Towards K-Nearest Neighbor Search in Time-Dependent Spatial Network Databases by U. Demiryurek, F. Banaei-Kashani, and C. Shahabi (*Databases in Networked Information Systems*, Lecture Notes in Computer Science, 2010)

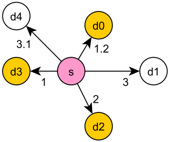
presented by
Jedidiah R. McClurg

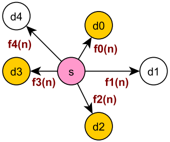
Northwestern University

November 28, 2011

Introduction

- The k-Nearest Neighbor problem plays a very important role in sensor network applications

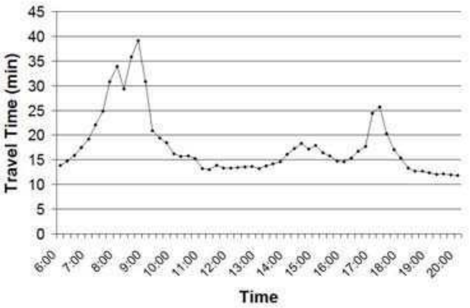




- The aforementioned paper [1] seeks to generalize this problem for networks where distances may change throughout time
- This talk will cover the paper in the following order:
  1. Background/Related Work
  2. Formal Preliminaries
  3. Proposed Algorithms
  4. Experimental Results
  5. Conclusion

Motivation

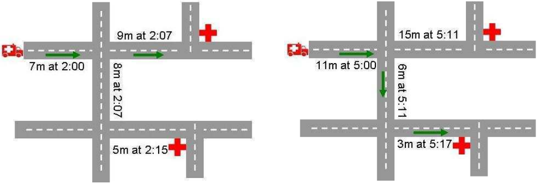
- How is k-NN useful?
  - Google Maps, GPS navigation systems
- Why do we need time-dependent k-NN?
  - Travel time ("distance") is not static:
    (Weekday Travel Time on I-405 in Los Angeles)



  - We can readily obtain such time-dependent sensor data

Using Time-Dependent k-NN

- Here is an example in which we can use TD-kNN:



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

- Can we simply store one of these snapshots for each t_s?
  - The data is continuous, so this would require excessive storage
- Given a time t_s, can we just reload the edge weights and use regular k-NN on the resulting graph?
  - This would be very slow if the network is large
- The paper presents two TD-kNN algorithms which attempt to circumvent these limitations

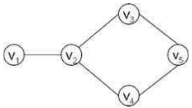
Some Related Work

- Related work regarding KNN queries
  - Various adaptations of Dijkstra's Algorithm, such as Incremental Network Expansion [3]
  - All of these rely on static edge weights
- Related work regarding time-dependent shortest path (TDSP)
  - Problem shown to be NP-Hard in non-FIFO networks
  - Belman-Ford adapted to TDSP with piecewise-linear functions as edge weights [2]
- These results can be used to build TD-kNN algorithms if we adopt the FIFO and piecewise linearity restrictions

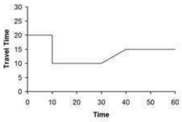
Preliminaries - Time-Dependent Graph

Definition (Time-Dependent Graph)

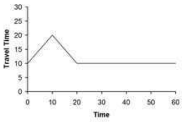
A Time-Dependent Graph is a directed graph $G_T(V, E)$ in which vertices represent the network nodes and edges represent the node connections. For each edge (v_i, v_j), there is a travel time function $c_{i,j}(t)$ which represents the time to travel from v_i to v_j starting at time t.



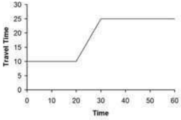
(a) Graph G_T



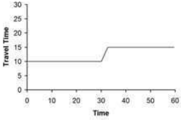
(b) $c_{1,2}(t)$



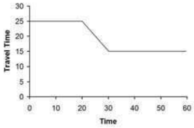
(c) $c_{2,3}(t)$



(d) $c_{2,4}(t)$



(e) $c_{4,5}(t)$



(f) $c_{3,5}(t)$ change

Preliminaries - Travel Time

Definition (Travel Time)

Let $s \rightsquigarrow d$ denote a path, i.e. a sequence of nodes $v_1, v_2, \cdots, v_k$ such that $s = v_1, d = v_k$ and $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$. The Travel Time from s to d starting at t_s is then denoted as $tt(s \rightsquigarrow d, t_s)$.

We can see that the Travel Time can be calculated as follows:

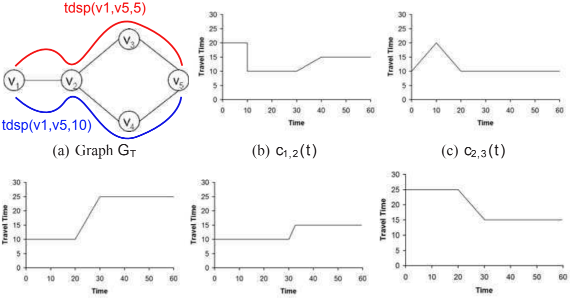
$$tt(s \rightsquigarrow d, t_s) = \sum_{i=1}^{k-1} c_{i,j}(t_i)$$

where $t_1 = t_s$ and $t_{i+1} = t_i + c_{i,j}(t_i)$.

Preliminaries - Time-Dependent Shortest Path

Definition (Time-Dependent Shortest Path)

The Shortest Path between nodes s and d starting at time t_s is denoted $tdsp(s, d, t_s)$. Since Travel Time is our distance metric, the Shortest Path is defined as the path with the smallest Travel Time between s and d starting at t_s.



(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

Preliminaries - Time-Dependent k-NN Query

Definition (Time-Dependent k-Nearest Neighbor Query)

A Time-Dependent k-NN Query with respect to a node s at time t is one which finds the set of k closest neighboring nodes. That is, given a node s, the Time-Dependent k-NN Query will return a set P such that

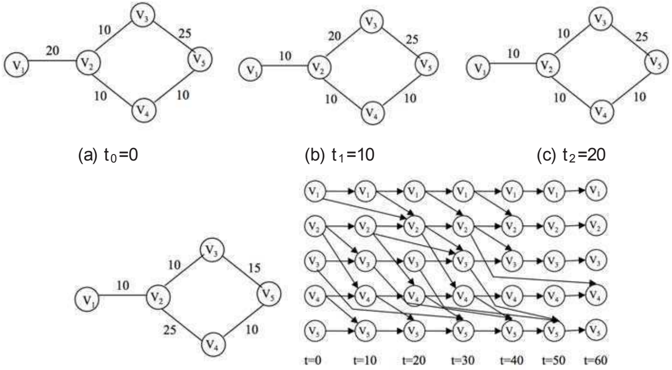
1. $|P| = k$
2. $tdsp(s, p, t) \leq tdsp(s, q, t)$ *for all* $p \in P, q \notin P$

TD-KNN with Time-Expanded Networks (Algorithm 1)

- Earlier, we saw that these two naive ideas will not work:
  - Create a copy of the network for each time
  - Recompute the edge weights on demand
- We can attempt to balance runtime and space usage by making these two ideas work together
- Algorithm 1 subdivides the time domain into n equally-spaced instants and constructs the graph of $n+1$ appropriately connected copies of the nodes of G_T
- This results in a bounded graph that can be searched using k-NN methods such as Incremental Network Expansion

Algorithm 1 (TE) Details

This diagram shows the time-expanded graph generated for $n = 6$:



(a) t_0=0 (b) t_1=10 (c) t_2=20 (d) t_3=30 (e) Time-expanded model

To execute a TD-kNN query with respect to node v_i at time t, we find the closest time instant, and do a k-NN search starting at the copy of v_i in that instant.

Analysis of Algorithm 1 (TE)

- If the query time t does not coincide exactly with one of the time instants, we will have an error ϵ that will propagate through the search
- This error is especially noticeable in the Results
- The storage requirement will be $O(|G_T| \cdot N)$ where N is the number of time instants

TD-KNN with Network Expansion (Algorithm 2)

- This approach adapts the Incremental Network Expansion method to handle TD-kNN queries starting at q
- It maintains a set S of explored nodes v_j which have their minimum distance from q (denoted by $t(v_j)$) correct
- After each iteration it picks the v_j adjacent to S with minimal $l(v_j)$ as the new v_i to add to S
- The labels on unexplored nodes $l(v_j)$ are relaxed with $min(l(v_j), f(v_i, v_j))$, where $f(v_i, v_j) = tt(q \rightsquigarrow v_i, t_q) + c_{i,j}(t(v_i))$

Algorithm 2 (TD-NE) Details

The following algorithm gives the specifics of the preceding idea:

TD-kNN(q, k, t_q)

$NN \leftarrow \emptyset; \quad S \leftarrow \{q\}; \quad t(q) \leftarrow 0; \quad l(v) \leftarrow \infty$ for all $v \notin S$
$v_i \leftarrow q$
$tt(q \rightsquigarrow v_i, t_q) \leftarrow 0$
while $|NN| < k$ **do**
  for all $v_j \notin S$ **do**
    $l(v_j) = min(l(v_j), f(v_i, v_j))$
  end for
  $v_i \leftarrow v_j \notin S$ such that $l(v_j)$ is minimal
  $S \leftarrow S \cup \{v_i\}; \quad t(v_i) \leftarrow l(v_i)$
  $NN \leftarrow NN \cup \{v_i\}$
  $tt(q \rightsquigarrow v_i, t_q) \leftarrow t(v_i)$
end while
return NN

Analysis of Algorithm 2 (TD-NE)

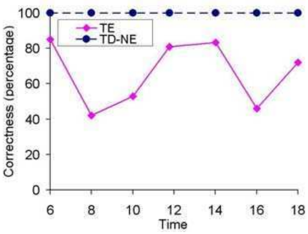
- Unlike Algorithm 1, this algorithm returns exact rather than approximate results
- The storage requirement is greatly reduced, since the algorithm does not generate multiple copies of the network

Experimental Setup

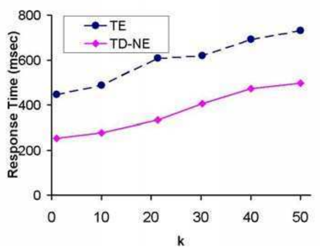
- Road network topology for Los Angeles County was obtained from the U.S. Census Bureau geography databases
- Traffic sensor data (over the period of 1 year) for these roads was utilized to create edge weight functions
- The resulting model was then loaded into a Java-based simulator running on a desktop workstation
- The simulator has the ability to vary the number of objects, the number of queries, and the k value independently
- Objects are represented as specially-marked nodes in the simulated network

Correctness and Impact of k

- For the correctness experiment, 10K objects and 3K queries were uniformly distributed throughout the network, and a query size of $k = 20$ was used
- For the experiment regarding k, the simulator used the preceding parameters, but varied k between 1 and 50



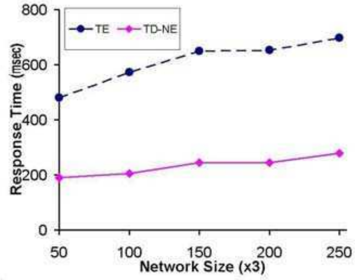
(a) Correctness versus time



(b) Impact of k on response time

Impact of Network Size

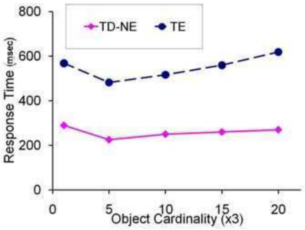
For the experiment regarding network size, the simulator used the preceding parameters, but varied the network size (50K - 250K segments) by examining subsets of the LA road model



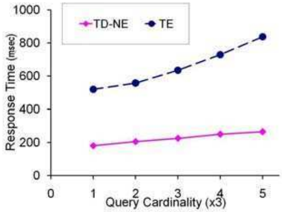
(b) Impact of network size

Impact of Object/Query Cardinality

- For the object cardinality experiment, the preceding parameters were used while varying the number of objects between 1K and 20K
- For the query cardinality experiment, the number of queries was varied between 1K and 5K



(a) Impact of object cardinality



(b) Impact of query cardinality

Conclusion

- Summary of the paper
  - This paper formalizes the notion of TD-kNN queries
  - It presents two algorithms for performing such queries
  - Algorithm 1 (TE) shows significant incorrectness, especially during rush hours
  - Algorithm 2 (TD-NE) shows a performance increase over TE, and demonstrates 100% correctness
  - TD-kNN techniques will help to improve the accuracy of navigation systems
- Future work
  - New data models for representing spatiotemporal networks
  - Preprocessing/indexing to reduce response time in query processing
  - Other spatial queries (ranges, etc.)

Thanks!

Questions/comments?

Bibliography

- Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi.
  Towards k-nearest neighbor search in time-dependent spatial network databases.
  In *Databases in Networked Information Systems*, volume 5999 of *Lecture Notes in Computer Science*, pages 296–310. Springer, 2010.

- Ariel Orda and Raphael Rom.
  Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length.
  J. ACM, 37:607–625, July 1990.

- Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao.
  Query processing in spatial network databases.
  In *Proceedings of VLDB - Volume 29*, VLDB '2003, pages 802–813. VLDB Endowment, 2003.