Journal of Korea Multimedia Society Vol. 22, No. 12, December 2019(pp. 1466-1480)
https://doi.org/10.9717/kmms.2019.22.12.1466

Efficient Processing of All-farthest-neighbors Queries in Spatial Network Databases

Hyung-Ju Cho†

ABSTRACT

This paper addresses the efficient processing of all-farthest-neighbors (AFN) queries in spatial network databases. Given a set of data points $P=\{p_1,p_2,\cdots,p_{|P|}\}$ in a spatial network, where the distance between two data points p and s, denoted by $dist(p,s)$, is the length of the shortest path between them, an AFN query is defined as follows: find the farthest neighbor $\omega(p)\in P$ of each data point p such that $dist(p,\omega(p)) \geq dist(p,s)$ for all $s\in P$. In this paper, we propose a shared execution algorithm called FAST (for All-Farthest-neighbors Search in spatial neTworks). Extensive experiments on real-world roadmaps confirm the efficiency and scalability of the FAST algorithm, while demonstrating a speedup of up to two orders of magnitude over a conventional solution.

Key words: All-farthest-neighbors Query, Maximum Distance, Spatial Network Database

1. INTRODUCTION

Computing the nearest, farthest, or some intermediate neighbors is a fundamental problem in computational geometry and location-based services (LBS), and the results of these computations can be applied to many applications. Given a set of data points $P=\{p_1,p_2,\cdots,p_{|P|}\}$ in a spatial network, the farthest neighbor of a data point p in P is defined as $max\{dist(p,s)|s\in P\}$, where $dist(p,s)$ is the length of the shortest path between two data points p and s. The problem of computing the farthest neighbor of each data point in P is referred to as the all-farthest-neighbors (AFN) search problem, which is the logical opposite of the all-nearest-neighbors (ANN) search problem [1, 2].

Nearest neighbor search has been studied extensively, because of its importance and overwhelming impact in LBS [1-4]. However, little attention has been paid to farthest neighbor search, even though it has real-life applications in a large number of fields, including marketing, facility location, clustering, and recommendation systems. Let us consider some real-life scenarios in which farthest neighbor search is valuable. For example, if a user wants to purchase a facility that has a sufficient service range, such as a transceiver or telescope, the user can use the distance to the farthest neighbor to determine which facility to buy. In addition, in contrast to the nearest neighbor, the farthest neighbor can be of particular interest to a user just because of its remoteness. A farthest neighbor search can be used, for example, to determine the quiet places farthest from a noisy factory. Some tourists may be interested in the greatest distance between two cities in a country that can be traversed on land.

Fig. 1 shows an example of an AFN query in a spatial network, where data points p_1 through p_5 denote places such as service stations. In this ex-

※ Corresponding Author : Hyung-Ju Cho, Address: (37224) Gyeongsang-daero 2559, Sangju-si, Gyeongsangbuk-do, Republic of Korea, TEL : +82-54-530-1455, FAX : +82-54-530-1459, E-mail : hyungju@knu.ac.kr
Receipt date : July 29, 2019, Revision date : Oct. 24, 2019
Approval date : Nov. 19, 2019
† Department of Software, Kyungpook National University
※ This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (NRF-2019R1H1A2080073)

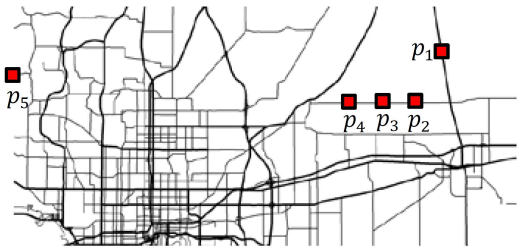


Fig. 1. Example of an AFN query in a road network where $P=\{p_1,p_2,p_3,p_4,p_5\}$.

ample, the AFN query retrieves the farthest neighbor, denoted by $\omega(p)$, of each data point p in P. We can compute the query result $\Omega(P)=\{\langle p,\omega(p)\rangle|p\in P\}$ $=\{\langle p_1,p_5\rangle,\langle p_2,p_5\rangle,\langle p_3,p_5\rangle,\langle p_4,p_5\rangle,\langle p_5,p_1\rangle\}$. Unfortunately, few studies have been carried out on the AFN search problem [5, 6]. Bae *et al.* [6] proposed an efficient solution to AFN search in the L_1 plane with the Manhattan metric, in the presence of highways and obstacles. However, this approach cannot be extended to AFN search over spatial networks, because of inconsistencies in problem requirements between the L_1 plane and a spatial network. Spatial queries based on farthest neighbor search have gained significant attention in recent years in the form of farthest neighbor queries [7], reverse farthest neighbor queries [8, 9], and aggregate farthest neighbor queries [10, 11]. However, existing solutions cannot be readily applied to AFN queries in spatial networks because of differences in the problem specification and distance metrics.

The simplest search algorithm for AFN queries in spatial networks involves computing the network distance between every ordered pair $\langle p,s\rangle\in P\times P$, which corresponds to the all-pairs-shortest-path problem [12], and then spending additional $O(|P|^2)$ time to choose the farthest ordered pairs among all pairs in $P\times P$. This simple solution is too computationally expensive to be of any practical use, particularly for large datasets, because of the very large number of distance computations required between data points in spatial networks of even moderate size. Therefore, we propose a new algorithm called FAST, for All-Farthest-neighbors Search in spatial neTworks. Our proposed solution is to cluster adjacent data points into a group and then optimize shared computation for the group, to rapidly filter candidates by computing the maximum distance between two groups. Although the shared computation of spatial queries has received much attention [13-15], no shared computation strategy has been applied to AFN queries in spatial networks to date. In this study, we optimize the shared execution strategy to process AFN queries in spatial networks. The FAST algorithm is easy to implement, facilitating its integration with popular network distance algorithms, such as transit node routing (TNR) [16] and G-tree [17], for spatial networks. We summarize the main contributions of this study as follows.

- We propose an efficient algorithm, called FAST, that exploits the shared execution of groups to minimize the number of network distance computations.
- We present a mathematical formula to compute the maximum distance between two groups. We also present effective pruning techniques based on the maximum distance between the two groups.
- We conduct experiments under several different sets of conditions to demonstrate the efficiency and scalability of the FAST algorithm and demonstrate speedups of up to two orders of magnitude over a conventional solution.

The remainder of this paper is organized as follows. In Section 2, we review related research. In Section 3, we provide essential background knowledge. In Section 4, we present a method for converting adjacent data points into a group, and detail how to compute the maximum distance between two groups. In Section 5, we present the FAST algorithm for performing AFN queries in

spatial networks using shared execution of the network distance computations. In Section 6, we report a comprehensive experimental study comparing the FAST algorithm with a conventional solution under different circumstances. Finally, we discuss our conclusions in Section 7.

2. RELATED WORK

Many studies have been performed on the processing of sophisticated spatial queries based on farthest neighbor search. Curtin *et al.* [7] presented an approximate farthest neighbor search algorithm that selects a set of candidate data points using data distributed in Euclidean space. They also developed an information-theoretic entropy measure to investigate the difficulty of the farthest neighbor search problem. Lu *et al.* [18] formulated the farthest dominated location query, which retrieves a location such that the distance to its nearest dominating object is maximized, for spatial decision support applications. Gao *et al.* [10] and Wang *et al.* [11] studied aggregate k-farthest neighbor queries in Euclidean space and spatial networks, respectively. Given a set of data points P and a set of query data points Q, an aggregate k-farthest neighbor query returns the k data points in P that have the largest aggregate distances to all query data points in Q. Reverse farthest neighbors queries have also been studied, both in Euclidean space [8, 9] and in spatial networks [19, 20]. Yao *et al.* [9] first studied reverse farthest neighbor query problem in Euclidean space. They proposed progressive farthest cell and convex hull farthest cell algorithms to support reverse farthest neighbors queries using the R-tree [21]. Wang *et al.* [8] presented a solution to support reverse k-farthest neighbors queries in Euclidean space for arbitrary values of k. Tran *et al.* [19] studied reverse farthest neighbor queries in spatial networks using network Voronoi diagrams and precomputed network distances. Xu *et al.* [20] presented efficient algorithms based on landmarks and hierarchical partitioning to process monochromatic reverse farthest neighbors queries, as well as bichromatic reverse farthest neighbors queries, in spatial networks. However, existing solutions in Euclidean space cannot be readily applied to our situation because it is very difficult to exploit R-trees and convex hulls in spatial networks.

The AFN search problem has been recently examined from a theoretical point of view [6, 22]. Bae *et al.* [6] proposed an efficient algorithm for AFN search in the L_1 plane in the presence of highways and obstacles. This approach cannot be extended to AFN search in spatial networks, because of inconsistent problem requirements. Daescu *et al.* [22] presented solutions to farthest-point problems in the plane and showed how the solutions can be used to efficiently solve other complex problems, such as simplifying polygonal paths or determining the data point farthest from a query segment. Katoh and Iwano [23] addressed the problems of enumerating the k closest or farthest bichromatic pairs of red and blue data points in the Euclidean space. However, large-scale spatial networks need practical and efficient algorithms for the evaluation of AFN queries, which limits the application of these theoretical approaches [6, 22] in our situation.

To process a large number of queries in a database system, the shared execution strategy has attracted considerable attention, because of its low processing cost [13-15]. The key idea of shared query execution is to cluster queries that share some common execution path into a group, and then execute the group as a single query in the system. These shared execution methods have been found to be effective in many applications involving high load conditions. In this study, we exploit a shared execution strategy to increase the efficiency of AFN queries in spatial networks. Our proposed solution differs from existing studies in several ways. First, it represents the first attempt to evaluate AFN queries efficiently in spatial

networks. Second, it employs a shared execution strategy to rapidly filter candidates while processing AFN queries. Finally, it can be implemented easily using popular network distance algorithms [16, 17] in spatial networks, a highly desirable property in practice. A brief survey and empirical comparison of some of the most popular network distance algorithms are presented in [24]. The techniques presented in [1, 2, 17, 25] for nearest neighbor search cannot be directly applied to farthest neighbor search, because the nearest neighbor search and farthest neighbor search have opposite goals.

3. PRELIMINARIES

We first define the principal terms related to farthest neighbor and AFN queries, and summarize the notation used in this paper.

Definition 1. (Farthest neighbor): The farthest neighbor of a data point p from a set of data points P is defined as $\omega(p)$ such that $\omega(p)\in P$ and $dist(p,\omega(p)) \geq dist(p,s)$ for $\forall s\in P$. Ties are broken arbitrarily.

Definition 2. (AFN query): Given a set of data points P, an AFN query retrieves a set of ordered pairs, each consisting of a data point p in P and its farthest neighbor $\omega(p)$. The AFN query result $\Omega(P)$ is formally defined as $\Omega(P)=\{\langle p,\omega(p)\rangle | p\in P\}$.

Definition 3. (Spatial network): We represent a spatial network as an undirected weighted graph $G=(V,E,W)$, where V is the set of vertices, E is the set of edges, and the weight $W: E\rightarrow R^+$ associates each edge with a positive real number representing the network distance or travel time. Each data point p is located on an edge $\overline{v_iv_j}$ in the spatial network.

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

Definition 5. (Vertex sequence and segment): A vertex sequence $\overline{v_lv_{l+1}\cdots v_m}$ denotes a path between two vertices v_l and v_m, such that each of v_l and v_m is either an intersection vertex or a terminal vertex, and the other vertices in the path, $v_{l+1},\cdots,v_{m-1}$, are intermediate vertices. The length of a vertex sequence is the total weight of the edges in the vertex sequence. A part of a vertex sequence is referred to as a segment. By definition, a vertex sequence is also a segment. Table 1 presents the symbols and notation used throughout the

Table 1. Symbols and their meanings

Symbol	Definition
$dist(p,s)$	Length of the shortest path connecting two data points p and s in the spatial network
$len(p,s)$	Length of the segment connecting two data points p and s that are located in a vertex sequence
$\overline{v_lv_{l+1}\cdots v_m}$	Vertex sequence where neither v_l nor v_m has a degree of two and the other vertices, $v_{l+1},\cdots,v_{m-1}$, have a degree of two
$\overline{p_ip_{i+1}\cdots p_j}$	Data segment in which data points $p_i,p_{i+1},\cdots,p_j$ are in a vertex sequence
$\omega(p,P)$	Farthest neighbor of a data point p in a set of data points P such that for all $s\in P$
$maxdist(\overline{p_ip_j},\overline{s_ls_m})$	Maximum distance between two data segments $\overline{p_ip_j}$ and $\overline{s_ls_m}$, i.e., $max\{dist(p,s)\|p\in\overline{p_ip_j},s\in\overline{s_ls_m}\}$,
$mindist(\overline{p_ip_j},\overline{s_ls_m})$	Minimum distance between two data segments $\overline{p_ip_j}$ and $\overline{s_ls_m}$, i.e., $min\{dist(p,s)\|p\in\overline{p_ip_j},s\in\overline{s_ls_m}\}$,

paper. To simplify the presentation, we use $\overline{p_ip_j}$ to denote $\overline{p_ip_{i+1}\cdots p_j}$ and use $\omega(p)$ to denote $\omega(p,P)$ when P denotes the set of all data points.

4. PROCESSING AFN QUERIES IN SPATIAL NETWORK DATABASES

4.1 Grouping adjacent data points in a vertex sequence

Fig. 2 presents an example of an AFN query in a spatial network, which will be used as the example throughout this section. As shown in Figure 2(a), given a set of data points $P=\{p_1,p_2,\cdots,p_7\}$, an AFN query retrieves the farthest neighbor of each data point p in P. Fig. 2(b) shows the sample grouping of adjacent data points in a vertex sequence. As shown in Fig. 2(b), four data points p_1, p_2, p_5, and p_6 in a vertex sequence $\overline{v_1v_2v_3}$ are grouped into a segment $\overline{p_2p_1p_6p_5}$ and two data points p_3 and p_4 in a vertex sequence $\overline{v_1v_4v_5v_3}$ are grouped into another segment $\overline{p_3p_4}$. Naturally, the data point p_7 in vertex sequence $\overline{v_1v_3}$ is not grouped because there is no other data point in the vertex sequence $\overline{v_1v_3}$. Therefore, a set of data points $P=\{p_1,p_2,\cdots,p_7\}$ can be transformed into a set of segments $\overline{P}=\{\overline{p_2p_1p_6p_5},\overline{p_3p_4},p_7\}$.

4.2 Computation of the distance between two segments

In this section, we describe the computation of the minimum and maximum distances between two data segments, $\overline{p_ip_j}$ and $\overline{s_ls_m}$, which are denoted as $mindist(\overline{p_ip_j},\overline{s_ls_m})$ and $maxdist(\overline{p_ip_j},\overline{s_ls_m})$, respectively. Clearly, $mindist(\overline{p_ip_j},\overline{s_ls_m})$ and $maxdist(\overline{p_ip_j},\overline{s_ls_m})$ are formally defined as $mindist(\overline{p_ip_j},\overline{s_ls_m})=min\{dist(p,s)|p\in\overline{p_ip_j},s\in\overline{s_ls_m}\}$ and $maxdist(\overline{p_ip_j},\overline{s_ls_m})=max\{dist(p,s)|p\in\overline{p_ip_j},s\in\overline{s_ls_m}\}$, respectively.

Corollary 1. $mindist(\overline{p_ip_j},\overline{s_ls_m})$ is a lower bound ($maxdist(\overline{p_ip_j},\overline{s_ls_m})$ is an upper bound) on the distance between two data points $p\in\overline{p_ip_j}$ and $s\in\overline{s_ls_m}$. Therefore, $mindist(\overline{p_ip_j},\overline{s_ls_m})\le dist(p,s)\le maxdist(\overline{p_ip_j},\overline{s_ls_m})$ for $\forall p\in\overline{p_ip_j}$ and $\forall s\in\overline{s_ls_m}$. ■

We describe how to compute $mindist(\overline{p_ip_j},\overline{s_ls_m})$ and $maxdist(\overline{p_ip_j},\overline{s_ls_m})$. If there exists an overlap between $\overline{p_ip_j}$ and $\overline{s_ls_m}$ (i.e., $\overline{p_ip_j}\cap\overline{s_ls_m}\neq\varnothing$), the minimum distance between $\overline{p_ip_j}$ and $\overline{s_ls_m}$ is $mindist(\overline{p_ip_j},\overline{s_ls_m})=0$; otherwise, the minimum distance between $\overline{p_ip_j}$ and $\overline{s_ls_m}$ is $mindist(\overline{p_ip_j},\overline{s_ls_m})=min\{dist(p_i,s_l),dist(p_i,s_m),dist(p_j,s_l),dist(p_j,s_m)\}$. Unlike the computation of $mindist(\overline{p_ip_j},\overline{s_ls_m})$, it is not intuitive to compute $maxdist(\overline{p_ip_j},\overline{s_ls_m})$. We first describe how to compute the maximum distance between a segment $\overline{p_ip_j}$ and a data point s in $\overline{s_ls_m}$. For this, we investigate the distance between two data points p and s where $p\in\overline{p_ip_j}$ and $s\in\overline{s_ls_m}$.

In Fig. 3, let us assume that the data point p_i corresponds to the origin of the XY coordinate system. The X-axis and Y-axis then represent $len(p,p_i)$ and $dist(p,s)$, respectively, where $p\in\overline{p_ip_j}$. For convenience, data point p is represented by a rectangle, whereas data point s is represented by a

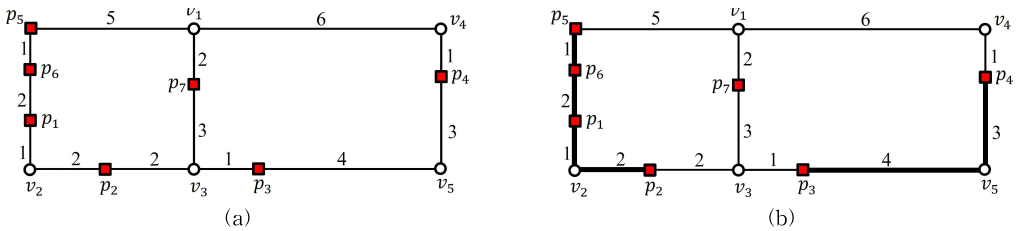


Fig. 2. Grouping adjacent data points in a vertex sequence. (a) $P=\{p_1,p_2,\cdots,p_7\}$ (b) $\overline{P}=\{\overline{p_2p_1p_6p_5},\overline{p_3p_4},p_7\}$.

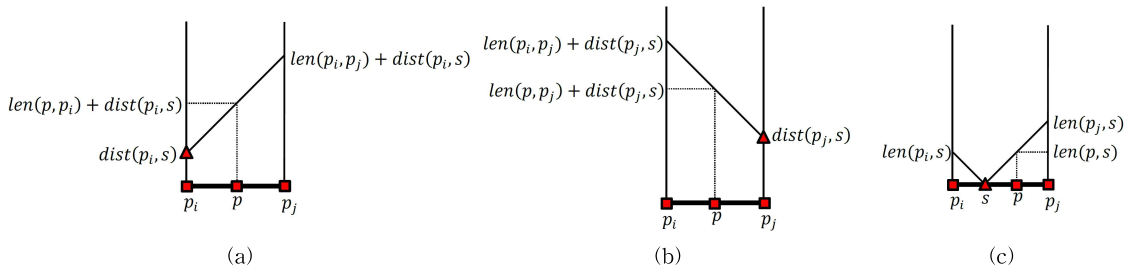


Fig. 3. Determination of the distance from p to s where $p \in \overline{p_i p_j}$. (a) $dist(p,s) = \text{len}(p,p_i) + dist(p_i,s)$ (b) $dist(p,s) = \text{len}(p,p_j) + dist(p_j,s)$ (c) $dist(p,s) = \text{len}(p,s)$.

triangle. If there is a path from data point p to data point s via p_i (i.e., $p \rightarrow p_i \rightarrow s$), the distance between p and s is computed as $dist(p,s) = len(p,p_i) + dist(p_i,s)$, as shown in Fig. 3(a). Similarly, if there is a path from p to s via p_j (i.e., $p \rightarrow p_j \rightarrow s$), then $dist(p,s)$ is computed as $dist(p,s) = len(p,p_j) + dist(p_j,s)$, as shown in Fig. 3(b). If the data point s is located in $\overline{p_i p_j}$, then $dist(p,s)$ is computed as $dist(p,s) = len(p,s)$, as shown in Fig. 3(c). Because $dist(p,s)$ is the length of the shortest path among multiple paths between p and s, it is computed as follows:

$$dist(p,s) = \begin{cases} \min\{len(p,p_i) + dist(p_i,s), len(p,p_j) + dist(p_j,s)\} & \text{if } s \notin \overline{p_i p_j} \\ \min\{len(p,p_i) + dist(p_i,s), len(p,p_j) + dist(p_j,s), len(p,s)\} & \text{otherwise} \end{cases}$$

Given a segment $\overline{p_i p_j}$ and a data point s, let $\omega(s, \overline{p_i p_j}) = s^*$ be the farthest data point of s to a set of data points in $\overline{p_i p_j}$, which means that there exists a data point s^* in $\overline{p_i p_j}$ such that $maxdist(\overline{p_i p_j}, s) = dist(s^*, s)$. We can then easily locate s^* in $\overline{p_i p_j}$ us-

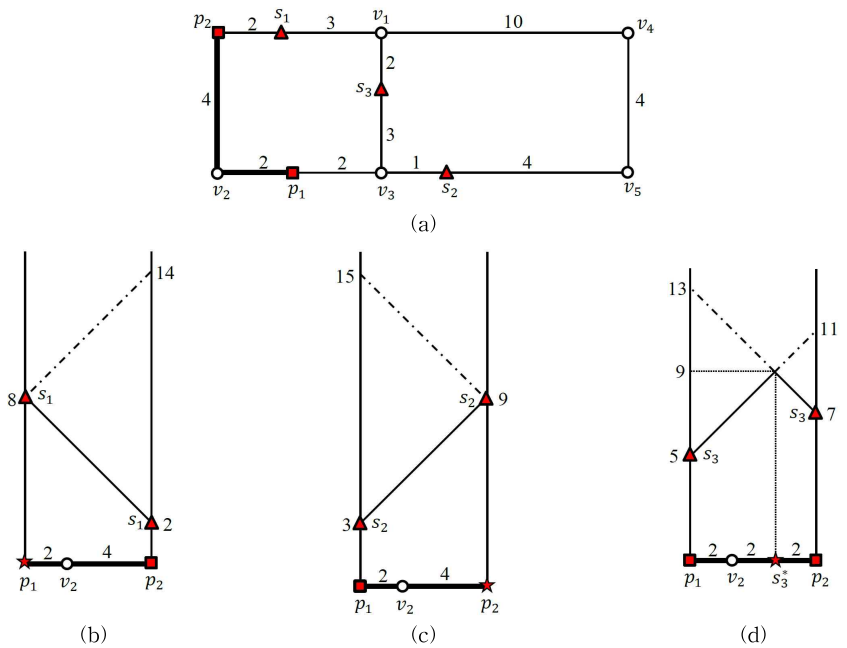


Fig. 4. Evaluation of $maxdist(\overline{p_1 p_2}, s_1)$, $maxdist(\overline{p_1 p_2}, s_2)$, and $maxdist(\overline{p_1 p_2}, s_3)$. (a) Data segment $\overline{p_1 p_2}$ and three data points s1, s2, and s3 (b) $maxdist(\overline{p_1 p_2}, s_1) = 8$ (c) $maxdist(\overline{p_1 p_2}, s_2) = 9$ (d) $maxdist(\overline{p_1 p_2}, s_3) = 9$.

ing the linear equation $maxdist(\overline{p_ip_j}, s) = dist(s^*, s)$. In Fig. 4, let us compute the farthest data point s^* of each data point $s \in \{s_1, s_2, s_3\}$ such that $s^* \in \overline{p_1p_2}$ and $maxdist(\overline{p_1p_2}, s) = dist(s^*, s)$. Because $dist(s_1, p_1) = 8$, $dist(s_1, p_2) = 2$, $len(p_1, p_2) = 6$, and $dist(s_1, p_1) = dist(s_1, p_2) + len(p_1, p_2)$, we have $maxdist(\overline{p_1p_2}, s) = 8$ and $\omega(s_1, \overline{p_1p_2}) = p_1$, as illustrated in Fig. 4 (b). Similarly, because $dist(s_2, p_1) = 3$, $dist(s_2, p_2) = 9$, $len(p_1, p_2) = 6$, and $dist(s_2, p_2) = dist(s_2, p_1) + len(p_1, p_2)$, we have $maxdist(\overline{p_1p_2}, s_2) = 9$ and $\omega(s_2, \overline{p_1p_2}) = p_2$, as illustrated in Fig. 4(c). Finally, because $dist(s_3, p_1) = 5$, $dist(s_3, p_2) = 7$, and $len(p_1, p_2) = 6$, the maximum distance between $\overline{p_1p_2}$ and s_3 is evaluated as $maxdist(\overline{p_1p_2}, s_3) = 9$ and the farthest data point of s_3 is marked as s_3^* in Fig. 4(d). The dashed/dotted lines in Fig. 4 indicate the lengths of redundant paths that are not the shortest.

Fig. 5 illustrates the process of computing $maxdist(\overline{p_ip_j}, \overline{s_ls_m})$. This process operates in two steps, which correspond to Fig. 5(a) and 5(b). In the first step, we find the farthest data point p_i^* (respectively, p_j^*) of p_i (respectively, p_j) such that $\omega(p_i, \overline{s_ls_m}) = p_i^*$ (respectively, $\omega(p_j, \overline{s_ls_m}) = p_j^*$), i.e., $maxdist(\overline{s_ls_m}, p_i) = dist(p_i, p_i^*)$ (respectively, $maxdist(\overline{s_ls_m}, p_j) = dist(p_j, p_j^*)$), as shown in Fig. 5(a). In the second step, we compute $maxdist(\overline{p_ip_j}, p_i^*)$ and $maxdist(\overline{p_ip_j}, p_j^*)$, as shown in Fig. 5(b), where data point p_i^{**} (respectively, p_j^{**}) indicates the farthest data point of p_i^* (respectively, p_j^*) such that $\omega(p_i^*, \overline{p_ip_j}) = p_i^{**}$ (respectively, $\omega(p_j^*, \overline{p_ip_j}) = p_j^{**}$) – that is, $maxdist(\overline{p_ip_j}, p_i^*) = dist(p_i^*, p_i^{**})$ – as shown in Fig. 5(b). Returning to the example, let us compute the maximum distances of the three segments $\overline{p_2p_1p_6p_5}$, $\overline{p_3p_4}$, and p_7 in Fig. 2(b). Specifically, we evaluate $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})$, $maxdist(\overline{p_2p_1p_6p_5}, p_7)$, $maxdist(\overline{p_3p_4}, p_7)$, $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_2p_1p_6p_5})$, $maxdist(\overline{p_3p_4}, \overline{p_3p_4})$, and $maxdist(p_7, p_7)$ in this order.

Fig. 6 illustrates how to compute $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})$. As shown in Fig. 6(a), the maximum distance between $\overline{p_2p_1p_6p_5}$ and p_3 is evaluated as $maxdist(\overline{p_2p_1p_6p_5}, p_3) = 9$ because we have $dist(p_2, p_3) = 3$, $dist(p_5, p_3) = 9$, and $len(p_2, p_5) = 6$. Therefore, the farthest data point of p_3 among $\overline{p_2p_1p_6p_5}$ is marked as $p_3^* = p_5$; that is, $\omega(p_3, \overline{p_2p_1p_6p_5}) = p_5$. As shown in Fig. 6(b), the maximum distance between $\overline{p_2p_1p_6p_5}$ and p_4 is evaluated as $maxdist(\overline{p_2p_1p_6p_5}, p_4) = 14$ because we have $dist(p_2, p_4) = 10$, $dist(p_5, p_4) = 12$, and $len(p_2, p_5) = 6$. Therefore, the farthest data point of p_4 among $\overline{p_2p_1p_6p_5}$ is marked as p_4^*; that is, $\omega(p_4, \overline{p_2p_1p_6p_5}) = p_4^*$. As shown in Fig. 6(c), the maximum distance between $\overline{p_3p_4}$ and p_5 is evaluated as $maxdist(\overline{p_3p_4}, p_5) = 14$ because we have $dist(p_3, p_5) = 9$, $dist(p_4, p_5) = 12$, and $len(p_3, p_4) = 7$. Therefore, the farthest data point of p_5 among $\overline{p_3p_4}$ is marked as p_5^*; that is, $\omega(p_5, \overline{p_3p_4}) = p_5^*$. As shown in Fig. 6(d), the maximum distance between $\overline{p_3p_4}$ and p_4^* is evaluated as $maxdist(\overline{p_3p_4}, p_4^*) = 14$ because we have

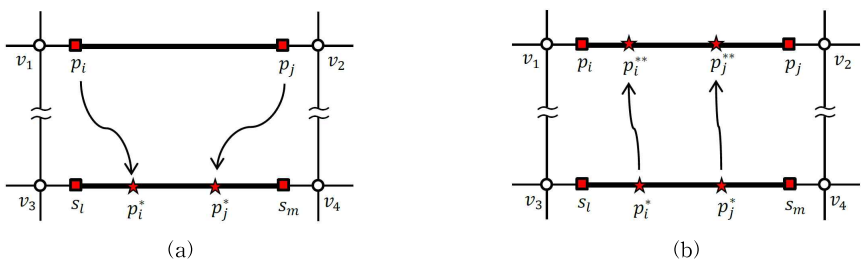


Fig. 5. Evaluation of $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) = max\{maxdist(\overline{p_ip_j}, p_i^*), maxdist(\overline{p_ip_j}, p_j^*)\}$. (a) Finding $\omega(p_i, \overline{s_ls_m}) = p_i^*$ and $\omega(p_j, \overline{s_ls_m}) = p_j^*$ (b) Evaluation of $maxdist(\overline{p_ip_j}, p_i^*)$ and $maxdist(\overline{p_ip_j}, p_j^*)$.

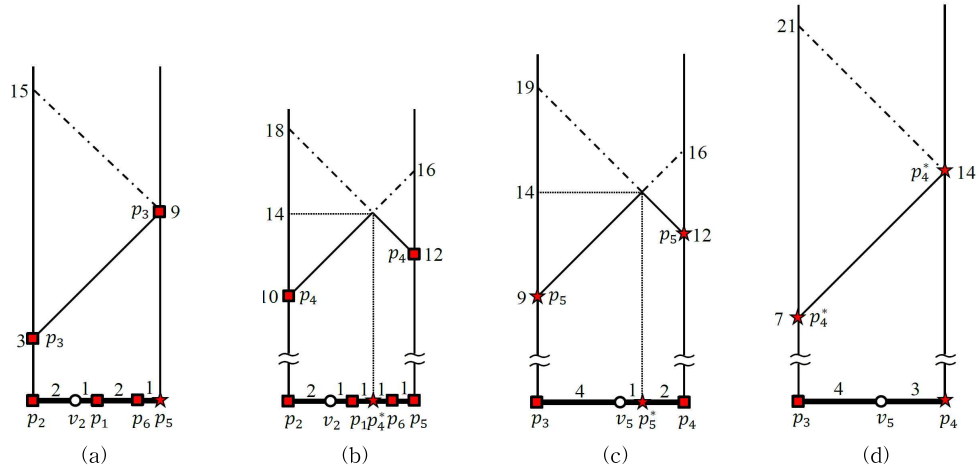


Fig. 6. $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})=14$. (a) $maxdist(\overline{p_2p_1p_6p_5}, p_3)=9$ and $\omega(p_3, \overline{p_2p_1p_6p_5})=p_5$ (b) $maxdist(\overline{p_2p_1p_6p_5}, p_4)=14$ and $\omega(p_4, \overline{p_2p_1p_6p_5})=p_4^*$ (c) $maxdist(\overline{p_3p_4}, p_5)=14$ and $\omega(p_5, \overline{p_3p_4})=p_5^*$ (d) $maxdist(\overline{p_3p_4}, p_4^*)=14$ and $\omega(p_4^*, \overline{p_3p_4})=p_4$.

$dist(p_3, p_4^*)=7$, $dist(p_4, p_4^*)=14$, and $len(p_3, p_4)=7$. Therefore, the farthest data point of p_4^* among $\overline{p_3p_4}$ becomes p_4; that is, $\omega(p_4^*, \overline{p_3p_4})=p_4$. Consequently, the maximum distance between $\overline{p_2p_1p_6p_5}$ and $\overline{p_3p_4}$ is evaluated as $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})=max\{dist(p_4^*, p_4), dist(p_5^*, p_5)\}=max\{14,14\}=14$. Clearly, the minimum distance between $\overline{p_2p_1p_6p_5}$ and $\overline{p_3p_4}$ is evaluated as $mindist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})=min\{dist(p_2, p_3), dist(p_5, p_3), dist(p_2, p_4), dist(p_5, p_4)\}=min\{3,9,10,12\}=3$. Table 2 summarizes the minimum and maximum distances between the segments in Fig. 2(b).

4.3 Sorting segments in decreasing order of their maximum distance

Fig. 7 illustrates sorting all segments in $\overline{P}$ for each segment $\overline{p_ip_j}$, for the example $\overline{P}=\{\overline{p_2p_1p_6p_5}, \overline{p_3p_4}, p_7\}$. Specifically, the three segments in $\overline{P}$ are sorted and plotted in decreasing order of the maximum distance to each segment in $\overline{P}$. If segments with the same maximum distance are found, they are repeatedly sorted in decreasing order of their minimum distance. As shown in Fig. 7(a), $\overline{p_3p_4}$, p_7, and $\overline{p_2p_1p_6p_5}$ are sorted in this order for $\overline{p_2p_1p_6p_5}$ because we have $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})=14$, $maxdist(\overline{p_2p_1p_6p_5}, p_7)=9$, and $maxdist(\overline{p_2p_1p_6p_5}, \overline{p_2p_1p_6p_5})=6$.

Table 2. Minimum and maximum distances between pairs of segments in $\overline{P}$

$\overline{p_ip_j}$	$\overline{s_ls_m}$	$maxdist(\overline{p_ip_j}, \overline{s_ls_m})$	$mindist(\overline{p_ip_j}, \overline{s_ls_m})$
$\overline{p_2p_1p_6p_5}$	$\overline{p_2p_1p_6p_5}$	$maxdist(\overline{p_2p_1p_6p_5}, \overline{p_2p_1p_6p_5})=6$	$mindist(\overline{p_2p_1p_6p_5}, \overline{p_2p_1p_6p_5})=0$
	$\overline{p_3p_4}$	$maxdist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})=14$	$mindist(\overline{p_2p_1p_6p_5}, \overline{p_3p_4})=3$
	p_7	$maxdist(\overline{p_2p_1p_6p_5}, p_7)=9$	$mindist(\overline{p_2p_1p_6p_5}, p_7)=5$
$\overline{p_3p_4}$	$\overline{p_2p_1p_6p_5}$	$maxdist(\overline{p_3p_4}, \overline{p_2p_1p_6p_5})=14$	$mindist(\overline{p_3p_4}, \overline{p_2p_1p_6p_5})=3$
	$\overline{p_3p_4}$	$maxdist(\overline{p_3p_4}, \overline{p_3p_4})=7$	$mindist(\overline{p_3p_4}, \overline{p_3p_4})=0$
	p_7	$maxdist(\overline{p_3p_4}, p_7)=10$	$mindist(\overline{p_3p_4}, p_7)=4$
p_7	$\overline{p_2p_1p_6p_5}$	$maxdist(p_7, \overline{p_2p_1p_6p_5})=9$	$mindist(p_7, \overline{p_2p_1p_6p_5})=5$
	$\overline{p_3p_4}$	$maxdist(p_7, \overline{p_3p_4})=10$	$mindist(p_7, \overline{p_3p_4})=4$
	p_7	$maxdist(p_7, p_7)=0$	$mindist(p_7, p_7)=0$

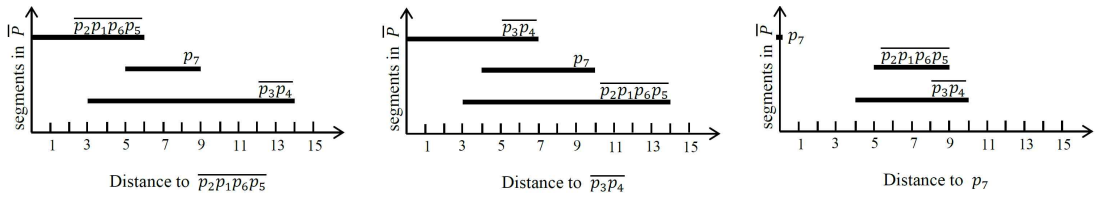


Fig. 7. Sorting all segments in decreasing order of their maximum distance to $\overline{p_ip_j} \in \{\overline{p_2p_1p_6p_5}, \overline{p_3p_4}, p_7\}$. (a) Sorting all segments for $\overline{p_2p_1p_6p_5}$ (b) Sorting all segments for $\overline{p_3p_4}$ (c) Sorting all segments for p_7.

As shown in Fig. 7(b), $\overline{p_2p_1p_6p_5}$, p_7, and $\overline{p_3p_4}$ are sorted in this order for $\overline{p_3p_4}$ because we have $maxdist(\overline{p_3p_4}, \overline{p_2p_1p_6p_5})=14$, $maxdist(\overline{p_3p_4}, p_7)=10$, and $maxdist(\overline{p_3p_4}, \overline{p_3p_4})=7$. Finally, as shown in Fig. 7(c), $\overline{p_3p_4}$, $\overline{p_2p_1p_6p_5}$, and p_7 are sorted in this order for p_7 because we have $maxdist(p_7, \overline{p_3p_4})=10$, $maxdist(p_7, \overline{p_2p_1p_6p_5})=9$, and $maxdist(p_7, p_7)=0$.

5. ALL-FARTHEST-NEIGHBORS SEARCH ALGORITHM OVER SPATIAL NETWORKS

Algorithm 1 describes the FAST algorithm for AFN search over spatial networks. The result set $\Omega(P)$ is initialized to the empty set (line 1). In the first step, adjacent data points $p_i, p_{i+1}, \cdots, p_j$ in a vertex sequence are grouped into a data segment $\overline{p_ip_j}$ and a set P of data points is converted to a set $\overline{P}$ of segments, which is explained in Section 4.1 (lines 2-3). The AFN search is performed for each segment in $\overline{P}$ to find the farthest neighbor of each data point in the segment (line 6). The result $\Omega(\overline{p_ip_j})$ of the AFN search for $\overline{p_ip_j}$ is added to the query result, where $\Omega(\overline{p_ip_j}) = \{\langle p, \omega(p)\rangle | p \in \overline{p_ip_j}\}$ (line 7). Finally, the query result $\Omega(P)$ is reported after the AFN search for all segments in $\overline{P}$ is performed (line 8).

Algorithm 2 describes the AFN search algorithm for finding the farthest neighbor of each data point in $\overline{p_ip_j}$. This algorithm sequentially traverses each of the sorted segments in $\overline{P}$. Note that all segments in $\overline{P}$ are sorted in decreasing order of their maximum distance to $\overline{p_ip_j}$. First, the result set $\Omega(\overline{p_ip_j})$ is initialized to the empty set (line 1). The pruning distance of each data point p in $\overline{p_ip_j}$ is initialized to $prundist(p)=0$ (line 2), where $prundist(p)$ indicates the distance from p to its most probable farthest neighbor. Similarly, the pruning distance of $prundist(\overline{p_ip_j})$ is the minimum pruning distance of all data points in $\overline{p_ip_j}$; that is, $prundist(\overline{p_ip_j}) = min\{prundist(p) | p \in \overline{p_ip_j}\}$. Based on Corollary 1, we pres-

Algorithm 1. FAST (P)

Input: P: set of data points

Output: : set of ordered pairs of each data point $p \in P$ and its farthest neighbor, i.e., $\Omega(P) = \{\langle p, \omega(p)\rangle | p \in P\}$.

1 $\Omega(P) \leftarrow \varnothing$ // the result set $\Omega(P)$ is initialized to the empty set.

2 // adjacent data points in each vertex sequence are grouped into a segment, as explained in Section 4.1.

3 $\overline{P} \leftarrow group\_points$ (P) // data points $p_i, p_{i+1}, \cdots, p_j$ in a vertex sequence are grouped into a data segment $\overline{p_ip_j}$.

4 // for each data segment, it retrieves the FN of each data point in the segment, which is detailed in Algorithm 2.

5 **for** each segment $\overline{p_ip_j} \in \overline{P}$ **do**

6 $\Omega(\overline{p_ip_j}) \leftarrow$ AFN_search$(\overline{p_ip_j}, \overline{P})$ // $\Omega(\overline{p_ip_j}) = \{\langle p, \omega(p)\rangle | p \in \overline{p_ip_j}\}$.

7 $\Omega(P) \leftarrow \Omega(P) \cup \Omega(\overline{p_ip_j})$ // the AFN search result for $\overline{p_ip_j}$ is added to $\Omega(P)$.

8 **return** $\Omega(P)$ // $\Omega(P)$ is returned after the AFN search for each data segment $\overline{p_ip_j} \in \overline{P}$ is executed.

Algorithm 2. AFN_search($\overline{p_ip_j}$, $\overline{P}$)

Input: $\overline{p_ip_j}$: data segment, $\overline{P}$: set of data segments

Output: $\Omega(\overline{p_ip_j})$: set of ordered pairs of each data point in $\overline{p_ip_j}$ and its farthest neighbor

1 $\Omega(\overline{p_ip_j}) \leftarrow \varnothing$ // the result set $\Omega(\overline{p_ip_j})$ is initialized to the empty set.

2 $prundist(p) \leftarrow 0$ for $\forall p \in \overline{p_ip_j}$ // the pruning distance of each data point p in $\overline{p_ip_j}$ is initialized to 0.

3 // the maximum and minimum distances from $\overline{p_ip_j}$ to each segment in $\overline{P}$ are computed as explained in Section 4.2.

4 **for** each data segment $\overline{s_ls_m} \in \overline{P}$ **do**

5 compute $maxdist(\overline{p_ip_j}, \overline{s_ls_m})$ and $maxdist(\overline{p_ip_j}, \overline{s_ls_m})$

6 // all the segments in $\overline{P}$ are sorted in decreasing order of their maximum distance to $\overline{p_ip_j}$ as explained in Section 4.3.

7 $\overline{P} \leftarrow sort\_by\_dec\_order\ (\overline{P})$ // $\overline{P}$ contains the sorted segments for the data segment $\overline{p_ip_j}$.

8 **for** each data segment $\overline{s_ls_m} \in \overline{P}$ **do**

9 $prundist(\overline{p_ip_j}) \leftarrow min\{prundist(p) | p \in \overline{p_ip_j}\}$ // the pruning distance of $\overline{p_ip_j}$ is updated.

10 **if** $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) < prundist(\overline{p_ip_j})$ **then** // note that all the segments are sorted in decreasing order.

11 **return** $\Omega(\overline{p_ip_j})$ // $\Omega(\overline{p_ip_j})$ is returned if $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) < prundist(\overline{p_ip_j})$.

12 **else** // it means that $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) \geq prundist(\overline{p_ip_j})$.

13 **for** each data point $p \in \overline{p_ip_j}$ **do**

14 **if** $maxdist(\overline{s_ls_m}, p) \geq prundist(p)$ **then** // $prundist(p)$ is updated to $dist(p, \omega(p))$.

15 **for** each data point $s \in \overline{s_ls_m}$ **do**

16 **if** $dist(p,s) > prundist(p)$ **then**

17 $prundist(p) \leftarrow dist(p,s)$ // the pruning distance of p is updated to $dist(p,s)$.

18 $\Omega(\overline{p_ip_j}) \leftarrow \Omega(\overline{p_ip_j}) \cup \{\langle p, s\rangle\}$ // an updated ordered pair $\langle p, s\rangle$ is added to $\Omega(\overline{p_ip_j})$ $\{\langle p, \omega(p)\rangle\}$

19 **return** $\Omega(\overline{p_ip_j})$ // $\Omega(\overline{p_ip_j})$ is returned if all segments in $\overline{P}$ have been explored

ent the following two corollaries to filter the remaining unexplored segments, because these segments can be safely neglected in the AFN search.

Corollary 2. If $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) < prundist(\overline{p_ip_j})$, the remaining unexplored segments can be safely ignored for the AFN search of the segment $\overline{p_ip_j}$, because all segments in $\overline{P}$ are sorted in decreasing order of their maximum distance to $\overline{p_ip_j}$. ■

Corollary 3. If $maxdist(\overline{s_ls_m}, p) < prundist(p)$, the distance computation between data point p and each data point in $\overline{s_ls_m}$ can be safely ignored for the AFN search of p, because no data point in $\overline{s_ls_m}$ can be the farthest neighbor of p. ■

All segments are explored sequentially to find the farthest neighbor of each data point p in $\overline{p_ip_j}$. Whenever a data segment $\overline{s_ls_m}$ is investigated, the pruning distance of $\overline{p_ip_j}$ is updated to the minimum pruning distance of all data points in $\overline{p_ip_j}$ (line 9). According to Corollary 2, if $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) < prundist(\overline{p_ip_j})$, the AFN search algorithm terminates by returning the result set $\Omega(\overline{p_ip_j})$; otherwise, $\overline{s_ls_m}$ is explored to find the farthest neighbor of each data point in $\overline{p_ip_j}$. According to Corollary 3, if $maxdist(\overline{s_ls_m}, p) < prundist(p)$, no data point in $\overline{s_ls_m}$ can be the farthest neighbor of p, and so the distance computation between p and all data points in $\overline{s_ls_m}$ can be safely ignored for p; otherwise, for each data point s in $\overline{s_ls_m}$, we compute $dist(p,s)$. If $dist(p,s) > prundist(p)$, the farthest neighbor of p and its pruning distance are updated to s and $dist(p,s)$, respectively (lines 17-18). Accordingly, the pruning distance of p is also updated to $prundist(p) = dist(p,s)$. Finally, the result set $\Omega(\overline{p_ip_j})$ is returned if $maxdist(\overline{p_ip_j}, \overline{s_ls_m}) < prundist(\overline{p_ip_j})$ (lines 10-11) or

if all segments in $\overline{P}$ have been explored (line 19).

6. PERFORMANCE STUDY

In this section, we report the results of an empirical analysis of our proposed solution. We present our experimental settings in Section 6.1, followed by our experimental results in Section 6.2.

6.1 Experimental settings

In the experiments, we use three real-world roadmaps [26], which are described in Table 3. These real-world roadmaps have different sizes and each is a part of the US road network. Specifically, the first roadmap includes major roads (such as city streets) in San Francisco (SF), California and corresponds to a data universe of approximately 6×10^2 km^2. The second roadmap includes major roads (such as highways) in North America (NA) and corresponds to a data universe of approximately 2.5×10^7 km^2. The third roadmap includes major roads (e.g., city streets) in San Joaquin (SJ), California and corresponds to a data universe of approximately 3.6×10^3 km^2.

The experimental parameter settings are given in Table 4. For convenience, each dimension of the data universe is normalized independently to unit length [0,1]. The data points have either centroid or uniform distributions. The centroid-based dataset is generated to resemble the real-world data. First, five centroids are selected randomly. The data points around each centroid follow a normal distribution, where the mean is set to the centroid and the standard deviation is set to 1% of the side length of the data universe.

In the performance study, we evaluated the performance of FAST using the query processing time. To the best of our knowledge, there is no competitive solution to efficiently evaluate AFN queries in spatial network databases, so we considered a baseline method as a benchmark to verify the performance of the FAST method. The baseline method finds the farthest neighbor of each data point in P by computing the network distance between every ordered pair $\langle p,s\rangle\in P\times P$ and then spending additional time $O(|P|^2)$ to identify the farthest ordered pairs among all pairs in $P\times P$. Both methods were implemented in C++ (Microsoft Visual Studio 2017), using common subroutines for similar tasks. We conducted experiments on a desktop computer running Windows 10 with a 4.2 GHz processor and 32 GB of memory. We believe that indexing structures of all techniques should be resident in memory to ensure responsive query processing; this is assumed in many recent studies [3, 24] and is crucial to commercial LBS as well as online map services. We determined the average values for each method based on repetitions of the experiments. Finally, we employed the TNR method [16] to rapidly compute the network distance between two data points. According to the performance comparison in [24], the TNR method

Table 3. Real-world roadmaps

Name	Description	Vertices	Edges	Vertex sequences
SF	City streets in San Francisco, California	174,956	223,001	192,276
NA	Highways in North America	175,813	179,179	12,416
SJ	City streets in San Joaquin, California	18,263	23,874	20,040

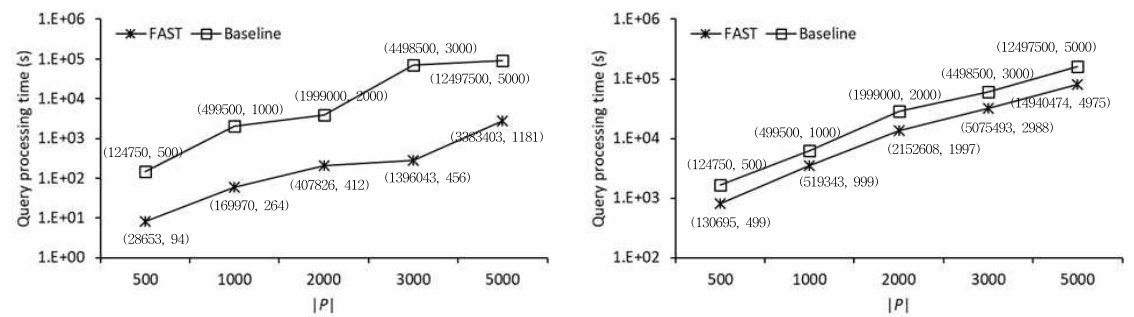
Table 4. Experimental parameter settings

Parameter	Range
Number of data points ($\|P\|$)	500, 1000, 2000, 3000, 5000
Distribution of data points	(C)entroid, (U)niform
Roadmap	SF, NA, SJ

achieves competitive performance regardless of the benchmark where it is tested.

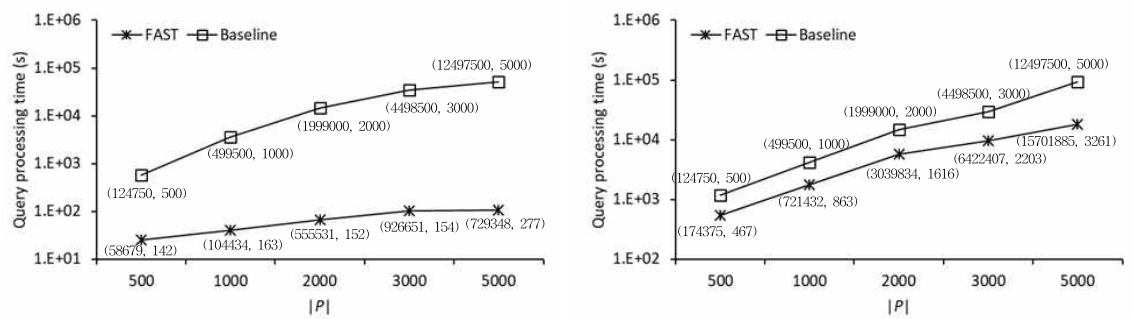
6.2 Experimental results

Fig. 8 shows a comparison of the query processing times of the FAST and baseline methods for the evaluation of AFN queries in the SF roadmap. To evaluate the performance of the FAST method, we examined the number of network distance computations and data segments generated from adjacent data points for each experimental setup. The two values in parentheses in Fig. 8, 9, and 10 indicate the number of network distance computations performed by the FAST method and the number of data segments generated from adjacent data points. Similarly, we have also represented the number of network distance computations for the baseline method and the number of data points to Fig. 8, 9, and 10 for comparison. As shown in Fig. 8(a), the query processing time of the FAST method for a centroid distribution of data points is less than that of the baseline algorithm by up to 22.8 times for $|P| = 5{,}000$. The difference in performance between the FAST and baseline algorithms tends to increase with because the FAST method exploits the shared execution of AFN searches during the query evaluation. The shared execution of the FAST method aims to minimize the number of network distance computations. To do this, the FAST method groups the adjacent data points into a joint data segment and exploits shared computation for this data segment to rapidly filter candidates by computing the maximum distance between two data segments. Therefore, as the number of data segments generated from the adjacent data points decreases, the difference between the FAST and baseline methods in terms of performance increases. The query processing time of the



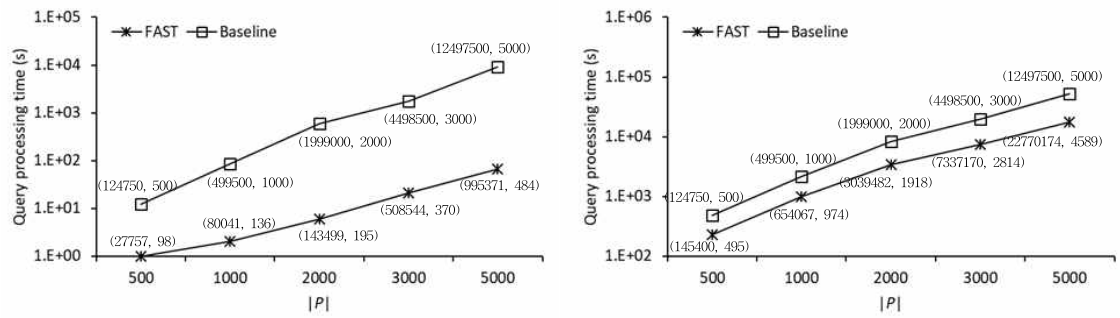
(a) (b)

Fig. 8. Comparison of query processing time for SF roadmap. (a) Centroid distribution (b) Uniform distribution.



(a) (b)

Fig. 9. Comparison of query processing time for NA roadmap. (a) Centroid distribution (b) Uniform distribution.



(a) (b)

Fig. 10. Comparison of query processing time for SJ roadmap. (a) Centroid distribution (b) Uniform distribution.

baseline method increases steeply with $|P|$ because the distance computation, to determine the farthest neighbor of each data point in P, has $O(|P|^2)$ time complexity. As shown in Fig. 8(b), the two methods have similar performance for uniformly distributed data points in the SF roadmap. This is expected because the data points are widely scattered, which obstructs the shared execution processing of the FAST method. Despite the uniform distribution of the data points, the query processing time of the FAST method is up to 2.2 times less than that of the baseline method in all cases.

Fig. 9 shows a comparison of the query processing times of the two methods for AFN queries in the NA roadmap. As shown in Fig. 9(a), the FAST method requires a shorter query processing time than the baseline method by up to 268 times for $|P|=2,000$. The FAST method requires the smallest query processing time for $|P|=2,000$, indicating that the shared execution of the network distance computations is significantly affected by the distribution of data points. The query processing time of the baseline method increases rapidly with $|P|$ because it requires a large number, $O(|P|^2)$, of network distance computations to determine the farthest neighbor of each data point in P. However, as shown in Fig. 9(b), the two methods show similar performance when the data points follow a uniform distribution in the NA roadmap. The reason is that the data points are widely scattered, which obstructs shared execution processing of the FAST method. Nevertheless, the FAST method clearly outperforms the baseline method by up to 5.7 times in all cases for the uniform distribution.

Fig. 10 shows a comparison of the query processing times of the two methods for evaluating AFN queries in the SJ roadmap. As shown in Fig. 10(a), the FAST method has shorter query processing times than the baseline method by up to 101 times for $|P|=5,000$. The query processing time of the FAST method is less sensitive to $|P|$ than the baseline method for a centroid distribution of data points. Specifically, the FAST method has a longer query execution time for $|P|=3,000$ than for $|P|=5,000$ whereas the query processing time of the baseline method increases with $|P|$. This is because the FAST method exploits the shared execution of the network distance computations and the baseline method requires a large number, $O(|P|^2)$, of distance computations to determine the farthest neighbor of each data point in P. However, as shown in Fig. 10(b), the difference in performance between the FAST and baseline algorithms decreases for a uniform distribution of data points compared to a centroid distribution. This is because the data points are widely scattered, which obstructs shared execution processing of the FAST method. The FAST method clearly outperforms the baseline method by up to 3.1 times in all cases of the uniform distribution of the data points.

7. CONCLUSIONS

In this research, we studied algorithms for the efficient processing of AFN queries in spatial network databases. The processing of AFN queries in spatial networks cannot be achieved by the straightforward application of previous approaches (e.g., [5, 6]) in Euclidean space because of the complexity of network distance computations, as opposed to Euclidean distance computations. To overcome this difficulty, we propose the FAST method, which employs grouping of adjacent data points and shared execution to avoid redundant network distance computations. We compared the performance of the FAST and baseline methods using several real-life roadmaps in a wide range of problem settings. The empirical results confirmed that the FAST algorithm is efficient and scalable with the number of data points and is significantly superior to the baseline algorithm, particularly when data points are not distributed uniformly over the spatial network.

REFERENCES

[1] Y. Chen and J.M. Patel, "Efficient Evaluation of All-nearest-neighbor Queries," *Proceeding of International Conference on Data Engineering*, pp. 1056-1065, 2007.

[2] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-nearest-neighbors Queries in Spatial Databases," *Proceeding of International Conference on Scientific and Statistical Database Management*, pp. 297-306, 2004.

[3] T. Abeywickrama, M.A. Cheema, and D. Taniar, "K-nearest Neighbors on Road Networks: A Journey in Experimentation and In-Memory Implementation," *Proceeding of International Conference on Very Large Data Bases Endowment*, Vol. 9, No. 6, pp. 492-503, 2016.

[4] O.S. Kwon and H.J. Cho, "Design and Implementation of Real-time Shortest Path Search System in Directed and Dynamic Roads," *Journal of Korea Multimedia Society*, Vol. 20, No. 4, pp. 649-659, 2017.

[5] A. Aggarwal and D. Kravets, "A Linear Time Algorithm for Finding All Farthest Neighbors in a Convex Polygon," *Information Processing Letters*, Vol. 31, No. 1, pp. 17-20, 1989.

[6] S.W. Bae, M. Korman, and T. Tokuyama, "All Farthest Neighbors in the Presence of Highways and Obstacles," *Proceeding of International Workshop on Algorithms and Computation*, pp. 71-82, 2009.

[7] R.R. Curtin, J. Echauz, and A.B. Gardner, "Exploiting the Structure of Furthest Neighbor Search for Fast Approximate Results," *Information Systems*, Vol. 80, pp. 124-135, 2019.

[8] S. Wang, M.A. Cheema, X. Lin, Y. Zhang, and D. Liu, "Efficiently Computing Reverse k Furthest Neighbors," *Proceeding of International Conference on Data Engineering*, pp. 1110-1121, 2016.

[9] B. Yao, F. Li, and P. Kumar, "Reverse Furthest Neighbors in Spatial Databases," *Proceeding of International Conference on Data Engineering*, pp. 664-675, 2009.

[10] Y. Gao, L. Shou, K. Chen, and G. Chen, "Aggregate Farthest-neighbor Queries over Spatial Data," *Proceeding of International Conference on Database Systems for Advanced Applications*, pp. 149-163, 2011.

[11] H. Wang, K. Zheng, H. Su, J. Wang, S.W. Sadiq, and X. Zhou, "Efficient Aggregate Farthest Neighbour Query Processing on Road Networks," *Proceeding of the Australasian Database Conference*, pp. 13-25, 2014.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, The Massachusetts Institute of Technology Press, London, England, 2009.

[13] H. Mahmud, A.M. Amin, M.E. Ali, T. Hashem, and S. Nutanong, "A Group Based Approach for Path Queries in Road Networks," *Pro-*

ceeding of International Symposium on Spatial and Temporal Databases, pp. 367-385, 2013.

[14] J.R. Thomsen, M.L. Yiu, and C.S. Jensen, "Effective Caching of Shortest Paths for Location-Based Services," *Proceeding of International Conference on Management of Data*, pp. 313-324, 2012.

[15] 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," *Distributed and Parallel Databases*, Vol. 31, No. 2, pp. 259-287, 2013.

[16] H. Bast, S. Funke, and D. Matijevic, "TRANSIT: Ultrafast Shortest-path Queries with Linear-time Preprocessing," *Proceeding of the 9th DIMACS Implementation Challenge*, pp. 175-192, 2006.

[17] 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.

[18] H. Lu and M.L. Yiu, "On Computing Farthest Dominated Locations," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 23, No. 6, pp. 928-941, 2011.

[19] Q.T. Tran, D. Taniar, and M. Safar, "Reverse k Nearest Neighbor and Reverse Farthest Neighbor Search on Spatial Networks," *Transactions on Large-scale Data- and Knowledge-centered Systems*, pp. 353-372, 2009.

[20] X.J. Xu, J.S. Bao, B. Yao, J. Zhou, F. Tang, M. Guo, et al., "Reverse Furthest Neighbors Query in Road Networks," *Journal of Computer Science and Technology*, Vol. 32, No. 1, pp. 155-167, 2017.

[21] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proceeding of International Conference on Management of Data*, pp. 47-57, 1984.

[22] O. Daescu, N. Mi, C.S. Shin, and A. Wolff, "Farthest-point Queries with Geometric and Combinatorial Constraints," *Computational Geometry*, Vol. 33, No. 3, pp. 174-185, 2006.

[23] N. Katoh and K. Iwano, "Finding k Farthest Pairs and k Closest/Farthest Bichromatic Pairs for Points in the Plane," *International Journal of Computational Geometry and Applications*, Vol. 5, No. 01n02, pp. 37-51, 1995.

[24] 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," *The Proceeding of the Very Large Data Bases Endowment*, Vol. 5, No. 5, pp. 406-417, 2012.

[25] 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.

[26] Real Datasets for Spatial Databases, https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm (accessed July 20, 2019).

Hyung-Ju Cho

is an associate professor at the department of software, Kyungpook National University. His current research interests include moving object databases and query processing in mobile peer-to-peer networks.