IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 14, 2020, accepted June 1, 2020, date of publication June 15, 2020, date of current version June 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002263

Group Processing of Multiple *k*-Farthest Neighbor Queries in Road Networks

HYUNG-JU CHO[1] AND MUHAMMAD ATTIQUE[2]
[1]Department of Software, Kyungpook National University, Sangju 37224, South Korea
[2]Department of Software, Sejong University, Seoul 05006, South Korea

Corresponding authors: Hyung-Ju Cho (hyungju@knu.ac.kr) and Muhammad Attique (attique@sejong.ac.kr)

The work of Hyung-Ju Cho was supported by the National Research Foundation of Korea (NRF) funded by the Korean Government, Ministry of Science and ICT (MSIT) under Grant NRF-2019R1H1A2080073. The work of Muhammad Attique was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education under Grant 2020R1G1A1013221.

ABSTRACT Advances in mobile technologies and map-based applications enables users to utilize sophisticated spatial queries, including *k*-nearest neighbor and shortest path queries. Often, location-based servers are used to handle multiple simultaneous queries because of the popularity of map-based applications. This study focuses on the efficient processing of multiple concurrent *k*-farthest neighbor (*k*FN) queries in road networks. For a positive integer *k*, query point *q*, and set of data points *P*, a *k*FN query returns *k* data points farthest from the query point *q*. For addressing multiple concurrent spatial queries, traditional location-based servers based on one-query-at-a-time processing are unsuitable owing to high redundant computation costs. Therefore, we propose a group processing of multiple *k*FN (GMP) algorithm to process multiple *k*FN queries in road networks. The proposed GMP algorithm uses group computation to avoid the redundant computation of network distances between the query and data points. The experiments using real-world roadmaps demonstrate the proposed solution's effectiveness and efficiency.

INDEX TERMS Spatial databases, group processing, multiple *k*-farthest neighbor query, road network.

I. INTRODUCTION

The proliferation of smartphones with GPS and Wi-Fi functionality has enabled mobile users to exploit various location-based services (LBS), such as mobile guides, intelligent transport systems, location-based gaming, and assistive technology to support people with health problems [17], [18], [32], [33], [41], [51]. Because of the popularity of LBSs, LBS servers often respond to multiple simultaneous user queries. For multiple applications, traditional LBS servers based on one-query-at-a-time processing are unsuitable because they cannot guarantee an inexpensive and real-time response in high-load conditions. Consequently, the group processing of spatial queries has become an important LBS research topic [7], [8], [27], [36], [37], [52], [53].

We focus on the group processing of multiple *k*-farthest neighbor (M*k*FN) queries in road networks; M*k*FN queries are the logical opposite of *k*-nearest neighbor (*k*NN) queries. We considered road networks rather than a Euclidean space because the road network constrains both people and vehicles. The farthest neighbor search is used in as many real-life applications as the nearest neighbor search, including computational geometry, artificial intelligence, pattern recognition, and information retrieval. In particular, the farthest neighbor search can determine the minimum radius of a circle centered at a query point *q* that includes all data points. For a *k*FN query, let us consider a real-life scenario in which a team of commandos is on a mission, and the leader commands all team members to be less than 1 km away from him. Typically, team members who are farther from the team leader require more of the leader's attention. Furthermore, the leader might be interested in the farthest team members to monitor their activities and advise them not to move further away from him.

Aggregate *k*FN (A*k*FN) query in road networks [47] is similar to the M*k*FN query. However, in A*k*FN queries, for a set of query points *Q* and a set of data points *P*, an A*k*FN query reports *k* data points having the largest aggregate network distance such as the largest sum of network distances from all query points in *Q*. However, an M*k*FN query reports *k* data points farthest from each query point *q* in *Q*. Thus, the existing solutions for A*k*FN queries cannot be directly used to

The associate editor coordinating the review of this manuscript and approving it for publication was Gianmaria Silvello.

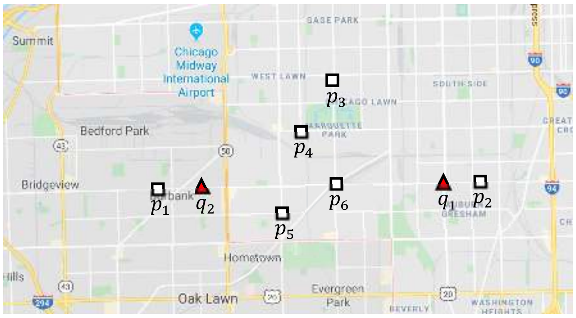


FIGURE 1. **Multiple *k*FN queries in a road network where a set of query points $Q = \{q_1, q_2\}$ and a set of data points $P = \{p_1, p_2, \cdots, p_6\}$ are provided.**

evaluate M*k*FN queries. For example, as shown in Figure 1, given a set of query points $Q = \{q_1, q_2\}$ and a set of data points $P = \{p_1, p_2, \cdots, p_6\}$ for the M*k*FN query result, data points p_1 and p_2 are the farthest neighbors of q_1 and q_2 query points, respectively. However, for the A*k*FN query result, data point p_3 is the farthest neighbor with the largest sum of network distances from query points q_1 and q_2.

A one-query-at-a-time approach that sequentially computes the k farthest data points from each query point in Q is a straightforward solution for M*k*FN queries. However, this solution involves computing the network distance from the query point q to each data point p in P with additional $O(|P|\log|P|)$ time to determine a set of k data points farthest from q, which is compute-intensive. Therefore, in road networks, we propose an innovative algorithm for the group processing of multiple *k*FN (GMP) queries. The GMP algorithm clusters adjacent query and data points into a query and data group, respectively, and then optimizes shared computation for the query group to eliminate redundant candidates by computing the maximum and minimum distances between query and data groups. Although the group computation of spatial queries has received considerable attention [7], [8], [27], [36], [37], [52], [53], group computation has not been applied to M*k*FN queries in road networks our knowledge. In this study, we utilized shared execution to efficiently evaluate M*k*FN queries in road networks in which it is assumed that query and data points arbitrarily move. The proposed solution is batch processing for M*k*FN queries, and the straightforward one-query-at-a-time solution is sequential processing. The GMP algorithm is orthogonal to the network distance methods [3], [13], [22], [23], [38], [54] and easy to implement, thereby facilitating its integration with existing network distance methods.

The primary contributions of this study are listed below:

- We propose the GMP algorithm, an efficient algorithm for the group processing of multiple *k*FN queries in road networks. To our knowledge, this attempt is the first to study M*k*FN queries in road networks.
- We present shared computation techniques to avoid the redundant computation of network distances from the query to data points. Furthermore, we present effective pruning techniques to utilize the maximum distance from the query to data groups.
- We conducted extensive experiments using real-world roadmaps to demonstrate the efficiency and scalability of the proposed solution.

The remainder of this study is organized as follows. In Section II, the related studies are reviewed. In Section III, we introduce preliminaries and formally define the M*k*FN query. In Section IV, we explain the grouping of adjacent points into segments, and then describe the computation of two different segments. In Section V, we present the GMP algorithm for the efficient processing of M*k*FN queries in road networks. In Section VI, we compare the GMP algorithm and its conventional solution with different setups. Finally, in Section VII, the conclusions and suggestions for future work are provided.

II. RELATED STUDIES

In this section, we describe the farthest neighbor search and group processing algorithms in Sections II.A and II.B, respectively.

A. FARTHEST NEIGHBOR SEARCH ALGORITHMS

Multiple studies have focused on the efficient processing of sophisticated spatial queries based on the farthest neighbor search [6], [10], [12], [24]–[26], [42], [45], [47], [49], [50]. Curtin *et al.* [10] reported an approximate farthest neighbor search algorithm that selects a set of candidate data points using data distributed in a Euclidean space. Furthermore, to investigate the difficulty of the farthest neighbor search problem, they developed an information-theoretic entropy measure. Lu and Yiu [26] formulated a farthest-dominated location query for spatial decision support applications. The formulated query retrieves a location such that the distance to its nearest dominating object is maximized. Gao *et al.* [12] and Wang *et al.* [47] studied A*k*FN queries in a Euclidean space and spatial networks, respectively. For a set of data points P and a set of query points Q, an A*k*FN query returns k data points in P that have the largest aggregate distances to all query points in Q. Moreover, reverse farthest neighbor queries have been studied, in the Euclidean space [24], [50] and spatial networks [45], [49]. Yao *et al.* [50] proposed progressive farthest cell and convex hull farthest cell algorithms to support the reverse farthest neighbor queries using an R-tree [4], [16]. Wang *et al.* [46] presented a solution to support reverse *k*FN queries in the Euclidean space for the arbitrary values of k. Tran *et al.* [45] studied reverse farthest neighbor queries in spatial networks using network Voronoi diagrams and pre-computed network distances. Xu *et al.* [49] presented efficient algorithms based on landmarks and hierarchical partitioning to process monochromatic and bichromatic reverse farthest neighbor queries in spatial networks. However, the existing solutions in the Euclidean space cannot be applied to our situation because it is difficult to utilize R-trees and convex hulls in spatial networks.

Furthermore, the efficient processing of M*k*FN queries in road networks has not been extensively studied.

B. GROUP PROCESSING ALGORITHMS

Multi-query optimization was originally investigated with reference to relational database systems [40]. For a set of currently running queries, computational costs are reduced by executing shared expressions once, materializing them temporarily, and then reusing them to solve the remaining queries. Thus, common subexpressions are evaluated once. This approach was subsequently extended to include query result caches, materialized/cached views, intermediate query results, and query rewriting, which have been extensively studied for relational database systems [11], [14], [15], [28]–[31], [34], [35] and streaming processing systems [19]–[21]. Group processing algorithms have proven to be effective in multiple applications involving high-load conditions [7], [8], [15], [19]–[21], [27]–[31], [34]–[37], [52], [53].

The shared execution strategy has attracted considerable attention in spatial databases because of its low processing cost. A series of batch shortest-path algorithms [27], [36], [43], [44], [52], [53] have been developed to evaluate group shortest-path queries in road networks efficiently. Zhang *et al*. [52] studied the batch processing of shortest-path queries in dynamic road networks in which road segments weights (e.g., travel times) frequently change. Recently, Cho developed shared execution techniques to evaluate ε-distance join queries effectively [7] and *k*NN join queries [8] in road networks. Owing to the inherent difference between farthest neighbor search and nearest neighbor search, applying these algorithms for the studies in [7], [8] to evaluate M*k*FN queries is difficult. Ali *et al*. [2] proposed group query processing techniques using the movement patterns of continuous queries on 3D object databases. Boinski and Zakrzewicz [5] presented a new method for the concurrent processing of multiple spatial collocation pattern discovery queries. However, the existing algorithms cannot be applied to evaluate M*k*FN queries in road networks. The farthest neighbor search algorithms in Section II.A focused on improving the efficiency of evaluating a single farthest neighbor query. They did not consider the use of shared computation among multiple queries. When multiple *k*FN queries arrive simultaneously, query scalability becomes an issue. Our proposed solution differs from existing studies in several aspects. First, it represents the first attempt to evaluate M*k*FN queries in road networks efficiently. Second, it uses a shared execution strategy to filter candidates while processing M*k*FN queries rapidly. Finally, it can be easily implemented using popular network distance algorithms [3], [23], [54] in road networks, which is highly desirable.

III. PRELIMINARIES

Definition 1 (kFN Query): For a positive integer k, a query point q, and a set of data points P, the *k*FN query retrieves a set $P_k(q)$ of k data points in P that are farthest from the query point q, i.e., $dist\left(q, p^+\right) \geq dist(q, p^-)$ for $\forall p^+ \in P_k(q)$ and $\forall p^- \in P - P_k(q)$.

Definition 2 (MkFN Query): For a set of query points Q, the M*k*FN query retrieves a set $P_k(q)$ of k data points farthest from each query point q in Q. For simplicity, we assume that each query point q requires the same number k of data points farthest from q. However, it is not difficult to consider the distinct number k_q of data points farthest from each query point q, which will be discussed in Section V.

Definition 3 (Road network): We represent a road network as an undirected 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.

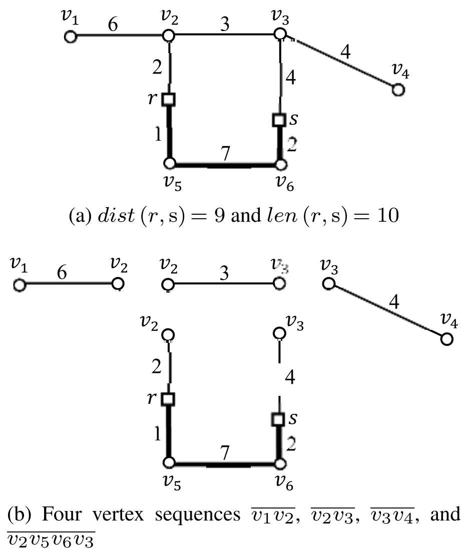
Definition 4 (Intersection, Intermediate, and Terminal Vertices): We divide vertices into three categories based on their degree. (1) If the degree is greater than or equal to three, then the vertex is referred to as an intersection vertex. (2) If the degree is two, then the vertex is an intermediate vertex. (3) If the degree is one, then the vertex is a terminal vertex.

Definition 5 (Vertex Sequence and Segment): A vertex sequence $\overline{v_l v_{l+1} \ldots v_m}$ denotes a path between two vertices v_l and v_m such that v_l (v_m) is either an intersection vertex or a terminal vertex, and then the other vertices in the path, $v_{l+1}, \ldots, v_{m-1}$ are intermediate vertices. The length of a vertex sequence is the total weight of the edges in the vertex sequence. One part of a vertex sequence is referred to as a segment. By definition, a vertex sequence is a segment.

Table 1 summarizes the typical symbols and notations used in this study. To simplify the presentation, we denote

TABLE 1. Definitions of symbols.

Symbol	Definition
k	Number of requested farthest neighbors
q	Query point
Q	Set of query points
P	Set of data points
$P_k(q)$	Set of k data points farthest from a query point q, i.e., $P_k(q)=\{p^+ \| dist\left(q,p^+\right) \geq dist(q,p^-)$ for $\forall p^- \in P - P_k(q)\}$
$dist\left(r,s\right)$	Length of the shortest path connecting two points r and s in the road network
$len\left(r,s\right)$	Length of the segment connecting two points r and s where both r and s are located 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, $v_{l+1}, \ldots, v_{m-1}$, are intermediate vertices
$\overline{q_i q_{i+1} \cdots q_j}$	Query segment that consists of query points $q_i, q_{i+1}, \ldots, q_j$ in a vertex sequence (in short, $\overline{q_i q_j}$)
$\overline{p_l p_{l+1} \cdots p_m}$	Data segment that consists of data points $p_l, p_{l+1}, \ldots, p_m$ in a vertex sequence (in short, $\overline{p_l p_m}$)
$maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$	Maximum distance between $\overline{q_i q_j}$ and $\overline{p_l p_m}$, i.e., $max\{dist(q,p) \| q \in \overline{q_i q_j}, p \in \overline{p_l p_m}\}$
$mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$	Minimum distance between $\overline{q_i q_j}$ and $\overline{p_l p_m}$, i.e., $min\{dist(q,p) \| q \in \overline{q_i q_j}, p \in \overline{p_l p_m}\}$
$\omega\left(q, \overline{p_l p_m}\right)=q^*$	Farthest point q^* of a query point q to all points in $\overline{p_l p_m}$ such that $maxdist\left(q, \overline{p_l p_m}\right) = dist\left(q, q^*\right)$ for $\exists q^* \in \overline{p_l p_m}$



(a) $dist\,(r, s) = 9$ and $len\,(r, s) = 10$

(b) Four vertex sequences $\overline{v_1v_2}$, $\overline{v_2v_3}$, $\overline{v_3v_4}$, and $\overline{v_2v_5v_6v_3}$

FIGURE 2. Difference between *dist* (*r*, s) and *len* (*r*, s).

$\overline{q_iq_{i+1}\cdots q_j}$ ($\overline{p_lp_{l+1}\cdots p_m}$) as $\overline{q_iq_j}$ ($\overline{p_lp_m}$), where query points $q_i, q_{i+1}, \ldots, q_j$ or data points $p_l, p_{l+1}, \cdots, p_m$ are located in the same vertex sequence. Figure 2 shows the difference between the network distance and the segment length between two points, r and s in a road network, where the numbers at the edges indicate the distance between two adjacent points (e.g., $dist\,(v_1, v_2) = 6$), as shown in Figure 2(a). The shortest path from r to s is $r \rightarrow v_2 \rightarrow v_3 \rightarrow s$, where the network distance between them is $dist\,(r, s) = 9$. The segment connecting r and s in a vertex sequence $\overline{v_2v_5v_6v_3}$ becomes $\overline{rv_5v_6s}$ with a length equal to $len\,(r, s) = 10$. Furthermore, $len\,(r, s)$ is defined only when both points r and s are in the same vertex sequence. Figure 2(b) shows how to disassemble a road network into four vertex sequences $\overline{v_1v_2}$, $\overline{v_2v_3}$, $\overline{v_3v_4}$, and $\overline{v_2v_5v_6v_3}$, where v_1 and v_4 are the terminal vertices, v_2 and v_3 are the intersection vertices, and v_5 and v_6 are the intermediate vertices, respectively.

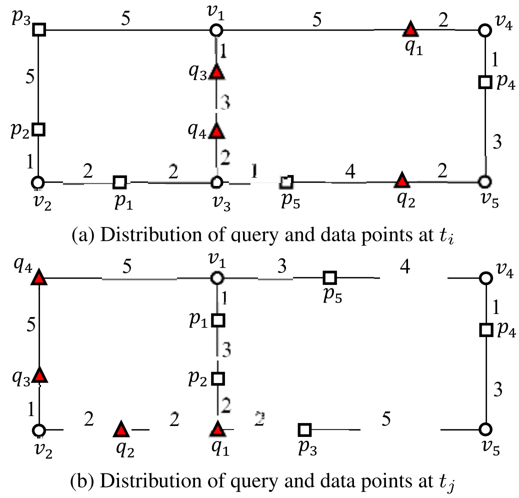
IV. GROUP PROCESSING OF MULTIPLE *K*-FARTHEST NEIGHBOR QUERIES IN ROAD NETWORKS

A. GROUPING QUERY AND DATA POINTS

In this section, we consider an MkFN query in a road network, which is shown in Figure 3. For $k = 2$, $Q = \{q_1, q_2, q_3, q_4\}$, and $P = \{p_1, p_2, p_3, p_4, p_5\}$, we consider a kFN query that retrieves two data points that are farthest from each query point q in Q.

Figure 3 shows the population of query and data points at timestamps t_i and t_j. Here, we assume that both the query and data points arbitrarily move along the road network. In this section, as shown in Figure 3(a), we focus on evaluating MkFN queries at timestamp t_i.

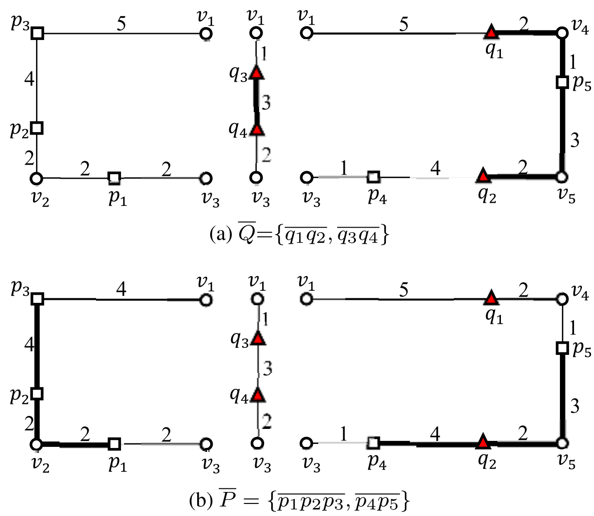
Figure 4 shows a sample grouping of adjacent query and data points. As shown in Figure 4(a), two query points q_1 and q_2 in a vertex sequence $\overline{v_1v_4v_5v_3}$ are grouped into a query segment $\overline{q_1q_2}$, whereas the other two query points q_3 and q_4 in a vertex sequence $\overline{v_1v_3}$ 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 segments $\overline{Q} = \{\overline{q_1q_2}, \overline{q_3q_4}\}$. Similarly, as shown in Figure 4(b). a set of data points $P = \{p_1, p_2, p_3, p_4, p_5\}$ can be transformed into a set of data segments $\overline{P} = \{\overline{p_1p_2p_3}, \overline{p_4p_5}\}$.



(a) Distribution of query and data points at t_i

(b) Distribution of query and data points at t_j

FIGURE 3. Population of query and data points at t_i and t_j.



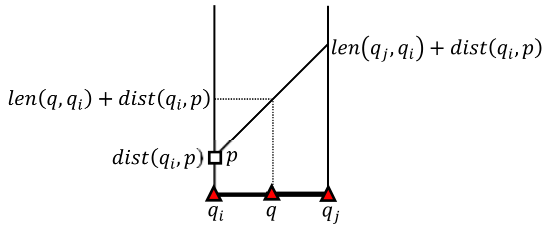
(a) $\overline{Q}=\{\overline{q_1q_2}, \overline{q_3q_4}\}$

(b) $\overline{P} = \{\overline{p_1p_2p_3}, \overline{p_4p_5}\}$

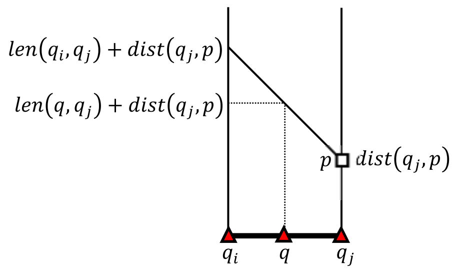
FIGURE 4. Grouping of adjacent query and data points.

B. COMPUTATION OF DISTANCE BETWEEN QUERY AND DATA SEGMENTS

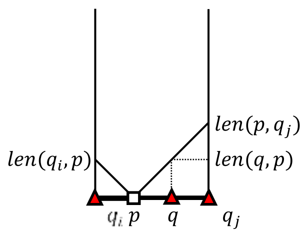
In this section, we describe the method to compute the minimum and maximum distances between a query segment $\overline{q_iq_j}$ and a data segment $\overline{p_lp_m}$ denoted by $mindist\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$ and $maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$, respectively. Note that $mindist\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$ and $maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$ are formally defined as $mindist\left(\overline{q_iq_j}, \overline{p_lp_m}\right) = min\{dist(q, p) | q \in \overline{q_iq_j}, p \in \overline{p_lp_m}\}$ and $maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right) = max\{dist(q, p) | q \in \overline{q_iq_j}, p \in \overline{p_lp_m}\}$.



(a) $dist\,(q, p) = len\,(q, q_i) + dist\,(q_i, p)$ for the path of $q \to q_i \to p$



(b) $dist\,(q, p) = len\,(q, q_j) + dist\,(q_j, p)$ for the path of $q \to q_j \to p$



(c) $dist\,(q, p) = len\,(q, p)$ for $p \in \overline{q_i q_j}$

FIGURE 5. Determination of distance from q to p, where $q \in \overline{q_i q_j}$.

Corollary 1: mindist $\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$ and *maxdist* $\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$ are the lower and upper bounds on the distance between a query point q and a data point p, respectively, where $\forall q \in \overline{q_i q_j}$ and $\forall p \in \overline{p_l p_m}$. Therefore, $mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) \leq dist(q, p) \leq maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$. ■

We describe the method to compute $mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$ and $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$. If $\overline{q_i q_j}$ and $\overline{p_l p_m}$ overlap (i.e., $\overline{q_i q_j} \cap \overline{p_l p_m} \neq \emptyset$), the minimum distance between $\overline{q_i q_j}$ and $\overline{p_l p_m}$ is $mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = 0$; otherwise, the minimum distance between them is $mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = min\left\{dist\,(q_i, p_l)\,, dist\,(q_i, p_m)\,, dist\,\left(q_j, p_l\right)\,, dist\,\left(q_j, p_m\right)\right\rangle$. Unlike the computation of $mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$, computing $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$ is not trivial. We first describe the method to compute the maximum distance between a query segment $\overline{q_i q_j}$ and a data point p in $\overline{p_l p_m}$. We investigate the distance $\mathrm{dist}\,(q, p)$ from a query point q in $\overline{q_i q_j}$ to a data point p.

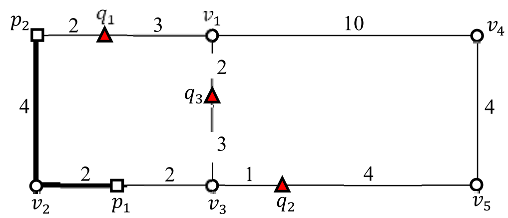
In Figure 5, assume that point q_i corresponds to the origin of the XY coordinate system. Subsequently, the Y-axis represents $dist\,(q, p)$ and the X-axis represents $len\,(q_i,\ q)$, where $q \in \overline{q_i q_j}$. As shown in Figure 5(a), if a path $q \to q_i \to p$ exists, then the distance from q to p is evaluated by $dist\,(q, p) = len\,(q, q_i) + dist\,(q_i, p)$. Similarly, as shown in Figure 5(b), if a path $q \to q_j \to p$ exists, then $dist\,(q, p)$ is evaluated by $dist\,(q, p) = len\left(q, q_j\right) + dist\left(q_j, p\right)$. If the data point p is located in $\overline{q_i q_j}$, then $dist\,(q, p)$ is evaluated by $dist\,(q, p) = len\,(q, p)$, as shown in Figure 5(c). Because $dist\,(q, p)$ is the length of the shortest path among multiple paths from q to p, it is computed as follows: If $p \notin \overline{q_i q_j}$, then $dist\,(q, p) = min\{len\,(q, q_i) + dist\,(q_i, p)\,, len\left(q, q_j\right) + dist\left(q_j, p\right)\}$; otherwise, $dist\,(q, p) = min\{len\,(q, q_i) + dist\,(q_i, p)\,, len\left(q, q_j\right) + dist\left(q_j, p\right), len\,(q, p)\}$.

For a data segment $\overline{p_l p_m}$ and a query point q, let $\omega\left(q, \overline{p_l p_m}\right) = q^*$ be the farthest point q^* of the query point q to all points in $\overline{p_l p_m}$. This indicates that a point q^* exists in $\overline{p_l p_m}$ such that $maxdist\left(q, \overline{p_l p_m}\right) = dist\,(q, q^*)$. Thus, we can easily locate q^* in $\overline{p_l p_m}$ using the linear equation $maxdist\left(q, \overline{p_l p_m}\right) = dist\,(q, q^*)$. Based on Figure 6, we compute the farthest point q^* from each query point $q \in \{q_1, q_2, q_3\}$ such that $maxdist\left(q, \overline{p_1 p_2}\right) = dist\,(q, q^*)$ for $\exists q^* \in \overline{p_1 p_2}$. Because $dist\,(q_1, p_1) = 8$, $dist\,(q_1, p_2) = 2$, $len\,(p_1, p_2) = 6$, and $dist\,(q_1, p_1) = dist\,(q_1, p_2) + len\,(p_2, p_1)$, we have $maxdist\left(q_1, \overline{p_1 p_2}\right) = 8$ and $\omega\left(q_1, \overline{p_1 p_2}\right) = p_1$, as shown in Figure 6(b). Similarly, because $dist\,(q_2, p_1) = 3$, $dist\,(q_2, p_2) = 9$, $len\,(p_1, p_2) = 6$, and $dist\,(q_2, p_2) = dist\,(q_2, p_1) + len\,(p_1, p_2)$, we have $maxdist\left(q_2, \overline{p_1 p_2}\right) = 9$ and $\omega\left(q_2, \overline{p_1 p_2}\right) = p_2$, as shown in Figure 6(c). Because $dist\,(q_3, p_1) = 5$, $dist\,(q_3, p_2) = 7$, and $len\,(p_1, p_2) = 6$, as shown in Figure 6(d), the maximum distance between q_3 and $\overline{p_1 p_2}$ is evaluated as $maxdist\left(\overline{p_1 p_2}, q_3\right) = 9$, and the farthest point of q_3 is marked as q_3^*. The dash-dotted lines in Figure 6 shows the lengths of redundant paths are not the shortest path.

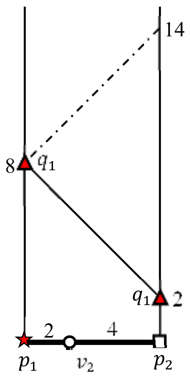
Figure 7 shows the process of computing *maxdist* $\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$. This process operates in two phases, which correspond to Figures 7(a) and 7(b). In the first phase, we obtain the farthest point q_i^* (q_j^*) of q_i (q_j) such that $maxdist\left(q_i, \overline{p_l p_m}\right) = dist\left(q_i, q_i^*\right)$ ($maxdist\left(q_j, \overline{p_l p_m}\right) = dist\left(q_j, q_j^*\right)$), i.e., $\omega\left(q_i, \overline{p_l p_m}\right) = q_i^*$ ($\omega\left(q_j, \overline{p_l p_m}\right) = q_j^*$), as shown in Figure 7(a). In the second phase, we compute $maxdist\left(\overline{q_i q_j}, q_i^*\right)$ and $maxdist\left(\overline{q_i q_j}, q_j^*\right)$, as shown in Figure 7(b), where points q_i^{**} (q_j^{**}) indicate the farthest point of q_i^* (q_j^*) such that $maxdist\left(\overline{q_i q_j}, q_i^*\right) = dist(q_i^{**}, q_i^*)$ ($maxdist\left(\overline{q_i q_j}, q_j^*\right) = dist(q_j^{**}, q_j^*)$), i.e., $\omega\left(\overline{q_i q_j}, q_i^*\right) = q_i^{**}$ ($\omega\left(\overline{q_i q_j}, q_j^*\right) = q_j^{**}$), as shown in Figure 7(b). Note that q_i^* and q_j^* belong to a data segment $\overline{p_l p_m}$, whereas q_i^{**} and q_j^{**} belong to a query segment $\overline{q_i q_j}$.

Lemma 1 proves that $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = max\,\{\,dist\left(q_i^*, q_i^{**}\right), dist\left(q_j^*, q_j^{**}\right)\}$, where $q_i^* = \omega\left(q_i, \overline{p_l p_m}\right)$, $q_i^{**} = \omega\left(q_i^*, \overline{q_i q_j}\right)$, $q_j^* = \omega\left(q_j, \overline{p_l p_m}\right)$, and $q_j^{**} = \omega\left(q_j^*, \overline{q_i q_j}\right)$, as shown in Figure 7.

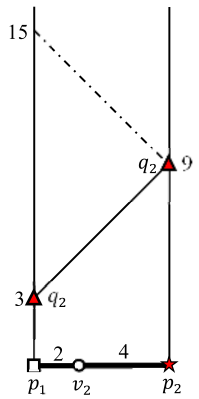
Lemma 1: maxdist $\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = max\{dist\left(q_i^*, q_i^{**}\right), dist\left(q_j^*, q_j^{**}\right)\}$, where $q_i^* = \omega\left(q_i, \overline{p_l p_m}\right)$, $q_i^{**} = \omega\left(q_i^*, \overline{q_i q_j}\right)$, $q_j^* = \omega\left(q_j, \overline{p_l p_m}\right)$, and $q_j^{**} = \omega\left(q_j^*, \overline{q_i q_j}\right)$.



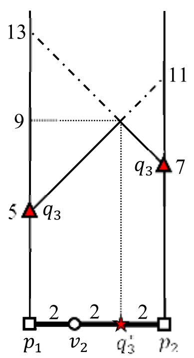
(a) Data segment $\overline{p_1 p_2}$ and three query points q_1, q_2, and q_3



(b) $maxdist\left(q_1, \overline{p_1 p_2}\right) = 8$

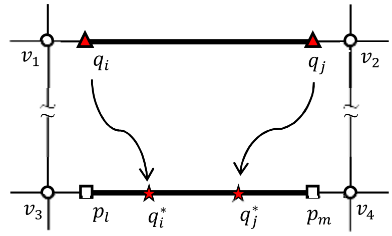


(c) $maxdist\left(q_2, \overline{p_1 p_2}\right) = 9$

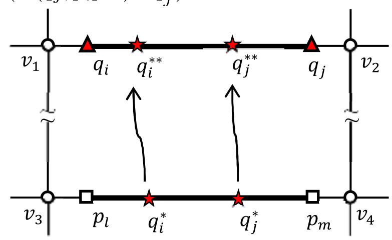


(d) $maxdist\left(q_3, \overline{p_1 p_2}\right) = 9$

FIGURE 6. Evaluation of $maxdist\left(q_1, \overline{p_1 p_2}\right)$, $maxdist\left(q_2, \overline{p_1 p_2}\right)$, and $maxdist\left(q_3, \overline{p_1 p_2}\right)$.



(a) Finding $\omega\left(q_i, \overline{p_l p_m}\right) = q_i^*$ $\left(\omega\left(q_j, \overline{p_l p_m}\right) = q_j^*\right)$



(b) Evaluation of $dist\left(q_i^*, q_i^{**}\right)$ and $dist\left(q_j^*, q_j^{**}\right)$

FIGURE 7. $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = max\{dist\left(q_i^*, q_i^{**}\right), dist(q_j^*, q_j^{**})\}$.

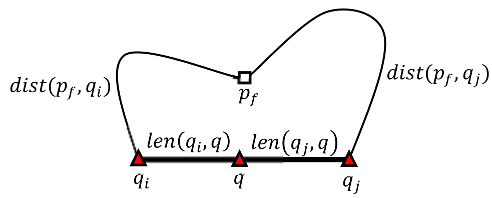


FIGURE 8. $dist\left(q, p_f\right) = min\{dist\left(p_f, q_i\right) + len\left(q_i, q\right), dist\left(p_f, q_j\right) + len\left(q_j, q\right)\}$.

Proof: We prove this lemma by contradiction. The maximum distance between two segments $\overline{q_i q_j}$ and $\overline{p_l p_m}$ can be represented by $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = max\left\{maxdist\left(\overline{q_i q_j}, p\right) \middle| maxdist\left(\overline{q_i q_j}, p\right) p \in \overline{p_l p_m}\right\rangle p \in \overline{p_l p_m}\rangle$. We assume that the farthest point p_f in $\overline{p_l p_m}$ exists such that $maxdist\left(\overline{q_i q_j}, p_f\right) > max\{maxdist\left(\overline{q_i q_j}, q_i^*\right), maxdist\left(\overline{q_i q_j}, q_j^*\right)\}$.

Generally, $maxdist\left(\overline{q_i q_j}, p_f\right) = max\{dist\left(q, p_f\right) | q \in \overline{q_i q_j}\}$, $maxdist\left(\overline{q_i q_j}, q_i^*\right) = max\{dist\left(q, q_i^*\right) | q \in \overline{q_i q_j}\}$, and $maxdist\left(\overline{q_i q_j}, q_j^*\right) = max\{dist\left(q, q_j^*\right) | q \in \overline{q_i q_j}\}$, As shown in Figure 8, the shortest path from p_f to $q \in \overline{q_i q_j}$ is either $p_f \rightarrow q_i \rightarrow q$ or $p_f \rightarrow q_j \rightarrow q$, and the distance from p_f to q is represented by $dist\left(q, p_f\right) = min\{dist\left(p_f, q_i\right) + len\left(q_i, q\right), dist\left(p_f, q_j\right) + len\left(q_j, q\right)\}$.

Based on the assumption that $maxdist\left(\overline{q_i q_j}, p_f\right) > max\left\{maxdist\left(\overline{q_i q_j}, q_i^*\right), maxdist\left(\overline{q_i q_j}, q_j^*\right)\right\}$, a point $q_x \in \overline{q_i q_j}$ exists such that $maxdist\left(\overline{q_i q_j}, p_f\right) = dist(q_x, p_f)$ and $dist\left(q_x, p_f\right) > max\left\{dist\left(q_x, q_i^*\right), dist\left(q_x, q_j^*\right)\right\}$. If the shortest path from p_f to q_x is $p_f \rightarrow q_i \rightarrow q_x$, then $dist\left(q_x, p_f\right) > dist\left(q_x, q_i^*\right)$, indicating $dist\left(q_i, p_f\right) > dist\left(q_i, q_i^*\right)$, this contradicts the given condition that

$q_i^* = \omega\left(q_i, \overline{p_l p_m}\right)$. Similarly, if the shortest path from p_f to q_x is $p_f \rightarrow q_j \rightarrow q_x$, then $dist\left(q_x, p_f\right) > dist\left(q_x, q_j^*\right)$, indicating $dist\left(q_j, p_f\right) > dist\left(q_j, q_j^*\right)$, this contradicts the given condition that $q_j^* = \omega\left(q_j, \overline{p_l p_m}\right)$. Therefore, no farthest point p_f exists such that $maxdist\left(\overline{q_i q_j}, p_f\right) > max\{maxdist\left(\overline{q_i q_j}, q_i^*\right), maxdist\left(\overline{q_i q_j}, q_j^*\right)\}$. Consequently, $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) = max\left\{dist\left(q_i^*, q_i^{**}\right), dist\left(q_j^*, q_j^{**}\right)\right\}$, where $q_i^* = \omega\left(q_i, \overline{p_l p_m}\right)$, $q_i^{**} = \omega\left(q_i^*, \overline{q_i q_j}\right)$, $q_j^* = \omega\left(q_j, \overline{p_l p_m}\right)$, and $q_j^{**} = \omega\left(q_j^*, \overline{q_i q_j}\right)$. ■

Returning to the example in Figure 4, we compute the maximum distance between a query segment $\overline{q_i q_j}$ and a data segment $\overline{p_l p_m}$, where $\overline{q_i q_j} \in \{\overline{q_1 q_2}, \overline{q_3 q_4}\}$ and $\overline{p_l p_m} \in \{\overline{p_1 p_2 p_3}, \overline{p_4 p_5}\}$. Specifically, we evaluate $maxdist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right)$, $maxdist\left(\overline{q_1 q_2}, \overline{p_4 p_5}\right)$, $maxdist\left(\overline{q_3 q_4}, \overline{p_1 p_2 p_3}\right)$, and $maxdist\left(\overline{q_3 q_4}, \overline{p_4 p_5}\right)$ in this order.

Figure 9 shows the method to compute $maxdist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right)$. In the first phase, Figures 9(a) and (b), we determine the farthest point q_1^* (q_2^*) of a query point q_1 (q_2) among $\overline{p_1 p_2 p_3}$, i.e., $q_1^* = \omega\left(q_1, \overline{p_1 p_2 p_3}\right)$ ($q_2^* = \omega\left(q_2, \overline{p_1 p_2 p_3}\right)$). In the second phase, as show in Figures 9(c) and (d), we compute $maxdist(q_1^*, \overline{q_1 q_2})$ ($maxdist(q_2^*, \overline{q_1 q_2})$) and then compute $maxdist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right) = max\{maxdist\left(q_1^*, \overline{q_1 q_2}\right), maxdist(q_2^*, \overline{q_1 q_2})\}$.

As shown in Figure 9(a), because query point q_1 is not located in the data segment $\overline{p_1 p_2 p_3}$ and the length of the shortest path from q_1 to p_1 (p_3) is $dist\left(q_1, p_1\right) = 13$ ($dist\left(q_1, p_3\right) = 9$), the maximum distance between query point q_1 and data segment $\overline{p_1 p_2 p_3}$ is evaluated as $maxdist\left(q_1, \overline{p_1 p_2 p_3}\right) = 15$. As shown in Figure 9(b), because query point q_2 is not located in the data segment $\overline{p_1 p_2 p_3}$ and the length of the shortest path from q_2 to p_1 (p_3) is $dist\left(q_2, p_1\right) = 7$ ($dist\left(q_2, p_3\right) = 15$), the maximum distance between query point q_2 and data segment $\overline{p_1 p_2 p_3}$ is evaluated as $maxdist\left(q_2, \overline{p_1 p_2 p_3}\right) = 15$. Consequently, the farthest point of q_1 among $\overline{p_1 p_2 p_3}$ is $q_1^* = \omega\left(q_1, \overline{p_1 p_2 p_3}\right) = v_2$, and the farthest point of q_2 among $\overline{p_1 p_2 p_3}$ is $q_2^* = \omega\left(q_2, \overline{p_1 p_2 p_3}\right) = p_3$.

As shown in Figure 9(c), because the point q_1^* is not located in the query segment $\overline{q_1 q_2}$ and the length of the shortest path from q_1^* to q_1 (q_2) is $dist\left(q_1^*, q_1\right) = 15$ ($dist\left(q_1^*, q_2\right) = 9$), the maximum distance between query segment $\overline{q_1 q_2}$ and a point q_1^* that matches a vertex v_2 is evaluated as $maxdist\left(q_1^*, \overline{q_1 q_2}\right) = 16$. As shown in Figure 9(d), because the point q_2^* is not located in the query segment $\overline{q_1 q_2}$ and the length of the shortest path from q_2^* to q_1 (q_2) is $dist\left(q_2^*, q_1\right) = 9$ ($dist\left(q_2^*, q_2\right) = 15$), the maximum distance between query segment $\overline{q_1 q_2}$ and a point q_2^* that matches a data point p_3 is evaluated as $maxdist\left(q_2^*, \overline{q_1 q_2}\right) = 16$. Consequently, the maximum distance between $\overline{q_1 q_2}$ and $\overline{p_1 p_2 p_3}$ is $maxdist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right) = max\left\{maxdist\left(q_1^*, \overline{q_1 q_2}\right), maxdist\left(q_2^*, \overline{q_1 q_2}\right)\right\} = \{16, 16\} = 16$. Furthermore, as shown in Figure 9, we compute $maxdist\left(\overline{q_1 q_2}, \overline{p_4 p_5}\right)$, $maxdist\left(\overline{q_3 q_4}, \overline{p_1 p_2 p_3}\right)$, and $maxdist\left(\overline{q_3 q_4}, \overline{p_4 p_5}\right)$. Table 2 summarizes the minimum and maximum distances between $\overline{q_i q_j}$ and $\overline{p_l p_m}$, where $\overline{q_i q_j} \in \{\overline{q_1 q_2}, \overline{q_3 q_4}\}$ and $\overline{p_l p_m} \in \{\overline{p_1 p_2 p_3}, \overline{p_4 p_5}\}$.

TABLE 2. Computation of minimum and maximum distances between $\overline{q_i q_j}$ and $\overline{p_l p_m}$.

$mindist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$	$maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right)$
$mindist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right) = 7$	$maxdist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right) = 16$
$mindist\left(\overline{q_1 q_2}, \overline{p_4 p_5}\right) = 0$	$maxdist\left(\overline{q_1 q_2}, \overline{p_4 p_5}\right) = 12$
$mindist\left(\overline{q_3 q_4}, \overline{p_1 p_2 p_3}\right) = 4$	$maxdist\left(\overline{q_3 q_4}, \overline{p_1 p_2 p_3}\right) = 10$
$mindist\left(\overline{q_3 q_4}, \overline{p_4 p_5}\right) = 3$	$maxdist\left(\overline{q_3 q_4}, \overline{p_4 p_5}\right) = 12$

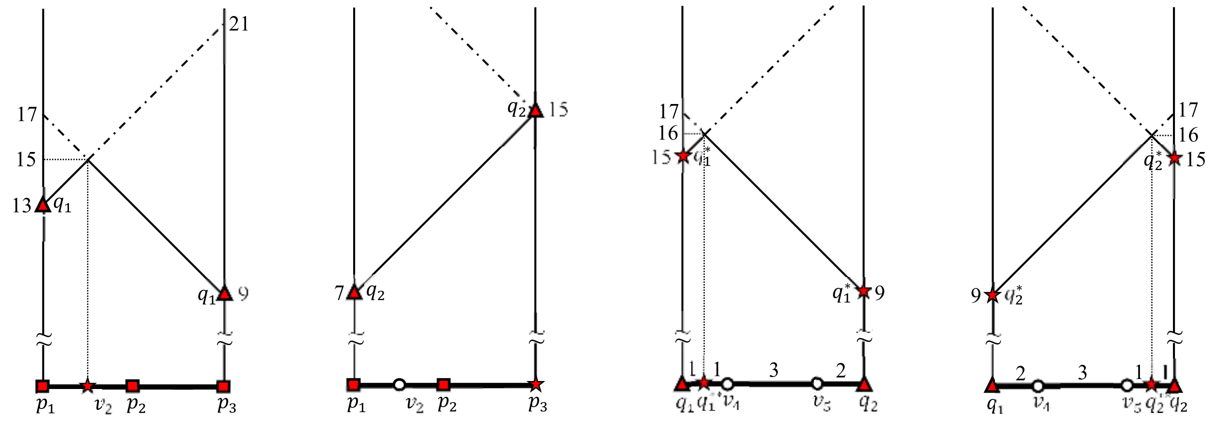
C. SORTING DATA SEGMENTS BY THE MAXIMUM DISTANCE

Figure 10 shows the sorting of $\overline{p_1 p_2 p_3}$ and $\overline{p_4 p_5}$ in $\overline{P}$ for each query segment $\overline{q_i q_j}$. Specifically, the two data segments were sorted and plotted in the decreasing order of the maximum distance to a query segment. If data segments with the same maximum distance are discovered, they are re-sorted in the decreasing order of their minimum distance. As shown in Figure 10(a), $\overline{p_1 p_2 p_3}$ and $\overline{p_4 p_5}$ are sorted in the decreasing order, as represented by $\langle\overline{p_1 p_2 p_3}, \overline{p_4 p_5}\rangle$, and then sequentially processed for $\overline{q_1 q_2}$. This is because the maximum distance of $\overline{p_1 p_2 p_3}$ to $\overline{q_1 q_2}$ (i.e., $maxdist\left(\overline{q_1 q_2}, \overline{p_1 p_2 p_3}\right) = 16$) is larger than the maximum distance of $\overline{p_4 p_5}$ (i.e., $maxdist\left(\overline{q_1 q_2}, \overline{p_4 p_5}\right) = 12$). Similarly, as shown in Figure 10(b), $\overline{p_4 p_5}$ and $\overline{p_1 p_2 p_3}$ are sorted in the decreasing order, as represented by $\langle\overline{p_1 p_2 p_3}, \overline{p_4 p_5}\rangle$, and then sequentially processed for $\overline{q_3 q_4}$. This is because the maximum distance of $\overline{p_4 p_5}$ to $\overline{q_3 q_4}$ (i.e., $maxdist\left(\overline{q_3 q_4}, \overline{p_4 p_5}\right) = 12$) is larger than the maximum distance of $\overline{p_1 p_2 p_3}$ (i.e., $maxdist\left(\overline{q_3 q_4}, \overline{p_1 p_2 p_3}\right) = 10$).

V. GMP ALGORITHM

Algorithm 1 describes the GMP algorithm for M*k*FN search in road networks. The result set $\Omega(Q)$ is initialized to an empty set (line 1). In the first step (lines 2–4), adjacent query points $q_i, q_{i+1}, \cdots, q_j$ and adjacent data points $p_l, p_{l+1}, \cdots, p_m$ in a vertex sequence are grouped into a query segment $\overline{q_i q_j}$ and data segment $\overline{p_l p_m}$, respectively. Therefore, as explained in Section IV, a set of query points Q and a set of data points P are converted to a set of query segments $\overline{Q}$ and a set of data segments $\overline{P}$, respectively. The M*k*FN search for a query segment $\overline{q_i q_j}$ is performed to identify the k farthest data points of each query point in $\overline{q_i q_j}$ (line 7). The result $\Omega(\overline{q_i q_j})$ of the M*k*FN search for $\overline{q_i q_j}$ is added to the query result, where $\Omega(\overline{q_i q_j}) = \{\langle q, P_k(q)\rangle \mid q \in \overline{q_i q_j}\}$ (line 8). Then, the query result $\Omega(Q)$ is returned after the M*k*FN search for all query segments is performed (line 9).

Algorithm 2 describes the M*k*FN search algorithm for obtaining the k farthest data points of each query point in $\overline{q_i q_j}$. The M*k*FN search algorithm sequentially traverses each of the sorted data segments in $\overline{P}$. Furthermore, all data segments in $\overline{P}$ are sorted in the decreasing order of their



(a) (b) (c) (d)

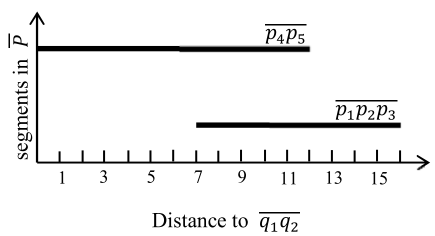
FIGURE 9. ***maxdist*** $(\overline{q_1q_2}, \overline{p_1p_2p_3}) = 16$. **(a)** ***maxdist*** $(q_1, \overline{p_1p_2p_3}) = 15$ **and** $q_1^* = v_2$ **(b)** ***maxdist*** $(q_2, \overline{p_1p_2p_3}) = 15$ **and** $q_2^* = p_3$ **(c)** ***maxdist*** $\left(q_1^*, \overline{q_1q_2}\right) = 16$ **(d)** ***maxdist*** $\left(q_2^*, \overline{q_1q_2}\right) = 16$.

Algorithm 1 GMP (Q, P)

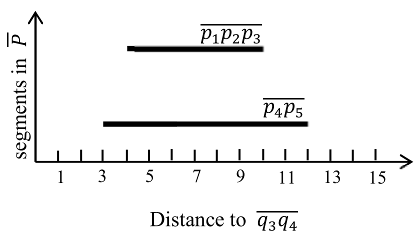
Input: Q: set of query points, P: set of data points

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

1: $\Omega(Q) \leftarrow \emptyset$ // the result set $\Omega(Q)$ is initialized to the empty set.
2: // adjacent points in a vertex sequence are grouped into a segment, which is explained in Section IV.A.
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: $\overline{P} \leftarrow$ *group_points*(P) // adjacent data points $p_l, p_{l+1}, \cdots, p_m$ in a vertex sequence are grouped into $\overline{p_lp_m}$.
5: // farthest data points from each query point in $\overline{q_iq_j}$ are retrieved, which is detailed in Algorithm 2.
6: **for** each query segment $\overline{q_iq_j} \in \overline{Q}$ **do**
7: $\quad\Omega(\overline{q_iq_j}) \leftarrow$ *MkFN_search*($\overline{q_iq_j}$, $\overline{P}$) // $\Omega(\overline{q_iq_j}) = \{\langle q, P_k(q)\rangle \mid q \in \overline{q_iq_j}\}$
8: $\quad\Omega(Q) \leftarrow \Omega(Q) \cup \Omega(\overline{q_iq_j})$ // the result for each query point in $\overline{q_iq_j}$ is added to $\Omega(Q)$
9: **return** $\Omega(Q)$ // $\Omega(Q)$ is returned after the M*k*FN search for all query segments in $\overline{Q}$ is executed.



(a) Sorting data segments for $\overline{q_1q_2}$, i.e., $\langle\overline{p_1p_2p_3}, \overline{p_4p_5}\rangle$



(b) Sorting data segments for $\overline{q_3q_4}$, i.e., $\langle\overline{p_4p_5}, \overline{p_1p_2p_3}\rangle$

FIGURE 10. Sorting data segments in decreasing order of their maximum distance to each query segment.

maximum distance to $\overline{q_iq_j}$. First, the result set $\Omega\left(\overline{q_iq_j}\right)$ is initialized to an empty set. Three general cases are then formally considered depending on the number of query points in $\overline{q_iq_j}$, i.e., $\left|\overline{q_iq_j}\right| = 1$, $\left|\overline{q_iq_j}\right| = 2$, and $\left|\overline{q_iq_j}\right| \geq 3$, where $\left|\overline{q_iq_j}\right|$ returns the number of query points in $\overline{q_iq_j}$. Thus, $\left|\overline{q_iq_j}\right| = 1$ denotes $\overline{q_iq_j}$ contains only a single query point q_i, $\left|\overline{q_iq_j}\right| = 2$ denotes $\overline{q_iq_j}$ contains only two query points q_i and q_j, and $\left|\overline{q_iq_j}\right| \geq 3$ denotes $\overline{q_iq_j}$ contains more than three query points $q_i, q_{i+1}, \ldots, q_j$. If $\left|\overline{q_iq_j}\right| = 1$. Subsequently, the *k*FN query from q_i is evaluated, which is detailed in Algorithm 3. The query result $P_k(q_i)$ for q_i is obtained and returned (lines 6−9). Similarly, if $\left|\overline{q_iq_j}\right| = 2$, then two *k*FN queries from q_i and q_j are evaluated. The query results $P_k(q_i)$ and $P_k(q_j)$ are obtained for q_i and q_j, respectively, and their union, $\Omega\left(\overline{q_iq_j}\right) = \{\langle q_i, P_k(q_i)\rangle\} \cup \{\left\langle q_j, P_k\left(q_j\right)\right\rangle\}$) is returned (lines 10−14). Finally, if $\left|\overline{q_iq_j}\right| \geq 3$, then two *k*FN queries from q_i and q_j are evaluated and their results are saved to $P_k(q_i)$ and $P_k(q_j)$, respectively. Note that the third argument of the *k*FN_search function in lines 19 and 20 is set to the length $len(q_i, q_j)$ of the query segment rather than 0 as in lines 8, 12, and 13. The k farthest data points of each query point q in $\overline{q_iq_j}$ is retrieved from candidate data points in $P_{k_{max}}(q_i) \cup P_{k_{max}}(q_j)$ (lines 15−25), as explained in Algorithm 4.

Algorithm 3 then describes the *k*FN search algorithm for finding candidate data points farthest from q. If the third input

Algorithm 2 *MkFN_search* ($\overline{q_iq_j}$, $\overline{P}$)

Input: $\overline{q_iq_j}$: query segment, $\overline{P}$: set of data segments
Output: $\Omega\left(\overline{q_iq_j}\right)$: set of ordered pairs of each query point q in $\overline{q_iq_j}$ and its query result, i.e., $\Omega(\overline{q_iq_j}) = \{\langle q, P_k(q)\rangle \mid q \in \overline{q_iq_j}\}$.
1: // the maximum and minimum distances from $\overline{q_iq_j}$ to each data segment in $\overline{P}$ are computed as explained in Section IV.B.
2: **for** each data segment $\overline{p_lp_m} \in \overline{P}$ **do**
3: compute *maxdist* $\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$ and *mindist* $\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$
4: // the data segments in $\overline{P}$ are sorted in a decreasing order of their maximum distance to $\overline{q_iq_j}$ as explained in Section IV.C.
5: $\overline{P} \leftarrow sort\_by\_dec\_order(\overline{P})$ // $\overline{P}$ contains a set of the sorted data segments for $\overline{q_iq_j}$.
6: **if** $\left|\overline{q_iq_j}\right| = 1$ **then**
7: // $\left|\overline{q_iq_j}\right| = 1$ denotes that $\overline{q_iq_j}$ consists of a query point q_i, i.e., $q_i = q_j$.
8: $P_{k_i}(q_i) \leftarrow k\text{FN\_search}(q_i, k_i, 0, \overline{P})$ // kFN search from q_i is performed and its result is saved to $P_{k_i}(q_i)$.
9: **return** $\{\langle q_i, P_{k_i}(q_i)\rangle\}$ // query result $P_{k_i}(q_i)$ is returned for q_i.
10: **else if** $\left|\overline{q_iq_j}\right| = 2$ **then**
11: // $\left|\overline{q_iq_j}\right| = 2$ denotes that $\overline{q_iq_j}$ consists of two query points q_i and q_j.
12: $P_{k_i}(q_i) \leftarrow k\text{FN\_search}(q_i, k_i, 0, \overline{P})$ // kFN search from q_i is performed and its result is saved to $P_{k_i}(q_i)$.
13: $P_{k_j}(q_j) \leftarrow k\text{FN\_search}(q_j, k_j, 0, \overline{P})$ // kFN search from q_j is performed and its result is saved to $P_{k_j}(q_j)$.
14: **return** $\{\langle q_i, P_{k_i}(q_i)\rangle\} \cup \{\langle q_j, P_{k_j}(q_j)\rangle\}$ // the union of query results, $P_{k_i}(q_i) \cup P_{k_j}(q_j)$, is returned for q_i and q_j.
15: **else if** $\left|\overline{q_iq_j}\right| \geq 3$ **then**
16: // $\left|\overline{q_iq_j}\right| \geq 3$ denotes that $\overline{q_iq_j}$ consists of more than three query points.
17: $\Omega(\overline{q_iq_j}) \leftarrow \emptyset$ // the result set $\Omega(\overline{q_iq_j})$ is initialized to the empty set..
18: $k_{max} \leftarrow max\{k_i, k_{i+1}, \cdots, k_j\}$ // assume that q_n requests k_n farthest data points for $i \leq n \leq j$.
19: $P_{k_{max}}(q_i) \leftarrow k\text{FN\_search}(q_i, k_{max}, len(q_i, q_j), \overline{P})$ // kFN search is performed for q_i with k_{max}.
20: $P_{k_{max}}(q_j) \leftarrow k\text{FN\_search}(q_j, k_{max}, len(q_i, q_j), \overline{P})$ // kFN search is performed for q_j with k_{max}.
21: // k FNs of query points $q_i, q_{i+1}, \cdots, q_j$ are retrieved from $P_k(q_i) \cup P_k(q_i)$.
22: **for** each query point $q \in \overline{q_iq_j}$ **do**
23: $P_k(q) \leftarrow choose\_k\text{FN}(q, k, P_{k_{max}}(q_i) \cup P_{k_{max}}(q_j))$ // *choose_k*FN is explained in Algorithm 4.
24: $\Omega\left(\overline{q_iq_j}\right) \leftarrow \Omega\left(\overline{q_iq_j}\right) \cup \{\langle q, P_k(q)\rangle\}$ // each query result for $q \in \overline{q_iq_j} - \{q_i, q_j\}$ is added to $\Omega\left(\overline{q_iq_j}\right)$.
25: **return** $\Omega\left(\overline{q_iq_j}\right)$ // the union of query results is returned for $q_i, q_{i+1}, \cdots, q_j$.

argument $l = 0$, then the kFN search function produces a set of k farthest data points for a query point q only. Otherwise (i.e., $l > 0$), the search function produces a set of candidate data points for all query points in $\overline{q_iq_j}$. Data segments are traversed in descending order based on maximum distance to q. Therefore, the data segments are explored in descending order based on the maximum distance to q. Furthermore, *prundist* is initialized to 0 and holds the difference between the distance from q to the current kth FN candidate and the segment length, i.e., $prundist = dist(q, p_{kth}) - l$. Note that *prundist* is used as the sentinel to determine whether the algorithm should be terminated (line 6). If the value of *prundist* is larger than the maximum distance of the current data segment $\overline{p_lp_m}$ to be analyzed, the algorithm terminates with the candidate set $P_k(q)$; otherwise, it analyzes whether each data point p in $\overline{p_lp_m}$ is a candidate for either the query point q when the segment length $l = 0$ or the query segment $\overline{q_iq_j}$ when $l > 0$. If $dist(q, p) \geq prundist$, then the data point p is included in the candidate set $P_k(q)$ and the *prundist* value is accordingly updated, as shown in line 12. Furthermore, if $|P_k(q)| < k$, $prundist = 0$. If $maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$ is less than *prundist* (lines 5−6) the data segments in $\overline{P}$ are investigated (lines 4−11), the algorithm returns the candidate set $P_k(q)$ that includes k data points farthest from q and then terminates. In Corollary 2, if $maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right) < prundist$, then the remaining unexplored data segments can be safely neglected for a query segment $\overline{q_iq_j}$.

Corollary 2: If $maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right) < prundist$, no data point p in $\overline{p_lp_m}$ belongs to a set P_k of the kFNs of q because $maxdist\left(q, \overline{p_lp_m}\right) \leq maxdist\left(\overline{q_iq_j}, \overline{p_lp_m}\right)$ holds. Note that, as shown in line 12, *prundist* is determined by $prundist = dist(q, p_{kth}) - l$. Therefore, data points in the remaining data segments can be safely disregarded because the maximum distance of these data segments is less than the distance to the kth FN candidate. ■

Algorithm 4 describes the process to determine the k farthest data points of q among the candidate data points in Σ_q that correspond to $\Sigma_q = P_{k_{max}}(q_i) \cup P_{k_{max}}(q_j)$. The set of kFNs of query point q, $P_k(q)$ is initialized to an empty set. The distance from q to a candidate data point p is computed as per the condition of p (lines 3−5). If $p \notin \overline{q_iq_j}$, then the distance from q to p is $dist(q, p) = min\{len(q, q_i) + dist(q_i, p), len(q, q_j) + dist(q_j, p)\}$. Otherwise (i.e., $p \in \overline{q_iq_j}$), the distance is $dist(q, p) = min\{len(q, q_i) + dist(q_i, p), len(q, q_j) + dist(q_j, p), len(q, p)\}$. After computing $dist(q, p)$, we can determine whether p will be added to the result set $P_k(q)$. If $|P_k(q)| < k$, then p is added to $P_k(q)$ (lines 7−8). If $|P_k(q)| = k$ and $dist(q, p) > dist(q, p_{kth})$, then p is added to $P_k(q)$ and p_{kth} is removed from $P_k(q)$. Here, p_{kth} denotes the current kth FN from q,

Algorithm 3 kFN_search $(q, k, l, \overline{P})$

Input: q: query point, k: number of requested FNs, l: segment length, $\overline{P}$: set of sorted data segments
Output: $P_k(q)$: set of farthest data points from q with consideration of the number k of requested FNs and offset distance l

1: // it explores sorted data segments sequentially to find data points farthest from q as early as possible.
2: $P_k(q) \leftarrow \emptyset$ // $P_k(q)$ keeps a set of candidate data points farthest from q.
3: $prundist \leftarrow 0$ // $prundist$ is used to determine whether to explore the other data segments.
4: **for** each data segment $\overline{p_l p_m} \in \overline{P}$ **do**
5: // assume that $\overline{q_i q_j}$ is the query segment containing the query point q.
6: **if** $maxdist\left(\overline{q_i q_j}, \overline{p_l p_m}\right) < prundist$ **then**
7: **go to** line 14 // the other data segments can be safely ignored according to Corollary 2.
8: **else**
9: **for** each data point $p \in \overline{p_l p_m}$ **do**
10: **if** $dist\ (q, p) \geq prundist$ **then**
11: $P_k\ (q) \leftarrow P_k\ (q) \cup \{p\}$ // if $dist\ (q, p) \geq prundist$, a data point p is included in $P_k\ (q)$.
12: $prundist \leftarrow dist\ (q, p_{kth}) - l$ // p_{kth} is the current kth FN among the candidates in $P_k\ (q)$.
13: // it removes redundant data points p from $P_k\ (q)$ because they cannot be included in $P_k(q)$ for all query points in $\overline{q_i q_j}$.
14: **for** each data point $p \in P_k(q)$ **do**
15: // p is removed from $P_k(q)$ if $dist\ (q, p) < prundist$.
16: **if** $dist\ (q, p) < prundist$ **then**
17: $P_k\ (q) \leftarrow P_k\ (q) - \{p\}$ // p cannot be a candidate for any query points in $\overline{q_i q_j}$.
18: **return** $P_k(q)$

Algorithm 4 *choose_k*FN(q, k, Σ_q)

Input: q: query point in $\overline{q_i q_j}$, k: number of farthest data points retrieved for q, Σ_q: set of candidate data points for q
Output: $P_k(q)$: set of k data points farthest from q

1: $P_k(q) \leftarrow \emptyset$ // $P_k(q)$ is initialized to the empty set.
2: **for** each data point $p \in \Sigma_q$ **do**
3: // step 1: $dist(q, p)$ is computed according to the condition of p.
4: **if** $p \notin \overline{q_i q_j}$ **then**
5: $dist(q, p) \leftarrow min\{len\ (q, q_i) + dist\ (q_i, p)\ , len\left(q, q_j\right) + dist\left(q_j, p\right)\}$ // see Figure 5.
6: **else**
7: $dist(q, p) \leftarrow min\{len\ (q, q_i) + dist\ (q_i, p)\ , len\left(q, q_j\right) + dist\left(q_j, p\right), len\ (q, p)\}$ // see Figure 5.
8: // step 2: p is added to $P_k(q)$ if it satisfies either of the two conditions below.
9: **if** $|P_k(q)| < k$ **then**
10: $P_k(q) \leftarrow P_k(q) \cup \{p\}$
11: **else if** $|P_k(q)| = k$ **and** $dist\ (q, p) > dist(q, p_{kth})$ **then**
12: $P_k(q) \leftarrow P_k(q) \cup \{p\} - \{p_{kth}\}$ // assume that p_{kth} is the current k-th farthest data point from q.
13: **return** $P_k(q)$

i.e., $P_k(q) \leftarrow P_k(q) \cup \{p\} - \{p_{kth}\}$ (lines 9−10). The query result $P_k(q)$ of the query point q is then returned after the candidate data points in Σ_q have been explored (line 13).

VI. PERFORMANCE STUDY

In this section, we report the results from an empirical analysis of the GMP algorithm. We describe the experimental settings in Section VI.A and present the experimental results in Section VI.B.

A. EXPERIMENTAL SETTINGS

In the experiments, we used three real-world roadmaps [55], which are described in Table 3. These real-world roadmaps have different sizes and are part of the road network in the United States. For convenience, each dimension of the data universe was independently normalized to a unit length [0, 1]. The query and data points exhibited either centroid or uniform distributions. The centroid-based dataset was generated to resemble the real-world data. First, a centroid was randomly selected for the query points, and five centroids were randomly selected for the 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. Table 4 shows the experimental parameter settings. In each experiment, we varied a single parameter within the range shown in Table 4 and maintained the other parameters at the bolded default values. Unless otherwise stated, both the query and data points exhibited a centroid distribution.

TABLE 3. **Real-world roadmaps [55].**

Name	Description	Number of vertices	Number of edges	Number of vertex sequences
NA	Highways in North America (NA)	175,813	179,179	12,416
SJ	City streets in San Joaquin (SJ), California	18,263	23,874	20,040
SF	City streets in San Francisco (SF), California	174,956	223,001	192,276

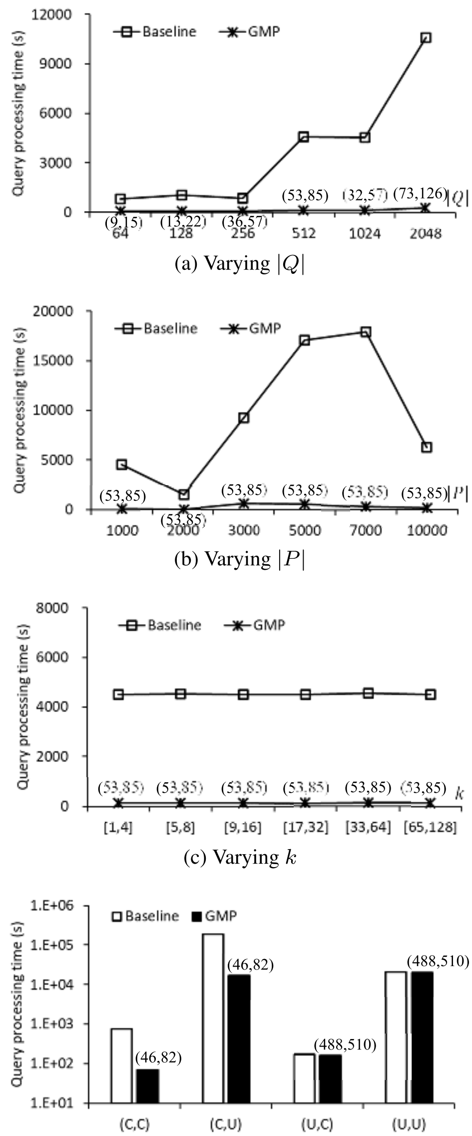
TABLE 4. **Experimental parameter settings.**

Parameter	Range
Number of query points ($\|Q\|$)	64, 128, 256, **512**, 1024, 2048, 4096, 8192, 16384
Number of data points ($\|P\|$)	1, 3, 5, 7, **10,** 20, 40, 80 ($\times$ 1,000)
Number of farthest data points (k)	$[1, 4]$, $[5, 8]$, $[\mathbf{9}, \mathbf{16}]$, $[17, 32]$, $[33, 64]$, $[65, 128]$
Distribution of query points	**(C)entroid**, (U)niform
Distribution of data points	**(C)entroid**, (U)niform
Roadmap	NA, SJ, SF

As a benchmark for evaluating the GMP method, we used a one-query-at-a-time approach called the baseline method, which sequentially computes the k farthest data points for each query point in Q. The GMP method processes query points in batches, whereas the benchmark method processes query points sequentially. In this study, the query and data points move freely within the road networks. Therefore, using pre-computation techniques such as presented in [39], is not applicable because the movement of query and data points might frequently invalidate the pre-computed distances between the query and data points in road networks. All methods were implemented using C++ in the Microsoft Visual Studio 2019 development environment. Note that C++ and the development environment use common subroutines for similar tasks. We then performed the experiments on a desktop computer running Windows 10 operating system with 32 GB RAM and a quad-core processor (i7-7700K) at 4.2 GHz. We believe that the indexing structures of all techniques reside in memory to ensure responsive query processing, which is assumed in many recent studies [1], [48] and is crucial for commercial LBSs and online map services. We performed the experiments using multiple queries for each method and determined the average processing time required to answer the queries. Finally, we used the TNR method [3] to rapidly compute the network distance between the query and data points because the TNR method was easy to implement and demonstrated performances comparable to those of other network distance methods [13], [23], [38], [54]. As stated previously, our solution is orthogonal to network distance methods, and existing network distance methods can be easily integrated with the GMP method to process M*k*FN queries.

B. EXPERIMENTAL RESULTS

Figure 11 shows the result of a comparison of query processing times using the baseline and GMP methods when evaluating M*k*FN queries in the NA roadmap. Here, each chart shows the effect of changing one of the parameters in Table 4. The two values in parentheses in Figures 11–14 show the number of query segments generated from the query points and the number of *k*FN queries evaluated using the GMP method. Because the baseline method sequentially evaluated a *k*FN query for each query point, the number of *k*FN queries



(a) Varying $|Q|$

(b) Varying $|P|$

(c) Varying k

(d) Varying point distribution

FIGURE 11. **Comparison of baseline and GMP methods for NA.**

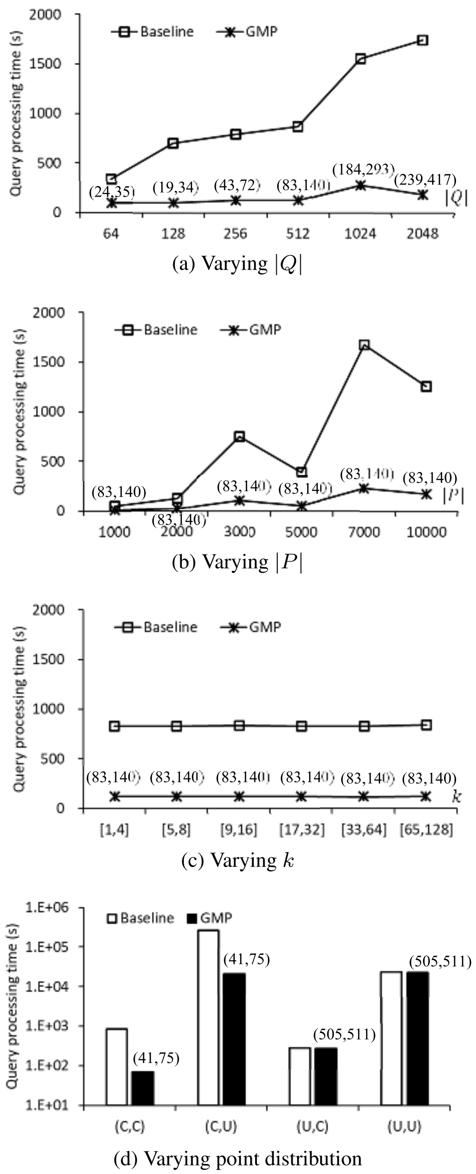


FIGURE 12. **Comparison of baseline and GMP methods for SJ.**

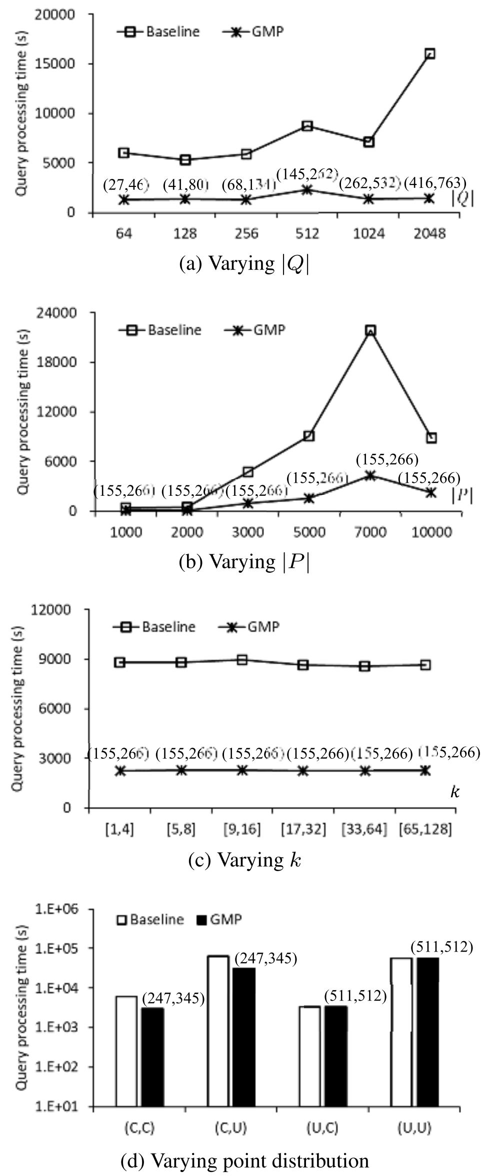
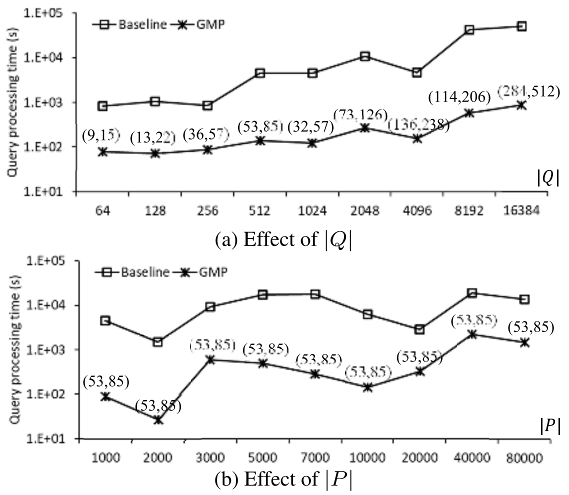


FIGURE 13. **Comparison of baseline and GMP methods for SF.**

evaluated using the baseline method to compute the MkFN query was equal to the number of query points. Note that these values are omitted for simplicity. Figure 11(a) shows the query processing times of the baseline and GMP methods with the number of query points ranging from 64 to 2048, i.e., $64 \leq |Q| \leq 2048$. The GMP method is less sensitive to $|Q|$ than the baseline method due to the shared execution of the GMP method. The processing times of queries using the GMP method were up to 40.6 times shorter than those using the baseline method in all cases. Note that the performance difference between the baseline and GMP methods significantly increased as the number of query points increased. Figure 11(b) shows the query processing times of the baseline and GMP methods with the number of data points ranging from 1000 to 10000, i.e., $10^3 \leq |P| \leq 10^4$. The query processing times using the GMP method at $|P| = 7000$ are up to 63.5 times shorter than those using the baseline method in all cases. Figure 11(c) shows the query processing times of the baseline and GMP methods with the number of data points farthest from a query point ranging from 1 to 128, i.e., $1 \leq k \leq 128$. The query processing times using the GMP method are up to 33.2 times shorter than those using the baseline method in all cases even if the query processing times of both methods are steady regardless of the k values. Figure 11(d) shows the query processing time for various distributions of both 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 query and data points. The GMP method is significantly superior to the baseline method for a centroid distribution of query points (i.e., $\langle$C, C$\rangle$ and $\langle$C, U$\rangle$).



(a) Effect of $|Q|$

(b) Effect of $|P|$

FIGURE 14. Scalability test using NA for various $|Q|$ and $|P|$.

However, the performance of the GMP method is similar to that of the baseline method for a uniformly distributed query points (i.e., ⟨U, C⟩ and ⟨U, U⟩). This is because the query points are widely scattered, and a query segment is typically generated with only a few query points, which limits the group processing of the GMP method. Furthermore, the processing times are extremely long for both the baseline and GMP methods, particularly when the data points are uniformly distributed (i.e., ⟨C, U⟩ and ⟨U, U⟩).

Figure 12 compares the query processing times using the baseline and GMP methods when evaluating M*k*FN queries in the SJ roadmap. Figure 12(a) shows the query processing time as a function of $|Q|$. The query processing times using the GMP method are up to 9.5 times shorter than those using the baseline method in all cases. The baseline method evaluates $|Q|$ *k*FN queries to answer M*k*FN queries, whereas the GMP method evaluates a maximum of $2 \times |\overline{Q}|$ *k*FN queries because of group processing. Furthermore, $|\overline{Q}| \ll |Q|$ for a centroid distribution of query points, whereas $|\overline{Q}| \cong |Q|$ for a uniform distribution of query points. In our experimental settings, for $|Q| = 2048$, the number of *k*FN queries evaluated using the GMP method is up to 4.9 times less than the number of *k*FN queries evaluated using the baseline method. Figure 12(b) shows the query processing time as a function of $|P|$. The GMP method is superior to the baseline method in all cases because it utilizes group processing of adjacent query points and requests a smaller number of *k*FN queries than the baseline method. The GMP methods utilize the shared execution processing of adjacent query points and request a smaller number of *k*FN queries than the baseline method irrespective of $|P|$. Figure 12(c) shows the query processing time as a function of k. The processing times using the GMP method are up to 7.2 times less than those using the baseline method in all cases. Note that the processing times of the baseline and GMP methods are stable irrespective of the k values. Figure 12(d) shows the query processing time for various distributions of both query and data points. For a centroid distribution of the query points (i.e., ⟨C, C⟩, and ⟨C, U⟩), the query processing time of the GMP method is up to 12.3 times less than that of the baseline method. However, for uniformly distributed query points (i.e., ⟨U, C⟩, and ⟨U, U⟩), the performance of the GMP method is similar to that of the baseline method because the query points are widely scattered, which limits the shared execution processing.

Figure 13 shows a comparison of the query processing times of the baseline and GMP methods when evaluating M*k*FN queries in the SF roadmap. Figure 13(a) shows the query processing time as a function of $|Q|$. The GMP method outperformed the baseline method in all cases. The difference in the number of *k*FN queries evaluated by the baseline and GMP methods increased with $|Q|$. In particular, the GMP method evaluated 72%, 62%, 52%, 51%, 52%, and 37% more *k*FN queries than the baseline method when $|Q| = 64$, 128, 256, 512, 1024, and 2048, respectively. Figure 13(b) shows the query processing time as a function of $|P|$. The GMP method significantly outperformed the baseline method in all cases because it utilized the group processing of adjacent query points and requested a smaller number of *k*FN queries. Figure 13(c) shows the query processing time as a function of k. The GMP method outperformed the baseline method in all cases because the baseline and GMP methods evaluated 512 and 266 *k*FN queries, respectively. The query processing times of the baseline and GMP methods were steady irrespective of the value of k. Figure 13(d) shows the query processing time for various distributions of query and data points. The GMP method outperformed the baseline method when the query points exhibited a centroid distribution, i.e., ⟨C, C⟩ and ⟨C, U⟩. However, the two methods showed similar performances when the query points exhibited a uniform distribution, i.e., ⟨U, C⟩ and ⟨U, U⟩.

We then analyzed the scalability of the GMP method by varying the number of query points $|Q|$ and the number of data points $|P|$. Figure 14 shows the query processing times for the baseline and GMP methods for various numbers of query and data points in the NA roadmap, i.e., $64 \leq |Q| \leq 16,384$ and $1,000 \leq |P| \leq 80,000$. As shown in Figure 14(a), the GMP method outperformed the baseline method for all cases in $|Q|$, and the performance difference between them increased as $|Q|$ increases. As shown in Figure 14(b), the GMP method outperformed the baseline method for all cases in $|P|$. The empirical results indicated that the GMP method scaled well with both $|Q|$ and $|P|$.

VII. CONCLUSION

In this study, we investigated methods to evaluate concurrent M*k*FN queries in road networks efficiently. Existing solutions for spatial queries using Euclidean distances are un suitable for answering M*k*FN queries in road networks. We proposed a group processing solution called the GMP method to process M*k*FN queries in road networks efficiently. The GMP method can be easily implemented using popular network distance algorithms [3], [23], [54], which is highly desirable. Extensive experimental studies demonstrated the efficiency and effectiveness of the GMP method compared with the baseline method based on one-query-at-a-time processing. In particular, the GMP method was up to 74.1 times faster than the baseline method. Furthermore, the GMP method scaled based on the number of query points, particularly when the query points exhibited a non-uniform distribution. However, the performance of the GMP method was similar to that of the baseline method when the query points exhibited a uniform distribution. For future studies, we plan to extend the group processing approach used in this study to problems on the processing of sophisticated spatial queries in road networks such as multi-way distance join queries [9] and AkFN queries [47]. These problems have not been adequately addressed in road networks despite their importance.

REFERENCES

[1] T. Abeywickrama, M. A. Cheema, and D. Taniar, "K-nearest neighbors on road networks: A journey in experimentation and in-memory implementation," *PVLDB*, vol. 9, no. 6, 492–503, 2016.

[2] M. E. Ali, E. Tanin, R. Zhang, and L. Kulik, "A motion-aware approach for efficient evaluation of continuous queries on 3D object databases," *VLDB J.*, vol. 19, no. 5, pp. 603–632, Oct. 2010.

[3] H. Bast, S. Funke, and D. Matijevic, "Ultrafast shortest-path queries via transit nodes," in *Proc. DIMACS Workshop Shortest Path Problem*, 2006, pp. 175–192.

[4] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1990, pp. 322–331.

[5] P. Boinski and M. Zakrzewicz, "Concurrent execution of data mining queries for spatial collocation pattern discovery," in *Proc. Int. Conf. Data Warehousing Knowl. Discovery*, 2013, pp. 184–195.

[6] Z. Chen and J. Van Ness, "Characterizations of nearest and farthest neighbor algorithms by clustering admissibility conditions," *Pattern Recognit.*, vol. 31, no. 10, pp. 1573–1578, Oct. 1998.

[7] H.-J. Cho, "Shared execution approach to ε-distance join queries in dynamic road networks," *ISPRS Int. J. Geo-Inf.*, vol. 7, no. 7, p. 270, Jul. 2018.

[8] H.-J. Cho, "Efficient shared execution processing of K-nearest neighbor joins in road networks," *Mobile Inf. Syst.*, vol. 2018, Apr. 2018, Art. no. 1243289.

[9] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Multi-way distance join queries in spatial databases," *GeoInformatica*, vol. 8, no. 4, pp. 373–402, 2004.

[10] R. R. Curtin, J. Echauz, and A. B. Gardner, "Exploiting the structure of furthest neighbor search for fast approximate results," *Inf. Syst.*, vol. 80, pp. 124–135, Feb. 2019.

[11] M. Eslami, Y. Tu, H. Charkhgard, Z. Xu, and J. Liu, "PsiDB: A framework for batched query processing and optimization," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2019, pp. 6046–6048.

[12] Y. Gao, L. Shou, K. Chen, and G. Chen, "Aggregate farthest-neighbor queries over spatial data," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, vol. 2, 2011, pp. 149–163.

[13] R. Geisberger, P. Sanders, D. Schultes, and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks," in *Proc. Int. Workshop Experim. Algorithms*, 2008, pp. 319–333.

[14] G. Giannikis, G. Alonso, and D. Kossmann, "SharedDB: Killing one thousand queries with one stone," *Proc. VLDB Endowment*, vol. 5, no. 6, pp. 526–537, Feb. 2012.

[15] G. Giannikis, D. Makreshanski, G. Alonso, and D. Kossmann, "Shared workload optimization," *Proc. VLDB Endowment*, vol. 7, no. 6, pp. 429–440, Feb. 2014.

[16] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. Int. Conf. Manage. Data*, 1984, pp. 47–57.

[17] X. Huang, C. S. Jensen, H. Lu, and S. Šaltenis, "S-GRID: A versatile approach to efficient query processing in spatial networks," in *Proc. Int. Symp. Adv. Spatial Temporal Databases*, 2007, pp. 93–111.

[18] C. S. Jensen, J. Kolárvr, T. B. Pedersen, and I. Timko, "Nearest neighbor queries in road networks," in *Proc. Int. Symp. Adv. Geographic Inf. Syst.*, 2003, pp. 1–8.

[19] A. Jonathan, A. Chandra, and J. Weissman, "Multi-query optimization in wide-area streaming analytics," in *Proc. ACM Symp. Cloud Comput.*, Oct. 2018, pp. 412–425.

[20] J. Karimov, T. Rabl, and V. Markl, "AStream: Ad-hoc shared stream processing," in *Proc. Int. Conf. Manage. Data SIGMOD*, 2019, pp. 607–622.

[21] J. Karimov, T. Rabl, and V. Markl, "AJoin: Ad-hoc stream joins at scale," *Proc. VLDB Endowment*, vol. 13, no. 4, pp. 435–448, Dec. 2019.

[22] K. C. K. Lee, W.-C. Lee, B. Zheng, and Y. Tian, "ROAD: A new spatial object search framework for road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 3, pp. 547–560, Mar. 2012.

[23] Z. Li, L. Chen, and Y. Wang, "G-tree: An efficient spatial index on road networks," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 268–279.

[24] J. Liu, H. Chen, K. Furuse, and H. Kitagawa, "An efficient algorithm for arbitrary reverse furthest neighbor queries," in *Proc. Asia–Pacific Web Conf. Web Technol. Appl.*, 2012, pp. 60–72.

[25] W. Liu and Y. Yuan, "New ideas for FN/RFN queries based nearest voronoi diagram," in *Proc. Int. Conf. Bio-Inspired Comput., Theories Appl.*, 2013, pp. 917–927.

[26] H. Lu and M. L. Yiu, "On computing farthest dominated locations," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 6, pp. 928–941, Jun. 2011.

[27] H. Mahmud, A. M. Amin, M. E. Ali, T. Hashem, and S. Nutanong, "A group based approach for path queries in road networks," in *Proc. Int. Symp. Adv. Spatial Temporal Databases*, 2013, pp. 367–385.

[28] D. Makreshanski, G. Giannikis, G. Alonso, and D. Kossmann, "MQJoin: Efficient shared execution of main-memory joins," *Proc. VLDB Endowment*, vol. 9, no. 6, pp. 480–491, Jan. 2016.

[29] D. Makreshanski, G. Giannikis, G. Alonso, and D. Kossmann, "Many-query join: Efficient shared execution of relational joins on modern hardware," *VLDB J.*, vol. 27, no. 5, pp. 669–692, Oct. 2018.

[30] R. Marroquín, I. Müller, D. Makreshanski, and G. Alonso, "Pay one, get hundreds for free: Reducing cloud costs through shared query execution," in *Proc. ACM Symp. Cloud Comput.*, Oct. 2018, pp. 439–450.

[31] P. Michiardi, D. Carra, and S. Migliorini, "In-memory caching for multi-query optimization of data-intensive scalable computing workloads," in *Proc. Workshops EDBT/ICDT Joint Conf.*, 2019, pp. 1–8.

[32] S. Nutanong and H. Samet, "Memory-efficient algorithms for spatial network queries," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 649–660.

[33] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proc. Int. Conf. Very Large Data Bases*, 2003, pp. 802–813.

[34] I. Psaroudakis, M. Athanassoulis, and A. Ailamaki, "Sharing data and work across concurrent analytical queries," *Proc. VLDB Endowment*, vol. 6, no. 9, pp. 637–648, Jul. 2013.

[35] R. Rehrmann, C. Binnig, A. Böhm, K. Kim, W. Lehner, and A. Rizk, "OLTPshare: The case for sharing in OLTP workloads," *Proc. VLDB Endowment*, vol. 11, no. 12, pp. 1769–1780, Aug. 2018.

[36] R. M. Reza, M. E. Ali, and T. Hashem, "Group processing of simultaneous shortest path queries in road networks," in *Proc. 16th IEEE Int. Conf. Mobile Data Manage.*, Jun. 2015, pp. 128–133.

[37] R. M. Reza, M. E. Ali, and M. A. Cheema, "The optimal route and stops for a group of users in a road network," in *Proc. 25th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, Nov. 2017, pp. 1–10.

[38] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 43–54.

[39] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Distance join queries on spatial networks," in *Proc. 14th Annu. ACM Int. Symp. Adv. Geographic Inf. Syst. (GIS)*, 2006, pp. 211–218.

[40] T. K. Sellis, "Multiple-query optimization," *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, 1988.

[41] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, "A road network embedding technique for K-nearest neighbor search in moving object databases," *Geoinformatica*, vol. 7, no. 3, pp. 255–273, 2015.

[42] S. Suri, "Computing geodesic furthest neighbors in simple polygons," *J. Comput. Syst. Sci.*, vol. 39, no. 2, pp. 220–235, Oct. 1989.

[43] J. R. Thomsen, M. L. Yiu, and C. S. Jensen, "Effective caching of shortest paths for location-based services," in *Proc. Int. Conf. Manage. Data SIGMOD*, 2012, pp. 313–324.

[44] J. R. Thomsen, M. L. Yiu, and C. S. Jensen, "Concise caching of driving instructions," in *Proc. 22nd ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst.*, 2014, pp. 23–32.

[45] Q. T. Tran, D. Taniar, and M. Safar, "Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems I* (Lecture Notes in Computer Science), vol. 5740, A. Hameurlain, J. Küng, and R. Wagner, Eds. Berlin, Germany: Springer-Verlag, 2009, doi: 10.1007/978-3-642-03722-1_14.

[46] S. Wang, M. A. Cheema, X. Lin, Y. Zhang, and D. Liu, "Efficiently computing reverse k furthest neighbors," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 1110–1121.

[47] H. Wang, K. Zheng, H. Su, J. Wang, S. W. Sadiq, and X. Zhou, "Efficient aggregate farthest neighbour query processing on road networks," in *Proc. Australas. Database Conf. Databases Theory Appl.*, 2014, pp. 13–25.

[48] L. Wu, X. Xiao, D. Deng, G. Cong, A. D. Zhu, and S. Zhou, "Shortest path and distance queries on road networks: An experimental evaluation," *Proc. VLDB Endowment*, vol. 5, no. 5, pp. 406–417, Jan. 2012.

[49] X.-J. Xu, J.-S. Bao, B. Yao, J.-Y. Zhou, F.-L. Tang, M.-Y. Guo, and J.-Q. Xu, "Reverse furthest neighbors query in road networks," *J. Comput. Sci. Technol.*, vol. 32, no. 1, pp. 155–167, Jan. 2017.

[50] B. Yao, F. Li, and P. Kumar, "Reverse furthest neighbors in spatial databases," in *Proc. IEEE 25th Int. Conf. Data Eng.*, Mar. 2009, pp. 664–675.

[51] D. Zhang, C.-Y. Chow, Q. Li, X. Zhang, and Y. Xu, "SMashQ: Spatial mashup framework for k-NN queries in time-dependent road networks," *Distrib. Parallel Databases*, vol. 31, no. 2, pp. 259–287, Jun. 2013.

[52] M. Zhang, L. Li, W. Hua, and X. Zhou, "Efficient batch processing of shortest path queries in road networks," in *Proc. 20th IEEE Int. Conf. Mobile Data Manage. (MDM)*, Jun. 2019, pp. 100–105.

[53] M. Zhang, L. Li, W. Hua, and X. Zhou, "Batch processing of shortest path queries in road networks," in *Proc. Australas. Database Conf. Databases Theory Appl.*, 2019, pp. 3–16.

[54] 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 Trans. Knowl. Data Eng.*, vol. 27, no. 8, pp. 2175–2189, Aug. 2015.

[55] *Real Datasets for Spatial Databases*. Accessed: Jun. 13, 2020. [Online]. Available: https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

HYUNG-JU CHO is currently an Associate Professor with the Department of Software, Kyungpook National University, South Korea. His current research interests include moving object databases and query processing in mobile peer-to-peer networks.

MUHAMMAD ATTIQUE is currently an Assistant Professor with the Department of Software, Sejong University, South Korea. His research interests include spatial computing, location-based services, big spatial data, and spatial query processing in mobile networks.