w_{i,j}(t_0)$ is first generated as a random number from $[0, \bar{w}]$, where $\bar{w}$ is a number to restrict the max value of $w_{i,j}(t)$. Within each subinterval $[t_{x-1}, t_x]$ $(1 \le x \le k)$, $w_{i,j}(t)$ is a linear function $w^x_{i,j}(t)$, $w^x_{i,j}(t_{x-1}) = w^{x-1}_{i,j}(t_{x-1})$ and $w^x_{i,j}(t_x)$ is generated as a random number from $[\max(0, w^x_{i,j}(t_{x-1}) - \Delta t_x), \bar{w}]$, where $\Delta t_x = t_x - t_{x-1}$. Then the time function $w_{i,j}(t)$ is guaranteed to be non-negative and FIFO. For TCSP-AWT, we sample one discrete time point every other time unit from the whole time interval. For example, if the whole time interval is $[0, 1000]$, then the sampled discrete time points are $[1, 3, 5, \cdots, 999]$. We randomly generate 1,000 pairs of vertices and query the cost-optimal path between every pair of vertices. The reported querying time is the average time on each dataset. We use TWO-STEP to denote TWO-STEP-SEARCH in the experimental results.

We are interested in the following aspects to evaluate the performance of TWO-STEP-SEARCH: (1) the impact of number of vertices; (2) the impact of number of edges; (3) the impact of distances between source and destination; (4) the impact of the length of time interval $[t_d, t_a]$ between the earliest departure time t_d and the latest arrival time t_a; and (5) the impact of the average number of piecewise intervals of $f_{i,j}(t)$. The parameters to be evaluated are: (1) querying time; (2) memory overhead; (3) average number of times that a vertex is dequeued from Q; (4) relative error $(c - c^*)/c^*$ of TCSP-AWT. Note that TCSP-AWT can be only used for the discrete



(a) querying time (b) memory overhead (c) average dequeue times (d) relative error

Figure 6: Impact of distance between source and destination



(a) querying time (b) memory overhead (c) average dequeue times (d) relative error

Figure 7: Impact of the length of time interval $[t_d, t_a]$

time model, thus the results of TCSP-AWT may not be optimal as the discussion in Section 2.2. Here, c is the cost of the path computed by TCSP-AWT and c^* is the cost of the optimal path. There is no error for TWO-STEP-SEARCH algorithm.

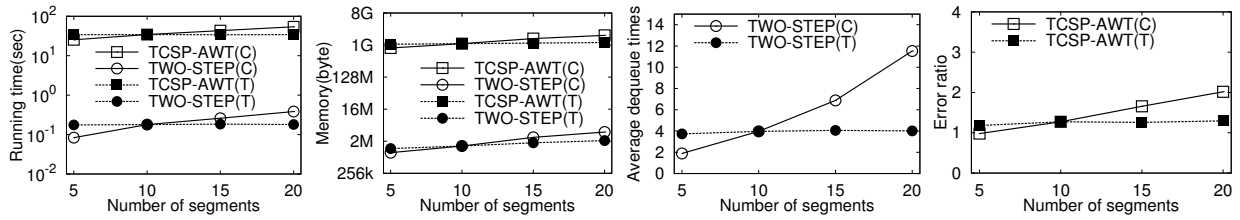
4.2 Experimental Results

Exp-1. Impact of the number of vertices: In Fig. 4, we study the impact of number of vertices of time dependent graph G_T. In this group of the experiments, the number of vertices increases from 2K to 100K, where the graphs with 2k to 20k vertices are generated from CARN dataset and the graph with 100k vertices is generate from EURN dataset. The number of piecewise intervals of $f_{i,j}(t)$ is 10. The time interval $[t_d, t_a]$ is $[0, 1000]$. As shown in Fig. 4(a) and Fig. 4(b), the querying time and memory overhead of TWO-STEP-SEARCH are always less than that of TCSP-AWT. TWO-STEP-SEARCH is nearly 20 times faster than TCSP-AWT. The memory overhead of TWO-STEP-SEARCH is nearly 500 times less than that of TCSP-AWT. The reason is TCSP-AWT needs to compute the arrival costs for all the vertices in G_T for every time point and then it can compute the minimum cost from source to destination based on these results. We find the querying time and memory overhead increase marginally with the number of vertices increasing. From Fig. 4(c), we find the average number of dequeuing times of vertices is not affected by the number of vertices. From Fig. 4(d), we find the relative error increases with the increasing of the number of vertices. If the number of vertices increases, the distance between source and destination increases. Thus the relative error cumulated in the path from source to destination increases.

Exp-2. Impact of the number of edges: We study the impact of the number of edges in Fig. 5. In this group of experiments, the number of vertices is fixed at 10K and the number of edges increases from 10K to 80K. The number of piecewise intervals of $f_{i,j}(t)$ is 10. The time interval $[t_d, t_a] = [0, 1000]$. As shown in Fig. 5(a) and Fig. 5(b), we find TWO-STEP-SEARCH always performs better than TCSP-AWT. From Fig. 5(c), we find the average number of dequeuing times decreases with the increasing of the number of edges. Intuitively, more edges results in larger density of graphs and then the distance between every two vertices decreases. Thus TWO-STEP-SEARCH can compute the minimum cost from source to destination faster and the average number of dequeuing times decreases. However, more edges indicates more outgoing neighbors of a vertex in a graph. When a vertex v_i is dequeued from Q, it takes more time to update the atmc-functions for all the outgoing neighbors of v_i. Therefore, in Fig. 5(a), the querying time increases with the number of edges increasing. From Fig. 5(d), we find the relative error of TCSP-AWT decreases with the number of edges increasing. It is because the increasing of the number of edges shortens the distance between source and destination. Then less relative error is cumulated in the path from source to destination.

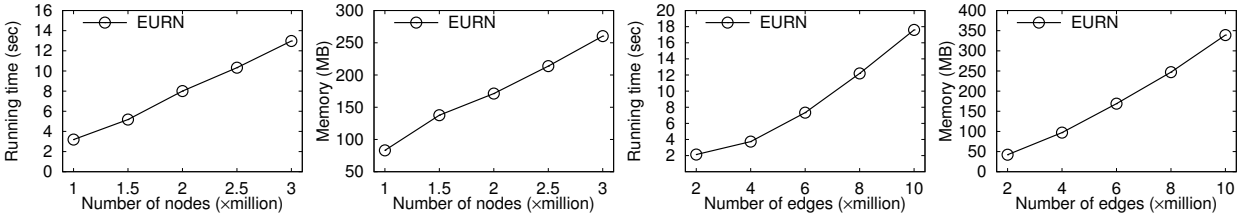
Exp-3. Impact of the distance between source and destination: In Fig. 6, we study the impact of the distance between source and destination. In this group of experiments, the number of vertices is fixed at 20K. The distance between source and destination ranges from 5 to 20. As shown in Fig. 6(a) and Fig. 6(b), we find the querying time and memory overhead of both algorithms are not affected by the distance between source and destination. The reason is the path with the minimum distance is not a cost-optimal path in many cases. Note that the density of a graph does not increase in this group of experiments. Thus, in Fig. 6(c), the average number of dequeuing times is not affected by the distance. From Fig. 6(d), we find the relative error of TCSP-AWT increases with the increasing of distance. The longer distance between source and destination, the more error cumulated in the path from source to destination.

Exp-4. Impact of the length of time interval $[t_d, t_a]$: In Fig. 7, we study the impact of the length of time interval $[t_d, t_a]$. In this group of experiments, the earliest departure time is fixed at time point 0 and the latest arrival time ranges from time point 600 to time point 1200. As shown in Fig. 7(a) and Fig. 7(b), the querying time and memory overhead of TWO-STEP-SEARCH are not affected by the length of time interval $[t_d, t_a]$. It is because the number of piecewise intervals of $f_{i,j}(t)$ does not increase. However, the number of sampled discrete time points for TCSP-AWT increases with the length of time interval $[t_d, t_a]$ increasing. Thus the querying time and memory overhead of TCSP-AWT increases. We also find the average number of dequeuing times is stable in Fig. 7(c) and the relative error does not change significantly in Fig. 7(d).



(a) querying time (b) memory overhead (c) average dequeue times (d) relative error

Figure 8: Impact of number of piecewise intervals of cost-function and time-function



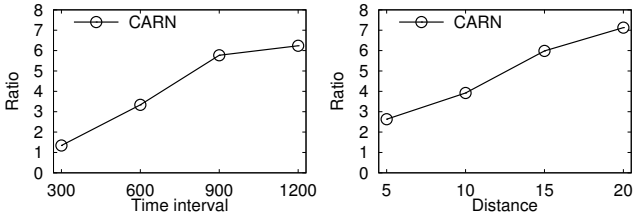
(a) querying time (nodes) (b) memory overhead (nodes) (c) querying time (edges) (d) memory overhead (edges)

Figure 9: Adaptivity to large graphs

Exp-5. Impact of the number of piecewise intervals of cost function and time function: In Fig. 8, we investigate the impact of the number of piecewise intervals of $f_{i,j}(t)$ by curves TCSP-AWT(C) and TWO-STEP(C) in Fig. 8. In this group of experiments, the number of piecewise intervals of $f_{i,j}(t)$ increases from 5 to 20. As shown in Fig. 8(a) and Fig. 8(b), the querying time and memory overhead of TWO-STEP-SEARCH increase with the increasing of the number of piecewise intervals. The reason is that TWO-STEP-SEARCH needs more time to update the atmc-function for vertices when the number of piecewise time intervals increases. We find TWO-STEP-SEARCH is always better than TCSP-AWT. From Fig. 8(c), we find the average number of dequeuing times increases with the increasing of the number of piecewise intervals. In Fig. 8(d), the relative error of TCSP-AWT increases with the increasing of the number of piecewise intervals. When the number of piecewise intervals increases, the number of arrival cost values between two consecutive discrete time points in TCSP-AWT increases. This results in that TCSP-AWT cannot find the optimal cost of traveling an edge. Thus the relative error of TCSP-AWT increases with the number of piecewise intervals increasing.

We also investigate the impact of the number of piecewise intervals of $w_{i,j}(t)$ by curves TCSP-AWT(T) and TWO-STEP(T) in Fig. 8. The number of piecewise intervals of $w_{i,j}(t)$ also increases from 5 to 20. The experimental results show the efficiency of TWO-STEP-SEARCH is not affected by the number increasing. It is because the increasing of this number does not incur extra operation on atmc-function during the whole process.

Exp-6. Scalability: We evaluate the scalability of TWO-STEP-SEARCH in Fig.9. We investigate the querying time and memory overhead by varying the number of vertices from one million to three millions and by varying the number of edges from two millions to ten millions on the EURN dataset. In this group of experiments, the number of piecewise intervals of $f_{i,j}(t)$ and $w_{i,j}(t)$ are both 10. The time interval $[t_d, t_a]$ is $[0, 1000]$. The experimental results show TWO-STEP-SEARCH can perform efficiently even though the number of vertices is larger than three millions or the number of edges is larger than ten millions. These experimental results indicates TWO-STEP-SEARCH are also suitable for large time-dependent graphs.



(a) varying time interval (b) varying distance

Figure 10: The cost-optimal path vs the fastest path

Exp-7. Comparing with the fastest path: In Fig. 10, we investigate the ratio $\frac{c'}{c*}$ on the CARN dataset with 20K vertices, where c' is the travel cost of the path with the minimum travel time from v_s to v_e and $c*$ is the cost of the path computed by TWO-STEP-SEARCH, i.e., the cost-optimal path with time constraint. $\frac{c'}{c*}$ reflects that how much cheaper the cost-optimal path is compared with the fastest path. In Fig. 10(a), we study $\frac{c'}{c*}$ by varying the length of time interval $[t_d, t_a]$. In this group of experiments, the earliest departure time is fixed to time point 0 and the latest arrival time ranges from time point 300 to time point 1200. We find $\frac{c'}{c*}$ increases with the length of time interval $[t_d, t_a]$ increasing. When time interval is $[0, 1200]$, $\frac{c'}{c*}$ is larger than 6. It indicates the fastest path is at least six times more expensive than the cost-optimal path. In Fig. 10(b), we study $\frac{c'}{c*}$ by varying the distance between source and destination. We also find $\frac{c'}{c*}$ increases with the increasing of distance. These experimental results show that the fastest path may not be a cost-optimal path and it is always much more expensive than a cost-optimal path.

5. RELATED WORK

Shortest path query is an important problem in graphs and has been well studied in static graphs. The existing works for the shortest path problem propose various index techniques to enhance the efficiency of the shortest path query for large graphs [23, 1, 8, 22, 5, 19]. All these works make a trade-off between the querying time

and index size. The main idea of these works are maintaining some shortest paths in index. Given a query, algorithms first retrieve the shortest path and then concatenate them by the shortest paths not in index. The recent literature [20] gives a full overview of the works on the shortest path problem.

Recently, there are several works on the shortest path problem in time-dependent graphs. However, these works are to solve the TDSP problem, that is, to find a path from source to destination that has the minimum travel time. These works are categorized as follows: one group is based on the discrete time model and the other one is based on the continuous time model. George et al. in [10] and [9] assign an aggregated attribute to each edge, called edge time series, which represents the time point at which the edge appears. Chabini et al. in [4] discretize the starting-time interval into k time points evenly, and construct a static graph by making k copies of each node and each edge, respectively. The TDSP problem can be solved as a static single-source shortest path problem in such a static graph, whose size is enlarged k times. The fundamental drawbacks of the discrete time model are two-fold. First, it cannot represent the state of networks between two discrete time points, which might yield inaccurate results. Second, the memory and processing requirement are high. Orda et al. propose a continuous time algorithm to solve the TDSP problem in [16]. This algorithm generalizes the BELLMAN-FORD shortest path algorithm. Sung et al. [21] present a Flow Speed Model that computes the travel time on each road segment as a piecewise linear function of time. Pfoser et al. [17, 18] contribute techniques that can derive the up-to-date speed associated with a road segment at a given time based on Floating Car Data. Kanoulas et al. in [13] propose an A*-extended algorithm. The main idea of this algorithm is to maintain a priority queue Q of all the paths to be expanded. For any vertex v_i, the algorithm estimates the travel time from v_i to destination v_e. By estimating the travel time, the algorithm computes a shortest path from source to destination with the minimum travel time. Gonzalez et al. in [11] apply some data mining techniques to derive driving and speed patterns that describe road speeds under a variety of conditions, such as time, weather, and vehicle type. Ding et al. in [7] propose an efficient 2S algorithm to solve the TDSP problem. All these methods utilize the following property: the earliest arrival time of a vertex v_i can be computed by the earliest arrival time of v_i's incoming neighbors. However, for the problem proposed in this paper, this property does not hold. Therefore, the solution to the TDSP problem cannot be used to solve the cost-optimal path problem proposed in this paper.

Several studies in the field of operation research consider the cost-optimal path problem under the discrete time model [3, 2]. In the discrete time model, the whole time interval is discretized to a set of time points, $\{t_1, t_2, \cdots, t_l\}$. For any edge (v_i, v_j), users only can select a specified time point t_x to depart from v_i. The main disadvantages of these works are as follows: (1) The optimal path cannot be found under the discrete time model. (2) These works need to compute the arriving cost for every vertex at every time point. The time and space costs are expensive.

6. CONCLUSION

In this paper, we study how to find a cost-optimal path with time constraint in time-dependent graphs. We first define the cost-optimal path with time constraint. Second, we propose an efficient TWO-STEP-SEARCH algorithm to compute a cost-optimal path with time constraint. We show that the time and space complexities of the algorithm are $O(kn \log n + mk)$ and $O((n + m)k)$ respectively. Finally, we confirm the effectiveness and efficiency of our algorithm through conducting experiments on real datasets.

7. ACKNOWLEDGMENTS

This work is partly supported by the grant of the National Program on Key Basic Research Project (973 Program) of China, No. 2012CB316200, the grant of the National Natural Science Foundation of China No. 61190115, 61173022, and the grant of the Research Grants Council of Hong Kong SAR, No. CUHK 418512.

8. REFERENCES

[1] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD Conference*, pages 349–360, 2013.

[2] X. Cai, T. Kloks, and C. Wong. Time-varying shortest path problems with constraints. *Networks*, 29(3):141–150, 1997.

[3] X. Cai, T. Kloks, and C. K. Wong. Shortest path problems with time constraints. In *MFCS*, pages 255–266, 1996.

[4] I. Chabini. Discrete dynamic shortest path problem in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Records*, 1645:170–175, 1998.

[5] J. Cheng, Y. Ke, S. Chu, and C. Cheng. Efficient processing of distance queries in large graphs: a vertex cover approach. In *SIGMOD Conference*, pages 457–468, 2012.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[7] B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, pages 205–216, 2008.

[8] J. Gao, R. Jin, J. Zhou, J. X. Yu, X. Jiang, and T. Wang. Relational approach for shortest path discovery over large graphs. In *PVLDB*, volume 5, pages 358–369, 2012.

[9] B. George, S. Kim, and S. Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. In *SSTD*, pages 460–477, 2007.

[10] B. George and S. Shekhar. Time-aggregated graphs for modeling spatio-temporal networks. *Journal on Data Semantics*, 11:191–212, 2006.

[11] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag. Adaptive fastest path computation on a road network: A traffic mining approach. In *VLDB*, pages 794–805, 2007.

[12] L. Jiang, S. Parekh, and J. Walrand. Time dependent network pricing and bandwidth trading. In *NOMS Workshops*, pages 193–200, 2008.

[13] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *ICDE*, pages 10–19, 2006.

[14] J. Leape. The london congestion charge. *Journal of Economic Perspectives*, 20(4):157–176, 2006.

[15] M. O'Mahony, W. Y. Szeto, and X. Q. Li. Modeling time-dependent tolls under transport, land use, and environment considerations. In *Applications of Advanced Technology in Transportation*, pages 852–857, 2006.

[16] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of ACM*, 37(3):607–625, 1990.

[17] D. Pfoser, S. Brakatsoulas, P. Brosch, M. Umlauft, N. Tryfona, and G. Tsironis. Dynamic travel time provision for road networks. In *GIS*, pages 68–71, 2008.

[18] D. Pfoser, N. Tryfona, and A. Voisard. Dynamic travel time maps - enabling efficient navigation. In *SSDBM*, pages 369–378, 2006.

[19] M. Qiao, H. Cheng, L. Chang, and J. X. Yu. Approximate shortest distance computing: A query-dependent local landmark scheme. In *ICDE*, pages 462–473, 2012.

[20] C. Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4):1–35, 2014.

[21] K. Sung, M. G. Bell, M. Seong, and S. Park. Shortest paths in a network with time-dependent flow speeds. *European Journal of Operational Research*, 121(1):32–39, 2000.

[22] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou. Shortest path and distance queries on road networks: An experimental evaluation. *PVLDB*, 5(5):406–417, 2012.

[23] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD Conference*, pages 857–868, 2013.