Authentication of k Nearest Neighbor Query on Road Networks

Yinan Jing, *Member, IEEE*, Ling Hu, Wei-Shinn Ku, *Senior Member, IEEE*, and Cyrus Shahabi

Abstract—Outsourcing spatial databases to the cloud provides an economical and flexible way for data owners to deliver spatial data to users of location-based services. However, in the database outsourcing paradigm, the third-party service provider is not always trustworthy, therefore, ensuring spatial query integrity is critical. In this paper, we propose an efficient road network k-nearest-neighbor query verification technique which utilizes the network Voronoi diagram and neighbors to prove the integrity of query results. Unlike previous work that verifies k-nearest-neighbor results in the Euclidean space, our approach needs to verify both the distances and the shortest paths from the query point to its kNN results on the road network. We evaluate our approach on real-world road networks together with both real and synthetic points of interest datasets. Our experiments run on Google Android mobile devices which communicate with the service provider through wireless connections. The experiment results show that our approach leads to compact verification objects ($\mathcal{VO}$) and the verification algorithm on mobile devices is efficient, especially for queries with low selectivity.

Index Terms—Spatial database outsourcing, location-based services, query authentication, road networks

1 Introduction

The combination of mobile devices and Cloud-based solutions is creating a versatile ecosystem for reshaping the way geospatial data are stored, managed, served, and shared. In this new ecosystem, also known as *database outsourcing*, the data owner (DO) delegates the management of its database to a third-party Cloud service provider (SP), and the SP server is responsible for indexing the data, answering client queries, and updating the data on requests from the DOs. Mobile clients, which are used to send their queries to DOs, now submit queries to SP and retrieve results from SP directly. For example, Microsoft Bing Maps partners with NAVTEQ, a major provider of base electronic navigable maps, to provide web mapping services for the public. In this case, NAVTEQ is the data owner (DO), and Bing Maps is the service provider (SP). The general architecture of the database outsourcing model is shown in Fig. 1.

Cloud computing provides flexible resources that can easily be scaled up or down based on user demands, effectively reducing the operational and maintenance expenses for data owners. The new *pay-as-you-go* business model and *Data as a Service* (DaaS) model have shown to be good fits for businesses with dynamic requirements and needs. According to a study from IDC [1], the spending on the IT Cloud services is expected to grow at the rate of 26% annually from 2009 to 2013, over six times the rate of traditional IT offerings. Consequently, outsourcing databases to the Cloud is becoming increasingly popular and has received considerable attention from the research community.

Although database outsourcing provides data owners with a more efficient, economical, and flexible solution, it also introduces new concerns [2]. In this paper, we study one of those concerns, in particular, the query integrity concern. That is, how to ensure that the query results returned by SP are still trustworthy. As SP is not the real owner of the data, it might return incorrect results to mobile clients out of its own interests, for example, an SP which hosts a collection of restaurants might favor some restaurants that pay more advertisement fees. Moreover, an SP might return suboptimal results to query clients by applying flawed or inferior algorithms in order to save computation resources. For example, as of today, the Google Maps [3] online service still provides users with k-nearest-neighbor results based on their Euclidean distances instead of their road network distances, not because they cannot compute the road network distances, but because it is much less expensive to compute the Euclidean distances. On the other hand, with the growing popularity of the Cloud, more and more security breaches and attacks on such systems have been brought to people's attention. For example, in June 2011, Dropbox temporarily allowed visitors to login to any of its 25 million user accounts due to a software glitch [4]. As a result, although commercial SPs are generally unlikely to provide inaccurate results to users, they could act involuntarily malicious and serve false results unintentionally to end users, e.g., when the SP server or the communication

- *Y. Jing is with the School of Computer Science, Fudan University, Shanghai 201203, China. E-mail: jingyn@fudan.edu.cn.*
- *L. Hu is with Google Inc, Mountain View, CA 94043-1351, USA. E-mail: lingb@google.com.*
- *W.-S. Ku is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA. E-mail: weishinn@auburn.edu.*
- *C. Shahabi is with the Computer Science Department, University of Southern California, Los Angeles, CA 90089, USA. E-mail: shahabi@usc.edu.*

Manuscript received 25 Mar. 2013; revised 13 Oct. 2013; accepted 26 Oct. 2013. Date of publication 7 Nov. 2013; date of current version 29 May 2014.
Recommended for acceptance by J. Pei.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier 10.1109/TKDE.2013.174

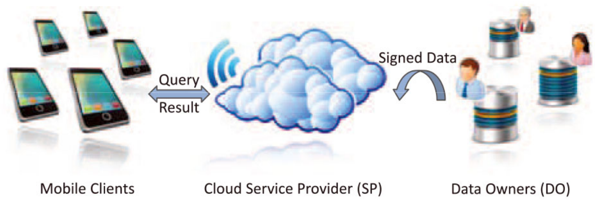


Fig. 1. Database outsourcing architecture.

channel between SP and clients is compromised by attackers. Therefore, providing a mechanism that allows clients to authenticate the *correctness* and *completeness* of the query result is necessary. Specifically, *correctness* means all data returned by SP originate from DO without any falsification and the query result is identical to that computed by DO. *Completeness* means all eligible results have been included by SP in the result set, i.e., there is no false missing of correct answers.

The general framework commonly used in the literature for query integrity assurance is based on digital signatures and utilizes a public-key cryptosystem, such as RSA [5]. Initially, DO obtains a *private* and a *public* key through a trusted key distribution center. The private key is kept secret at DO, whereas the public key is accessible by all clients. Using its private key, the DO digitally signs the data by generating a number of signatures. Then, it sends the signatures and the data to SP which constructs the necessary data structures for efficient query processing. When SP receives a query from a client, it generates a *verification object* ($\mathcal{VO}$) that contains the result set along with the corresponding authentication information. Finally, SP sends the $\mathcal{VO}$ to the client, which can verify the results using the public key of the DO.

Most existing works [6]–[14] solve query verification problems in the Euclidean space where the distance between two objects is measured by a straight line. That is, the distance depends only on the locations of the two points. Approaches proposed in this domain include the Merkle Hash Tree (MHT) augmented spatial indexes (such as MR-tree [7] and MR*-tree [8]) and the signature chain based approaches (for example, iDistance [6] and VN-Auth [12]). Unfortunately, Euclidean distance is not an appropriate distance measure for many real-world applications where objects can only move on predefined paths, such as streets on a road network. Furthermore, there might exist multiple paths between two points on a road network. Therefore, simply verifying distances is not sufficient for queries on road networks. We also need an approach which can verify the path between two points. Thus, existing approaches, which only focus on verification in the Euclidean space, cannot solve the spatial network query verification problem. In this work, we specifically focus on the k-nearest-neighbor (kNN) query verification on road networks and design verification schemes which support both *distance verification* and *path verification*. That is (i) the k resulting objects have the shortest distances to the query point among all the objects in the database, and (ii) the path from the query point to each k-nearest-neighbor result is the valid shortest path on the network.

In order to verify the kNN query result on a road network, a naïve solution would be to return the whole road network and the point of interest (POI) dataset to the client to show correctness and completeness of the result. However, this approach will incur a prohibitive communication overhead between SP and the client. This paper subsumes and extends our earlier work [15] by providing a sound proof for our kNN query verification scheme, which utilizes the network Voronoi diagram to generate a compact $\mathcal{VO}$, as well as ensuring the completeness and correctness of the kNN query result with regard to both distances and paths. Furthermore, this paper proposes a pre-computation based verification scheme to accelerate the verification on mobile clients by utilizing the distance pre-computation. In addition, this paper includes a discussion on updates of the outsourced database. Subsequently, we propose two update modes: the one-by-one update mode and the batch update mode. Finally, we conduct extensive experiments using real-world and synthetic datasets to evaluate the verification performance and the database update cost.

The remainder of the paper is organized as follows. Section 2 reviews related work and Section 3 discusses Voronoi diagrams in both the Euclidean space and the network space. Section 4 introduces the authentication data structure. Section 5 describes the verification approach for network k-nearest-neighbor queries. Section 6 discusses how to handle database updates. Experiment results are reported in Section 7. Finally, Section 8 concludes the paper and provides directions of future work.

2 Related Work

In this section, we review previous work related to k-nearest-neighbor query and query authentication methods for outsourced spatial databases.

2.1 Nearest Neighbor Query

The k-nearest-neighbor search is a very important query type for supporting Geospatial applications. Since most mobile users move on roads in reality, nearest neighbor search algorithms have been extended to support queries on spatial networks. Papadias *et al.* [16] proposed techniques for NN queries in spatial network databases by progressively expanding road segments around the query point. In addition, an approach based on the network Voronoi diagram was introduced in [17] for evaluating NN queries on spatial networks. The key idea is to partition a large network into a number of small Voronoi regions and then pre-compute distances both within and across the regions. Hu *et al.* [18] presented an index structure for distance computation and query processing over long distances. The index structure discretizes the distance between objects and network nodes into categories and then encodes these categories. In order to speed up NN search in a spatial network, Samet *et al.* [19] presented a solution to explore the entire spatial network by pre-computing the shortest paths between all the vertices in the network and using a shortest path quadtree to capture spatial coherence. By employing their approach, the shortest paths between various vertices can be computed only once to answer different NN queries in a given spatial network. However, these pre-computation-based approaches incur high I/O and computation costs.

To overcome the shortcomings, Lee *et al.* [20] proposed a query framework named ROAD, which organizes a large road network as a hierarchy of interconnected regional subnetworks. ROAD maintains objects separately from a given network and adopts an effective search space pruning technique to enhance search performance. Nevertheless, none of the aforementioned mechanisms has considered the query integrity problem.

2.2 Query Authentication for Outsourced Spatial Databases

Hacigümüs *et al.* were the first to propose the idea of outsourcing databases to third-party service providers in their pioneering work [2]. Afterward, numerous query authentication solutions have been proposed for outsourced relational databases [21]–[27]. Mykletun *et al.* [23] provided techniques based on digital signature aggregation to ensure data integrity and authenticity for outsourced databases. However, the techniques cannot assure completeness of the result set. Pang *et al.* [24] employed an aggregated signature in order to sign each record with the information from neighboring records by assuming that all the records are sorted in a certain order. Their mechanism helps users verify that query results are both complete and authentic. In addition, the challenge token scheme, introduced by Sion [25], is for a server running outsourced databases to provide a proof of the actual query execution, which is then checked at the client side for integrity verification. Compared to [24], the scheme also supports more query types without assuming that all the records are sorted. Nonetheless, none of the these techniques are specifically designed for spatial databases.

For authenticating spatial queries in the Euclidean space, Cheng and Tan [6] proposed VR-tree-based mechanisms for verifying *k*NN and several other spatial query types on multi-dimensional databases. Their mechanisms ensure that query results are complete, authentic, and minimal. Yang *et al.* [7], [8] introduced the MR- and MR*-trees, which are space-efficient authenticated data structures for spatial query verification. Their schemes extend the R-tree by computing hash digests on the concatenation of the binary representation of all the entries in a tree node. To validate the correctness and completeness of query results, the generated $\mathcal{VO}$ includes (i) all visited objects and (ii) MBRs and digests of all the pruned nodes. Although the MR- and MR*-tree based approaches are efficient in verifying certain types of spatial queries, they suffer from a few drawbacks, such as the efficiency-degraded index structure due to the storage of additional hashes, the bottleneck on the tree root node on updates, and the bulky $\mathcal{VO}$ sizes especially for queries with low selectivity, such as *k*NN queries. Please refer to [12], [13] for more discussions on the drawbacks and experimental results that show why this authenticated spatial index data structure is not suitable for light location-based queries. Furthermore, Papadopoulos *et al.* [28] extended the technique in [8] to authenticate multi-step nearest neighbor search involving high-dimensional data, but they did not tackle the distance verification and the path verification on road networks. The Partially Materialized Digest (PMD) scheme [9] verifies spatial queries and applies to databases with frequent updates. Similar to our approach, PMD utilizes separate indexes for data and the associated verification information in order to save resources when processing queries that do not request authentication. A common limitation of the aforementioned solutions is that they all require the modification of the spatial indices or query algorithms of the DBMS in order to support the embedded authentication information. However, the requirement may not be realistic in many applications. Ku *et al.* [10], [11] proposed a query integrity assurance technique that does not require any modifications in the DBMS. The solution first employs a spatial transformation method that encrypts the spatial data before outsourcing them to the SP server. Then, by probabilistically replicating a portion of the data and encrypting it with a different encryption key, clients are able to audit the trustworthiness of query results. Nevertheless, since the method in [10], [11] is not a deterministic solution, attacks may escape the auditing process. Therefore, Hu *et al.* [12], [13] designed the VN-Auth scheme, which is both efficient and deterministic, and does not require any modifications in the DBMS. However, all above-mentioned solutions cannot be employed to verify *k*NN queries on road networks.

Goodrich *et al.* [29] studied the authentication of connectivity queries on graphs, which can verify whether two points are connected in the graph. For verifying spatial queries on road networks, Yiu *et al.* [30] studied the shortest path verification on road networks where, given a path between two points on the road network, a client can verify whether this path is indeed the shortest among all the other paths. However, approaches proposed in [29], [30] do not apply to the network *k*NN query verification problem because their approaches cannot verify whether a POI is the closest point (or the i^{th} closest point) to the query point. A recent work [31] studied a challenging problem of privacy-preserving range query authentication in the context where the locations of queried objects (users) are sensitive. This is different from the application context discussed in this paper where queried objects are POIs whose locations are public and are incorporated into the $\mathcal{VO}$ to enable a query client to verify the distance and path to queried objects. Hence, our approach is not directly applicable to solve the problem presented in [31].

3 Preliminary

In this section, we discuss the definitions and important properties of Voronoi diagrams on both the Euclidean space and road networks.

3.1 Voronoi Diagrams

Given a set of distinct objects $P = \{p_1, p_2, \dots, p_t\}$ in $\mathbb{R}^2$, the *Voronoi diagram* of P, denoted as $\mathbb{VD}(P)$, partitions the space of $\mathbb{R}^2$ into t disjoint regions, such that each object p_i in P belongs to only one region and every point in that region is closer to p_i than to any other object of P in the Euclidean space. The region around p_i is called the *Voronoi polygon* or *Voronoi cell* of p_i, denoted as $VC(p_i)$, and p_i is the *generator* of the Voronoi cell. Therefore, the Voronoi diagram of P is the union of all Voronoi cells

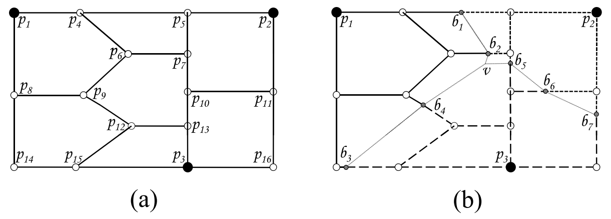


Fig. 2. Example of road network and network Voronoi diagram: (a) Road network. (b) Network Voronoi diagram.

$\mathbb{VD}(P) = \{VC(p_1), VC(p_2), \ldots, VC(p_t)\}$. If two generators share a common edge, they are Voronoi neighbors.

Property 1. *Given a set of distinct points $P = \{p_1, p_2, \ldots, p_t\} \subset \mathbb{R}^2$, the Voronoi diagram $\mathbb{VD}(P)$ of P is unique.*

Property 2. *The average number of edges per Voronoi polygon does not exceed six. That is, the average number of Voronoi neighbors per generator does not exceed six.*

Property 3. *Given the Voronoi diagram of P, the nearest neighbor of a query point q is p, if and only if $q \in VC(p)$.*

Property 4. *Let $p_1, \ldots, p_k$ be the k ($k > 1$) nearest neighbors in P to a query point q. Then, p_k is a Voronoi neighbor of at least one object $p_i \in \{p_1, \ldots, p_{k-1}\}$.*

Please refer to [32] (Properties 1 - 3) and [17] (Property 4) for proofs of the above properties.

3.2 Road Network Voronoi Diagrams

In many realistic applications, the distance between two points is measured by the road network distance instead of their Euclidean distance, assuming objects can only move along street systems. A road network system can be modeled as a weighted graph $\mathbb{G}(\mathbb{V}, \mathbb{E}, \mathbb{W})$ consisting of a set of vertices $\mathbb{V} = \{p_1, p_2, \ldots, p_n\}$ and a set of edges $\mathbb{E} = \{e_1, e_2, \ldots, e_m\}$ connecting vertices to form a graph. $\mathbb{W}$ represents the cost of each edge in $\mathbb{E}$, for example, the distance, the traveling time or the toll fees. Let $P \subset \mathbb{V}$ be a set of points of interest (POI). We assume all POIs are restricted on road network segments (edges).

An example of a road network, a set of POIs, and the corresponding Network Voronoi Diagram (**NVD**) are shown in Fig. 2 [32]. The original road network is represented as a graph in Fig. 2(a) where p_1, p_2, and p_3 are points of interest and p_4-p_{16} are intersections on the road network. For each POI p_i, we define the following:

Definition 1. Dominance Region

$$Dom(p_i, p_j) = \{p | p \in \bigcup_{i=1}^{m} e_i, d_n(p, p_i) \leq d_n(p, p_j), i \neq j\},$$

where p_i and p_j are POIs and $d_n(p, q)$ is the shortest path distance between p and q on the road network.

The dominance region of p_i over p_j defines all the points on all edges in $\mathbb{E}$ that are closer to p_i than to p_j (or have equal distances to p_i and p_j). Next, the network Voronoi cell (**NVC**) of p_i and the network Voronoi diagram can be defined as follows.

Definition 2. Network Voronoi Cell

$$V(p_i) = \bigcap_{j \neq i} Dom(p_i, p_j).$$

Definition 3. Network Voronoi Diagram

$$NVD(P) = \{V(p_1), \ldots, V(p_t)\}.$$

The network Voronoi cell $V(p_i)$ contains all points on edges that are closer to p_i than to any other POIs. It is actually a shortest path tree generated from p_i, and hence p_i is also called the *generator* of $V(p_i)$. Note that, different from the Voronoi Diagram in the Euclidean space where each Voronoi cell is a continuous area, the network Voronoi cell of each generator contains a set of road segments. In Fig. 2(b), $V(p_1)$, $V(p_2)$, and $V(p_3)$ are represented by line segments with different styles separated by points b_1-b_7. Since objects are restricted on road segments, the network Voronoi cell of a specific generator is unique. As a result, the network Voronoi diagram is unique. That is Property 1 is still valid for the network Voronoi diagram.

Given a set of points of interest, one can construct the network Voronoi diagram by expanding shortest path trees from each POI simultaneously until the shortest path trees meet. The meeting points, termed as *border points*, are also on the edges of the road network with the property that the costs (e.g., road network distances) from the meeting point to the two neighboring POIs are equal to each other.

Definition 4. Border points

$$B(p_i, p_j) = V(p_i) \cap V(p_j), i \neq j.$$

The border point set $B(p_i, p_j)$ contains all the points shared between $V(p_i)$ and $V(p_j)$. For each border point $b \in B(p_i, p_j)$, it has the property: $d_n(b, p_i) = d_n(b, p_j)$, that is, it has equal distance to p_i and p_j. In Fig. 2(b), $B(p_1, p_2) = B(p_2, p_1) = \{b_1, b_2\}$.

Definition 5. Voronoi Neighbors

$$Nbr(p_i) = \{p_j | B(p_i, p_j) \neq \emptyset, p_j \in P, i \neq j\}.$$

The Voronoi neighbors of p_i are those generators which share one or more border points with p_i. In Fig. 2(b), the Voronoi neighbors of p_1 are p_2 and p_3. Property 2 also holds for the network Voronoi diagram, i.e., the average number of Voronoi neighbors per POI does not exceed six [32].

In order to demonstrate the boundary of each network Voronoi cell more intuitively, one can properly connect all the border points of a generator p without crossing any road segments to form a polygon, termed network Voronoi polygon $NVP(p)$. For example, if two adjacent border points share the same generators (b_3 and b_4 are between p_1 and p_3), they can be directly connected with an arbitrary line without crossing any road segments. Three or more adjacent border points can be connected via adding auxiliary points (v is added to connect b_2, b_4 and b_5 in Fig. 2(b)). By adding arbitrary lines and auxiliary points, NVPs become non-unique. However, as objects are restricted on road segments, the NVPs contain exactly the same set of road segments as the NVCs and hence are unique from this perspective. Note that these hypothetical NVPs in the remainder of this paper are just for clear demonstration and

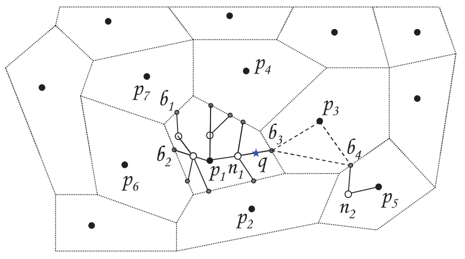


Fig. 3. k-nearest-neighbor query on a network Voronoi diagram.

explanation, but we do not need to compute them when constructing the NVD or verifying queries.

Given the network Voronoi diagram of a set of POIs P and a road network $\mathbb{G}$, the first nearest neighbor of a query point q has the following property.

Property 5. *Let $V(p)$ be the Voronoi cell of p on a road network. The nearest neighbor of a query point q is p, if and only if $q \in V(p)$.*

This property is the counterpart of the Property 3 in the Euclidean space. The proof follows directly from the definition of the network Voronoi diagram in Definition 1 and Definition 2. Furthermore, the following property holds for the k nearest neighbors of a query point q on a road network.

Property 6. *Given the network Voronoi diagram of P and a query point q on a road network, let $p_1, p_2, \dots, p_{k-1}$ be the $k-1$ nearest neighbors of q. Then, the k^{th} nearest neighbor of q is among the Voronoi neighbors of $p_1, p_2, \dots, p_{k-1}$.*

Proof. This property can be proven by contradiction. Suppose that the k^{th} NN, denoted as p_k, is not one of the adjacent neighbors of the first $k-1$ nearest neighbors of q. Then the shortest path from q to p_k must pass through some road segment(s) of a generator p_l which is not in $\{p_1, p_2, \dots, p_{k-1}\}$. Consider Fig. 3 as an example where p_1 is the first NN of q. Suppose p_5 is the second NN which is not an adjacent neighbor of p_1. We know that the shortest path from q to p_5 must pass through some Voronoi cell other than $V(p_1)$. In our example, assume the shortest path goes through $V(p_3)$. Suppose the shortest path from q to p_5 enters and exits $V(p_3)$ at b_3 and b_4, respectively. Then, the network distance from q to p_5 is $d_n(q, b_3) + d_n(b_3, b_4) + d_n(b_4, n_2) + d_n(n_2, p_5)$. According to the definition of network Voronoi diagram, the two network distances from the border point b_4 to generators p_3 and p_5 are equal, that is, $d_n(b_4, n_2) + d_n(n_2, p_5) = d_n(b_4, p_3)$. Therefore, $d_n(q, p_5) = d_n(q, b_3) + d_n(b_3, b_4) + d_n(b_4, p_3)$. We also know that the shortest path in both Euclidean and network distance functions obey triangular inequality, hence, $d_n(b_3, b_4) + d_n(b_4, p_3) \geq d_n(b_3, p_3)$. That is, the distance from q to p_5 is no smaller than the distance from q to p_3, which contradicts our assumption that p_5 is the second NN of q. As a consequence, Property 6 is true. Note that Property 6 is the generalization of Property 4 from the Euclidean space to road networks. □

4 Authentication Data Structure

To support the query verification, we need a well-defined *authentication data structure* (**ADS**) built on the outsourced data, which should be cryptographically signed by DO to ensure data integrity. Consider the outsourced database with a set of POIs P over an underlying road network $\mathbb{G}$. In this work, we propose an elaborate authentication data structure in order to support k-nearest-neighbor query verification on road networks where the distances from the query point to objects are measured by the shortest path on the graph with regard to the edge weight $\mathbb{W}$. Given a set of POIs and the graph $\mathbb{G}$, the network Voronoi diagram can be computed by applying parallel Dijkstra's algorithm [33] where POIs are treated as multiple sources. In this algorithm, we employ a Fibonacci heap to expand the shortest path tree from all POIs in the graph until shortest path trees meet. For each POI, a set of border points is discovered and all the road segments in between the border points form the network Voronoi cell for that POI (generator). Note that the network Voronoi cell for any generator on the graph is unique because it contains a unique set of road segments.

Before outsourcing the dataset to the Cloud, DO first transforms the original POI set P and the underlying road network $\mathbb{G}$ into a set of network Voronoi cell objects as follows:

$$o_{vc}(p) = \langle p, V(p), Nbr(p) \rangle,$$

where p is a generator in P, $V(p)$ corresponds to the network Voronoi cell of p and $Nbr(p)$ represents the set of Voronoi neighbors of p on the network Voronoi diagram. Each network Voronoi cell object o_{vc} corresponds to a POI p. Note that network Voronoi cells are mutually exclusive (they do not overlap, except at border points) and collectively exhaustive (every point on the road network belongs to at least one generator). That is, every point (except for border points) on the road network belongs to one and only one network Voronoi cell. Therefore, the union of $V(p)$ for all $p \in P$ corresponds to the original road network, and the collection of o_{vc} corresponds to the original dataset (P and $\mathbb{G}$) to be outsourced to the Cloud. Next, we introduce the authentication data structure which is used for kNN query verification.

After computing the network Voronoi cells, DO signs each POI p along with the Voronoi cell $V(p)$ and its neighbors $Nbr(p)$ to create an authenticated network Voronoi cell o_{anvc} as follows:

$$o_{anvc}(p) = \langle p, V(p), Nbr(p), \mathbb{S} \rangle$$
$$\mathbb{S} = sign\left(h\left(p|V(p)|Nbr(p)\right)\right),$$

where h is a one-way, collision-resistant hash function and '|' represents the concatenation of two binary strings. Taking the generator p_1 in Fig. 3 as an example, Fig. 4 shows the authenticated network Voronoi cell of p_1, which includes all road segments inside $V(p_1)$, the Voronoi neighbors of p_1, p_1 itself, and the signature.

Next, DO transmits the authenticated network Voronoi cells, denoted as $\mathbb{O}$, to SP which hosts the database service, builds spatial indexes for the data, and processes kNN queries on behalf of the DO. Different from existing approaches [30] which combine spatial indexes with authentication information, our approach separates these two parts. That is, the DO decides whether and how to integrate the authentication information to the dataset, while

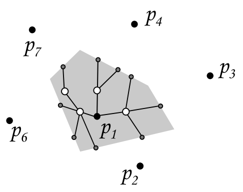


Fig. 4. Authenticated network Voronoi cell.

the SP server chooses which spatial index to construct and what algorithm to use for processing queries on the service provider side. Furthermore, since the authentication data structure is related to only one POI and its neighbors, updates will affect only a specific neighborhood around a POI rather than the whole database.

In the next section, we describe the proof generation at SP and the verification algorithm that can be used by clients to verify kNN queries.

5 Verifying NN on Road Networks

Nearest neighbor (NN) queries on road networks are common spatial query types for location-based services. Specifically, kNN queries enable mobile clients to retrieve the closest k POIs from the database with regard to the network distance to the location of the query point. A verifiable kNN query requests a verification object ($\mathcal{VO}$) from SP that contains not only the kNN query result, but also a proof to justify that the kNN result is correct and complete.

5.1 Proof Generation at SP

Without loss of generality, we can assume that SP builds a VoR-tree [34] or VQ-tree [35] based on network Voronoi cells and employs a query processing algorithm to retrieve the kNN result efficiently. Subsequently, the SP server constructs the verification object ($\mathcal{VO}$) for the kNN query result. Unlike the nearest neighbor verification in the Euclidean space where the path between two points is a straight line, on road networks, there are usually several paths between two points. Therefore, in order to verify the kNN query result, it is necessary to ensure that the path between two points is the shortest among all possible paths. On road networks, the shortest path from a query point to its first nearest neighbor has the following property.

Lemma 1. *Let $V(p)$ be the network Voronoi cell of p, the shortest path from any point $q \in V(p)$ to the generator p only goes through road segments within $V(p)$.*

Proof. We prove this property by using contradiction. Suppose the shortest path from q to its first NN p also passes through some road segments inside $V(p')$ where $p \neq p'$. Let b_1 be a border point where the shortest path exits $V(p)$. From the definition of network Voronoi diagram, we know that once the shortest path passes the border point b_1 and enters $V(p')$, it is closer to p' than to p, contradicting the fact that p is the first NN of q. Therefore, the shortest path from q to p cannot go through any road segment outside $V(p)$. □

Lemma 1 can be also generalized to the shortest path from a query point to its k^{th} nearest neighbors on a road network.

Lemma 2. *Given the network Voronoi diagram of P and a query point q on a road network, let $p_1, p_2, \ldots, p_{k-1}$ be the $k-1$ nearest neighbors of q and $V(p_1), V(p_2), \ldots, V(p_{k-1})$ be the corresponding network Voronoi cells. The shortest path from q to the k^{th} nearest neighbor p_k only goes through the union of $\{V(p_1), V(p_2), \ldots, V(p_{k-1}), V(p_k)\}$.*

Proof. Suppose the statement above is not true. That is, the shortest path from q to the k^{th} NN passes through the road segment of $V(p_j)$ where $p_j \notin \{p_1, p_2, \ldots, p_{k-1}, p_k\}$. Without loss of generality, we assume that the shortest path from q to p_k passes $V(p_1)$, $V(p_i)$, $V(p_j)$, and $V(p_k)$ in order where $p_i \in \{p_1, p_2, \ldots, p_{k-1}, p_k\}$. Then the part of the shortest path within $V(p_j)$ is closer to p_j than p_k. Therefore,q becomes closer to p_j than p_k, which contradicts the assumption that p_k is the k^{th} NN of q. Therefore, the shortest path to the k^{th} NN can only pass through the Voronoi cells of the k NNs of q, hence, Lemma 2 is proven. Note that the shortest path from q to the k^{th} NN of q must pass through $V(p_1)$ and $V(p_k)$, but does not necessarily pass through all the Voronoi cells of p_2-p_{k-1}. □

Assuming the kNN result is $\{p_1, p_2, \ldots, p_k\}$, according to Property 6, Lemma 1, and Lemma 2, a set of corresponding authenticated network Voronoi cells, $o_{anvc}(p_1), o_{anvc}(p_2), \ldots, o_{anvc}(p_k)$, is expected to be sufficient for the client to verify the kNN query result. The sound proof and the verification algorithm will be presented in the next section. Consequently, SP generates the verification object ($\mathcal{VO}$), which contains the kNN query result and the corresponding authenticated network Voronoi cell of every POI in the kNN result. Due to the fact that each o_{anvc} has a signature, these signatures may incur considerable communication cost, especially for queries with more results. Therefore, when there are multiple signatures, the SP server employs a signature aggregation technique [5], [23] to condense multiple signatures into one, thus to reduce the size of the $\mathcal{VO}$. Finally, the SP server transfers the $\mathcal{VO}$ back to the query client.

5.2 Network *k*NN Verification at the Client

Generally, a query verification process consists of two sequential steps, that is, signature verification and geometry verification. As the signature verification step is a standard and straightforward procedure, it is only briefly discussed here and omitted from the following sections. On receiving the result of a kNN query, the client first examines the signature attached to the $\mathcal{VO}$ to ensure that all objects in the result set originated from DO. On receiving an aggregate signature, the client verifies the signature by employing the corresponding aggregate signature verification algorithm (see details in [23]). Signature verification can ensure the proof data has not been falsified by SP or malicious attackers.

Next, the client verifies the geometry property of the kNN result. The verification starts with verifying the first NN of the result. Recall that in the $\mathcal{VO}$ returned by the SP server, each POI p in the kNN result set is accompanied by its network Voronoi cell $V(p)$ and its Voronoi neighbors $Nbr(p)$. According to Property 5, we know that the query point q must be on a road segment inside the Voronoi cell of

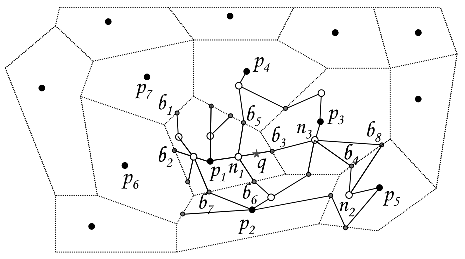


Fig. 5. Shortest path on road networks.

its first NN. Therefore, given the first NN p_1 and its Voronoi cell $V(p_1)$, a client verifies that the query point q falls on one of the road segments inside $V(p_1)$. If this is true, the generator p_1 is the first NN of q. Otherwise, p_1 is not the first NN of q and the result is not correct. The actual distance and the shortest path from q to the generator p_1 can be verified using Dijkstra's algorithm [36] or A* algorithm [37] on the subgraph inside $V(p_1)$. Note that Lemma 1 ensures the shortest path from q to p_1 is fully contained in $V(p_1)$. Therefore, the Voronoi cell $V(p_1)$ is sufficient for computing the shortest path from q to p_1 on the client. As a result, both the first NN p_1 and the shortest path from query point q to p_1 can be verified.

Next, we verify the second NN by utilizing the Voronoi neighbors $Nbr(p_1)$ of the first NN.

Lemma 3. *Given a query point q, the first NN p_1, the network Voronoi cell $V(p_1)$, and the Voronoi neighbors $Nbr(p_1)$ of p_1, the shortest path distance from q to the second NN of q can be verified.*

Proof. According to Property 6, the second NN must be one of the Voronoi neighbors of the first NN. Therefore, if we know the network distance from the query point to each of the Voronoi neighbors in $Nbr(p_1)$, then we know which generator is the second NN. Moreover, according to Lemma 2, if one specific Voronoi neighbor of p_1 is the second NN, then the shortest path from q to this POI must pass through $V(p_1)$. Therefore, we assume every Voronoi neighbor of p_1 is the possible second NN. Given the query point q, the first NN p_1, $V(p_1)$, and $Nbr(p_1)$, we can compute the shortest path distance from q to each Voronoi neighbor of p_1 based on the subgraph inside $V(p_1)$. Specifically, we first compute the distances from q to all the border points on $V(p_1)$ using Dijkstra's algorithm based on the subgraph inside $V(p_1)$. Next, because the two network distances from a border point to the two adjacent generators are equal to each other, the distance from each border point to the neighbor generators of p_1 can be replaced by the distance from the border point to p_1, which can be easily calculated based on the subgraph inside $V(p_1)$. As a result, the distance from q to each Voronoi neighbor of p_1 equals to the sum of the distance from q to a border point and from that border point to p_1. For example, in Fig. 5, the distance from q to p_3 is $d_n(q, p_3) = d_n(q, b_3) + d_n(b_3, p_1)$. In the case where there is more than one border point between two adjacent generators, we pick the one with the smallest distance (shortest path). For example, both b_6 and b_7 are border points between p_1 and p_2. The distance from q to p_2 is $d_n(q, p_2) = min\{d_n(q, b_6)+d_n(b_6, p_1), d_n(q, b_7)+d_n(b_7, p_1)\}$. Note that the distances computed on the subgraph inside $V(p_1)$ are not necessarily the shortest distances from q to all neighbor generators on the original graph $\mathbb{G}$. However, for the generator which is the second NN of q, the shortest path distance computed on the subgraph inside $V(p_1)$ is identical to the shortest distance on the original graph $\mathbb{G}$. (Lemma 2). □

Lemma 4. *Given a query point q, the $k-1$ nearest neighbors $p_1, p_2, \ldots, p_{k-1}$, the union of subgraphs inside $V(p_1), V(p_2), \ldots, V(p_{k-1})$, and the Voronoi neighbors of the $k-1$ NNs, the shortest path distance from q to the k^{th}NN can be verified.*

Proof. This is a generalization of Lemma 3. Given the network Voronoi cells of all $k-1$ NNs of p, by union all the road segments inside these Voronoi cells, we get a connected network g which is a subgraph of the whole road network $\mathbb{G}$. The shortest path from q to all the border points on the subgraph g can be computed using Dijkstra's algorithm. Next, as the two distances from the border point to its two neighbor generators are equal to each other, the shortest distance from q to all the Voronoi neighbors of $p_1, p_2, \ldots, p_{k-1}$ can be calculated based on g. Again, the shortest path distances computed based on the subgraph g are not necessarily equal to the shortest path distances on the original graph $\mathbb{G}$. However, for the k^{th} NN, p_k, Lemma 2 ensures that the shortest path distances between q and p_k on the subgraph g and the original graph $\mathbb{G}$ are equal. □

Reducing Verification Space by Euclidean Restriction: As the number of border points quickly grows with increasing k, the verification cost on computing the distances from q to border points and border points to generators increases. For example, in Fig. 5, to verify the second NN, we need to compute the network distance from q to all five Voronoi neighbors of p_1. To reduce computation cost, we employ the *Euclidean restriction* [16] to prune the verification space. We know that the Euclidean distance is the lower bound of the network distance between two points. Therefore, if the Euclidean distance from the query point q to p' is greater than the network distance from q to another point p, i.e., $d(q, p') > d_n(q, p)$, p is closer to q than p' on the network, and we do not need to compute the network distance between q and p'. Therefore, in order to verify the second NN (suppose it is p_2), we only need to consider Voronoi neighbors with a Euclidean distance no larger than the network distance from the candidate second NN to q. This is because neighbors with a larger Euclidean distance can never be closer to q than p_2 to q, and hence should be pruned without further investigation. In Fig. 5, suppose the network distance from q to p_2 is 6: $d_n(q, p_2) = 6$, and the Euclidean distances from q to other neighbors of p_1 are as follows: $d(q, p_3) = 4$, $d(q, p_4) = 5$, $d(q, p_6) = 9$, and $d(q, p_7) = 10$. By applying the Euclidean restriction rule, we only need to compute the network distance from q to p_3 and p_4, and discard p_6 and p_7 as they are further away from q than p_2.

The network k-nearest-neighbor query verification process is shown in Algorithm 1. The inputs to the algorithm are the query point q, the verification object $\mathcal{VO}$, and the parameter k. The $\mathcal{VO}$ contains the authenticated network Voronoi object of every POI in the kNN result, including the generator POI, the Voronoi cell of the generator and

Algorithm 1 VerifyNetwork$k$NN(q,$\mathcal{VO}$,k)

```
1. $H \leftarrow \emptyset$; $Visited \leftarrow \emptyset$; $g \leftarrow \emptyset$;
2. $\langle p, V(p), Nbrs \rangle \leftarrow \mathcal{VO}.getNN(1)$;
3. if ($q \notin V(p)$) then
4.    return false;{the 1$^{st}$ NN fails by Property 5}
5. end if
6. $Visited.add(p)$;
7. $g \leftarrow V(p)$; $H \leftarrow Nbrs$; $Visited \leftarrow Nbrs$;
8. for $i = 2$ to $k$ do
9.    $\langle p, V(p), Nbrs \rangle \leftarrow \mathcal{VO}.getNN(i)$;
10.   if $p \notin H$ then
11.      return false;{Property 6}
12.   end if
13.   $g \leftarrow g \cup V(p)$;
14.   $lb \leftarrow computeSP(g, q, p)$; {Lemma 2}
15.   $minDist \leftarrow MaxValue$; $minPt \leftarrow null$;
16.   for all ($h \in H$) do
17.      if ($h.ed > lb$) then
18.         break; {apply Euclidean restriction}
19.      else
20.         $h.nd = computeSP(g, q, h.poi)$;
21.         if ($h.nd < minDist$) then
22.            $minDist \leftarrow h.nd$;
23.            $minPt \leftarrow h$;
24.         end if
25.         if ($minDist < lb$) then
26.            return false; {$minPt$ is closer to $q$ than $p$}
27.         end if
28.      end if
29.   end for
30.   if ($p == minPt.poi$ && $p.nd == minDist$) then
31.      $H.remove(minPt)$;{the $i^{th}$NN is verified}
32.      for all ($nbr \in Nbrs$) do
33.         if ($nbr \notin Visited$) then
34.            $H \leftarrow H \cup nbr$; $Visited \leftarrow Visited \cup nbr$;
35.         end if
36.      end for
37.   else
38.      return false;{the $i^{th}$NN is not verified}
39.   end if
40. end for
41. return true;
```

its neighbors. The method $\mathcal{VO}$.*getNN(i)* returns the network Voronoi cell object o_{vc} of the i^{th} NN on line 2 and line 9. H is a min-heap containing all the Voronoi neighbors of already verified NNs and objects in H are sorted by their Euclidean distances to the query point q. Each object h in H contains a location (latitude and longitude), denoted as *h.poi*, the Euclidean distance to q, denoted as *h.ed*, and the network distance to q, represented as *h.nd*. The *h.ed* is calculated while the object is inserted into the min-heap on lines 7 and 34, and the *h.nd* is computed while the object is likely to be the next nearest neighbor on line 20. *Visited* is a set which maintains all the POIs that the client has seen so far to avoid examining the same object twice.

The algorithm starts with the first NN p_1 by checking whether the query point q belongs to the road segments inside the Voronoi cell $V(p_1)$ of p_1 (lines 3-5). If not, p_1 is not the first NN and the verification process fails. Otherwise, p_1 is verified as the first NN and is added to the *Visited* set on line 6. Next, the road segments inside the $V(p_1)$ are merged with g which is a subgraph of the road networks inside the Voronoi cells of already retrieved NNs, and all the neighbors of p_1 are inserted into H and *Visited* set. The subsequent *for* loop (lines 8-40) iterates through each object in the kNN result set obtained from $\mathcal{VO}$ (line 9). If the current NN p is not in H (the current NN returned is not one of the Voronoi neighbors of previously verified NNs), then p is not the i^{th} NN of q (according to Property 6), and the verification fails on line 11. Otherwise, p is one of the neighbors of already verified NNs, and we need to verify that the network distance of p is the smallest among all the Voronoi neighbors and identical to the distance in the query result returned by SP (lines 13-39).

To apply the Euclidean restriction rule to prune the verification space, we first compute the network distance and the corresponding shortest path from q to p on the subgraph g. Then, the distance is used as a *lower bound* (*lb*) in the next steps. For each object h in H, if the Euclidean distance *h.ed* is greater than the lower bound, then h is skipped. Otherwise, the network distance from h to q is computed and stored in *h.nd* (line 20). If p has the smallest network distance among all entries in H, p is verified as the i^{th} NN. Next, the corresponding entry is removed from H (line 31) and all the unvisited neighbors of p are added to H and *Visited* set (lines 32-36). Otherwise, p is not the i^{th} NN and the verification fails on line 38.

Theorem 1. *Given a query point q, the k-nearest-neighbor query result $p_1, p_2, \ldots, p_k$, and their corresponding authenticated network Voronoi cells $o_{anvc}(p_1), o_{anvc}(p_2), \ldots, o_{anvc}(p_k)$, the query client can use Algorithm 1 to verify the correctness and completeness of the query result with regard to both distance and path.*

Proof. Before proving this theorem, note that through the signature verification, we can ensure the data integrity of each given network Voronoi cell object $o_{vc}(p)$, which includes the POI p, the corresponding network Voronoi cell $V(p)$, and its Voronoi neighbors $Nbr(p)$. Next, we prove this theorem by induction as follows.

For the 1st NN p_1, Property 5 and Lemma 1 ensure the completeness and correctness of p_1, respectively.

For the 2nd NN p_2, we can prove its completeness and correctness as follows: (i) ***Completeness***. According to Property 6, the second NN must be among the Voronoi neighbors of p_1. Therefore, the valid second NN will not be missed. That is, the completeness of the 2nd NN can be ensured. (ii) ***Correctness***. According to Lemma 3, the correctness of the distance from q to p_2 can be verified (**Distance Verification**). Moreover, the correctness of the shortest path to p_2 can be verified according to Lemma 2 (**Path Verification**).

For the $k-1^{th}$ NN, we assume that its correctness and completeness can be verified by Algorithm 1. Next, we prove that the k^{th} NN can also be verified.

For the k^{th} NN p_k, we can prove: (i) ***Completeness***. Property 6 can ensure the completeness. (ii) ***Correctness***. Lemma 4 ensures the distance is correct (**Distance Verification**), and Lemma 2 ensures the shortest path from q to p_k is correct, too (**Path Verification**). □

5.3 Pre-computation Based Verification Scheme

During the verification in Algorithm 1, the client needs to compute the shortest distance and path from the query point q to candidate neighbor generators in lines 14 and 20 by using Dijkstra [36] or A* [37] algorithm.

Unfortunately, these distance computations are usually expensive, especially as the subgraph network g grows large. From Fig. 5, we can observe that each path from q to any POI, except the first nearest neighbor, always passes through one or more border points. Thus, if the client knows distances from each border point to other border points and to the generator, then the distance computation cost can be greatly reduced. Therefore, to improve the efficiency of the client verification algorithm, we adopt the network distance pre-computation approach from [17]. Specifically, for each network Voronoi cell, DO pre-computes the shortest distance and path between two types of point pairs: (i) border point-to-generator pairs from each border point to the generator, and (ii) border-to-border point pairs between all border points on a Voronoi cell. Taking $V(p_1)$ in Fig. 5 as an example, DO pre-computes the shortest distances and paths from the nine border points to p_1 and all pairs between the nine border points. After that, DO stores pre-computed distances of all point pairs in a *distance table* $DT(p)$. In order to reduce the storage cost, DO only stores the hash value of the shortest path between each pair of points in a separate *hashed path table* denoted $\Phi(p)$.

Next, in order to support the client's utilization of these pre-computed distances for accelerating the verification, we create a new ADS called authenticated distance table o_{adt} for each POI. Instead of incorporating the Voronoi cell $V(p)$ into $o_{adt}(p)$, the distance table $DT(p)$ is included. Furthermore, to ensure the integrity of paths, the hashed path table $\Phi(p)$ should be included into $o_{adt}(p)$. However, in order to keep the $\mathcal{VO}$ generated later by SP as compact as possible, an accumulative hash value $\varphi(p)$ is incorporated into $o_{adt}(p)$ instead of $\Phi(p)$. The value of $\varphi(p)$ is computed as the modular multiplication of all hashed paths in the pre-computed table $\Phi(p)$:

$$\varphi(p) = \left[\prod_{hp_i \in \Phi(p)} hp_i\right] mod\ \eta,$$

where hp_i is one hashed path value in $\Phi(p)$ and η is a 32-byte modulus that keeps $\varphi(p)$ the same size as one hash digest. In addition, η needs to be a large prime number to avoid collisions [38]. Thus, the new authentication data structure is defined as follows:

$$\begin{aligned} o_{adt}(p) &= \langle p, DT(p), \varphi(p), Nbr(p), \mathbb{S}\rangle \\ \mathbb{S} &= sign\left(h\left(p|DT(p)|\varphi(p)|Nbr(p)\right)\right). \end{aligned}$$

After computing the $o_{adt}(p)$ for each POI p, DO outsources each $o_{adt}(p)$ and $\Phi(p)$ to SP. Eventually, the outsourced database managed by SP contains every POI p along with its old ADS $o_{anvc}(p)$, its new ADS $o_{adt}(p)$, and $\Phi(p)$.

In the query answering phase, according to the query point q, SP first gets the query result, which contains k nearest neighbors $\{p_i | 1 \leqslant i \leqslant k\}$ and the shortest distance and path from q to each neighbor POI p_i. Next, SP generates the $\mathcal{VO}$ according to the result. Instead of returning the road segments inside Voronoi cells of the kNN result, distance tables (i.e., $DT(p_i)$) are included in the $\mathcal{VO}$ and transferred to the query client to boost the efficiency of the verification process. Note that the SP server still needs to return the road segments inside the Voronoi cell of the first NN, namely $V(p_1)$, in order to perform the verification on line 3 in Algorithm 1. Furthermore, to help the client verify the shortest paths from q to the kNN result, SP first incorporates the border points into the shortest paths in the query result. Next, instead of returning the entire hashed path table $\Phi(p_i)$, SP computes another accumulative hash value $\varphi'(p_i)$ as the modular multiplication of hashed paths between those irrelevant point pairs (i.e., the paths in $\Phi(p_i)$ do not appear in the shortest paths from q to k nearest neighbors). For example, in Fig. 5, we assume both p_3 and p_5 are in the result set and the shortest path from q to p_5 passes through the border points b_3 and b_4, then $\varphi'(p_3)$ is the modular multiplication of hashed paths in $\Phi(p_3)$ excluding hashed paths of b_3-b_4 and b_3-p_3 pairs. Consequently, the $\mathcal{VO}$ contains the query result (i.e., nearest neighbors and shortest paths), each NN's $DT(p_i)$, $\varphi'(p_i)$ and $Nbr(p_i)$, and the first NN's Voronoi cell $V(p_1)$. Subsequently, SP needs to compute a condensed signature for the $\mathcal{VO}$ by combining signatures in $o_{anvc}(p_1)$, $o_{adt}(p_1), \ldots, o_{adt}(p_k)$. After that, SP returns the $\mathcal{VO}$ to the client.

When authenticating query results, the client first use $V(p_1)$ to verify the first NN p_1 as Algorithm 1, and then verify the distances and paths of other nearest neighbors. According to Lemma 2, the possible shortest path from the query point to one candidate POI can be divided into following three parts: (i) from q to the border point, (ii) from one border point to another border point within one Voronoi cell, and (iii) from one border point to the generator POI. The first part can be computed on $V(p_1)$ by Dijkstra's algorithm. The other two parts can be obtained by a set of distance lookups from the distance table $DT(p_i)$. As a consequence, the shortest distance verification on lines 14 and 20 in Algorithm 1 can be accelerated. Moreover, the client can reconstruct $\varphi(p_i)$ by combining $\varphi'(p_i)$ with hashed paths between relevant point pairs that appear in the result. Continuing the above example in Fig. 5, we assume both b_3-b_4 and b_3-p_3 point pairs appear in the shortest paths from q to p_3 and p_5. The client can first use the detailed sub-paths b_3-b_4 and b_3-p_3 in the query result to compute hashed paths $hp(b_3\text{-}b_4)$ and $hp(b_3\text{-}p_3)$ respectively, and then reconstruct $\varphi(p_3)$ by computing the modular multiplication of $hp(b_3\text{-}b_4)$, $hp(b_3\text{-}p_3)$, and $\varphi'(p_3)$ that is given in the $\mathcal{VO}$. The client can also reconstruct other $\varphi(p_i)$, e.g., $\varphi(p_5)$, with the same procedure. Then through the signature verification, we can ensure that the shortest paths from q to the kNN results have not been falsified. Thus, the correctness of shortest paths can also be verified. Obviously, this pre-computation based verification scheme follows the similar process of Algorithm 1, therefore, Theorem 1 still holds. Due to the space limit, the proof is omitted.

6 Database Update

In this section, we discuss how DO updates its outsourced database for either POI updates or road network updates. For different types of updates, DO follows a similar update procedure as follows. First of all, DO needs to update the network Voronoi diagram. Secondly, it should update those affected authentication data structures and renew their

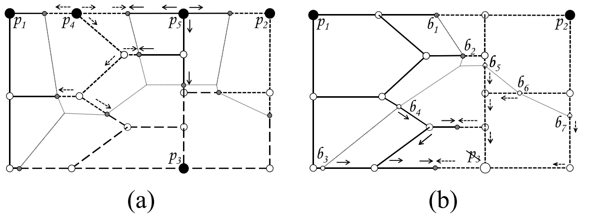


Fig. 6. Database update - POI update examples: (a) POI insertion. (b) POI deletion.

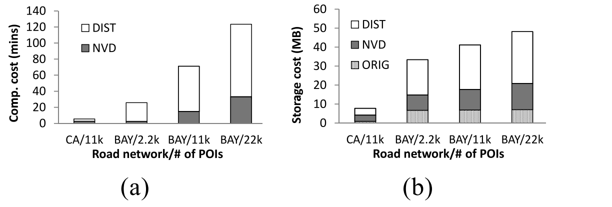


Fig. 7. Computation and storage cost on DO: (a) Computation cost. (b) Storage cost.

signatures. Thirdly, DO transmits all these updates to SP. Eventually, upon receiving the update request, SP updates its spatial index and database correspondingly.

Fig. 6 shows how to insert and delete POIs respectively on the original database shown in Fig. 2. To insert a new POI into the database, we need to construct a new network Voronoi cell for this POI at first. This can be computed by using Dijkstra's algorithm to build a shortest path tree with this new POI as root. As shown in Fig. 6(a), we expand nodes from new added POIs (e.g., p_4 and p_5) simultaneously, until no more nodes are closer to the new POIs than to other POIs. Then, we identify new border points between the new POIs and their surrounding Voronoi neighbors and delete those obsolete border points. After that, we update all neighbors around the new POIs and their affiliated authentication data structures, including network Voronoi cells, their neighborhood relationships, pre-computed distance tables and signatures. Deleting a POI follows a similar process. As shown in Fig. 6(b), the difference is that we expand inwardly from all border points ($b_3, \ldots, b_7$) of the deleted POI (p_3) in parallel, until we help each node in the deleted network Voronoi cell find its new nearest POI. Then, we update all the affected neighbors around the deleted POI.

The other category of updates is road network changes, including node/edge insertions and deletions, and updates to edge weights. As node insertion/deletion can be treated as a special case of edge insertion/deletion, we consider edge updates only. When an edge insertion, deletion, or weight update happens, the affected network Voronoi cells should be identified first. Next, those affected network Voronoi cells are rebuilt and re-signed, including re-generating all the pre-computed shortest paths and the hash on these paths. Note that either one POI update or the change of one edge only impacts a few nearby Voronoi cells. Hence, each update is processed locally without affecting other parts of the road network. According to Property 2, the average number of Voronoi neighbors per POI does not exceed six. Therefore, local updates are expected to be efficient. Please refer to Section 7 for experimental results on the update performance.

Furthermore, instead of performing updates one by one, we can perform the same type of updates in batch to accelerate the update process. Such a batch update has two advantages. First, we can utilize the parallel Dijkstra's algorithm [33] to add/remove/update network Voronoi cells for multiple POIs simultaneously. Second, some network Voronoi cells might be affected by multiple updates. If these updates were performed one by one, then these cells would be updated many times. However, in a batch update manner, they would be updated only once. Thus the entire update process can be accelerated by reducing the redundant updates. For instance, as shown in Fig. 6(a), the network Voronoi cell with respect to p_3 needs to be updated twice if we insert new POIs p_4 and p_5 one by one. However, if we insert p_4 and p_5 in batch, it would eventually be updated only once.

Finally, a malicious SP may ignore all updates generated from DO, and the client will remain oblivious to this fact. To ensure database freshness, we take the measures discussed in [8], [26] that enforce signatures to either expire periodically or be selectively revoked by DO.

7 Experimental Study

In this section, we evaluate the performance of the network Voronoi diagram-based k-nearest-neighbor query verification approach through experiments. Our database outsourcing framework contains two phases: the offline database transformation phase on DO, and the online phase on SP and clients. In the offline phase, DO computes the network Voronoi diagram on the POI set and the underlying road network, and it then generates a signature for each POI by incorporating the authentication information into each POI object. In the online phase, SP evaluates queries and sends results to query clients on behalf of the DO, and then the client verifies the query result. Our implementation for the online query evaluation and verification is in Java. The server-side program (SP) runs on a Linux Server with an Intel Core2 Duo 2.13GHz CPU and 4GB memory. Meanwhile, the client-side query verification program runs on a HTC EVO 3D mobile device with the Google Android OS which communicates with the SP server via Internet connections through WiFi. The cryptographic operations are implemented using the SHA-256 digest algorithm (with 32-byte hash digest) and RSA-1024 signature algorithm (with 128-byte keys) from [39]. Our experiments were performed on two real-world road networks obtained from [40], [41]: (i) **CA** which consists of the major freeways and highways in the state of California with a total of 21,048 nodes and 22,830 edges, and (ii) **BAY** which contains highways as well as surface streets with 174,956 nodes and 223,001 edges in the San Francisco Bay Area, California. Although the second network (BAY) covers a smaller geospatial region, the size of the network is larger than the California road network due to the inclusion of surface streets.

In addition, we use both real and synthetic POI datasets on these two road networks. For the CA road network, we use a real POI dataset originating from U.S. Census [41], [42] representing airports, hospitals, schools, etc. from the state of California. Additionally, to study the effect of the

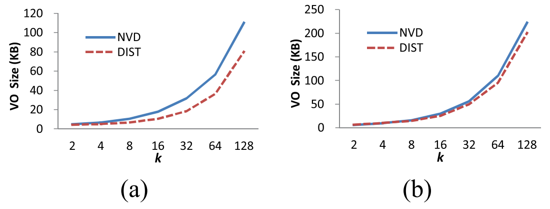


Fig. 8. $\mathcal{VO}$ size over k: (a) CA/11k. (b) BAY/11k.

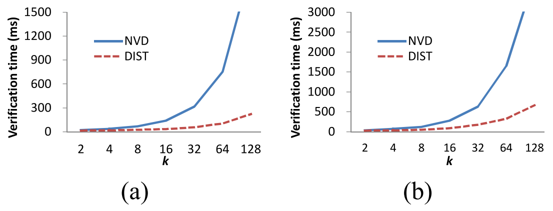


Fig. 10. Verification time over k on mobile clients: (a) CA/11k. (b) BAY/11k.

different density of POI datasets in our system, we experimented with a few synthetic POI datasets generated on the BAY road network. Specifically, the POIs are generated by repeatedly picking a road segment at random, and then choosing a random point on that road segment as a POI. The POI density ρ corresponds to the ratio of the number of POI points to the number of road segments in the network. As a result, the areas with a dense road network are also likely to contain dense POIs, which is consistent with our observation of the real-world, that is, more POIs are located in areas with more roads (e.g., urban areas) than areas with less roads (suburbs, national parks, etc.). Results of both the basic version (denoted as NVD) of our verification algorithm and the improved version with pre-computed distances as described in Section 5.3 (denoted as DIST) are reported in this section.

We first study the offline database transformation costs on DO, including the total computation cost and the extra storage incurred by the authentication information. Fig. 7(a) shows the computation cost for generating authenticated network Voronoi cells (NVD) as described in Section 4, and the additional cost for pre-computing the distance and hashed path table within each cell (DIST). Fig. 7(b) describes the size of the original datasets (ORIG), the extra storage overhead for storing the network Voronoi cells, neighbors, signatures (NVD) and the additional cost for storing the distance and hashed path table (DIST). As we can see in Fig. 7(a), the computation cost increases significantly with the size of the road network and the number of POIs, especially the time spent for pre-computation for the BAY network. This is because when there are more POIs on the network, more network Voronoi cells are generated, resulting in a higher time cost for computing the distances and paths. Similarly, extra storage overhead is required to store Voronoi cells, neighbors, and signatures in the NVD scheme. Additionally, more space is needed for storing the distance and hashed path tables in the DIST scheme as shown in Fig. 7(b).

In the second set of experiments, we investigate the size of the $\mathcal{VO}$ over the query parameter k. Fig. 8(a) presents the $\mathcal{VO}$ size over k on the CA road network with 11k POIs, and Fig. 8(b) shows the $\mathcal{VO}$ size on the BAY road network also with 11k POIs. The results show that on both road networks, the $\mathcal{VO}$ sizes in the DIST scheme are smaller than in the NVD scheme. This is because in the DIST scheme, the distance table is returned instead of the original road network inside the Voronoi cells of the kNN results (except for the 1^{st} NN), resulting in a smaller $\mathcal{VO}$ size especially for queries with k larger than 8.

Our next set of experiments studies the communication time for evaluating network k-nearest-neighbor queries on mobile devices. The communication time is measured by the round trip time cost per query for a client. Specifically, it starts with the client sending a query request and ends with that client receiving the result from SP, excluding the query processing time cost on SP. Fig. 9 shows the communication cost over k on both the CA road network and the BAY road network, each with 11k POIs.

The next set of experiments investigates the computation time cost for verifying k-nearest-neighbor results on mobile devices. After receiving the $\mathcal{VO}$, the mobile client starts the verification algorithm (Algorithm 1) to verify the correctness and completeness of the kNN results. Fig. 10 shows the verification time cost over k on the CA and BAY networks. Note that the verification time also includes the signature verification time, which comprises computing a hash digest based on the raw data and verifying the signature. Taking the BAY/11K dataset as an example, the average size of a network Voronoi cell is about 1.75KB, and hashing and verifying 1.75KB data takes about 1.3ms on a mobile device. Therefore, the signature verification cost is relatively inexpensive compared with other operations. The experiments demonstrate that on both networks, the DIST scheme is significantly more efficient in verifying the network kNN result than the NVD scheme, especially when the value of k is larger than 16. This is due to the fact that when k increases, the subgraph inside the union of the Voronoi cells of the kNN results grows. Therefore, using the Dijkstra's algorithm in verifying the shortest paths to the kNN results becomes less efficient, decelerating the whole verification process.

Our next set of experiments studies the $\mathcal{VO}$ size and verification time cost on the BAY road network by varying the

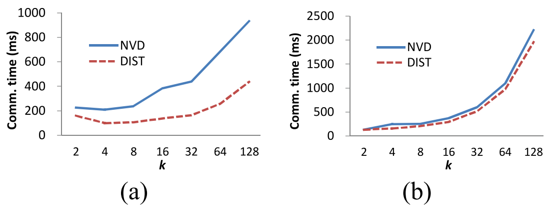


Fig. 9. Communication time over k on mobile clients: (a) CA/11k. (b) BAY/11k.

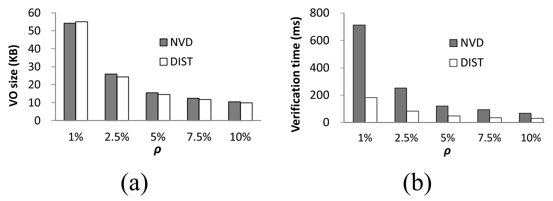


Fig. 11. Verification cost over density of POI: (a) $\mathcal{VO}$ size. (b) Verification time.

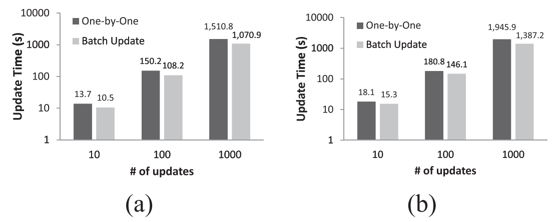


Fig. 12. Database update cost: (a) POI updates. (b) Edge updates.

value of ρ (POI density) from 1% (2.2k POIs) to 10% (22k POIs). Fig. 11(a) shows the size of $\mathcal{VO}$ over ρ and Fig. 11(b) presents the verification cost over ρ. All the experiments are performed by setting the query parameter k to 8. The results show that as the value of ρ increases, the size of $\mathcal{VO}$ declines. This is because when there are fewer POIs on the network, the network Voronoi cell of each POI covers a larger region with more road segments. Therefore, the size of each network Voronoi cell is bigger, resulting in larger $\mathcal{VO}$ size. For the same reason, the verification cost on a sparse POI dataset is higher than on a dense POI dataset, as shown in Fig. 11(b).

The next set of experiments studies the cost of database updates. Fig. 12 shows the update cost on DO to process a number of POI updates and edge updates on the BAY/11K dataset, respectively. The x-axis corresponds to the number of updates, and y-axis shows the update time. The update time includes costs on updating the network Voronoi diagram, updating all affected authenticated network Voronoi cells, the related pre-computation, and refreshing signatures. The results show that the update time increases linearly with the number of updates. On average, each update can be processed in 1.5-2 seconds (30-40 updates per minute) in the one-by-one mode. Furthermore, since our data structure supports parallel updates, the update cost in batch mode achieves 15%-30% performance improvement over the one-by-one update mode.

Our last set of experiments studies how database size affects the size of $\mathcal{VO}$. Two larger real-world road networks from [43] are used: (i) **COL**, the road network of Colorado with a total of 435, 666 nodes and 528, 533 edges, and (ii) **FLA**, the road network of Florida, with 1, 070, 376 nodes and 1, 356, 399 edges. A synthetic POI dataset with 5% density ($\rho = 5\%$) is generated on each network. Fig. 13 shows that the $\mathcal{VO}$ size remains roughly the same on the two road networks of different size. This is because the size of $\mathcal{VO}$ is directly related to the average size of Network Voronoi Cells (NVC) constructed based on the road network, not the total size of the road network. The average NVC size is around 1.75 KB on both networks. Hence the $\mathcal{VO}$ sizes are also similar over k.

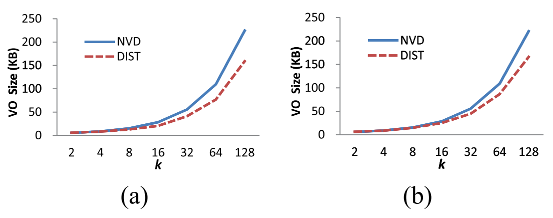


Fig. 13. Effects of database size on $\mathcal{VO}$ size: (a) COL. (b) FLA.

8 Conclusion

In this paper, we studied the query verification problem for k-nearest-neighbor queries on road networks. While existing approaches proposed in this domain cannot verify both the distance and the shortest path to the kNN results simultaneously, we present a network Voronoi diagram-based verification approach that utilizes the network Voronoi cell of each result object to verify the correctness and completeness of the kNN result with regard to both distance and path. In addition to the basic verification algorithm (NVD), an improved version (DIST) with pre-computed distances within each network Voronoi cell is presented for kNN query verification to further reduce the verification cost on mobile clients. Our experiments on real-world road networks show that both verification schemes generate compact verification objects and allow for efficient verification processes on modern mobile devices. The future work of this paper can explore a few different directions. First of all, how to handle more types of network spatial queries using the general framework and data structure discussed in this paper. Secondly, in this paper, we only considered one data owner party. However, in practice, there might be multiple data owners. For example, the POIs and the road networks can come from two different data owners. Hence, how to handle the query verification problem in the presence of multiple data owners is also an interesting direction to explore. Last but not least, how to handle highly frequent network updates, such as traffic changes during rush hours, is also a very challenging problem.

Acknowledgments

This work was supported in part by the US NSF under Grant IIS-1115153, in part by the USC Integrated Media Systems Center (IMSC), in part by HP Labs, and in part by unrestricted cash gifts from Google, Northrop Grumman, Microsoft and Oracle.

References

[1] F. Gens. (2011, Oct. 20). *IDC's new IT cloud services forecast: 2009-2013* [Online]. Available: http://blogs.idc.com/ie/?p=543
[2] H. Hacigümüs, S. Mehrotra, and B. R. Iyer, "Providing database as a service," in *Proc. 18th ICDE*, San Jose, CA, USA, 2002, pp. 29–38.
[3] *Google Maps* [Online]. Available: http://maps.google.com/
[4] C. Cachin and M. Schunter, "A cloud you can trust," *IEEE Spectrum*, vol. 48, no. 12, pp. 28–51, Dec. 2011.
[5] A. Fiat, "Batch RSA," *J. Cryptology*, vol. 10, no. 2, pp. 75–88, 1997.
[6] W. Cheng and K.-L. Tan, "Query assurance verification for outsourced multi-dimensional databases," *J. Comput. Secur.*, vol. 17, no. 1, pp. 101–126, 2009.
[7] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Spatial outsourcing for location-based services," in *Proc. IEEE 24th ICDE*, Cancun, Mexico, 2008, pp. 1082–1091.
[8] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *VLDB J.*, vol. 18, no. 3, pp. 631–648, Jun. 2009.
[9] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: An efficient verification method for outsourced databases," *VLDB J.*, vol. 18, no. 1, pp. 363–381, 2009.
[10] W.-S. Ku, L. Hu, C. Shahabi, and H. Wang, "Query integrity assurance of location-based services accessing outsourced spatial databases," in *Proc. 11th Int. Symp. SSTD*, Aalborg, Denmark, 2009, pp. 80–97.

[11] W.-S. Ku, L. Hu, C. Shahabi, and H. Wang, "A query integrity assurance scheme for accessing outsourced spatial databases," *GeoInformatica*, vol. 17, no. 1, pp. 97–124, 2013.

[12] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi, "Verifying spatial queries using voronoi neighbors," in *Proc. 18th GIS*, San Jose, CA, USA, 2010, pp. 350–359.

[13] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi, "Spatial query integrity with voronoi neighbors," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 863–876, Apr. 2013.

[14] M. L. Yiu, E. Lo, and D. Yung, "Authentication of moving kNN Queries," in *Proc. IEEE 27th ICDE*, Hannover, Germany, 2011, pp. 565–576.

[15] L. Hu, Y. Jing, W.-S. Ku, and C. Shahabi, "Enforcing k nearest neighbor query integrity on road networks," in *Proc. 20th SIGSPATIAL/GIS*, Redondo Beach, CA, USA, 2012, pp. 422–425 (short paper).

[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *Proc. 29th VLDB*, Berlin, Germany, 2003, pp. 802–813.

[17] M. R. Kolahdouzan and C. Shahabi, "Voronoi-based K nearest neighbor search for spatial network databases," in *Proc. 13th Int. Conf. VLDB*, Toronto, ON, Canada, 2004, pp. 840–851.

[18] H. Hu, D. L. Lee, and V. C. S. Lee, "Distance indexing on road networks," in *Proc. 32nd Int. Conf. VLDB*, 2006, pp. 894–905.

[19] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proc. SIGMOD*, New York, NY, USA, 2008, pp. 43–54.

[20] 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.

[21] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. SIGMOD*, Madison, WI, USA, 2002, pp. 216–227.

[22] H. Pang and K.-L. Tan, "Authenticating query results in edge computing," in *Proc. ICDE*, Washington, DC, USA, 2004, pp. 560–571.

[23] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *TOS*, vol. 2, no. 2, pp. 107–138, May 2006.

[24] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan, "Verifying completeness of relational query results in data publishing," in *Proc. SIGMOD Conf.*, Baltimore, MD, USA, 2005, pp. 407–418.

[25] R. Sion, "Query execution assurance for outsourced databases," in *Proc. 31st Int. Conf. VLDB*, Trondheim, Norway, 2005, pp. 601–612.

[26] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proc. SIGMOD*, , Chicago, IL, USA, 2006, pp. 121–132.

[27] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *Proc. 33rd Int. Conf. VLDB*, Vienna, Austria, 2007, pp. 782–793.

[28] S. Papadopoulos, L. Wang, Y. Yang, D. Papadias, and P. Karras, "Authenticated Multistep Nearest Neighbor Search," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 641–654, May 2011.

[29] M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen, "Authenticated data structures for graph and geometric searching," in *Proc. CT-RSA*, San Francisco, CA, USA, 2003, pp. 295–313.

[30] M. L. Yiu, Y. Lin, and K. Mouratidis, "Efficient verification of shortest path search via authenticated hints," in *Proc. ICDE*, Long Beach, CA, USA, 2010, pp. 237–248.

[31] H. Hu, J. Xu, Q. Chen, and Z. Yang, "Authenticating location-based services without compromising location privacy," in *Proc. SIGMOD*, Scottsdale, AZ, USA, 2012, pp. 301–312.

[32] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* New York, NY, USA: Wiley, 2000.

[33] M. Erwig, "The graph voronoi diagram with applications," *Netw.*, vol. 36, no. 3, pp. 156–163, Oct. 2000.

[34] M. Sharifzadeh and C. Shahabi, "VoR-Tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries," *PVLDB*, vol. 3, no. 1, pp. 1231–1242, Sept. 2010.

[35] U. Demiryurek and C. Shahabi, "Indexing network voronoi diagrams," in *Proc. 17th Int. Conf. DASFAA*, Busan, South Korea, 2012, pp. 526–543.

[36] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Num. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[37] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* Cambridge, MA, USA: MIT Press, 2009.

[39] Oracle Corp. *Java SE Security*. Retrieved on June 12, 2012 [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html

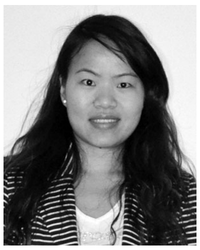
[40] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *Proc. SSTD*, Angra dos Reis, Brazil, 2005, pp. 273–290.

[41] F. Li. (2012 Apr. 2). *Real Datasets for Spatial Databases* [Online]. Available: http://www.cs.utah.edu/~lifeifei/SpatialDataset.htm

[42] *U.S. Geological Survey* [Online]. Available: http://www.usgs.gov/

[43] DIMACS. (2013 Jun. 13). *9th DIMACS Implementation Challenge - Shortest Paths* [Online]. Available: http://www.dis.uniroma1.it/challenge9/download.shtml

Yinan Jing received the Ph.D. degree in computer science from Fudan University, Shanghai, China in 2007. He is an Assistant Professor with the School of Computer Science at Fudan University. He was also a Visiting Researcher with the Department of Computer Science at the University of Southern California. His current research interests include spatial and temporal data management, geographic information systems, mobile computing, and security and privacy. He is a member of ACM and IEEE.



Ling Hu is a Software Engineer at Google Inc, Mountain View, CA, USA. She received the M.S. degree in computer science from Northeastern University, Boston, and the Ph.D. degree in computer science from the University of Southern California in 2012. Her current research interests include spatial and temporal data management, geographical information systems, query integrity and query optimization, and data warehousing.

Wei-Shinn Ku received the M.S. degrees in computer science and also in electrical engineering from University of Southern California (USC) in 2003 and 2006, respectively. He also received the Ph.D. degree in computer science from the USC in 2007. He is an Associate Professor with the Department of Computer Science and Software Engineering at Auburn University, Auburn, AL, USA. His current research interests include data management systems, data analytics, geographic information systems, and mobile computing. He is a senior member of IEEE and a member of ACM.

Cyrus Shahabi is a Professor of Computer Science and Electrical Engineering and the Director of the NSF's Integrated Media Systems Center (IMSC) at the University of Southern California, Los Angeles, CA, USA. He was also the CTO and Co-Founder of a USC spin-off, Geosemble Technologies, which was acquired in June 2012. He received the B.S. degree in computer engineering from Sharif University of Technology in 1989 and then the M.S. and the Ph.D. degrees in computer science from the University of Southern California in May 1993 and August 1996, respectively. He authored two books and more than two hundred research papers in the areas of databases, GIS and multimedia. He is currently on the Editorial Boards of the *VLDB Journal* and *IEEE Transactions on Knowledge and Data Engineering*. He is a recipient of the ACM Distinguished Scientist Award and the U.S. Presidential Early Career Awards for Scientists and Engineers (PECASE).