Finding the K Nearest Objects over Time Dependent Road Networks

Muxi Leng[1], Yajun Yang[1](✉), Junhu Wang[2], Qinghua Hu[1], and Xin Wang[1]

[1] School of Computer Science and Technology, Tianjin University, Tianjin, China
{mxleng,yjyang,huqinghua,wangx}@tju.edu.cn

[2] School of Information and Communication Technology, Griffith University, Brisbane, Australia
j.wang@griffith.edu.cn

Abstract. *K* nearest neighbor (kNN) search is an important problem and has been well studied on static road networks. However, in real world, road networks are often time-dependent, i.e., the time for traveling through a road always changes over time. Most existing methods for kNN query build various indexes maintaining the shortest distances for some pairs of vertices on static road networks. Unfortunately, these methods cannot be used for the time-dependent road networks because the shortest distances always change over time. To address the problem of kNN query on time-dependent road networks, we propose a novel voronoi-based index in this paper. Moreover, we propose an algorithm for pre-processing time-dependent road networks such that the waiting time is not necessary to be considered. We confirm the efficiency of our method through experiments on real-life datasets.

1 Introduction

With the rapid development of mobile devices, k nearest neighbor (kNN) search on road networks has become more and more important in location-based services. Given a query location and a set of objects (e.g., restaurants) on a road network, it is to find k nearest objects to the query location. kNN search problem has been well studied on static road networks. However, road networks are essentially time-dependent but not static in real world. For example, the Vehicle Information and Communication System (VICS) and the European Traffic Message Channel (TMC) are two transportation systems, which provide real-time traffic information to users. Such road networks are time-dependent, i.e., travel time for a road varies with taking "rush hour" into account.

The existing works propose various index techniques for answering k nearest object query on road networks. The main idea behind these indexes is to partition the vertices into several clusters, and then the clusters are organized as a voronoi diagram or a tree (e.g., R-tree, G-tree, etc.). All these methods pre-compute and maintain the shortest distances for some pairs of vertices to facilitate kNN query. Unfortunately, these indexes cannot be used for time-dependent road networks.

© Springer International Publishing AG, part of Springer Nature 2018
Y. Cai et al. (Eds.): APWeb-WAIM 2018, LNCS 10988, pp. 334–349, 2018.
https://doi.org/10.1007/978-3-319-96893-3_25

The reason is that the minimum travel time between two vertices often varies with time. For example, u and v are in the same cluster for one time period but they may be in two distinct clusters for another time period because of the minimum travel time varying with time. Therefore, the existing index techniques based on the static shortest distance cannot handle the case that the minimum travel time is time-dependent. Moreover, the waiting time is allowed on time-dependent road networks, i.e., someone can wait a time period to find another faster path. When the waiting time is considered, it is more difficult to build an index for kNN query by existing methods because it is difficult to estimate an appropriate waiting time for pre-computing the minimum travel time between two vertices.

Recently, there are some works about kNN query on time-dependent graphs [4–6,13]. Most of these works utilize A* algorithm to expand the road networks by estimating an upper or lower bound of travel time. There are two main drawbacks of these methods. First, in these works, the FIFO (first in first out) property is required for the networks and the waiting time is not allowed. Second, the indexes proposed by these works are based on the estimated value of travel time. However, these indexes cannot facilitate query effectively for large networks because the deviation are always too large between the estimated and actual travel time.

In this paper, we study k nearest object query on time dependent road networks. A time-dependent road network is modeled as a graph with time information. The weight of every edge is a time function $w_{i,j}(t)$ which specifies how much time it takes to travel through the edge (v_i, v_j) if departing at time point t. The main idea of our method is to pre-compute minimum travel time functions (or mtt-function for short) instead of concrete values for some pairs of vertices and then design a "*dynamic*" voronoi-based index based on such functions. Here "dynamic" means that in a time-dependent network it can be easily decided which cluster a vertex should be in for any given time point t. Different to previous works, our index can facilitate query effectively for large networks. Moreover, our method does not require the FIFO property for networks and we allow waiting time on every vertex.

The main contributions of this paper are summarized as below. First, we propose an algorithm to process $w_{i,j}(t)$ for every edge such that the waiting time is not necessary to be considered. Let G_T and G_T^* be the original graph and the graph after processing $w_{i,j}(t)$. We can prove that a shortest path with consideration of waiting time on G_T is one-one mapped to a shortest path without waiting time on G_T^*. Furthermore, we show how to compute the mtt-function for two vertices. Second, we propose a novel voronoi-based index for time-dependent road networks and an algorithm to answer kNN query using our index. Finally, we confirm the efficiency of our method through extensive experiments on real-life datasets.

The rest of this paper is organized as follows. Section 2 gives the problem statement. Section 3 describes how to process $w_{i,j}(t)$ and compute the mtt-function. Section 4 explains how to build the voronoi-based index for time-dependent networks and Sect. 5 proposes the kNN query algorithm. The experimental results are presented in Sect. 6. The related work is in Sect. 7. Finally, we conclude this paper in Sect. 8.

2 Problem Statement

Definition 1 (Time-Dependent Road Network): *A time-dependent road network is a simple directed graph, denoted as $G_T(V, E, W)$ (or G_T for short), where V is the set of vertices; $E \subseteq V \times V$ is the set of edges; and W is a set of non-negative value functions. For every edge $(v_i, v_j) \in E$, there is a time-function $w_{i,j}(t) \in W$, 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 one departs from v_i at time point t.*

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

The are several works that study how to construct time function $w_{i,j}(t)$, which is always modeled as a piecewise linear function [7,8,11] and it can be formalized as follows:

$$w_{i,j}(t) = \begin{cases} a_1 t + b_1, & t_0 \leq t < t_1 \\ a_2 t + b_2, & t_1 \leq t < t_2 \\ \cdots \\ a_p t + b_p, & t_{p-1} \leq t \leq t_p \end{cases}$$

Given a path p, the travel time of p is time-dependent. In order to minimize the travel time, some waiting time ω_i is allowed at every vertex v_i in p. That is, when arriving at v_i, one can wait a time period ω_i if the travel time of 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 v_i in p, we have

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

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

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

Thus the travel time of path p is $w(p) = \mathsf{arrive}(v_h) - t$. Given two vertices v_i and v_j in G_T, the minimum travel time from v_i to v_j with departure time t is defined as $m_{i,j}(t) = \min\{w(p)|p \in P_{i,j}\}$, where $P_{i,j}$ is the set of all the paths from v_i to v_j in G_T. Obviously, $m_{i,j}(t)$ is also a function related to the departure time t. We call $m_{i,j}(t)$ the ***minimum travel time function*** (or mtt-function shortly) from v_i to v_j. Let $|V|$ be n, in the following, we use $m_{i,n+j}(t)$ to represent mtt-function from a vertex v_i to an object o_j, in order to distinguish from $m_{i,j}(t)$ from v_i to a vertex v_j. Note that an object o_i is also a vertex regarded as v_{n+i} in the network.

Next, we give the definition of kNN query over time-dependent road networks.

Definition 2 (k Nearest Objects on Time-Dependent Road Networks): *Given a time-dependent road network $G_T(V, E, W)$, a set of the objects $O = \{o_1, o_2, \cdots\}$, a query point $v_q \in V$ and a departure time t_d, k nearest objects query of v_q is to find a k-size subset $O(v_q) \subseteq O$, such that $m_{q,n+j}(t_d) \geq \max\{m_{q,n+i}(t_d)|o_i \in O(v_q)\}$ for every object $o_j \in O \setminus O(v_q)$.*

3 Minimum Travel Time Function

We pre-compute mtt-functions for some pairs of vertices and then build the index to facilitate kNN query over time-dependent road networks. In this section, we first describe how to process the time function $w_{i,j}(t)$ for every edge in G_T such that the waiting time is not necessary to be considered when computing mtt-function and then explain how to compute mtt-function without waiting time.

3.1 Pre-processing Time Function for Every Edge

Given a path p, the waiting time ω_i is allowed for any vertex $v_i \in p$. However, it is not easy to find an appropriate value of ω_i for every $v_i \in p$ to minimize the travel time of p. In this section, we propose an algorithm to convert time function $w_{i,j}(t)$ to a new function $w^*_{i,j}(t)$ for every edge $(v_i, v_j) \in E$. We call $w^*_{i,j}(t)$ the "no waiting time function" of edge (v_i, v_j) (or nwt-function for short). The waiting time can be considered as zero when nwt-function is used to compute the minimum travel time of path p. The nwt-function $w^*_{i,j}(t)$ is defined by the following equation.

$$w^*_{i,j}(t) = \min_{\omega_i}(\omega_i + w_{i,j}(t + \omega_i)) \tag{1}$$

The following theorem guarantees the nwt-function $w^*_{i,j}(t)$ can be used to compute the minimum travel time for any path p in G_T without waiting time.

Theorem 1. *Given two time-dependent graphs $G_T(V, E, W)$ and $G^*_T(V, E, W^*)$, where W^* is the set of nwt-functions of all edges in E, for any path p in G_T, the minimum travel time of p in G_T* ***with consideration of waiting time*** *equals to the minimum travel time of p in G^*_T* ***without waiting time****.*

PROOF: Let $p = v_1 \to v_2 \to \cdots \to v_h$ be a given path with the departure time t. ω^*_i is the waiting time on v_i ($1 \leq i \leq h$) minimizing the travel time of p in G_T. We have $\mathsf{depart}(v_i) = \mathsf{arrive}(v_i) + \omega^*_i$ and $\mathsf{arrive}(v_{i+1}) = \mathsf{depart}(v_i) + w_{i,i+1}(\mathsf{depart}(v_i))$. Similarly, we have $\mathsf{depart}^*(v_i) = \mathsf{arrive}^*(v_i)$ and

Algorithm 1. Nwt-Function $(G_T(V, E, W))$

```
Input:  $G_T(V, E, W)$.
Output: $W^*$.

 1: $W^* \leftarrow \emptyset$;
 2: for every $w_{i,j}(t) \in W$ do
 3:     $\phi \leftarrow w_{i,j}(t_p)$, $w^*_{i,j}(t_p) \leftarrow w_{i,j}(t_p)$;
 4:     for $k = p$ to 1 do
 5:         $a^* \leftarrow -1$, $b^* \leftarrow t_k + \phi$;
 6:         $w^*_{i,j}(t) \leftarrow a^*t + b^*$ for $t \in [t_{k-1}, t_k)$;
 7:         $w^*_{i,j}(t) \leftarrow \min\{w^*_{i,j}(t), w_{i,j}(t) | t \in [t_{k-1}, t_k)\}$;
 8:         $\phi \leftarrow \min\{w^*_{i,j}(t_{k-1}), w^-_{i,j}(t_{k-1})\}$;
 9:     $W^* \leftarrow W^* \cup \{w^*_{i,j}(t)\}$;
10: return $W^*$
```

$\mathsf{arrive}^*(v_{i+1}) = \mathsf{depart}^*(v_i) + w^*_{i,i+1}(\mathsf{depart}^*(v_i))$ for G^*_T. We only need to prove $\mathsf{arrive}(v_h) = \mathsf{arrive}^*(v_h)$. It can be easily proved by induction on v_i. We omit it due to the space limitation. □

The algorithm to compute nwt-function is shown in Algorithm 1. For every $w_{i,j}(t) \in W$, Algorithm 1 computes $w^*_{i,j}(t)$ backward from $[t_{p-1}, t_p]$ to $[t_0, t_1]$ iteratively. In each iteration, $w^*_{i,j}(t)$ for $t \in [t_{k-1}, t_k)$ is computed. Algorithm 1 first sets $w^*_{i,j}(t)$ as $a^*t + b^*$, where $a^* = -1$ and $b^* = t_k + \phi$. ϕ is the minimum value between $w^*_{i,j}(t_k)$ and $w^-_{i,j}(t_k)$. $w^-_{i,j}(t_k)$ is the left limit value of $w_{i,j}(t)$ on t_k. Note that $w^*_{i,j}(t_k)$ and ϕ have been computed in the last iteration, i.e., the iteration for computing $w^*_{i,j}(t)$ on $[t_k, t_{k+1})$. ϕ is initialized as $w_{i,j}(t_p)$. Next, Algorithm 1 updates $w^*_{i,j}(t)$ as $\min\{w^*_{i,j}(t), w_{i,j}(t)\}$ for $t \in [t_{k-1}, t_k)$ and then ϕ is updated as $\min\{w^*_{i,j}(t_{k-1}), w^-_{i,j}(t_{k-1})\}$. The algorithm terminates when $w^*_{i,j}(t)$ has been computed for $t \in [t_0, t_1)$.

The time and space complexities analysis for Algorithm 1 are given below. Let n and m be the number of the vertices and edges in G_T respectively. For every edge (v_i, v_j), Algorithm 1 needs to compute $w^*_{i,j}(t)$ on $[t_{k-1}, t_k)$ iteratively from $k = p$ to 1. For every time interval $[t_{k-1}, t_k)$, $w^*_{i,j}(t)$ can be computed in constant time. Therefore, the time complexity of Algorithm 1 is $O(mp)$. Moreover, Algorithm 1 needs to maintain $w^*_{i,j}(t)$ and then the space complexity is also $O(mp)$.

Example 1. We illustrate how to compute $w^*_{i,j}(t)$ by an example in Fig. 1. As the solid black line in Fig. 1(a), $w_{i,j}(t)$ is a piecewise linear function:

$$w_{i,j}(t) = \begin{cases} t + 5, & 0 \le t < 10 \\ 15, & 10 \le t < 20 \\ -2t + 55, & 20 \le t \le 25 \end{cases}$$

In the first iteration, ϕ is initialized as $w_{i,j}(25) = 5$ and then $b^* = 25 + \phi = 30$. As the dashed red line in the right-side of Fig. 1(a), we find $a^*t + b^* = -t + 30$ is

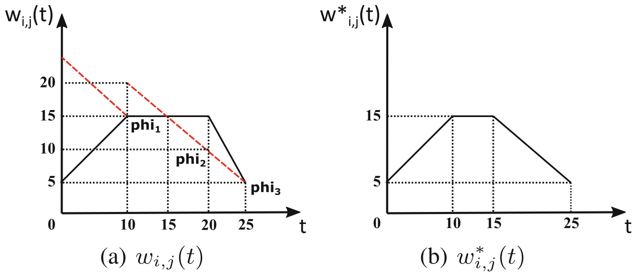


Fig. 1. Computing $w^*_{i,j}(t)$ (Color figure online)

always less than $w_{i,j}(t)$ on $[20, 25]$, then $w^*_{i,j}(t) = -t+30$ for $t \in [20, 25]$ and ϕ is updated as 10. Similarly, in the second iteration, $w^*_{i,j}(t)$ on $[10, 20)$ is computed as $\min\{15, -t + 30\}$, i.e., $w^*_{i,j}(t) = 15$ for $t \in [10, 15)$ and $w^*_{i,j}(t) = -t + 30$ for $t \in [15, 20)$. Then ϕ is updated as $\min\{w^*_{i,j}(10), w^-_{i,j}(10)\} = 15$. In the final iteration, as the dashed red line in the left-side of Fig. 1(a), $a^*t + b^* = -t + 25$ is always larger than $t + 5$ on $[0, 10)$, we have $w^*_{i,j}(t) = t + 5$ for $t \in [0, 10)$. Then $w^*_{i,j}(t)$ is given below and depicted in Fig. 1(b)).

$$w^*_{i,j}(t) = \begin{cases} t+5, & 0 \le t < 10 \\ 15, & 10 \le t < 15 \\ -t+30, & 15 \le t \le 25 \end{cases}$$

The following theorem guarantees the correctness of Algorithm 1.

Theorem 2. *The $w^*_{i,j}(t)$ computed by Algorithm 1 is exactly the* nwt*-function $w^*_{i,j}(t)$ given by Eq. (1).*

Proof: We proved it by induction on p.

Basis. We need to prove that $w^*_{i,j}(t)$ on time interval $[t_{p-1}, t_p]$ can be correctly computed by Algorithm 1. First, ω_i can only be zero when $t = t_p$, then we have $w_{i,j}(t_p) = w^*_{i,j}(t_p)$ and $\phi_p = w_{i,j}(t_p)$. Next, we consider the case of $t \in [t_{p-1}, t_p)$. By the definition of $w^*_{i,j}(t)$, we have

$$\begin{aligned} w^*_{i,j}(t) &= \min_{\omega_i}(\omega_i + w_{i,j}(t + \omega_i)) \\ &= \min_{\omega_i}(\omega_i + a_p(t + \omega_i) + b_p) \\ &= \min_{\omega_i}((a_p + 1)\omega_i + a_p t + b_p) \\ &= \min_{\omega_i}((a_p + 1)\omega_i + w_{i,j}(t)) \end{aligned}$$

For $t \in [t_{p-1}, t_p)$, if $a_p \ge -1$, $w^*_{i,j}(t)$ cannot decrease with ω_i increasing. It means $(a_p + 1)\omega_i + w_{i,j}(t)$ is minimum when $\omega_i = 0$ and then $w^*_{i,j}(t) = w_{i,j}(t)$. If $a_p < -1$, $w^*_{i,j}(t)$ will decrease with ω_i increasing and thus $(a_p + 1)\omega_i + w_{i,j}(t)$

is minimum when $\omega_i = t_p - t$, which is the longest waiting time on v_i for $t \in [t_{p-1}, t_p)$. Then we have

$$w^*_{i,j}(t) = (a_p + 1)(t_p - t) + a_p t + b_p = -t + t_p + \phi_p$$

Obviously, $w_{i,j}(t) \leq -t + t_p + \phi_p$ when $a_p \geq -1$ and $w_{i,j}(t) \geq -t + t_p + \phi_p$ when $a_p < -1$. Then we have $w^*_{i,j}(t) = \min\{w_{i,j}(t), -t + t_p + \phi_p\}$ for $t \in [t_{p-1}, t_p]$.

Induction. Assume the correct $w^*_{i,j}(t)$ can be computed by Algorithm 1 for $t \in [t_k, t_p]$, then we need to prove it also can be correctly computed for $t \in [t_{k-1}, t_k)$. We consider the following two cases: (1) $\omega_i \geq t_k - t$; and (2) $\omega_i < t_k - t$.

For case (1), the departure time $t + \omega_i \in [t_k, t_p]$ because $\omega_i \geq t_k - t$. By the assumption, nwt-function $w^*_{i,j}(t)$ has been correctly computed for $t \in [t_k, t_p]$, then $w^*_{i,j}(t_k)$ is the minimum travel time for edge (v_i, v_j) with departure time t_k. Therefore, $w^*_{i,j}(t)$ for $t \in [t_{k-1}, t_k)$ can be computed by the following equation:

$$w^*_{i,j}(t) = t_k - t + w^*_{i,j}(t_k)$$

For case (2), because $\omega_i < t_k - t$, then $t + \omega_i \in [t_{k-1}, t_k)$. Similar to the proof of basis, we have

$$w^*_{i,j}(t) = \min\{w_{i,j}(t), -t + t_k + w^-_{i,j}(t_k)\}$$

Note that, when $a_k < -1$, $w^*_{i,j}(t) = -t + t_k + w^-_{i,j}(t_k)$ because $w_{i,j}(t)$ may be noncontinuous at t_k. Therefore, we have

$$w^*_{i,j}(t) = \min\{w_{i,j}(t), -t + t_k + w^-_{i,j}(t_k), -t + t_k + w^*_{i,j}(t_k)\}$$

The proof is completed. □

3.2 Computing Minimum Travel Time Function

We adopt a Dijkstra-based algorithm proposed in [7] to compute mtt-function for two vertices v_i and v_j in G_T. This algorithm is only used for the case that the waiting time is not allowed. After converting $w_{i,j}(t)$ to nwt-function $w^*_{i,j}(t)$ for every edge in G_T by Algorithm 1, this algorithm can be used for time-dependent graphs with waiting time.

The main idea of this Dijkstra-based algorithm is to refine a function $g_{i,j}(t)$ iteratively for every $v_j \in V$, where $g_{i,j}(t)$ represents the earliest arrival time on v_j if departing from v_i at time point t. In every iteration, algorithm selects a vertex $v_x \in V$ and then refine $g_{i,x}(t)$ by extending a time domain I_x to a larger I'_x, where $I_x = [t_0, \tau_x]$ is a subinterval of the whole time domain T. $g_{i,x}(t)$ is regarded as well-refined in I_x if it specifies the earliest arrival time at v_x from v_i for any departure time $t \in I_x$. The algorithm repeats time-refinement process till $g_{i,j}(t)$ of destination v_j has been well-refined in the whole time domain T and then mtt-function $m_{i,j}(t)$ can be computed as $m_{i,j}(t) = g_{i,j}(t) - t$. The more details about this Dijkstra-based algorithm is given in [7]. As shown in [7], the time and space complexities are $O((n \log n + m)\alpha(T))$ and $O((n + m)\alpha(T))$ respectively, where $\alpha(T)$ is the cost required for each function (defined in interval T) operation.

4 The Novel Voronoi-Based Index

We propose a novel voronoi-based index for kNN query over time-dependent road networks. In static road networks, the voronoi diagram divides the network (or space) into a group of disjoint subgraphs (or sub-spaces) where the nearest object of any vertex inside a subgraph is the object generating this subgraph. However, in time-dependent road networks, the nearest object of a vertex may be dynamic. The nearest object of a vertex v may be o_i for departure time $t \in [t_1, t_2]$ but it may be o_j for $t \in [t_3, t_4]$. The main idea of our novel voronoi-based index is also to divide the vertex set V into some vertex subsets V_i and every subset V_i is associated with one object $o_i \in O$. Different to static road networks, our voronoi-based index are time-dependent, that is, every vertex v inside a subset is with a time interval indicating when the object o_i is nearest to v. Next, we describe what is the novel voronoi-based index and how to construct it.

4.1 What Is the Voronoi-Based Index?

Given a vertex v and an object o_i, $I_i(v)$ is called v's ***maximum time interval*** about o_i if it satisfies the following two conditions: (1) o_i is the nearest object of v for any departure time $t \in I_i(v)$; and (2) there does not exist another $I'_i(v) \supset I_i(v)$ satisfying the condition (1). Note that $I_i(v)$ may not be a continuous time interval, that is, if o_i is nearest to v for two disjoint departure time intervals $[t_1, t_2]$ and $[t_3, t_4]$, then $[t_1, t_2] \cup [t_3, t_4] \subseteq I_i(v)$. The voronoi-based index maintains a set C_i for every object $o_i \in O$, where C_i is a set of the tuples $(v, I_i(v))$ for all the vertices v with non-empty $I_i(v)$, i.e.,

$$C_i = \{(v, I_i(v)) | v \in V \wedge I_i(v) \neq \emptyset\}$$

We call C_i the ***closest vertex-time pair set*** of o_i. For simplicity, we say v is a vertex in C_i if $(v, I_i(v)) \in C_i$. Next, we give the definition of the border vertex.

Definition 3 (Border Vertex): *A vertex v_x in C_i is called a* ***border vertex*** *of C_i if there exist $v_y \in N^+(v_x)$ such that $(v_y, I_y) \notin C_i$ for any $I_y \supseteq f_{x,y}(I_i(v_x))$, where $N^+(v_x)$ is the outgoing neighbor set of v_x and $f_{x,y}(I_i(v_x))$ is the time interval mapped from $I_i(v_x)$ by the function $f_{x,y}(t) = t + w^*_{x,y}(t)$.*

The border vertex v_x of C_i indicates there exist a time point $t \in f_{x,y}(I_i(v_x))$ such that o_i is not the nearest object of v_y if one departs at time point t.

We use B_i to denote the set of all the border vertices of C_i. For every C_i, D_i is the set of mtt-functions $m_{x,n+i}(t)$ for all vertices v_x in C_i, that is,

$$D_i = \{m_{x,n+i}(t) | v_x \text{ is a vertex in } C_i\}$$

and M_i is a matrix of size $|C_i| \times |B_i|$ to maintain mtt-function $m_{x,y}(t)$ for all pairs of vertex v_x and border vertex v_y in C_i, i.e.,

$$M_i = \{m_{x,y}(t) | v_x \in C_i \wedge v_y \in B_i\}$$

The voronoi-based index is $\{\mathsf{C}, \mathsf{B}, \mathsf{D}, \mathsf{M}\}$, where C, B, D and M are the collections of all C_i, B_i, D_i and M_i respectively.

4.2 How to Construct the Voronoi-Based Index?

We have explained how to compute mtt-function in Sect. 3. Next, we describe how to compute C_i and B_i for every $o_i \in O$.

For every vertex $v_x \in V$, $I_i(v_x)$ is initialized as the whole time domain T. We refine $I_i(v_x)$ iteratively by removing the sub-intervals on which $m_{x,n+i}(t)$ is larger than $m_{x,n+j}(t)$ for another object o_j. It means o_i is not the nearest object of v_x when departure time is in these sub-intervals. For every $o_j \in O$ ($o_j \neq o_i$), let $T_j(v_x)$ denote the maximum time interval on which $m_{x,n+j}(t) < m_{x,n+i}(t)$, $I_i(v_x)$ is updated as $I_i(v_x) - T_j(v_x)$. After removing $T_j(v_x)$ for every other object o_j, if $I_i(v_x)$ is not empty, then the pair $(v_x, I_i(v_x))$ is inserted into C_i.

For every vertex v_x in C_i, if there exists an outgoing neighbor v_y of v_x, such that v_y is not in C_i or $f_{x,y}(I_i(v_x)) \nsubseteq I_i(v_y)$, then v_x must be a border vertex of C_i and it is inserted into B_i.

Algorithm 2. kNN-Query (G_T^*, v_q, t_d, k)

```
Input:  time-dependent graph $G_T^*$, query vertex $v_q$, departure time $t_d$ and $k$
Output: the $k$ nearest neighbor set $O(v_q)$

 1: $O(v_q) \leftarrow \emptyset$, $Q \leftarrow \{C_q\}$; $E_q \leftarrow \{v_q\}$
 2: while $|O(v_q)| < k$ do
 3:    $C_i \leftarrow$ Dequeue $(Q)$, $O(v_q) \leftarrow O(v_q) \cup \{o_i\}$;
 4:    for each $v_y \in B_i$ do
 5:       for each $v_x \in E_i$ do
 6:          $m_{q,y} \leftarrow \min\{m_{q,y}, m_{q,x} + m_{x,y}(t_d + m_{q,x})\}$;
 7:       for each $v_z \in N^+(v_y)$ do
 8:          if $m_{q,z} > m_{q,y} + w^*_{y,z}(t_d + m_{q,y})$ then
 9:             $m_{q,z} \leftarrow m_{q,y} + w^*_{y,z}(t_d + m_{q,y})$;
10:             Let $C_j$ be the set including $v_z$ when $t = t_d + m_{q,z}$;
11:             if $C_j \notin O(v_q)$ then
12:                $E_j \leftarrow E_j \cup \{v_z\}$;
13:                if $m_{q,n+j} > m_{q,z} + m_{z,n+j}(t_d + m_{q,z})$ then
14:                   $m_{q,n+j} \leftarrow m_{q,z} + m_{z,n+j}(t_d + m_{q,z})$;
15:                   if $C_j \notin Q$ then
16:                      Enqueue$(Q, C_j)$;
17:                   else
18:                      Update$(Q, C_j)$;
19: return $O(v_q)$
```

5 Query Processing

Algorithm 2 describes how to find the k nearest objects for a query vertex v_q with departure time t_d. In Algorithm 2, $O(v_q)$ is a set to maintain the objects that have been found so far and Q is a priority queue to maintain a candidate set of C_i whose o_i is possible to be an object in kNN set. All $C_i \in Q$ are sorted in an ascending order by the minimum travel time $m_{q,n+i}$ from v_q to o_i. The top C_i in Q is with the minimum $m_{q,n+i}$ and it can be easily done using Fibonacci Heap. $O(v_q)$ and Q are initialized as $\emptyset$ and $\{C_q\}$ respectively, where C_q contains v_q for the departure time t_d, i.e., $(v_q, I_q(v_q)) \in C_q$ and $t_d \in I_q(v_q)$. $O(v_q)$ is expanded iteratively by inserting objects one by one from Q until $|O(v_q)| = k$. In each iteration, if $|O(v_q)| < k$, Algorithm 2 first dequeues the top C_i from Q with the minimum $m_{q,n+i}$. The object o_i of C_i must be one of k nearest objects of v_q. It can be guaranteed by Theorem 3. Then o_i will be inserted into $O(v_q)$. For every border vertex v_y in C_i, Algorithm 2 computes $m_{q,y}$ as $\min\{m_{q,x} + m_{x,y}(t_d + m_{q,x}) | v_x \in E_i\}$, where E_i is the entry set of C_i. The "entry" means any path entering into C_i must go through a vertex in E_i. E_i will be updated when Algorithm 2 runs. For every $v_z \in N^+(v_y)$, if $m_{q,z} > m_{q,y} + w^*_{y,z}(t_d + m_{q,y})$, then $m_{q,z}$ will be updated as $m_{q,y} + w^*_{y,z}(t_d + m_{q,y})$. Next, if v_z is in C_j ($C_j \neq C_i$ and $C_j \notin O(v_q)$) at the time point $t_d + m_{q,z}$, then v_z will be inserted into E_j as an entry of C_j. For the object o_j of C_j, $m_{q,n+j}$ will be updated as $m_{q,z} + m_{z,n+j}(t_d + m_{q,z})$ when $m_{q,n+j} > m_{q,z} + m_{z,n+j}(t_d + m_{q,z})$. If C_j is not in Q, then C_j will be enqueued into Q. Otherwise, C_j has been in Q and Q will be updated by C_j with new $m_{q,n+j}$. Algorithm 2 terminates when the size of $O(v_q)$ is k.

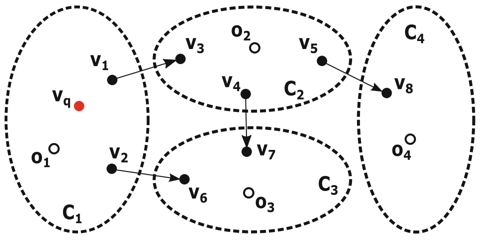


Fig. 2. Query processing

Example 2. We use the example in Fig. 2 to illustrate the kNN querying process for $k = 3$. In this example, v_q is the query vertex and it is in C_1 for the departure time t_d. Q and $O(v_q)$ are initialized as $\{C_1\}$ and $\emptyset$ respectively. In the first iteration, C_1 is dequeued from Q and then o_1 is inserted into $O(v_q)$. Because v_1 is a border vertex of C_1 and v_3 is an outgoing neighbor of v_1, Algorithm 2 computes $m_{q,1}(t_d)$ and $m_{q,3}(t_d) = m_{q,1}(t_d) + w^*_{1,3}(t_d + m_{q,1}(t_d))$. Note that v_3 is in C_2 when $t = t_d + m_{q,3}(t_d)$ and then it is an entry of C_2. Therefore, C_2 is enqueued into Q. Similarly, C_3 is also enqueued into Q and $Q = \{C_2, C_3\}$. Assume that o_2 is nearer to v_q than o_3, in the second iteration, C_2 is dequeued and $O(v_q)$ is updated as $\{o_1, o_2\}$. In the same way, C_4 will be enqueued into Q in this iteration. In the final iteration, C_3 will be dequeued due to o_3 is nearer to v_q and then $O(v_q) = \{o_1, o_2, o_3\}$. Because $|O(v_q)| = 3$, Algorithm 2 terminates and returns $O(v_q)$.

The next theorem guarantees the correctness of Algorithm 2.

Theorem 3. *In Algorithm 2, the object o_i of C_i dequeued from Q in the k-th iteration must be the k-th nearest object of query vertex v_q for the departure time t_d.*

Proof: We prove it by induction on k.

Basis. Obviously, C_q is dequeued from Q in the first iteration. By the definition of C_q, o_q is the nearest object of v_q when the departure time is t_d.

Induction. Assume that the i-th nearest neighbor of v_q is dequeued from Q in the i-th iteration for $i < k$. We need to prove it also hold for $i = k$. We prove it by contradiction. Let C_k be the closest vertex-time pair set dequeued from Q in the k-th iteration and o_k is the object of C_k. Suppose that the k-th nearest object of v_q is $o_{k'}$ and $o_{k'} \neq o_k$. Let p be the shortest path from v_q to $o_{k'}$ with the departure time t_d. Because $k > 1$, then $C_{k'}$ is not C_q and there must exist an entry v_e of $C_{k'}$ in p. Let v_b be the predecessor of v_e in p, then v_b must be a border vertex of C_b at time point $t_d + m_{q,b}$ and $C_b \neq C_{k'}$. There are two cases for the object o_b of C_b: (1) o_b is not in the k nearest object set of v_q; and (2) o_b is in the k nearest object set of v_q.

For case (1), by the definition of C_b, o_b is the nearest neighbor of v_b at time point $t_d + m_{q,b}$, then we have

$$m_{q,b} + m_{b,n+b}(t_d + m_{q,b}) < m_{q,b} + m_{b,n+k'}(t_d + m_{q,b})$$

Thus o_b is nearer to v_q than $o_{k'}$ when the departure time is t_d. It means o_b must be in the k nearest object set of v_q, which is a contradiction.

For case (2), Let o_b be the i-th ($i < k$) nearest object of v_q, by the inductive assumption, C_b is dequeued from Q in i-th iteration. According to the Algorithm 2, $C_{k'}$ is enqueued into Q in this iteration. Therefore, $C_{k'}$ will be dequeued from Q in k-th iteration instead of C_k, which is a contradiction. The proof is completed □

The time and space complexities of Algorithm 2 are given below. Let b and e be the average size of B_i and E_i respectively. In every iteration, Algorithm 2 upadates $m_{q,y}$ as $\min\{m_{q,y}, m_{q,x} + m_{x,y}(t_d + m_{q,x})\}$ for every border vertex v_y in C_i. It will cost $O(be)$ time. For every outgoing neighbor v_z of border vertex v_y, Algorithm 2 needs to compute $m_{q,z}$ and then it will cost $O(bd)$ time, where d is the average out-degree of the vertices in G_T. Therefore, the time complexity of Algorithm 2 is $O(kb(d + e))$. On the other hand, because Algorithm 2 needs to maintain $m_{q,y}$ and $m_{q,z}$, then the space complexity is $O(k(b + e))$.

6 Experiements

We compare our voronoi-based index method (marked as VI) with FTTI (Fast-Travel-Time Index) method [13] and TLNI (Tight-and-Loose-Network Index) method [6] on the real-life datasets. FTTI and TLNI are the state of the art index-based methods for kNN query over time-dependent road networks. Note that FTTI and TLNI are used on G^*_T in which every edge is an nwt-function $w^*_{i,j}(t)$ because FTTI and TLNI do not allow the waiting time. Although some algorithms are proposed in recent works [1,4], they are only to find the nearest object (i.e., $k = 1$) and they cannot be used for general kNN query on time-dependent graphs. All the experiments are conducted on a 2.6 GHz Intel Core i7 CPU PC with the 16 GB main memory, running on Windows 7.

6.1 DataSets and Experiment Setup

We tested the voronoi-based index method on California road network (CARN) with 196,5206 vertices and 553,3214 edges. We extracted five time-dependent graphs with different size using the CARN dataset. The number of vertices ranges from 100k to 500k. The time domain is set as $T = [0, 2000]$, i.e., the departure time t can be selected from $[0, 2000]$ for any vertex. Here, 2000 means 2000 time units. For every $w_{i,j}(t)$, we split the time domain T to p subintervals and assign a linear function randomly for every sub-interval and then $w_{i,j}(t)$ is a piecewise linear function.

6.2 Experimental Results

Exp-1. Impact of Network Size: In this group of experiments, we study the impact of time-dependent network size. The number of the vertices increases from 100k to 500k and the number of objects is fixed at 10k. We investigate the querying time for $k = 7$. The number of piecewise intervals of $w_{i,j}(t)$ is set as 4. As shown in Fig. 3(a) and (b), the querying time of our method is always less than FTTI and TLNI. Specifically, the querying time of TLNI is always much more than our method even though TLNI has the smallest index size. The reason is TLNI index only maintain the vertices for an object o_i that the upper bound of travel time to o_i are less than the lower bound to the other objects. It cannot facilitate query effectively in large networks.

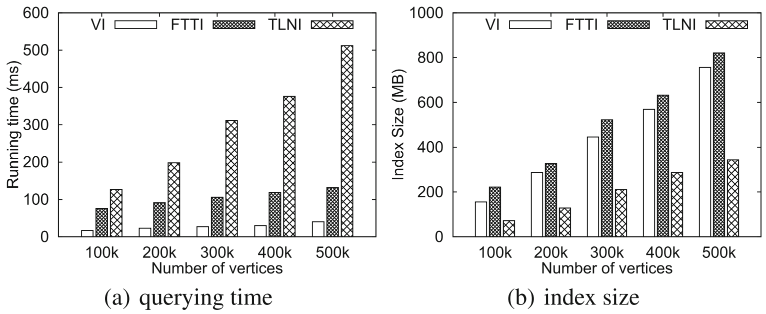


Fig. 3. Impact of the network size

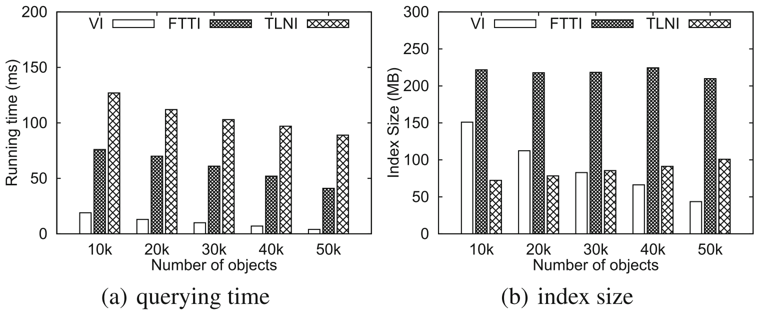


Fig. 4. Impact of the object set size

Exp-2. Impact of Object Set Size: In this group of experiments, the number of the vertices is fixed at 100k and the number of objects ranges from 10k to 50k. As shown in Fig. 4(a) and (b), the querying time of our method are always less than FTTI and TLNI. Moreover, the querying time and index size decrease with the increasing of the object set size. There are two reasons as follows: (1) the average size of C_i and B_i decrease if the object set size increases; (2) the increasing of object size results in that the objects become nearer to v_q and then querying time decreases.

Exp-3. Impact of the Time Domain: In Fig. 5, we study the impact of time domain. In this group of experiments, the number of vertices and objects are fixed at 100k and 10k respectively. The time domain ranges from $[0, 1000]$ to $[0, 3000]$. We investigate the querying time for $k = 7$. As shown in Fig. 5(a) and (b), the querying time and index size of our method are not affected by the expanding of time domain. However, for FTTI and TLNI, the querying time increases with the the expanding of time domain. It is because they need to maintain the estimated value about travel time in index to facilitate kNN query. If the time domain becomes larger, the deviation between the estimation and actual travel time will become larger too. It cannot facilitate query effectively.

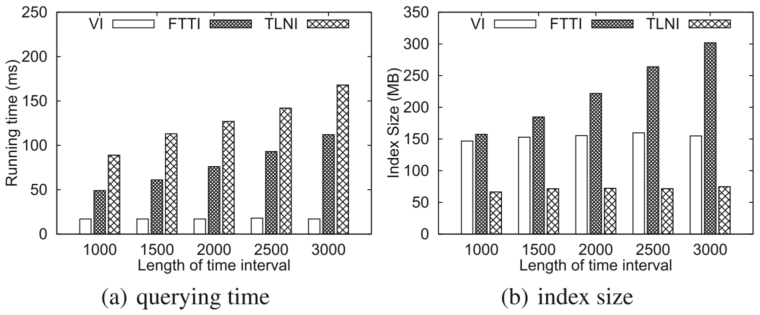
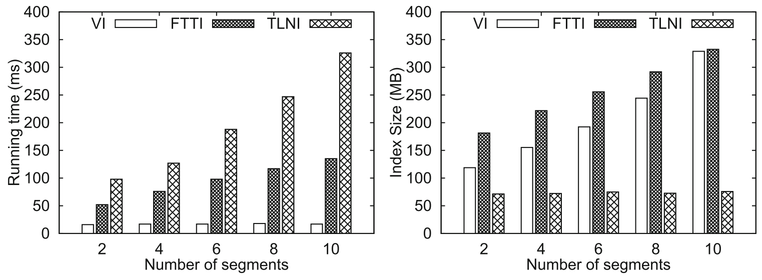


Fig. 5. Impact of the length of time interval



(a) querying time (b) index size

Fig. 6. Impact of the number of piecewise interval of time function

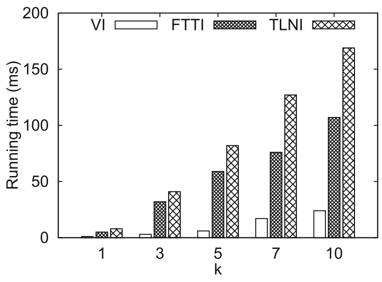
Exp-4. Impact of the Number of Piecewise Intervals: In Fig. 6, we investigate the impact of the number of piecewise intervals of $w_{i,j}(t)$. In this group of experiments, the number of piecewise intervals of $w_{i,j}(t)$ increases from 2 to 10. The number of the vertices and objects are fixed at 100k and 10k, respectively. As shown in Fig. 6(a) and (b), the querying time and index size always increase with the increasing of the number of piecewise intervals. The reason is that the more piecewise intervals of $w_{i,j}(t)$ results in more piecewise intervals of mtt-function and then the more border vertices will be maintained in the index.

Exp-5. Impact of k: In Fig. 7, we study the querying time by varying k from 1 to 10 on two different networks with 10k vertices and 50k vertices respectively. In this group of experiments, the number of objects are fixed at 10k and 50k for two different networks respectively. As shown in Fig. 7(a) and (b), the querying time always increases marginally with the increasing of k for our index method.

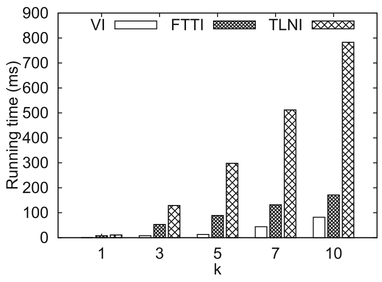
7 Related Work

kNN query has been well-studied on static road networks. Most of the existing works propose various index techniques. The main ideas of these methods are to partition the vertices into several clusters, and then the clusters are organized as a voronoi diagram or a tree (e.g., R-tree) [9,10,12,14–16,18–20]. These methods pre-compute and maintain the shortest distances for some pairs of vertices to facilitate kNN query. Unfortunately, these index techniques cannot be used for the time-dependent road networks because the minimum travel time between two vertices always varies with time.

kNN query has also been studied on time-dependent road networks [1,3–6,13]. Most of these works are based on A* algorithm. The authors in [1,4] study the problem to find nearest (i.e., $k = 1$) object on time-dependent networks. In [4], A virtual node v is inserted into the graph G with the zero-cost edges connecting to all the objects. The nearest object can be found on the shortest path from the query vertex to v. The authors in [2] study problem of finding k POIs that minimize the aggregated travel time from a set of query points.



(a) the number of vertices is 10k



(b) the number of vertices is 50k

Fig. 7. Impact of k

The index-based methods are proposed in [6,13]. In [6], A* algorithm is utilized to expand the road networks by estimating an upper or lower bound of travel time. An index is built to facilitate kNN query using these estimated bounds. In [13], time domain is divided to several sub-intervals. For every sub-interval, C nearest objects of every vertex are found by an estimation of minimum travel time. There are two main drawbacks of these methods. First, in these works, the FIFO (first in first out) property is required for networks and waiting time is not allowed. Second, the indexes proposed by these works are based on the estimated value of travel time. However, these indexes cannot facilitate query effectively for the large networks because the deviations are always too large between the estimated and actual travel time.

Recently, there are some works about the shortest path query between two given vertices over time-dependent graphs [7,17]. However, these works does not study any index that can be used in kNN query over time-dependent road networks. The method in [7] is used to compute mtt-function between two vertices in our paper.

8 Conclusion

In this paper, we study the problem of k nearest objects query on time-dependent road networks. We first give an algorithm for processing time-dependent road networks such that the waiting time is not necessary to be considered and then propose a novel voronoi-based index to facilitate kNN query. We explain how to construct the index and complete the querying process using our index. We confirm the efficiency of our method through extensive experiments on real-life datasets.

Acknowledgments. This work is supported by the grant of the National Natural Science Foundation of China No. 61402323, 61572353 and the Australian Research Council Discovery Grant DP130103051.

References

1. Chucre, M.R.R.B., do Nascimento, S.M., de Macêdo, J.A.F., Monteiro, J.M., Casanova, M.A.: Taxi, please! A nearest neighbor query in time-dependent road networks. In: MDM, pp. 180–185 (2016)
2. Costa, C.F., Machado, J.C., Nascimento, M.A., de Macêdo, J.A.F.: Aggregate k-nearest neighbors queries in time-dependent road networks. In: SIGSPATIAL, pp. 3–12 (2015)
3. Costa, C.F., Nascimento, M.A., de Macêdo, J.A.F., Machado, J.C.: A*-based solutions for KNN queries with operating time constraints in time-dependent road networks. In: MDM, pp. 23–32 (2014)
4. Cruz, L.A., Lettich, F., Júnior, L.S., Magalhães, R.P., de Macêdo, J.A.F.: Finding the nearest service provider on time-dependent road networks. In: ECML-PKDD, pp. 21–31 (2017)
5. Cruz, L.A., Nascimento, M.A., de Macêdo, J.A.F.: K-nearest neighbors queries in time-dependent road networks. JIDM **3**(3), 211–226 (2012)
6. Demiryurek, U., Kashani, F.B., Shahabi, C.: Efficient k-nearest neighbor search in time-dependent spatial networks. In: DEXA, pp. 432–449 (2010)
7. Ding, B., Yu, J.X., Qin, L.: Finding time-dependent shortest paths over large graphs. In: EDBT, pp. 205–216 (2008)
8. George, B., Shekhar, S.: Time-aggregated graphs for modeling spatio-temporal networks. J. Data Semant. **11**, 191–212 (2006)
9. Hu, H., Lee, D.L., Xu, J.: Fast nearest neighbor search on road networks. In: EDBT, pp. 186–203 (2006)
10. Huang, X., Jensen, C.S., Saltenis, S.: The islands approach to nearest neighbor querying in spatial networks. In: SSTD, pp. 73–90 (2005)
11. Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on a road network with speed patterns. In: ICDE, p. 10 (2006)
12. Kolahdouzan, M.R., Shahabi, C.: Voronoi-based K nearest neighbor search for spatial network databases. In: VLDB, pp. 840–851 (2004)
13. Komai, Y., Nguyen, D.H., Hara, T., Nishio, S.: kNN search utilizing index of the minimum road travel time in time-dependent road networks. In: SRDS, pp. 131–137 (2014)
14. Lee, K.C.K., Lee, W., Zheng, B.: Fast object search on road networks. In: EDBT, pp. 1018–1029 (2009)
15. Wei-Kleiner, F.: Finding nearest neighbors in road networks: a tree decomposition method. In: EDBT, pp. 233–240 (2013)
16. Yang, S., Cheema, M.A., Lin, X., Zhang, Y., Zhang, W.: Reverse k nearest neighbors queries and spatial reverse top-k queries. VLDB J. **26**(2), 151–176 (2017)
17. Yang, Y., Gao, H., Yu, J.X., Li, J.: Finding the cost-optimal path with time constraint over time-dependent graphs. Proc. VLDB Endow. **7**, 673–684 (2014)
18. Zheng, Y., Guo, Q., Tung, A.K.H., Wu, S.: Lazylsh: approximate nearest neighbor search for multiple distance functions with a single index. In: SIGMOD, pp. 2023–2037 (2016)
19. Zhong, R., Li, G., Tan, K., Zhou, L.: G-tree: an efficient index for KNN search on road networks. In: CIKM, pp. 39–48 (2013)
20. Zhu, H., Yang, X., Wang, B., Lee, W.: Range-based obstructed nearest neighbor queries. In: SIGMOD, pp. 2053–2068 (2016)