JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 4, pp. 63-74, April 2021
https://doi.org/10.9708/jksci.2021.26.04.063

A Batch Processing Algorithm for Moving *k*-Nearest Neighbor Queries in Dynamic Spatial Networks

Hyung-Ju Cho*

*Professor, Dept. of Software, Kyungpook National University, Sangju, Korea

[Abstract]

Location-based services (LBSs) are expected to process a large number of spatial queries, such as shortest path and *k*-nearest neighbor queries that arrive simultaneously at peak periods. Deploying more LBS servers to process these simultaneous spatial queries is a potential solution. However, this significantly increases service operating costs. Recently, batch processing solutions have been proposed to process a set of queries using shareable computation. In this study, we investigate the problem of batch processing moving *k*-nearest neighbor (M*k*NN) queries in dynamic spatial networks, where the travel time of each road segment changes frequently based on the traffic conditions. LBS servers based on one-query-at-a-time processing often fail to process simultaneous M*k*NN queries because of the significant number of redundant computations. We aim to improve the efficiency algorithmically by processing M*k*NN queries in batches and reusing sharable computations. Extensive evaluation using real-world roadmaps shows the superiority of our solution compared with state-of-the-art methods.

▸ **Key words**: Spatial databases, Moving *k*-nearest neighbor query, Batch processing, Dynamic spatial network

[요 약]

위치 기반 서비스(LBS)는 가장 바쁜 시간에 동시에 도착하는 최단 경로 및 *k*-최근접 이웃 질의를 포함한 다양한 공간 질의를 효과적으로 처리한다. 동시에 도착하는 공간 질의를 빠르게 처리하기 위한 간단한 해결 방법은 LBS 서버를 추가하는 것이다. 이 방법은 서비스 운영 비용을 많이 증가시킨다. 최근에는 공유 가능한 계산을 사용하여 일련의 질의를 한꺼번에 모아서 처리하는 일괄 처리 방법이 제안되었다. 본 연구에서는 교통 상황에 따라 각 도로 구간의 이동 시간이 빈번하게 변하는 동적 공간 네트워크에서 움직이는 *k*-최근접 이웃 질의를 한꺼번에 처리하는 방법을 연구한다. 순차적 질의 처리를 기반으로 하는 LBS 서버는 중복 계산으로 인해 한꺼번에 요청이 들어오는 움직이는 *k*-최근접 이웃 질의를 효과적으로 처리하지 못한다. 본 연구의 목표는 움직이는 *k*-최근접 이웃 질의를 한꺼번에 처리하고 공유 가능한 계산을 재사용하여 알고리즘을 효율성을 개선한다. 실제 지도 데이터를 사용한 실험 평가는 최신 방법보다 제안된 방법이 우수하다는 것을 보여준다.

▸ **주제어**: 공간 데이터베이스, 움직이는 *k*-최근접 이웃 질의, 일괄 처리, 동적 공간 네트워크

• First Author: Hyung-Ju Cho, Corresponding Author: Hyung-Ju Cho
*Hyung-Ju Cho (hyungju@knu.ac.kr), Dept. of Software, Kyungpook National University
• Received: 2021. 02. 18, Revised: 2021. 03. 31, Accepted: 2021. 03. 31.

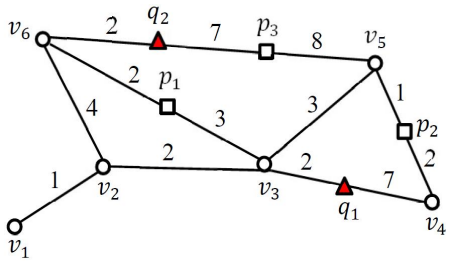
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

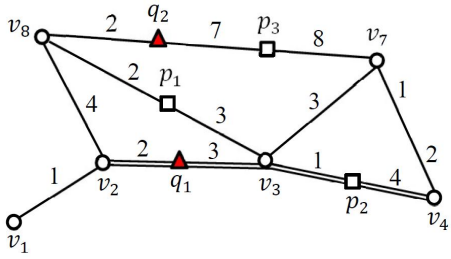
Currently, location-based services (LBSs), such as taxi-calling and ridesharing services, utilize real-time spatial data to find k points of interest (POI) closest to a query point based on the length of the shortest path from the query point to the POI. For example, a taxi client wishes to be served by available taxicabs that can reach them quickly. LBS servers based on one-query-at-a-time processing often fail to process a large number of simultaneous spatial queries reaching the servers at the peak time. Hence, batch processing algorithms have been introduced to address this critical problem in LBSs [1, 2].

Here, we investigate the batch processing of moving k-nearest neighbor (MkNN) queries in dynamic spatial networks, where the travel time for each road segment changes frequently based on the traffic conditions such as the traffic volume and accidents. MkNN queries in a dynamic spatial network have many potential applications for LBSs, such as ride-hailing and car parks. For example, 14 million Uber trips for ridesharing were completed each day in 2019, demonstrating the significance of scalable and efficient solutions to promptly match Uber cabs with passengers. Another example is real-time parking management, which helps drivers find parking spaces nearest to them. It is often difficult for drivers to find available parking spaces when they reach their destinations.

Figure 1 shows two snapshots at timestamps t_i and t_j of a dynamic spatial network, where a set Q of moving query points and a set P of moving data points are expressed as $Q=\{q_1,q_2\}$ and $P=\{p_1,p_2,p_3\}$, respectively. Note that for the convenience of presentation, two road segments $\overline{v_2v_3}$ and $\overline{v_3v_4}$ are identified using a double solid line to represent changes in the travel time of these road segments, as shown in Figure 1(b). In Figure 1(a), data point p_1 is closest to both q_1 and q_2 at timestamp t_i. However, in Figure 1(b), data point p_1 (p_2) is closest to q_2 (q_1) at timestamp t_j. A simple solution for MkNN queries uses a one-query-at-a-time method, which computes k data points that are closest to each query point in Q sequentially. This solution introduces a prohibitive overhead because of redundant network traversal for adjacent query points, despite utilizing efficient kNN search algorithms [3, 4, 5, 6] for retrieving a set of k data points closest to the query point.



(a) Snapshot of query and data points at timestamp t_i



(b) Snapshot of query and data points at timestamp t_j

Fig. 1. Example of MkNN queries in a dynamic spatial network

All nearest neighbor (ANN) queries [7] are similar to MkNN queries. However, ANN queries retrieve only one data point closest to each query point q in Q, indicating $k=1$ for each $q\in Q$. Contrarily, MkNN queries retrieve a different number of k data points closest to each query point q. Furthermore, we consider a highly dynamic situation where both the query and data points move freely in dynamic spatial networks. Herein, we propose an efficient algorithm known as BANK for the batch processing of MkNN queries in dynamic spatial networks. The BANK algorithm first groups adjacent query points into a query group and performs batch computation for the query group to avoid

redundant network traversal. To our study, the batch computation approach has not been applied to M*k*NN queries in dynamic spatial networks; however, the batch computation of spatial queries has received significant attention.

The primary contributions of this study are listed as follows:

- We propose an efficient algorithm called BANK for the batch processing of M*k*NN queries in dynamic spatial networks. To our study, the BANK algorithm is the first to consider the batch processing of M*k*NN queries in dynamic spatial networks.
- We present group computation techniques to avoid the redundant computation of network distances for adjacent query points. Furthermore, we present a theoretical analysis to prove the advantage of the BANK algorithm over one-query-at-a-time methods.
- We conduct extensive experiments using real-world roadmaps to demonstrate the efficiency of the proposed solution.

The remainder of this paper is organized as follows. In Section II, we review related studies and introduce the background of the study. In Section III, we explain the method for clustering adjacent query points into a query group and present the BANK algorithm for the batch processing of M*k*NN queries in dynamic spatial networks. In Section IV, we compare the BANK algorithm and its conventional solutions with different setups. Conclusions are presented in Section V.

II. Preliminaries

1. Related works

Nearest neighbor (NN) queries have been investigated extensively in spatial networks. NN query processing for spatial networks involves a high cost for computing the length of the shortest path between two points, in which graph traversal may be required. Studies regarding NN queries in spatial networks have presented various techniques to reduce the shortest-path-distance computation. Papadias *et al.* [4] introduced the incremental Euclidean restriction (IER) and incremental network expansion (INE). IER is based on the assumption that the length of the shortest path between two points cannot be less than their Euclidean distance. INE involves network expansion from the query point in a manner similar to Dijkstra's algorithm and examines the data points in the sequence encountered. The distance browsing (DisBrw) algorithm [8] uses the spatially induced linkage cognizance index, which stores the shortest path distance between every pair of vertices. The route overlay and association directory (ROAD) [3] algorithm hierarchically partitions the spatial network and precomputes the shortest path distance between border vertices within each partition, where border vertices of a partition are the vertices connecting to other partitions. The G-tree [6] partitions the spatial network; however, it differs from the ROAD in terms of the tree structure and searching paradigm. The V-tree [5] employs a hierarchical structure similar to that of the G-tree; it identifies border nodes at the boundaries of subgraphs. Efficient techniques are used to answer *k*NN queries by maintaining the lists of data points closest to the border nodes. Abeywickrama *et al.* [9] performed a thorough experimental evaluation of several *k*NN search algorithms for spatial networks, including G-tree [6], IER [4], INE [4], DisBrw [8], and ROAD [3]. Cao *et al.* [10] proposed a scalable in-memory processing method to answer snapshot *k*NN queries over moving objects in a spatial network. Unfortunately, existing solutions in [9, 11, 12, 13] focused on improving the efficiency of a *k*NN query, and are referred to as one-query-at-a-time solutions for *k*NN queries.

ANN queries were investigated in [7]. Unlike M*k*NN queries, ANN queries stipulate that every query point q in Q retrieves only one data point closest to q, which means $k=1$. Most studies regarding ANN queries have been conducted in

Euclidean spaces. Several previous studies have solved the continuous *k*NN query problem in spatial networks [14, 15, 16]. Some models [14] have assumed moving query points and stationary data points. However, the models used in [15, 16] assumed the opposite. These studies are orthogonal to ours and focus on the efficient maintenance of *k*NN results. The current study considers multiple snapshot *k*NN queries such as Uber taxi services, where query and data points correspond to passengers and taxicabs, respectively, and both freely move along a dynamic spatial network.

2. Background

Definition 1. (*k*NN query) For a positive integer k, query point q, and set of data points P, the *k*NN query retrieves a set $P_k(q)$ of k data points in P that are closest to q, $dist(q,p^+) \leq dist(q,p^-)$ for $p^+ \in P_k(q)$ and $p^- \in P - P_k(q)$.

Definition 2. (M*k*NN query) For a set of query points Q, the M*k*NN query retrieves set $P_k(q)$ of k data points closest to each query point q in Q. When query point q_i (q_j) retrieves k_i (k_j) data points closest to q_i (q_j), the k_i value may differ from the k_j value for $i \neq j$ and $1 \leq i,j \leq |Q|$. For simplicity, we assume that each query point, q, requires the same number of k data points closest to q. However, it is not difficult to consider a different number of k data points closest to the query point, q, which is discussed in Section III.2.

Definition 3. (Spatial network) A dynamic spatial network can be described as a dynamic weighted graph $G = \langle V, E, W \rangle$, where V, E, and W indicate the vertex set, edge set, and edge distance matrix, respectively. Each edge $\overline{v_i v_j}$ has a non-negative weight representing the network distance, such as the travel time, and frequent changes in its weight.

Definition 4. (Intersection, intermediate, and terminal vertices) We categorize vertices into three categories based on their degree. (1) If the degree of a vertex is greater than or equal to three, then the vertex is an intersection vertex. (2) If the degree is two, then it is an intermediate vertex. (3) If the degree is one, then it is a terminal vertex.

The symbols and notations used in this study are listed in Table 1. To simplify the presentation, we denote $\overline{q_i q_{i+1} \cdots q_j}$ by $\overline{q_i q_j}$, where query points $q_i, q_{i+1}, \cdots, q_j$ are located in the same vertex sequence.

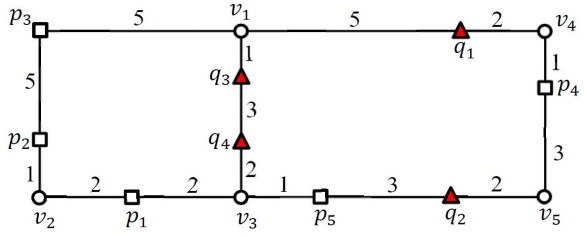
Table 1. Definitions of symbols

Symbol	Definition
k	Number of requested NNs
q	Query point
p	Data point
Q	Set of query points
P	Set of data points
$P_k(q)$	Set of k data points closest to a query point
$P(\overline{\alpha\beta})$	Set of data points that belong to a segment
$dist(p,q)$	Length of the shortest path connecting two points p and q in the spatial network
len(*p*,*q*)	Length of the segment connecting two points p and q such that both p and q are in the same vertex sequence
$\overline{v_l v_{l+1} \cdots v_m}$	Vertex sequence where v_l and v_m are not intermediate vertices, and the other vertices, $v_{l+1}, \cdots, v_{m-1}$, are intermediate vertices
$\overline{q_i q_{i+1} \cdots q_j}$	Query segment comprising query points $q_i, q_{i+1}, \cdots, q_j$ in a vertex sequence (in short, $\overline{q_i q_j}$)
adj_seqs(*v*)	Number of query segments adjacent to a vertex v

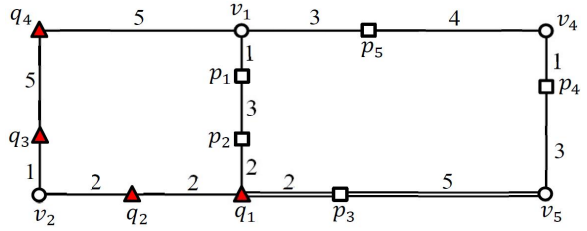
III. The Proposed Scheme

1. Grouping adjacent query points

In this section, we consider an M*k*NN query in a spatial network, as shown in Figure 2. For $Q = \{q_1, q_2, q_3, q_4\}$, and $P = \{p_1, p_2, p_3, p_4, p_5\}$, we consider a *k*NN query that retrieves data points closest to each query point q in Q. For simplicity, we assume that q_1, q_2, q_3, and q_4 request one, two, one, and two data points closest to them, respectively, which means that $k_1 = k_3 = 1$ and $k_2 = k_4 = 2$.



(a) Distribution of query and data points at t_1



(b) Distribution of query and data points at t_2

Fig. 2. Population of query and data points at timestamps t_1 and t_2

Figure 2 shows the population of the query and data points at timestamps t_1 and t_2. Here, we assume that both the query and data points move arbitrarily along the spatial network. In this section, we focus on evaluating M*k*NN queries at timestamp, t_1, in Figure 2(a).

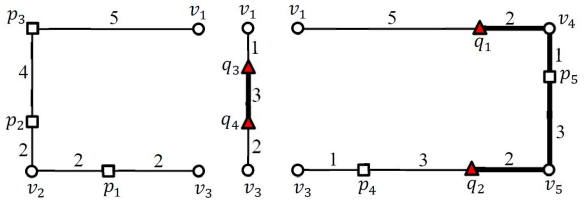


Fig. 3. Grouping of adjacent query points into query segments

Figure 3 illustrates a sample grouping of adjacent query points. Two query points, q_1 and q_2, in a vertex sequence, $\overline{v_1v_4v_5v_3}$, are transformed into a query segment, $\overline{q_1q_2}$, and the other two query points, q_3 and q_4, in a vertex sequence, $\overline{v_1v_3}$, are grouped into another query segment, $\overline{q_3q_4}$. Therefore, a set of query points, $Q=\{q_1,q_2,q_3,q_4\}$ can be transformed into a set of query groups, $\overline{Q}=\{\overline{q_1q_2},\overline{q_3q_4}\}$.

2. BANK algorithm

Algorithm 1 describes the BANK algorithm for the M*k*NN search over spatial networks. The result set $\Pi(Q)$ is initialized to an empty set (line 1). In the first step (lines 2−3), adjacent query points $q_i,q_{i+1},\cdots,q_j$ in the same vertex sequence are grouped into a query segment $\overline{q_iq_j}$. Therefore, a set Q of query points is converted into a set $\overline{Q}$ of query groups. The batch *k*NN (B*k*NN) search for a query segment $\overline{q_iq_j}$ is performed to obtain data points closest to each query point in $\overline{q_iq_j}$ (line 6). Result $\Pi(\overline{q_iq_j})$ for $\overline{q_iq_j}$ is added to the query result $\Pi(Q)$, where $\Pi(\overline{q_iq_j})=\{\langle q,P_k(q)\rangle|q\in\overline{q_iq_j}\}$ (line 7). Subsequently, the query result $\Pi(Q)$ is returned after performing the B*k*NN search for all query groups in $\overline{Q}$ (line 8).

Algorithm 2 describes the B*k*NN search algorithm. The B*k*NN search groups query points and batch execution to avoid redundant network traversal. This algorithm comprises two steps. First, two *k*NN queries are issued for the query segment $\overline{q_iq_j}$. We carefully determined the location of *k*NN queries

Algorithm 1: BANK(Q,P)
Input: Q: set of query points, P: set of data points
Output: $\Pi(Q)$: set of ordered pairs of each query point q in Q and its *k*NN set, i.e., $\Pi(Q)=\{\langle q,P_k(q)\rangle \mid q\in Q\}$.
1 $\Pi(Q)\leftarrow\varnothing$ // the result set $\Pi(Q)$ is initialized to the empty set.
2 // adjacent query points in the same vertex sequence are grouped into a query group, which is explained in Section III.1.
3 $\overline{Q}\leftarrow$ group_points(Q) // adjacent query points $q_i,q_{i+1},\cdots,q_j$ in a vertex sequence are grouped into $\overline{q_iq_j}$.
4 // data points closest to each query point in $\overline{q_iq_j}$ are retrieved, which is detailed in Algorithm 2.
5 **for** each query segment $\overline{q_iq_j}\in\overline{Q}$ **do**
6 $\Pi(\overline{q_iq_j})\leftarrow$ B*k*NN_search($\overline{q_iq_j},P$) // $\Pi(\overline{q_iq_j})=\{\langle q,P_k(q)\rangle \mid q\in\overline{q_iq_j}\}$
7 $\Pi(Q)\leftarrow\Pi(Q)\cup\Pi(\overline{q_iq_j})$ // the result for each query point in a query group $\overline{q_iq_j}$ is added to $\Pi(Q)$.
8 **return** $\Pi(Q)$ // $\Pi(Q)$ is returned after the B*k*NN search for all query groups in $\overline{Q}$ is executed.

Algorithm 2: BkNN_search$(\overline{q_iq_j}, P)$

Input: $\overline{q_iq_j}$: query segment, P: set of data points

Output: $\Pi(\overline{q_iq_j})$: set of ordered pairs of each query point q in $\overline{q_iq_j}$ and its kNN set, i.e., $\Pi(\overline{q_iq_j}) = \{\langle q, P_k(q)\rangle \mid q \in \overline{q_iq_j}\}$.

$\Pi(\overline{q_iq_j}) \leftarrow \varnothing$ // $\Pi(\overline{q_iq_j})$ is initialized to an empty set

// assume that a query segment $\overline{q_iq_j}$ belongs to a vertex sequence $\overline{v_lv_m}$ and that q_i (q_j) is closer to v_l (v_m) than q_i (q_j).

// step 1: a set of candidate data points is computed for all query points in $\overline{q_iq_j}$.

if adj_seqs$(v_l) \geq 2$ **and** *adj_seqs*$(v_m) \geq 2$ **then**

  $K_{v_l} \leftarrow \max\{k_a, k_{a+1}, \cdots, k_b\}$ // K_{v_l} is the max k value of $q_a, q_{a+1}, \cdots, q_b$ in query segments adjacent to v_l.

  $P_{K_{v_l}}(v_l) \leftarrow$ kNN_query(K_{v_l}, v_l, P) // kNN query is evaluated for v_l and its kNN set is saved to $P_{K_{v_l}}(v_l)$.

  $K_{v_m} \leftarrow \max\{k_c, k_{c+1}, \cdots, k_d\}$ // K_{v_m} is the max k value of $q_c, q_{c+1}, \cdots, q_d$ in query segments adjacent to v_m.

  $P_{K_{v_m}}(v_m) \leftarrow$ kNN_query(K_{v_m}, v_m, P) // kNN query is evaluated for v_m and its kNN set is saved to $P_{K_{v_m}}(v_m)$.

  $P_{can} \leftarrow P_{K_{v_l}}(v_l) \cup P_{K_{v_m}}(v_m) \cup P(\overline{v_lv_m})$ // P_{can} is the set of candidate data points for $\overline{q_iq_j}$.

else if *adj_seqs*$(v_l) \geq 2$ **and** *adj_seqs*$(v_m) = 1$ **then**

  $K_{v_l} \leftarrow \max\{k_a, k_{a+1}, \cdots, k_b\}$ // K_{v_l} is the max k value of $q_a, q_{a+1}, \cdots, q_b$ in query segments adjacent to v_l.

  $P_{K_{v_l}}(v_l) \leftarrow$ kNN_query(K_{v_l}, v_l, P) // kNN query is evaluated for v_l and its kNN set is saved to $P_{K_{v_l}}(v_l)$.

  $K_{q_j} \leftarrow \max\{k_i, k_{i+1}, \cdots, k_j\}$ // K_{q_j} is the max k value of query points $q_i, q_{i+1}, \cdots, q_j$ in $\overline{q_iq_j}$.

  $P_{K_{q_j}}(q_j) \leftarrow$ kNN_query(K_{q_j}, q_j, P) // kNN query is evaluated for q_j and its kNN set is saved to $P_{K_{q_j}}(q_j)$.

  $P_{can} \leftarrow P_{K_{v_l}}(v_l) \cup P_{K_{q_j}}(q_j) \cup P(\overline{v_lq_j})$ // P_{can} is the set of candidate data points for $\overline{q_iq_j}$.

else if *adj_seqs*$(v_l) = 1$ **and** *adj_seqs*$(v_m) \geq 2$ **then**

  $K_{q_i} \leftarrow \max\{k_i, k_{i+1}, \cdots, k_j\}$ // K_{q_i} is the max k value of query points $q_i, q_{i+1}, \cdots, q_j$ in $\overline{q_iq_j}$.

  $P_{K_{q_i}}(q_i) \leftarrow$ kNN_query(K_{q_i}, q_i, P) // kNN query is evaluated for q_i and its kNN set is saved to $P_{K_{q_i}}(q_i)$.

  $K_{v_m} \leftarrow \max\{k_c, k_{c+1}, \cdots, k_d\}$ // K_{v_m} is the max k value of $q_c, q_{c+1}, \cdots, q_d$ in query segments adjacent to v_m.

  $P_{K_{v_m}}(v_m) \leftarrow kNN_{query}(K_{v_m}, v_m, P)$ // kNN query is evaluated for v_m and its kNN set is saved to $P_{K_{v_m}}(v_m)$.

  $P_{can} \leftarrow P_{K_{q_i}}(q_i) \cup P_{K_{v_m}}(v_m) \cup P(\overline{q_iv_m})$ // P_{can} is the set of candidate data points for $\overline{q_iq_j}$.

else

  $K_{q_i} \leftarrow \max\{k_i, k_{i+1}, \cdots, k_j\}$ // K_{q_i} is the max k value of query points $q_i, q_{i+1}, \cdots, q_j$ in $\overline{q_iq_j}$.

  $P_{K_{q_i}}(q_i) \leftarrow$ kNN_query(K_{q_i}, q_i, P) // kNN query is evaluated for q_i and its kNN set is saved to $P_{K_{q_i}}(q_i)$.

  $K_{q_j} \leftarrow K_{q_i}$ // K_{q_j} is the same value as K_{q_i}.

  $P_{K_{q_j}}(q_j) \leftarrow$ kNN_query(K_{q_j}, q_j, P) // kNN query is evaluated for q_j and its kNN set is saved to $P_{K_{q_j}}(q_j)$.

  $P_{can} \leftarrow P_{K_{q_i}}(q_i) \cup P_{K_{q_j}}(q_j) \cup P(\overline{q_iq_j})$ // P_{can} is the set of candidate data points for $\overline{q_iq_j}$.

// step 2: a set of k NNs for each query point in $\overline{q_iq_j}$ is computed using the set P_{can} of candidate data points.

for each query point $q \in \overline{q_iq_j}$ do

  // kNN search for a query point q is performed using a set P_{can} of candidate data points for $\overline{q_iq_j}$.

  $P_k(q) \leftarrow$ kNN_search(k, q, P_{can}) // kNN search for q is performed using a set P_{can} of candidate data points.

  $\Pi(\overline{q_iq_j}) \leftarrow \Pi(\overline{q_iq_j}) \cup \{\langle q, P_k(q)\rangle\}$ // the kNN set for q is saved to $P_k(q)$, which is added to $\Pi(\overline{q_iq_j})$.

return // query result $\Pi(\overline{q_iq_j})$ is returned for $\overline{q_iq_j}$.

using the number of query segments adjacent to an intersection vertex v_l (v_m) in $\overline{v_lv_m}$ to share the results of kNN queries among the query segments adjacent to an intersection vertex. We assume that $\overline{q_iq_j}$ belongs to $\overline{v_lv_m}$ and that q_i (q_j) is closer to v_l (v_m) than q_j (q_i). The location of one kNN query is either q_i or v_l, and the location of another kNN query is either q_j or v_m. If more than two query segments are adjacent to v_l, i.e., *adj_segs*$(v_l) \geq 2$, v_l issues a kNN query with $K_{v_l} = \max\{k_a, k_{a+1}, \cdots, k_b\}$ assuming that $q_a, q_{a+1}, \cdots, q_b$ belong to query segments adjacent to v_l and $q_a, q_{a+1}, \cdots, q_b$ have $k_a, k_{a+1}, \cdots, k_b$, respectively; otherwise, v_l issues a kNN query with $K_{q_i} = \max\{k_i, k_{i+1}, \cdots, k_j\}$ assuming that $q_i, q_{i+1}, \cdots, q_j$ constitute $\overline{q_iq_j}$ and $q_a, q_{a+1}, \cdots, q_b$ have $k_a, k_{a+1}, \cdots, k_b$, respectively. Similarly, if more than

two query segments are adjacent to v_m, i.e., *adj_segs* $(v_m) \geq 2$, v_m issues another *k*NN query with $K_{v_m} = \max\{k_c, k_{c+1}, \cdots, k_d\}$ assuming that $q_c, q_{c+1}, \cdots, q_d$ belong to query segments adjacent to v_m and $q_c, q_{c+1}, \cdots, q_d$ have $k_c, k_{c+1}, \cdots, k_d$, respectively; otherwise, q_j issues another *k*NN query with $K_{q_j} = \max\{k_i, k_{i+1}, \cdots, k_j\}$. Therefore, we have the following four cases depending on the locations of the two *k*NN queries evaluated for a query segment $\overline{q_i q_j}$: $\langle v_l, v_m \rangle$, $\langle v_l, q_j \rangle$, $\langle q_i, v_m \rangle$, and $\langle q_i, q_j \rangle$. Specifically, the first case $\langle v_l, v_m \rangle$ is described in lines 4−9 of the algorithm. The second case $\langle v_l, q_j \rangle$ is described in lines 10−15. The third case $\langle q_i, v_m \rangle$ is described in lines 16−21, and the fourth case $\langle q_i, q_j \rangle$ is described in lines 22−27. After these two *k*NN queries are evaluated for a query segment $\overline{q_i q_j}$, their results are saved to be included in a set P_{can} of the candidate data points for $q_i, q_{i+1}, \cdots, q_j$ in $\overline{q_i q_j}$. Next, the *k*NN set $P_k(q)$ for each query point q in $\overline{q_i q_j}$ is retrieved from candidate data points in P_{can}. Subsequently, a pair of query point q and its *k*NN set $P_k(q)$ is added to the result, as follows: $\Pi(\overline{q_i q_j}) \leftarrow \Pi(\overline{q_i q_j}) \cup \{\langle q, P_k(q) \rangle\}$.

Algorithm 3 describes the *k*NN search for finding k data points closest to query point q in $\overline{q_i q_j}$ among the candidate data points in $P_{K_\alpha}(\alpha) \cup P_{K_\beta}(\beta) \cup P(\overline{\alpha\beta})$. First, the set $P_k(q)$ of *k*NNs of query point q is initialized to an empty set. The distance from q to the candidate data point p is computed based on condition p (lines 3−10). After computing $dist(q,p)$, we can determine whether p is added to the candidate *k*NN set $P_k(q)$. If $|P_k(q)| < k$, then p is added to $P_k(q)$ (lines 12−13). If $|P_k(q)| = k$ and $dist(q,p) < dist(q,p_{kth})$, then it is added to $P_k(q)$ and p_{kth} is removed from $P_k(q)$, where p_{kth} denotes the kth data point closest to q, i.e., $P_k(q) \leftarrow P_k(q) \cup \{p\} - \{p_{kth}\}$ (lines 14−15). The *k*NN set $P_k(q)$ for q is returned after all the candidate data points in $P_{K_\alpha}(\alpha) \cup P_{K_\beta}(\beta) \cup P(\overline{\alpha\beta})$ have been examined (line 16).

Furthermore, we present a theoretical analysis to prove the advantages of the BANK algorithm over sequential processing. The time complexity of the BANK algorithm is $O(|\overline{Q}| \cdot (|E| + |V| \log |V|))$,

Algorithm 3: *k*NN_search$(k, q, P_{K_\alpha}(\alpha) \cup P_{K_\beta}(\beta) \cup P(\overline{\alpha\beta}))$

Input: k: number of data points requested by q, q: query point in $\overline{q_i q_j}$, $P_{K_\alpha}(\alpha) \cup P_{K_\beta}(\beta) \cup P(\overline{\alpha\beta})$: set of candidate data points for q
Output: $P_k(q)$: set of k data points closest to q

```
1  P_k(q) ← ∅                    // P_k(q) is initialized to the empty set.
2  for each candidate data point p ∈ P_{K_α}(α) ∪ P_{K_β}(β) ∪ P(ᾱβ) do
3      // step 1: dist(q,p) is computed according to the condition of p.
4      if p ∈ P_{K_α}(α) ∩ P_{K_β}(β) ∩ P(ᾱβ) then dist(q,p) ← min{len(q,α)+dist(α,p), len(q,β)+dist(β,p), len(q,p)}
5      else if p ∈ P_{K_α}(α) ∩ P_{K_β}(β) − P(ᾱβ) then dist(q,p) ← min{len(q,α)+dist(α,p), len(q,β)+dist(β,p)}
6      else if p ∈ P_{K_α}(α) ∩ P(ᾱβ) − P_{K_β}(β) then dist(q,p) ← min{len(q,α)+dist(α,p), len(q,p)}
7      else if p ∈ P_{K_β}(β) ∩ P(ᾱβ) − P_{K_α}(α) then dist(q,p) ← min{len(q,β)+dist(β,p), len(q,p)}
8      else if p ∈ P_{K_α}(α) − (P_{K_β}(β) ∪ P(ᾱβ)) then dist(q,p) ← len(q,α)+dist(α,p)
9      else if p ∈ P_{K_β}(β) − (P_{K_α}(α) ∪ P(ᾱβ)) then dist(q,p) ← len(q,β)+dist(β,p)
10     else if p ∈ P(ᾱβ) − (P_{K_α}(α) ∪ P_{K_β}(β)) then dist(q,p) ← len(q,p)
11     // step 2: p is added to P_k(q) if it satisfies either of the following two conditions.
12     if |P_k(q)| < k then
13         P_k(q) ← P_k(q) ∪ {p}
14     else if |P_k(q)| = k and dist(q,p) < dist(q,p_kth) then
15         P_k(q) ← P_k(q) ∪ {p} − {p_kth}
16 return P_k(q)                 // the kNN set P_k(q) for q is returned after all the candidate data points have been examined.
```

Table 2. Real-world roadmaps

Name	Description	Vertices	Edges	Vertex sequences
SF	San Francisco, California	174,956	223,001	192,276
COL	Colorado	435,666	521,200	374,355
FLA	Florida	1,070,376	1,343,951	1,100,675

where $|\overline{Q}|$ is the number of query segments and $O(|E|+|V|\log|V|)$ is the time complexity for evaluating a single *k*NN query. Conversely, the time complexity of the simple solution based on sequential processing is $O(|Q|\cdot(|E|+|V|\log|V|))$. This theoretical analysis shows that the BANK algorithm is superior to the simple solution when the query points exhibit a skewed distribution. This is because $|\overline{Q}| \ll |Q|$ is for highly skewed query points.

IV. Performance study

In this section, we present the results of an empirical analysis of the BANK algorithm. We describe the experimental settings in Section IV.1 and present the experimental results in Section IV.2.

1. Experimental settings

For the performance study, we used the three real-world road networks [17, 18] presented in Table 2. These real-world road networks have different sizes and are part of the road network of the US. Table 3 shows the range of each variable used in the experiments with defaults written in bold. For convenience, each dimension of the data universe was normalized independently to a unit length $[0,1]$. The query and data points exhibited either a centroid or uniform distribution. A centroid-based dataset was generated to resemble real-world data. First, five centroids were randomly selected for the query and data points. The points around each centroid exhibited a normal distribution, where the mean was set to the centroid, and the standard deviation was set to 1% of the side length of the data universe. Unless otherwise stated, the query points exhibited a centroid distribution, whereas the data points exhibited a uniform distribution.

As a benchmark for the evaluation of the BANK algorithm, we used INE [4] as a one-query-at-a-time solution, which sequentially computes the *k*NN set for each query point in *Q*. We implemented and evaluated two versions of the BANK algorithm: BANK_{OPT} and BANK_{GRP}. BANK_{OPT} was implemented using the proposed algorithms. Conversely, BANK_{GRP} grouped query points in a road segment into a query segment and generated two *k*NN queries at the endpoints of the query segment. The BANK algorithm processed the query points in a batch, whereas INE processed them sequentially. In this study, we assumed that the query and data points can move freely within dynamic spatial networks. Therefore, it was not feasible to use precomputation techniques because the movements of the query and data points might frequently invalidate the precomputed distances in dynamic spatial networks. The methods were implemented in C++ in the Microsoft Visual Studio 2019 development environment. Note that C++ and the development environment use common subroutines for similar tasks. We performed the experiments on a desktop computer with a Windows 10 operating system with a 32 GB RAM and a 3.1 GHz processor (i9-9900).

Table 3. Experimental parameter settings

Parameter	Range
Number of query points	1, 3, 5, 7, **10** (×1000)
Number of data points	1, 3, 5, 7, **10** (×1000)
Number of requested NNs	[1,4], [5,8], **[9,16]**,…, [65,128]
Distribution of query points	**(C)entroid**, (U)niform
Distribution of data points	(C)entroid, **(U)niform**
Roadmap	SF, COL, FLA

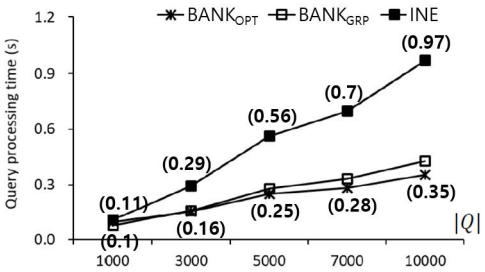
2. Experimental results

Figure 4 shows a comparison of query processing times using BANK_{OPT}, BANK_{GRP}, and INE

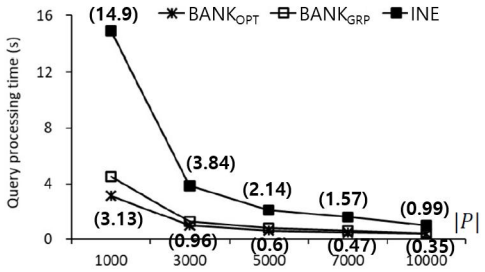
when evaluating M*k*NN queries in the SF roadmap. Each chart illustrates the effects of varying one of the parameters in Table 3. The two values in parentheses in Figures 4 and 5 show the query processing times of $BANK_{OPT}$ and INE. Figure 4(a) shows the query processing times using $BANK_{OPT}$, $BANK_{GRP}$, and INE when the number $|Q|$ of query points varied between 1000 and 10000, i.e., $1000 \le |Q| \le 10000$. $BANK_{OPT}$ outperformed INE owing to the batch processing of query points as $|Q|$ increased. $BANK_{OPT}$ evaluated 47%, 25%, 26%, 19%, and 17% of the number of *k*NN queries evaluated by INE when $|Q|$ = 1000, 3000, 5000, 7000, and 10000, respectively.

Figure 4(b) shows the query processing times using the three algorithms when the number $|P|$ of data points varied between 1000 and 10000, i.e., $1000 \le |P| \le 10000$. The query processing times using $BANK_{OPT}$ were up to 4.8 times shorter than those using INE in all cases. As $|P|$ decreased, the difference in the query processing times between $BANK_{OPT}$ and INE increased. $BANK_{OPT}$ evaluated 17% of the number of *k*NN queries evaluated by INE regardless of $|P|$.

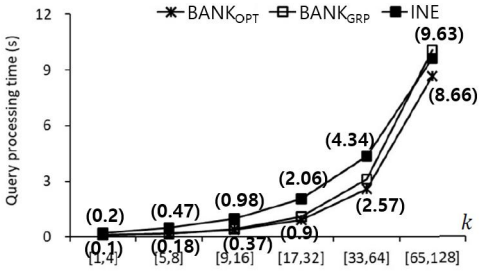
Figure 4(c) shows the query processing times using the three algorithms when the number of k data points requested by the query points varied between 1 and 128, i.e., $1 \le k \le 128$. The query processing times increased with the k value. The query processing times using $BANK_{OPT}$ were up to 2.7 times shorter than those using INE in all cases. Figure 4(d) shows the query processing times for various distributions of the query and data points, where each ordered pair (i.e., $\langle C, C \rangle$, $\langle C, U \rangle$, $\langle U, C \rangle$, and $\langle U, U \rangle$) denotes a combination of the distributions of the query and data points. $BANK_{OPT}$ outperformed INE for $\langle C, C \rangle$ and $\langle C, U \rangle$ distributions when the query points exhibited a centroid distribution. However, $BANK_{OPT}$ and INE demonstrated similar performances for $\langle U, C \rangle$ and $\langle U, U \rangle$ distributions when the query points exhibited a uniform distribution.



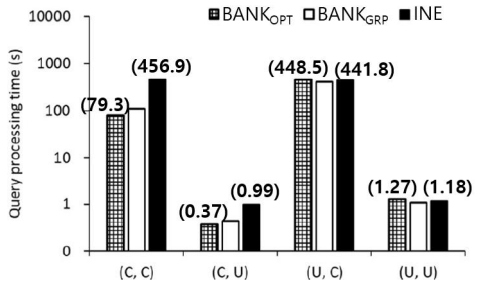
(a) Varying $|Q|$



(b) Varying $|P|$

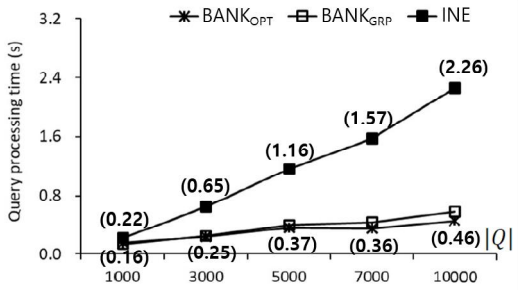


(c) Varying k

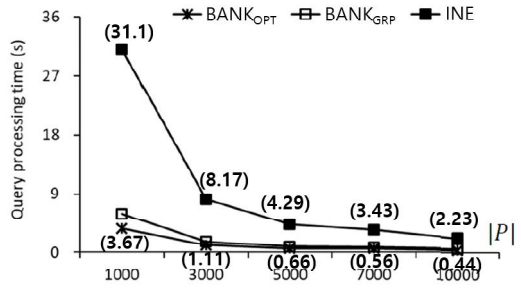


(d) Varying point distribution

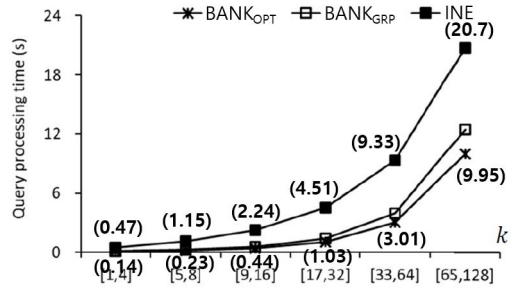
Fig. 4. Comparison of $BANK_{OPT}$, $BANK_{GRP}$, and INE for SF



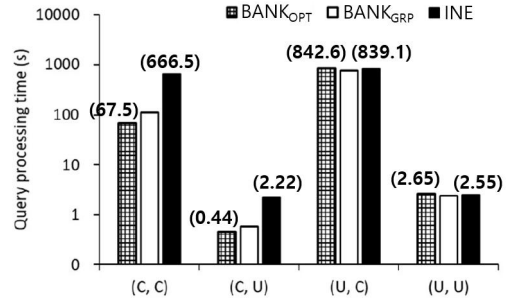
(a) Varying $|Q|$



(b) Varying $|P|$



(c) Varying k



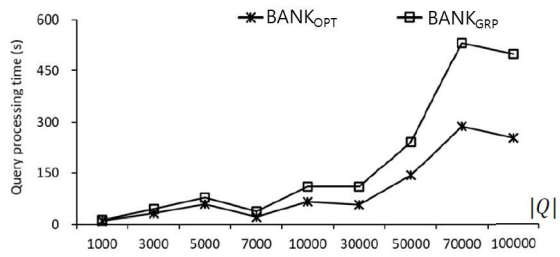
(d) Varying point distribution

Fig. 5. Comparison of $BANK_{OPT}$, $BANK_{GRP}$, and INE for COL

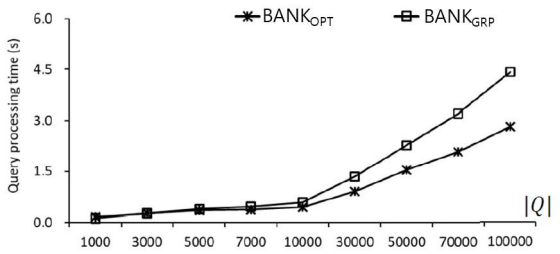
Figure 5 shows a comparison of query processing times using $BANK_{OPT}$, $BANK_{GRP}$, and INE when evaluating MkNN queries in the COL roadmap. Figure 5(a) shows the query processing time as a function of $|Q|$. The query processing times using $BANK_{OPT}$ were up to 4.9 times shorter than those using INE in all cases. INE evaluated $|Q|$ kNN queries to answer MkNN queries, whereas $BANK_{OPT}$ evaluated $2 \times |\overline{Q}|$ kNN queries at the maximum because of the batch processing. Figure 5(b) shows the query processing time as a function of $|P|$. $BANK_{OPT}$ was superior to INE in all cases. This is because $BANK_{OPT}$ utilizes batch processing of adjacent query points and requests fewer kNN queries than INE. Figure 5(c) shows the query processing time as a function of k. The query processing times using $BANK_{OPT}$ were up to 5.1 times less than those using INE in all cases. Figure 5(d) shows the query processing times for various distributions of the query and data points. For a centroid distribution of query points, i.e., $\langle C, C \rangle$ and $\langle C, U \rangle$, the query processing times using $BANK_{OPT}$ were up to 9.9 times shorter than those using INE. However, for a uniform distribution of query points (i.e., $\langle U, C \rangle$ and $\langle U, U \rangle$), $BANK_{OPT}$ and INE demonstrated similar performances because the query points were widely scattered, and $BANK_{OPT}$ and INE performed similarly.

Subsequently, we analyzed the scalability of $BANK_{OPT}$ and $BANK_{GRP}$ by varying the number $|Q|$ of query points when the data points exhibited uniform and centroid distributions. We did not include the query processing times of INE for the scalability test because INE yielded poor performance as $|Q|$ increased. Figure 6 shows the query processing times using $BANK_{OPT}$ and $BANK_{GRP}$ for $10^3 \leq |Q| \leq 10^5$. As shown in Figures 6(a) and 6(c), $BANK_{OPT}$ outperformed $BANK_{GRP}$ for the COL and FLA roadmaps, respectively, using the $\langle C, C \rangle$ distributions. The difference in query processing times between $BANK_{OPT}$ and $BANK_{GRP}$ increased with $|Q|$. Similarly, as shown in Figures 6(b) and 6(d), $BANK_{OPT}$ outperformed $BANK_{GRP}$ for the COL, and FLA roadmaps, respectively, using the $\langle C, U \rangle$ distributions. The difference in query processing times between $BANK_{OPT}$ and $BANK_{GRP}$ increased with $|Q|$. The empirical results indicate

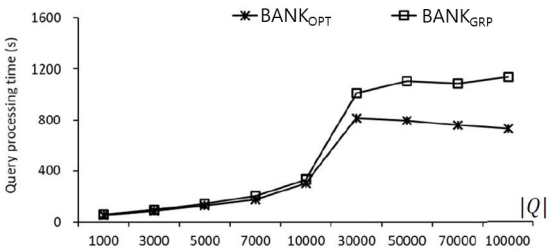
that $BANK_{OPT}$ scaled with $|Q|$ better than $BANK_{GRP}$.



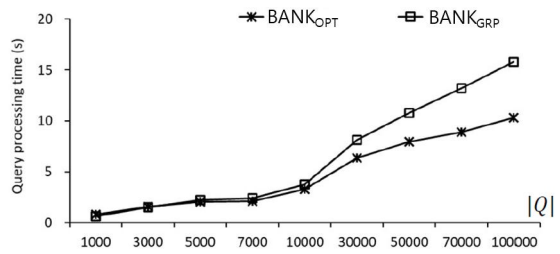
(a) COL and $\langle C, C \rangle$



(b) COL and $\langle C, U \rangle$



(c) FLA and $\langle C, C \rangle$



(d) FLA and $\langle C, U \rangle$

Fig. 6. Scalability test for $10^3 \leq |Q| \leq 10^5$

V. Conclusions

In this study, we proposed a batch processing algorithm, known as BANK, to efficiently process M*k*NN queries in dynamic spatial networks. The BANK algorithm was the first attempt at batch processing of M*k*NN queries in dynamic spatial networks and aimed to minimize the number of *k*NN queries requested for highly skewed query points. Our extensive evaluation using real-world roadmaps confirmed that the BANK algorithm clearly outperformed INE based on one-query-at-a-time processing and scaled well with the number of query points, particularly when the query points exhibited a non-uniform distribution. Notably, the BANK algorithm was up to 9.9 times faster than INE. However, the BANK and INE showed similar performances when the query points exhibited a uniform distribution. For future studies, we plan to extend the batch processing approach used in this study to problems on the processing of sophisticated spatial queries for large dynamic spatial networks.

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2020R1I1A3052713).

REFERENCES

[1] T. Kim, H.-J. Cho, H. J. Hong, H. Nam, H. Cho, G. Y. Do, and P. Jeon, "Efficient processing of k-farthest neighbor queries for road networks," *Journal of The Korea Society of Computer and Information*, vol. 24, no. 10, pp. 79−89, 2019.

[2] F. M. Choudhury, J. S. Culpepper, Z. Bao, and T. Sellis, "Batch processing of top-k spatial-textual queries," *ACM Transactions on Spatial Algorithms and Systems*, vol. 3, no. 4, pp. article ID 13, 2018.

[3] K. C. K. Lee, W.-C. Lee, B. Zheng, and Y. Tian, "ROAD: a new spatial object search framework for road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 3, pp. 547−560, 2012.

[4] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," In *Proc. of International Conference on Very Large Data Bases*, pp. 802−813, 2003.

[5] B. Shen, Y. Zhao, G. Li, W. Zheng, Y. Qin, B. Yuan, and Y. Rao, "V-tree: efficient knn search on moving objects with road-network constraints," In *Proc. of International Conference*

on Data Engineering, pp. 609−620, 2017.

[6] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: an efficient and scalable index for spatial search on road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2175−2189, 2015.

[7] Y. Xu, J. Qi, R. Borovica-Gajic, and L. Kulik, "Finding all nearest neighbors with a single graph traversal," In *Proc. of International Conference on Database Systems for Advanced Applications*, pp. 221−238, 2018.

[8] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," In *Proc. of International Conference on Mobile Data Management*, pp. 43−54, 2008.

[9] T. Abeywickrama and M. A. Cheema, "Efficient landmark-based candidate generation for knn queries on road networks," In *Proc. of International Conference on Database Systems for Advanced Applications*, pp. 425−440, 2017.

[10] B. Cao, C. Hou, S. Li, J. Fan, J. Yin, B. Zheng, and J. Bao, "SIMkNN: a scalable method for in-memory knn search over moving objects in road networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1957−1970, 2018.

[11] T. Dong, Y. Lulu, Y. Shang, Y. Ye, and L. Zhang, "Direction-aware continuous moving k-nearest-neighbor query in road networks," *ISPRS International Journal of Geo-Information*, vol. 8, no. 9, article ID 379, 2019.

[12] S. Luo, B. Kao, G. Li, J. Hu, R. Cheng, and Y. Zheng, "TOAIN: a throughput optimizing adaptive index for answering dynamic knn queries on road networks," *PVLDB*, vol. 11, no. 5, pp. 594−606, 2018.

[13] Y. Yang, H. Li, J. Wang, Q. Hu, X. Wang, and M. Leng, "A novel index method for k nearest object query over time-dependent road networks," *Complexity*, vol. 2019, article ID 4829164, 2019.

[14] B. Zheng, K. Zheng, X. Xiao, H. Su, H. Yin, X. Zhou, and G. Li, "Keyword-aware continuous knn query on road networks," In *Proc. of International Conference on Data Engineering*, pp. 871−882, 2016.

[15] U. Demiryurek, F. B. Kashani, and C. Shahabi, "Efficient continuous nearest neighbor query in spatial networks using Euclidean restriction," In *Proc. of International Symposium on Advances in Spatial and Temporal Databases*, pp. 25−43, 2009.

[16] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," In *Proc. of International Conference on Very Large Data Bases*, pp. 43−54, 2006.

[17] 9th DIMACS Implementation Challenge: Shortest Paths. Available online: http://www.dis.uniroma1.it/challenge9/download.shtml (accessed on 17 Feb. 2021).

[18] Real Datasets for Spatial Databases. Available online: https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm (accessed on 17 Feb. 2021).

Authors

Hyung-Ju Cho is an associate professor at the department of software, Kyungpook National University. His current research interests include moving object databases, query processing in mobile peer-to-peer networks, and real-time maintenance of the high definition digital map for autonomous vehicles.