Towards K-Nearest Neighbor Search in Time-Dependent Spatial Network Databases [⋆]

Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi

University of Southern California
Department of Computer Science
Los Angeles, CA 90089-0781
[demiryur, banaeika, shahabi]@usc.edu

Abstract. The class of k Nearest Neighbor (kNN) queries in spatial networks is extensively studied in the context of numerous applications. In this paper, for the first time we study a generalized form of this problem, called the Time-Dependent k Nearest Neighbor problem (TD-kNN) with which edge-weights are time variable. All existing approaches for kNN search assume that the weight (e.g., travel-time) of each edge of the spatial network is constant. However, in real-world edge-weights are time-dependent (i.e., the arrival-time to an edge determines the actual travel-time on that edge) and vary significantly in short durations. We study the applicability of two baseline solutions for TD-kNN and compare their efficiency via extensive experimental evaluations with real-world data-sets, including a variety of large spatial networks with real traffic-data recordings.

1 Introduction

With the ever growing popularity of online map services (e.g., Google Maps) and their wide deployment in hand-held devices (e.g.,iPhone) and car-navigation systems, more and more users search for geographical points of interests (e.g., restaurants, hospitals) and the corresponding directions and travel-times to these locations. Consequently, many recent research studies (e.g., [23, 5, 18, 16, 22, 2, 12, 13, 17, 24]) focus on developing techniques to accurately and efficiently compute the distance and route between objects in large road-networks. However, a majority of these studies rely on pre-computation of distances in the network and assume that the cost of traveling each edge of the road-network is constant (e.g., corresponding to the length of the edge).

On the other hand, we are witnessing an increase in the instrumentation of roads in major cities for collecting real-time traffic data. For example, we are working with LA METRO [1] from whom we are receiving real-time traffic data from more than 6500 sensors on various freeways and artillery roads in Los Angeles (LA) county. Studying this real-world traffic data, we observe that the actual travel-time on a road heavily depends on the

[⋆] This research has been funded in part by NSF grants IIS-0534761 and CNS-0831505 (CyberTrust), the NSF Center for Embedded Networked Sensing (CCR-0120778) and in part from the METRANS Transportation Center, under grants from USDOT and Caltrans. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[1] http://www.metro.net/

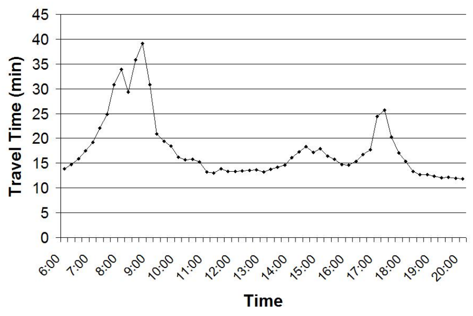


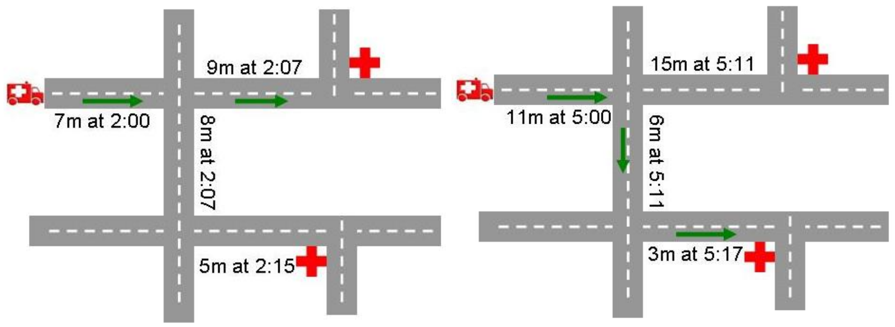
Fig. 1. Real-world travel-time for a weekday on a segment of I-405 in Los Angeles County

traffic congestion on the edges and is a function of the time of the day. To illustrate, consider Figure 1 that shows the graph of real-world travel-time on a segment of I-405 freeway in LA between 6am and 8pm on a weekday. Some interesting observations can be made from this figure. First, the travel-time of a segment is a function of the time of the day, hence, the term *time-dependent* travel-time. Second, the change in travel-time is significant, for example from 9:00am to 9:30am, the travel-time of this segment changes from 40 minutes to 20 minutes (100% change). Note that certain network segments (e.g., bridges) can be unavailable during during certain instants of time. Hence, the fastest path from a source to a destination may vary significantly depending on the time of the day. Third, the duration of the change in travel-time is rather short, e.g., within 30 minutes, and the change is continuous and not abrupt. Therefore, the travel-time of an (future) edge may change during a trip. These simple observations have a major computation implication: the time that one arrives at the segment entry determines the travel-time on that segment. Hence, to compute the fastest path from a source to a destination, all combinations of arrival-times at all possible segment entrances must be considered. We call this phenomenon *"arrival-dependency"* and we observe that because of this fact, naive approaches may find incorrect shortest paths (thus incorrect nearest neighbors), especially at the boundaries of traffic rush-hours. Fourth, we remark that there are only a handful of unique travel-time graphs for a given segment (e.g., weekday-graph, weekend-graph, holiday-graph) and hence we can assume that at any given time for any given segment we know the travel-time a priori[2].

Moreover, in addition to the time-dependent traffic data collected by governmental agencies (such as the data we receive from LA Metro), recently an increasing number of navigation companies are also releasing time-dependent travel-time information for road networks. For example, NAVTEQ [19], a leading provider of navigation services, offers a Predictive Flow Service that provides time-dependent travel-times up to one year. Also, INRIX [14] recently announced its new service that provides future traffic (at the temporal granularity of five minutes) computed based on the historical traffic data and local information like weather, school schedules, and sporting events.

[2] Traffic prediction is an active area of research and beyond the scope of this paper.

Figure 2 illustrates an example of time-dependent k nearest neighbor search. With this example, an ambulance is looking for the nearest hospital at 2 PM and 5 PM on the same day on a particular road network. The time-dependent travel-time (in minutes) and the arrival time for each edge are shown on the edges. Note that the travel-times on the edges change with arrival time to the edges in Figures 2(a) and 2(b). Therefore, the query launched by the ambulance at 2pm and 5pm would return different results.



(a) 1-NN Query at 2 PM (b) 1-NN Query at 5 PM

Fig. 2. Time-dependent 1-NN search

From Figure 2, one can come up with a naive approach to answer time-dependent kNN problem by applying an existing kNN search algorithm (e.g., [22, 16, 2, 12]) on different snapshots of the graph generated at discrete times. However, there are fundamental shortcomings with such a naive approach. First, the naive approach needs to update the edge weights and hence, the pre-computation (if any) for every snapshot. This is not realistic for real-world scenarios where the spatial network is large. Second, the naive approach can provide inaccurate results since the computations are done on discrete times rather than in continuous time. Specifically, the shortest path between the objects is derived based on the weights known at the query time for all network edges, disregarding the probably changing weight information during the trip (recall that the network edge weights change over the time). For example, consider the travel-time of all the edges are equal to five minutes at 5 PM in Figure 2(b). Finally, it is very hard to decide on the effective choice of snapshot intervals for real-world applications.

Considering afore-mentioned observations on the impact of time-dependency on fastest path computation and the availability of time-dependent travel-time information for road networks, the need for computational techniques that can answer time-dependent spatial network queries (e.g., time-dependent kNN query) is apparent and immediate. Unfortunately, once we consider the road networks with time-varying edge weights, all the techniques assuming constant edge-weights and/or relying on distance pre-computation would fail to answer k nearest neighbor queries in time-dependent road networks.

In this paper, for the first time, we study the problem of *Time-dependent k Nearest Neighbor* (TD-kNN) search. TD-kNN finds the k static nearest neighbors of a query object which is moving on a *time-dependent network* (i.e., a network where edge weights are variable functions of time). We discuss and compare two different baseline methods to answer time-dependent k nearest neighbor queries in both discrete and continuous time. With the first approach, we use time-expanded graphs to model the time-dependent network. This allows us to exploit previously developed techniques for static networks to solve TD-kNN problem with approximate results. With the second approach, we adopt incremental network expansion by generalizing it to time-dependent networks.

The remainder of this paper is organized as follows. In Section 2, we review the related work on both kNN queries and time-dependent shortest path algorithms. In Section 3, we formally define the time-dependent k nearest neighbor query in spatial networks. In Section 4, we introduce two different baseline approaches for time-dependent kNN queries. In Section 5, we present the results of our experimental evaluation of our proposed approaches with a variety of spatial networks with large number of data and query objects. Finally, in Section 6 we conclude and discuss our future work.

2 Related Work

In this section we review previous studies on kNN query processing in road networks as well as time-dependent shortest path computation.

2.1 kNN Queries in Spatial Networks

In [22], Papadias et al. introduced Incremental Network Expansion (INE) and Incremental Euclidean Restriction (IER) methods to support kNN queries in spatial networks. While INE is adaption of Dijkstra's algorithm, IER exploits the Euclidean restriction principle in which the results are first computed in Euclidean space and then refined by using the network distance. In [16], Kolahdouzan and Shahabi proposed first degree *network Voronoi diagrams* to partition the spatial network to network Voronoi polygons (NVP), one for each data object. They indexed the NVPs with a spatial access method to reduce the problem to a point location problem in Euclidean space and minimize the on-line network distance computation by precomputing the NVPs. Cho et al. [2] presented a system UNICONS where the main idea is to integrate the precomputed k nearest neighbors into the Dijkstra algorithm. Hu et al. [4] proposed a distance signature approach that precomputes the network distance between each data object and network vertex. The distance signatures are used to find a set of candidate results and Dijkstra algorithm is employed to compute their exact network distance. Huang et al. addressed the kNN problem using *Island* approach [12] where each vertex is associated to all the data points that are centers of given radius r (so called islands) covering the vertex. With their approach, they utilized a restricted network expansion from the query point while using the precomputed islands. In [13], Huang et al. introduced *S-GRID* where they partition the spatial network to disjoint sub-networks and precompute the shortest path for each pair of border points. To find the k nearest neighbors, they first perform a network expansion within the sub-networks and then proceed to outer expansion between the border points by utilizing the precomputed information. Recently Samet et al. [23] proposed a method where they associate a label to each edge that represents all nodes to which a shortest path starts with this particular edge. They use these labels to traverse *shortest path quadtrees* that enables geometric pruning to find the network distance. With all these studies, the network edge weights are assumed to be static (i.e., travel-time functions of all edges are constants) and hence the shortest path computations and precomputations are invalidated with time-varying edge weights. Unlike the previous approaches, we make a fundamentally different assumption that the weight of the network edges are time-dependent rather than fixed. Our assumption yields a much more realistic scenario and versatile approach.

2.2 Time-dependent Shortest Path Studies

Cooke and Halsey [3] introduced the first time-dependent shortest path (TDSP) solution where they formulated the problem in discrete time and use dynamic programming. Dreyfus [8] proposed a generalization of Dijkstra algorithm, but his algorithm is showed (by Halpren [11]) to be true in only FIFO networks. If the FIFO property does not hold in a time-dependent network, then the problem is NP-Hard as shown in [20]. In [1], Chabini proposed a discrete time one-to-all and all-to-one TDSP algorithm that allows waiting at network nodes. In [9], George and Shekhar proposed a time-aggregated graph where they aggregate the travel-times of each edge over the time instants into a time series. Their model has less storage requirements than the time-expanded networks. All these studies assume that the edge weight functions are defined over a finite discrete window of time $t \in t_0, t_1, .., t_n$, where t_n is determined by the total duration of time interval under consideration. Therefore, the problem is reduced the problem of computing, for each time window, minimum-weight paths through the static network where one can apply any of the well-known shortest path algorithms. Although discrete-time algorithms are easy to implement, they have numerous shortcomings mainly on storage (discussed in Section 1). Orda and Rom [20] proposed a Bellman-Ford based solution where edge weights are piece-wise linear functions. In [4], Dean proposed a label-setting algorithm where arrival times are considered as the labels of the nodes. In [7], Ding et al. also used a variation of label-setting algorithm to solve the time-dependent shortest path problem. Their algorithm decouples the path-selection and time-refinement by scanning a sequence of time steps of which the size depends on the values of the arrival time functions. The focus of this algorithm is to find a fastest path in a time-dependent graph for a given start time interval (e.g., between 7:30 AM and 8:30 AM). In [15], Kanoulas et al. introduced a Time-Interval All Fastest Path (allFP) algorithm based on A* algorithm in time-dependent networks. Instead of sorting the priority queue by scalar values, they maintain a priority queue of all paths to be expanded. Therefore, they enumerate all the paths from the source to a destination node which incurs exponential running time in the worst case. In addition, their algorithm is efficient when estimation (heuristic function in A*) can enable effective search space pruning. It is difficult to find such estimation in time-dependent graphs.

3 Problem Definition

In this section, we will formally define the problem of time-dependent kNN search in spatial networks. We assume a spatial network (e.g. the Los Angles road network) containing a set of static data objects (i.e., points of interest such as restaurants, hospitals) as well as moving query objects searching for their kNN. We model the spatial network as a time-dependent weighted graph whose edge and node properties as well as topological structure are time varying. We assume that the non-negative edge weights are time-dependent travel-times between the nodes and the time-dependent edge weights are given priori. We assume both data and query objects lie on the network edges and all relevant information about the objects is maintained by a central server. We model the data objects as the network nodes. As a query object moves, the central server is updated with the new location of the object. Below, we formally define our terminology.

Definition 1. *Time-dependent Graph*
A Time-dependent Graph (G_T) is defined as $G_T(V, E)$ where $V = \{v_i\}$ is a set of nodes representing the intersections and terminal points, and E ($E \subseteq V \times V$) is a set of edges representing the network segments each connecting two nodes. Each edge e is represented by $e(v_i, v_j)$ where v_i and v_j are starting and ending nodes, respectively, and $v_i \neq v_j$. For every edge $e(v_i, v_j) \in E$, there is an edge travel-time function $c_{i,j}(t)$, where t is the time variable in time domain T. An edge travel-time function $c_{i,j}(t)$ specifies how much time it takes to travel from v_i to v_j starting at time t. For example, Figure 3 depicts a road network modeled as a time-dependent graph $G_T(V, E)$. Figure 3(a) shows the graph structure with five edges and corresponding time-dependent travel-times (as piece-wise linear functions) for each edge.

Definition 2. *Travel-Time*
Let $\{s = v_1, v_2, ..., v_k = d\}$ represent a path which contains a sequence of nodes where $e(v_i, v_{i+1}) \in E$, $i = 1, ..., k-1$. Given a time-dependent graph G_T, a path ($s \rightsquigarrow d$), and a departure-time from the source t_s, the time-dependent travel time $TT(s \rightsquigarrow d, t_s)$ is the time it takes to travel along the path. Since the travel-time of an edge varies depending on the arrival-time to that edge (i.e., arrival-dependency), the travel time is computed as follows:

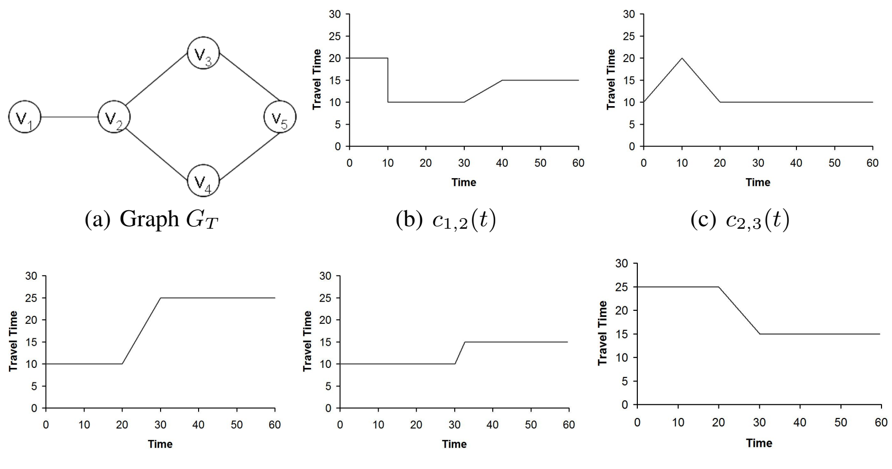
$$TT(s \rightsquigarrow d, t_s) = \sum_{i=1}^{k-1} c_{(v_i,v_{i+1})}(t_i) \text{ where } t_1 = t_s,\ t_{i+1} = t_i + c_{(v_i,v_{i+1})}(t_i),\ i = 1, .., k.$$

For example, in Figure 3 the travel-time of path $\{(v_1, v_2, v_3, v_5)\}$ with departure-time $t = 5$ is $TT(v_1 \rightsquigarrow v_5, 5) = 45$.

Definition 3. *Time-dependent Shortest (Fastest) Path*
Given a G_T, a source $s \in V$, a destination $d \in V$, and a departure-time t_s from the source, the time-dependent shortest path $TDSP(s, d, t_s)$ is a path with the minimum travel-time among all paths from s to d. Since we consider the travel-time between nodes as the distance measure to find the shortest path, we refer to $TDSP(s, d, t_s)$ as time-dependent fastest path $TDFP(s, d, t_s)$ and use them interchangeably in the rest of the paper. In a time-dependent graph, the fastest path from s to d changes based on the departure-time from the source. For instance, in Figure 3, suppose a query looking for the fastest path from node v_1 to v_5 for $t_s = 5$. In this case, $TDFP(v_1, v_5, 5) = \{v_1, v_2, v_3, v_5\}$. However, the same query will return $TDFP(v_1, v_5, 10) = \{v_1, v_2, v_4, v_5\}$ for $t_s = 10$. This is why all the methods assuming constant edge-weights and/or relying on distance pre-computation would fail to answer k nearest neighbors in time-dependent road networks. Obviously, with constant edge weights (i.e., time-independent), regardless of the query time the query would always return the same path (and path travel-time) as the result.

Definition 4. *Time-dependent k Nearest Neighbor Query (TD-kNN)*
A time-dependent k nearest neighbor query on spatial networks is defined as a query that finds the k nearest neighbors of a query object moving on a time-dependent network G_T. Considering a set of n data objects $P = \{p_1, p_2, ..., p_n\}$, the TD-kNN query with respect to a query point q finds a subset $P^{'} \subseteq P$ of k objects with minimum time-dependent travel-time to q, i.e., for any object $p^{'} \in P^{'}$ and $p \in P - P^{'}$, $TDFP(q, p^{'}, t) \leq TDFP(q, p, t)$.

In the rest of this paper, we assume that the edge travel-time functions are given as positive piece-wise linear functions of time and all piece-wise functions have a finite number of pieces. This is consistent with how traffic trends are reported for a given edge in real-world road networks. We also assume that the spatial network G_T satisfies the First-In-First-Out(FIFO) property [4]. This property suggests that moving objects exit from an edge in the same order they enter the edge. Finally, with our algorithm, we do not allow objects to wait at a node, because, in most real-world applications, waiting at a node is not realistic as it requires the moving object to get out of the current road (e.g., the exit freeway) and find a place to park and wait.



(a) Graph G_T (b) $c_{1,2}(t)$ (c) $c_{2,3}(t)$

(d) $c_{2,4}(t)$ (e) $c_{4,5}(t)$ (f) $c_{3,5}(t)$ change

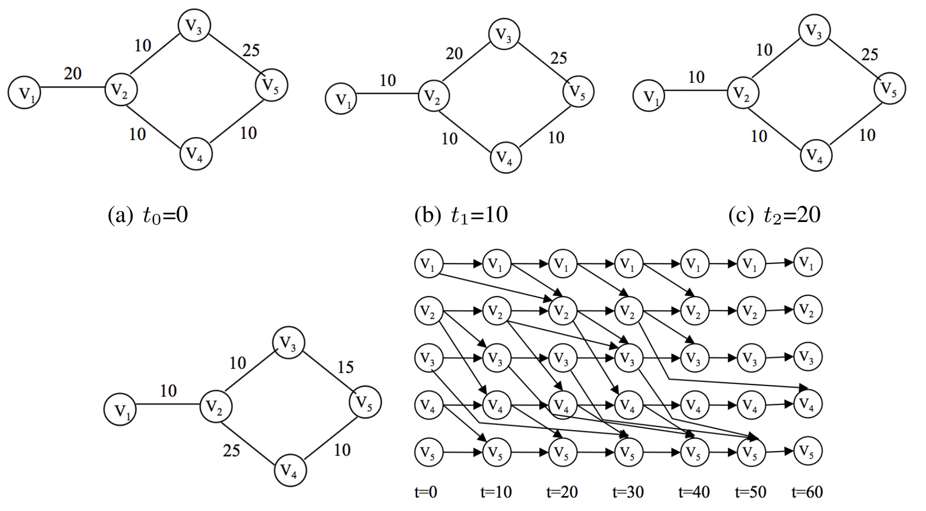
Fig. 3. A Time-dependent Graph $G_T(V, E)$

4 Baseline TD-KNN Algorithms

In this section, we explain two different algorithms to evaluate time-dependent k nearest neighbor queries in spatial networks. With the first approach, we model the time-dependent road network as a time-expanded graph [21] that approximates the time-dependent network with a snapshot of the network in each time interval. With the second approach, we exploit a generalization of incremental network expansion [22] method where we use time-dependent arrival times as the labels of the nodes to form the greedy search. While the former enables us to use existing nearest neighbor algorithms but with approximate results, the latter allows for obtaining exact results in FIFO time-dependent networks.

4.1 TD-kNN with Time-Expanded Networks

Given a time-dependent graph $G_T(V, E)$, a time-expanded model discretizes the time domain $T = [t_0; t_n]$ into n points of time, and constructs a static graph $G(V, E)$ by making n copies of each node and each edge, respectively. Specifically, time-expanded network replicates the original network for each discrete time unit $t = 0, 1, ..., t_n$, where t_n is determined by the total duration of the time interval under consideration. This model connects a node and its copy at the next instant in addition to the edges in the original network, replicated for every time instant. The weight of an edge in time-expanded network is the time difference between the time events associated with its endpoints. Therefore, a time-varying edge cost can be interpreted as a static flow in the corresponding time-expanded network. Figure 4 shows four consecutive snapshots and the corresponding time-expanded model of the time-dependent network in Figure 3. In this figure, for example, the weight (i.e., travel-time) of edge (v_1, v_2) at $t = 0$ is represented by connecting the copy of node v_1 at $t = 0$ to the copy of node v_2 at $t = 20$.



(a) t_0=0 (b) t_1=10 (c) t_2=20

(d) t_3=30 (e) Time-expanded model

Fig. 4. A Time-expanded graph

The time-expanded network approach enables time-dependent k nearest neighbor problem to be solved by applying techniques developed for static networks (e.g., INE). However, there are two drawbacks with any solution based on time-expanded networks. First, since the original network is replicated across time instants, the size of the network increases hence, resulting in high storage overhead and slower response time. The storage requirement for a time expanded-network is $O(|V|T) + O(|V| + |E|T)$, where T is the total number of snapshots. Second, the difference between the shortest path obtained using time-expanded model and the optimal shortest path is very sensitive to the parameter

n, and is unbounded. Because, the query time (or the arrival-time to an edge) can always be between any two time points (e.g., between t_0 and t_1), but the edge weights are only captured in either of the time points. For example, consider a shortest path query executed at $t = 12$ in Figure 4, and an error ϵ between the optimal path and the path found using time-expanded network model. In this case, the network snapshot at $t = 10$ is used to compute the shortest path for $t = 12$ and ϵ is accumulated on each edge along the path. Our experiments show that the error rate is especially high during rush hours (see 5.2), hence causing time-expanded models to generate inaccurate results.

4.2 TD-kNN with Network Expansion

In this section, we propose an algorithm that generalizes the incremental network expansion method [22] (originally proposed for static road networks) to answer k nearest neighbor queries in time-dependent road networks. With this algorithm, starting from the query object q all network nodes reachable from q in every direction are visited in order of their proximity (i.e., time-dependent travel-time) to q until all k nearest data objects are located. We use four main data structures to enable the network expansion solution. The *adjacency component* captures the network connectivity. The *edge component* includes the poly-line representation of each network edge (u, v), length of the edge, and a pair of pointers to the disk pages containing the adjacency lists of its endpoints u, v. The *travel-time component* includes the travel-time functions of each network edge. We use hash table to associate travel-time functions to network edges. The last component is R-tree [10] that indexes the edges' MBRs. Each leaf entry of R-tree contains a pointer to the disk page storing the corresponding edge.

We outline our proposed approach in Algorithm 1. The algorithm takes three parameters as the input, i.e., query location q, number of desired nearest neighbors k, and query time t_q. We first execute a $findEdge(q)$ operation to find the edge that contains q by performing a point location query on R-tree index. Then, we expand the network based on the time-dependent travel-time to each node around q until k objects are found. Specifically, we keep track of the arrival time at the end node of an edge e and use this arrival time to determine the (time-dependent) travel cost of the adjacent edges of edge e. In Algorithm 1, let $t(v)$ denote the time taken to travel from q to node v along the fastest path in a time-dependent network, i.e., time to travel for $TDFP(q, v, t_q)$. Analogous to shortest path distances, for every edge $e(u, v)$, we have $t(v) \leq f_e(t(u))$ where $f_e(t(u))$ is the time taken to travel from q to u plus the time travel from u to v (see Section 3 for time-dependent travel-time computation). The core idea (as initially shown in [8, 4]) with this algorithm is that, analogous to shortest path distances, we now have $t(v) \leq f_e(t(u))$ and this will form the basis of our greedy algorithm. We use a priority queue S to keep track of the nodes to be examined. With S, we maintain the set *explored* nodes (which includes the nodes for which we have calculated the actual $t(v)$) as well as a label $l(v)$ for each node not in S, where the label is our current estimate for the least time it takes to reach v from s. We update $l(v)$ by $min(l(v), f_e(t(u)))$ (Line 8) only considering the edges (u, v) where $u \in S$. Finally, we pick the node $w \notin S$ with smallest $l(.)$ value and add it to the set S. If the recently added node is a data object (recall that data objects are modeled as network nodes), we add that data object to nearest neighbor array NN and accordingly compute its travel time (Line 11-13). This process is repeated until the algorithm finds k data objects. It is important to note that Algorithm 1 holds for FIFO networks in which greedy property is maintained.

Algorithm 1 TDFP(q,k,t_q)

```
1:  // $S$: set of nodes, $q$: query location, $dt$: departure-time from $q$
2:  // $tt$: travel-time of the fastest path, $v_i$: last node added to $S$
3:  // $NN$: array of current NNs
4:  Initialize $S = \{q\}, t(q) = 0, l(v) = \infty$ for all $v \notin S$
5:  $v_i = q$
6:  **while** $S \neq V$ **do**
7:      $for\ each\ e(v_i, v_j) \in E\ where\ v_j \notin S$
8:      $l(v_j) = min(l(v_j), f_e(t(v_i)))$
9:      $Let\ w \notin S\ such\ that\ l(w) = min_{v_j \notin S} l(v_j)$
10:     $S = S \bigcup \{w\}, t(w) = l(w), v_i = w$
11:     If $v_i\ is\ a\ dataObject$ Then
12:     $add\ v_i\ to\ NN$;
13:     $tt = t(v_i) - t_q$; //compute travel-time to $v_i$
14:     End If;
15:     If $NN.size() = k$ Then break;
16: **end while**
17: $return\ NN$ and travel times
```

5 Experimental Evaluation

5.1 Experimental Setup

We conducted several experiments with different spatial networks and various parameters (see Table 1) to evaluate the performance of both TD-kNN algorithms. As our dataset, we used Los Angeles (LA) and San Joaquin County (SJ) road network data with 304,162 and 24,123 road segments, respectively. We obtained these datasets from TIGER/Line [1]. Both of these datasets fit in the memory of a typical machine with 4GB of memory space.

To create realistic time-dependent edge weights, it is necessary to collect huge amounts of data about the network edge behaviors. Towards that end, for the past 1 year, we have been continuously collecting speed, occupancy, volume sensor data from a collection of approximately 6000 sensors located on the road network of Los Angeles County. The sampling rate of the data is 1 reading/sensor/min. Currently, our database consists of about 750 million sensor reading representing speed profiles on the road network segments of LA. We used the historical sensor data to create travel-time functions for LA network. In order to create the time-dependent edge weights of SJ network, we developed a traffic model [6] that synthetically generates time-dependent edge weights for SJ.

We generated the parameters represented in Table 1 using a simulator prototype developed in Java. We conducted our experiments on a workstation with 2.7 GHz Pentium Core Duo processor and 12GB RAM memory.

We computed the time-expanded network model of both LA and SJ networks by discritizing the networks for each 15 minutes. Similar to Algorithm 1 (denoted by TD-NE), we implemented a network expansion method to find k nearest neighbors in time-expanded networks (denoted by TE). We continuously monitored each query for 50 timestamps in both of the implementations. For each set of experiments, we only vary one parameter and fix the remaining to the default values in Table 1.

[1] http://www.census.gov/geo/www/

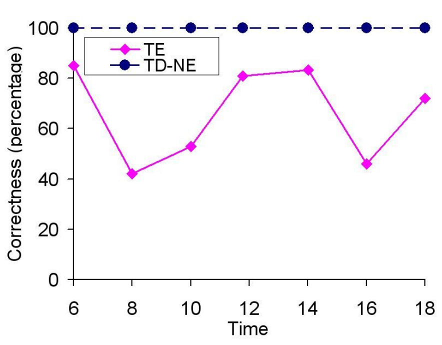
Table 1. Experimental parameters

Parameters	Default	Range
Number of objects	10 (K)	1,5,10,15,20(K)
Number of queries	3 (K)	1,2,3,4,5 (K)
Number of k	20	1,10,20,30,40,50
Object Distribution	Uniform	Uniform, Gaussian
Query Distribution	Uniform	Uniform, Gaussian

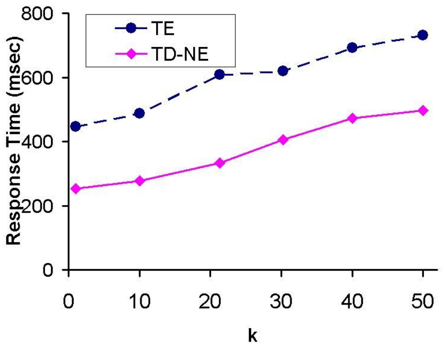
Since the experimental results with both LA and SJ networks differ insignificantly and due to space limitations, we only present the results from LA dataset.

5.2 Results

Correctness and Impact of k With this experiment, we compare the correctness of the two algorithms (i.e., percentage of correctly identified nearest neighbors). Figure 5(a) plots the correctness versus time ranging from 6 am to 6 pm, while using default settings in Table 1 for all other parameters. As shown, while TD-NE returns correct results all the time, TE's correctness is substantially low around rush hours (i.e., 7-9 am, 4-6 pm). This is because time-dependent weights of each network segment change rapidly especially at the boundaries of the traffic peak periods, resulting the error accumulating along the path.



(a) Correctness versus time



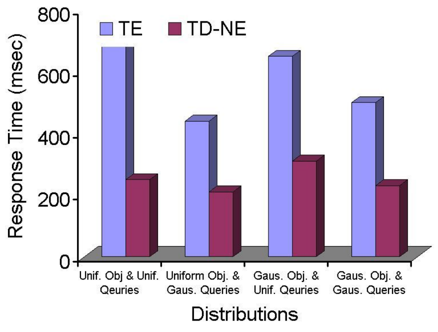
(b) Impact of k on response time

Fig. 5. Correctness and impact of k

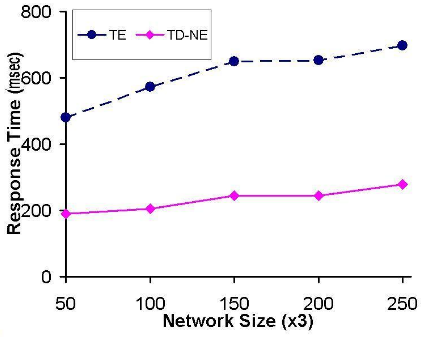
Next, we compare the performance of the two algorithms with regard to k. Figure 5(b) shows the average query efficiency versus k ranging from 1 to 50. The results indicate that TD-NE outperforms TE with all values of k and the response time of both algorithms increases with the large values of k. Note that the slower response time of TE in this and the following experiments is due to increased size of the network because of replication. One can implement pre-computation techniques to accelerate the response time of TE.

Impact of Object/Query Distribution and Network Size With this experiment, we study the impact of object and query distribution as well as network size. Figure 6 shows the response time of both algorithms where the objects and queries follow either uniform or Gaussian distributions. As illustrated, TD-NE yields better performance for queries with Gaussian distribution. This is because as queries are clustered in the spatial network with Gaussian distribution, their nearest neighbor would overlap; hence, allowing TD-NE to save computation.

In addition, we measured the performance of both algorithms with respect to the network size. In order to evaluate the impact of network size, we conducted experiments with the sub-networks of LA dataset ranging from 50K to 250K segments. Figure 6(b) illustrates the response time of both algorithms with different network sizes. In general, with the default parameters in the Table 1, the response time increases for both algorithms as the network size increases.



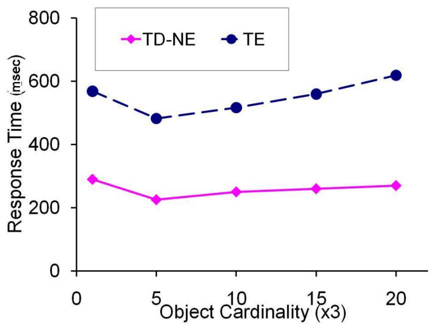
(a) Impact of object distribution



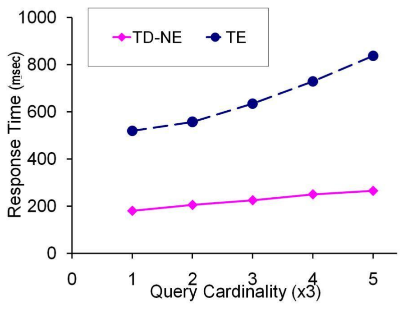
(b) Impact of network size

Fig. 6. Response time versus distribution and network size

Impact of Object and Query Cardinality With this set of experiments, we compare the performance of the two algorithms by varying the cardinality of the data objects (P) from 1K to 20K while using default settings in Table 1 for all other parameters. Figure 7(a) illustrates the impact of the growing object cardinality on response time. The results indicate that the response time linearly increases with the number of data objects in both methods, where TD-NE outperforms TE for all cases. From P=1K to 5K, the response time is slower. Because, since the objects are sparsely distributed when P is small, network expansion visits more redundant network nodes causing extra processing time. Figure 7(b) shows the impact of the query cardinality (Q), ranging from 1K to 5K, on response time. As shown, TD-NE scales better with large number of Q and the performance gap between the approaches increases as Q grows.



(a) Impact of object cardinality



(b) Impact of query cardinality

Fig. 7. Response time versus query/object cardinality and agility

6 Conclusion and Future Work

In this paper, for the first time we studied the problem of time-dependent k nearest neighbor search (TD-kNN) in spatial networks. We formulated a generalized type of k nearest neighbor query where we, unlike the existing studies, assume the edge weights of the network are time varying rather than fixed. We studied two baseline solutions exploiting time-expanded network and network expansion frameworks and compared their efficiency with real-world data-sets, including a variety of large spatial networks with real traffic-data. Although time-expanded network framework provides a mechanism to use existing kNN algorithms for static networks, the experimental results suggest that the error rate (incorrectly identified nearest neighbors) of this approach is very high especially during traffic peak hours. On the other hand, while network expansion yields correct results at all times, the overhead of executing network expansion is very high particularly in large networks with a sparse set of data objects, hence the need for efficient time-dependent search algorithms.

We intend to pursue this study in three different directions. First, we intend to investigate new data models for effective representation of spatiotemporal road networks. This is critical in supporting development of efficient and accurate time-dependent algorithms (e.g., shortest path), while minimizing the storage and cost of the computation. Second, given that online nearest neighbor queries require near real-time response time, we plan to develop novel preprocessing and indexing techniques that can be used to accelerate kNN computation in time-dependent networks. Third, we plan to study a variety of other spatial queries (including range queries and skyline queries) in time-dependent road networks.

Given the importance of time-dependency for accurate and realistic spatial query processing in road networks, as well as increasing use of traffic sensors and, hence; availability of time-varying traffic data, we predict rapid growth of interest (at academia and industry) in developing various query processing solutions for spatial queries in time-dependent road networks.

References

1. I. Chabini. The discrete-time dynamic shortest path problem: Complexity, algorithms, and implementations. In *Transportation Research Record*, 1999.
2. H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *VLDB*, 2005.
3. L. Cooke and E. Halsey. The shortest route through a network with timedependent internodal transit times. In *Journal of Mathematical Analysis and Applications*, 1966.
4. B. C. Dean. Algorithms for minimum cost paths in time-dependent networks. In *Networks*, 1999.
5. U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. Efficient continuous nearest neighbor query in spatial networks using euclidean restriction. In *SSTD*, 2009.
6. U. Demiryurek, B. Pan, F. B. Kashani, and C. Shahabi. Towards modeling the traffic data on road networks. In *GIS-IWCTS*, 2009.
7. B. Ding, J. X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *EDBT*, 2008.
8. S. E. Dreyfus. An appraisal of some shortest-path algorithms. In *Operations Research Vol. 17, No. 3*, 1969.
9. B. George, S. Kim, and S. Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. In *SSTD*, 2007.
10. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
11. J. Halpern. Shortest route with time dependent length of edges and limited delay possibilities in nodes. In *Mathematical Methods of Operations Research*, 1969.
12. X. Huang, C. S. Jensen, and S. Saltenis. The island approach to nearest neighbor querying in spatial networks. In *SSTD*, 2005.
13. X. Huang, C. S. Jensen, and S. Saltenis. S-grid: A versatile approach to efficient query processing in spatial networks. In *SSTD*, 2007.
14. Inrix. http://www.inrix.com. Last visited January 2, 2010.
15. E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding fastest paths on a road network with speed patterns. In *ICDE*, 2006.
16. M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, 2004.
17. U. Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *Geoinformation and Mobilitat*, 2004.
18. K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, 2006.
19. Navteq. http://www.navteq.com. Last visited January 2, 2010.
20. A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *J. ACM*, 1990.
21. S. Pallottino and M. G. Scutell. Shortest path algorithms in transportation models: Classical and innovative aspects. In *Equilibrium and Advanced Transportation Modelling*, 1998.
22. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, 2003.
23. H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *SIGMOD*, 2008.
24. D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. *Algorithms-ESA*, 2003.