Spatial Query Integrity with Voronoi Neighbors

Ling Hu, *Student Member, IEEE,* Wei-Shinn Ku, *Member, IEEE,* Spiridon Bakiras, and Cyrus Shahabi, *Senior Member, IEEE*

Abstract—With the popularity of location-based services and the abundant usage of smart phones and GPS-enabled devices, the necessity of outsourcing spatial data has grown rapidly over the past few years. Meanwhile, the fast arising trend of Cloud storage and Cloud computing services has provided a flexible and cost-effective platform for hosting data from businesses and individuals, further enabling many location-based applications. Nevertheless, in this database outsourcing paradigm, the authentication of the query results at the client remains a challenging problem. In this paper, we focus on the Outsourced Spatial Database (OSDB) model and propose an efficient scheme, called *VN-Auth*, which allows a client to verify the correctness and completeness of the result set. Our approach is based on neighborhood information derived from the Voronoi diagram of the underlying spatial dataset and can handle fundamental spatial query types, such as k nearest neighbor and range queries, as well as more advanced query types like reverse k nearest neighbor, aggregate nearest neighbor, and spatial skyline. We evaluated VN-Auth based on real-world datasets using mobile devices (Google Droid smart phones with Android OS) as query clients. Compared to the current state-of-the-art approaches (i.e., methods based on Merkle hash trees), our experiments show that VN-Auth produces significantly smaller verification objects and is more computationally efficient, especially for queries with low selectivity.

Index Terms—Spatial database outsourcing, location-based services, query authentication, spatial queries.

1 Introduction

The amount of digital spatial information available for day-to-day use has grown at an exceptional pace over the past decade. This large amount of information, as well as the complexity of the data, demand sophisticated data management systems that are beyond the capabilities of many small businesses or individuals. Additionally, the cost of running a state-of-the-art database management system may be significant, far exceeding the initial data acquisition cost. On the other hand, cloud computing provides flexible resources that can easily scale up or down (based on user demand), effectively reducing the operational and maintenance expenses for data owners (DOs). Consequently, the *database outsourcing* paradigm is becoming increasingly popular and has received a lot of attention in the research community. In this paradigm, the data owner delegates the management and maintenance of its database to a third-party cloud storage service provider (SP), and the SP is responsible for indexing the data and answering client queries.

In this work, we focus on the Outsourced Spatial Database (OSDB) model, as shown in Fig. 1. We assume that the clients are mobile users who issue location-based queries (e.g., k nearest neighbor (kNN) or range queries), in order to discover points of interest (POIs) in their neighborhood. However, there exist two major concerns with this model. First, as the SP is not the real owner of the data, it might return dishonest results out of its own interests. Second, query results might be tampered with by malicious attackers who could substitute one or more records with fake ones. Consequently, *query integrity assurance* is an important (and challenging) problem that has to be carefully addressed. To differentiate from traditional queries, we term spatial queries with integrity assurance as *verifiable queries*. In particular, for a verifiable query, the client must be able to prove that 1) all data returned from the SP originated at the DO and 2) the result set is correct and complete.

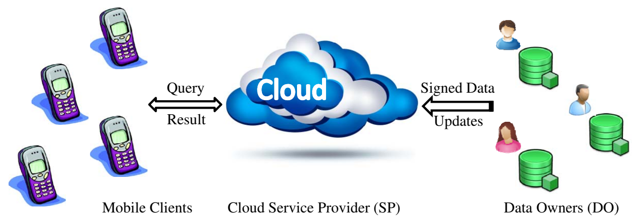


Fig. 1. System architecture.

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 [6]. Initially, the DO obtains a *private* and a *public* key through a trusted key distribution center. The private key is kept secret at the 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

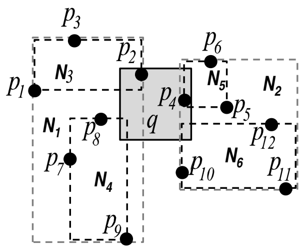
Ling Hu and Cyrus Shahabi are both with the Computer Science Department, University of Southern California, Los Angeles, California, USA 90089. E-mail: {lingh, shahabi}@usc.edu
Wei-Shinn Ku is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama, USA 36849. E-mail: weishinn@auburn.edu
Spiridon Bakiras is with the Department of Mathematics and Computer Science, John Jay College, CUNY, New York, NY, USA 10019. E-mail: sbakiras@jjay.cuny.edu

and the data to the SP which constructs the necessary data structures for efficient query processing. When the 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, the SP sends the $\mathcal{VO}$ to the client which can verify the results using the public key of the DO.

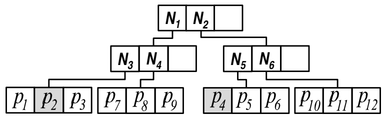
The current state-of-the-art solution for authenticating spatial queries is the Merkle R-tree (MR-tree) [34]. The MR-tree is essentially an R-tree that is augmented with authentication information, i.e., hash digests. In particular, every leaf node of the tree stores a digest that is computed on the concatenation of the binary representation of all objects in the node. Internal nodes are assigned a digest that summarizes the child nodes' MBRs (minimum bounding rectangles) and digests. Digests are computed in a bottom-up fashion, and the single digest at the root is signed by the DO. Range queries on the MR-tree are handled by a depth-first traversal of the tree. The resulting $\mathcal{VO}$ contains 1) all the objects in every leaf node visited and 2) the MBRs and digests of all the pruned nodes. Having this information, the client can reconstruct the root digest and compare it against the one that was signed by the owner. In addition, the client also examines the spatial relations between the query and each object/MBR included in the $\mathcal{VO}$, in order to verify the correctness of the result.

We argue that the structure of the MR-tree, as well as the verification process, suffer from several drawbacks. First, the authentication information (hash digests) embedded in the MR-tree reduces the node fanout, leading to more I/O accesses during query processing. Second, in the presence of updates from the DO, all digests on the path from an *affected* leaf node to the root have to be recomputed. Consequently, when updates are frequent, query performance is degraded, as discussed in [23]. Finally, the overhead of the $\mathcal{VO}$ can be significant, especially for queries that return only a few objects. This is due to the fact that the SP has to return all objects lying inside the leaf nodes that are visited during query processing. As an example, consider the range query q in Fig. 2. Even though the result set includes only two objects (p_2, p_4), the corresponding $\mathcal{VO}$ has to return all 12 objects in the database. An extension of the MR-tree, called MR*-tree [35], mitigates this last drawback by ordering the entries of each node and constructing hierarchical relationships of the digests therein. Nevertheless, it does not eliminate the $\mathcal{VO}$ overhead entirely, and it increases the verification cost at the client.

Motivated by the above observations, we propose VN-Auth, a novel approach that authenticates arbitrary spatial queries based on neighborhood information derived from the Voronoi diagram of the underlying spatial dataset. In particular, before delegating its database to the SP, the owner transforms each



(a) Points and MBRs



(b) MR-tree

Fig. 2. Range query on the MR-tree.

data object by creating a signature of the object itself along with information about its Voronoi neighbors. A key aspect of our method is that it separates the authentication information from the spatial index. As a result, the efficiency of the spatial index is not compromised, and updates affect only the neighborhoods of the updated objects. Furthermore, for kNN and range queries the $\mathcal{VO}$ is extremely compact, since it *only* includes the transformed objects that belong to the result set. We implemented our verification algorithms on Android mobile devices and run experiments using real-world datasets. Our experiments and results show that, compared to the MR-tree variants, VN-Auth produces significantly smaller verification objects and is more computationally efficient, especially for queries with low selectivity.

This paper subsumes our earlier work in [10] by extending the VN-Auth approach to handle more advanced spatial queries, such as reverse kNNs, k aggregate NNs and spatial skylines because these spatial query types are expected to be employed in advanced location-based services in the future [15, 28]. Specifically, we propose the corresponding query verification algorithms and investigate their performance through extensive experimentation with real-world datasets. To the best of our knowledge, this is the first paper that tackles the query verification problem for these types of queries.

The remainder of the paper is organized as follows. Section 2 reviews related work, while Section 3 discusses Voronoi diagrams and signature aggregation techniques. Section 4 describes the data transformation process, and Section 5 introduces the verification algorithms for different spatial query types. Experimental results are reported in Section 6. Section 7 introduces some additional features of VN-Auth and, finally, Section 8 concludes the paper.

2 Related Work

The idea of outsourcing databases to a third-party service provider was first introduced by Hacigümüs et al. [9]. Since then, numerous query authentication solutions have been proposed for auditing query results in outsourced relational databases [8, 14, 17, 21, 22, 29, 32, 33]. The first mechanism for verifying query results in *multi-dimensional* databases was proposed in [3]. The idea is to add authentication information into a spatial data structure by constructing certified chains [21] on the data points within each partition as well as on all the partitions in the data space. For a given range query, this approach generates a proof that every data point (within the intervals of the certified chains that overlap the query window) is either returned as a result or falls outside the query range. Based on [3], Cheng and Tan designed a mechanism for authenticating kNN queries on multi-dimensional databases, ensuring that the result set is complete, authentic, and minimal [4, 5]. Nevertheless, both solutions incur significant authentication overhead, and the required verification information consumes considerable client-server communication bandwidth.

Yang et al. [34, 35] introduced the MR- and MR*-trees, which are space-efficient authenticated data structures supporting fast query processing and verification. The MR-tree augments the standard R-tree [7], by computing hash digests on the concatenation of the binary representation of all the entries in a tree node. To verify the correctness and completeness of range query results, the generated $\mathcal{VO}$ includes 1) all visited objects and 2) MBRs and digests of all the pruned nodes. In addition to static queries, in [36] the MR-tree is utilized for authenticating moving kNN queries. The MR*-tree improves the MR-tree by ordering the entries of each node and constructing hierarchical relationships of the digests therein. Entries are sorted according to an in-order traversal of a KD-tree. As a result, when a query intersects an MBR, not all entries are required for query verification, and some of them can be pruned. The idea is similar to building a small Merkle tree on each node of the MR-tree. The MR*-tree significantly reduces the $\mathcal{VO}$ size but incurs some CPU overhead due to the embedded information. Nevertheless, neither the MR-tree nor the MR*-tree are able to handle data updates efficiently.

Efficient verification in the presence of frequent updates has been studied in the context of relational data. The Partially Materialized Digest scheme (PMD) [16] verifies one-dimensional queries and applies to both static and dynamic databases. Similar to our proposed approach, PMD employs separate indexes for the data and their associated verification information in order to avoid unnecessary costs when processing queries that do not request verification. Moreover, the authors designed two verification methods for spatial queries namely the Merkle R-tree (MR-tree) and the Partially Materialized KD-tree (PMKD). The first is an extension of the MB-tree method [14] to R-tree indexes, and the second an adaptation of the PMD methodology for spatial data. Furthermore, Pang et al. [23] introduced a protocol, based on signature aggregation, that verifies the authenticity, completeness and freshness of the query result. An important property of the protocol is that it allows new data to be disseminated immediately, while ensuring that outdated values (beyond a pre-set age) can be detected. In addition, the authors also implemented an efficient verification technique for ad-hoc equi-joins. Papadopoulos et al. [26] designed a solution for the authentication of continuous spatial queries, i.e., queries that are constantly evaluated on a highly dynamic database (consisting of moving objects). The proposed mechanism achieves both correctness and *temporal completeness* and aims at reducing the transmission overhead between the service provider and the clients.

All the aforementioned solutions require changes to the DBMS software (modifying the spatial indices or query algorithms) in order to support the embedded authentication information. This may not be realistic in many applications. On the other hand, Ku et al. [13] proposed a query integrity assurance technique that does not require any modifications in the DBMS software. The solution first employs a spatial transformation method that encrypts the spatial data before outsourcing it to the SP. 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 the query results. However, since [13] is not a deterministic solution, attacks may escape the auditing process. The VN-Auth method introduced in this paper is both efficient and deterministic, and does not require any modifications in the DBMS software.

3 Preliminaries

3.1 Voronoi Diagrams

Given a set of distinct objects $P = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^m$, the *Voronoi diagram* of P, denoted as $\mathbb{VD}(P)$, partitions the space of $\mathbb{R}^m$ into n 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 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 $\mathbb{VD}(P) = \{VC(p_1), VC(p_2), \ldots, VC(p_n)\}$. If two generators share a common edge, they are Voronoi neighbors. If we connect all the Voronoi neighbors, we get the *Delaunay triangulation* $\mathbb{DG}(P)$, which is the dual graph of $\mathbb{VD}(P)$. Note that, in this paper, we utilize Euclidean distance functions in $\mathbb{R}^2$.

***Property** 1:* Given a set of distinct points $P = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$, the Voronoi diagram $\mathbb{VD}(P)$ and the corresponding Delaunay triangulation $\mathbb{DG}(P)$ of P are unique.

***Property** 2:* The average number of Voronoi 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 point $p_i \in \{p_1, \ldots, p_{k-1}\}$.

Please refer to [20] (Properties 1-3) and [12] (Property 4) for proofs of the above properties.

3.2 Signature Aggregation

In our approach, the DO generates one signature for every object in the database, which is computed on the hash digest of the concatenation of the binary representation of the object and its Voronoi neighbors. In this way, the client can verify the authenticity of each individual object and its neighborhood. Note that, in this work, we utilize RSA signatures [6] that are typically 128 bytes in size. Alternatively, signatures based on Elliptic Curve Cryptography (ECC) [2] can be significantly shorter, thus reducing the overall communication and storage cost. However, ECC algorithms are computationally intensive and would perform poorly on mobile devices with limited computational capabilities.

The drawback of having one signature per database object is that it may increase considerably the communication cost between the SP and the client. Specifically, the SP has to transmit one 128-byte signature for every object in the result set, so the overhead can be significant for queries with high selectivity (especially for mobile clients). To avoid this cost, we employ a technique called *signature aggregation*. In particular, given k digests and their corresponding signatures (generated by the same signer), the SP can replace them with a single *Condensed-RSA* signature. Condensed-RSA has the same size as the original signatures (128 bytes), and it is computed as the modular multiplication of the k signatures. Aggregate signatures are provably secure [1, 18, 19] and can be computed by any party that possesses the individual signatures.

4 Data Transformation

Consider a DO that has compiled a large collection of n POIs (e.g., restaurants) within a geographic region. Each POI i is represented as a unique object p_i in the database, which has the form $\langle p_i.location, p_i.tail\rangle$. The *location* attribute stores the spatial coordinates of the object, while the *tail* attribute stores some additional information about the object, such as name, address, phone number, web site, etc. Before transmitting the database to the SP, the DO transforms each object by attaching neighborhood and authentication information.

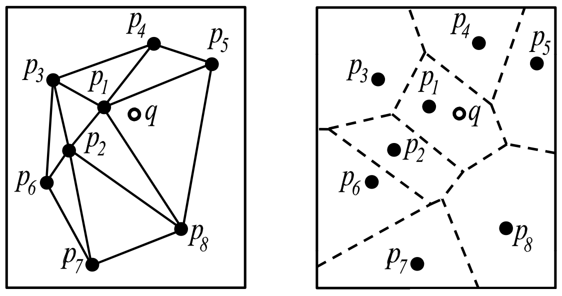
In particular, the DO initially computes the Voronoi diagram of the spatial dataset (as shown in Fig. 3) and retrieves the Voronoi neighbors of each POI. Then, it appends a *neighbors* attribute to every object in the database that stores the locations of all its Voronoi neighbors. For example, the *neighbors* attribute of p_5 from Fig. 3 is equal to:

$$p_5.neighbors = \{3, p_1.location, p_4.location, p_8.location\}$$

Note that, as the number of Voronoi neighbors for a POI is not fixed, the first value of the attribute specifies the exact number of neighbors. Furthermore, we assume that the DO stores the Voronoi neighbors in a clockwise or counter-clockwise order to facilitate the on-the-fly reconstruction of Voronoi cells at the client (as explained in Section 5.3.1). The final step is for the DO to sign each individual object, so that the client can verify the authenticity of the information stored therein. Specifically, for an object p_i, its signature $\mathbb{S}$ is computed as

$$\mathbb{S} = sign(h(p_i.location|p_i.tail|p_i.neighbors))$$

where h is a one-way, collision-resistant hash function and '|' denotes the concatenation of two binary strings. To summarize, each transformed object p_i at the DO's site has the form $\langle p_i.location, p_i.tail, p_i.neigbors, p_i.\mathbb{S}\rangle$.



(a) Delaunay triangulation (b) Voronoi diagram

Fig. 3. Spatial dataset example.

After the database transformation process completes, the DO transfers all objects to the SP. Upon receiving the database objects, the SP builds an appropriate spatial index and is then ready for query processing. Note that the leaf level of the index only stores pointers to the locations of the transformed objects on the disk. There are several advantages in our approach compared to the MR-tree variants. First, the DO is oblivious to the query processing mechanisms at the SP. Consequently, the SP may utilize any spatial index (and query processing algorithm) without informing the DO. Second, the spatial index does not store any authentication information, and thus remains compact and efficient. Third, database updates affect only their local regions, and they do not need to propagate to the root of the index. The only drawback of VN-Auth is the storage overhead of the transformed objects. However, according to Property 2 (Section 3.1), the number of neighbors for each generator in the Voronoi diagram does not exceed six, on average. Therefore, the expected overhead per object is $2 + 6 \times 16 + 128 = 226$ bytes (one 2-byte short integer, 16 bytes/point object, and one 128-byte signature), which is typically less than the size of the original object.

As mentioned earlier, one of the drawbacks of the MR-tree based verification techniques is that all updates propagate hash digest re-computations from the leaf nodes to the root. Consequently, frequent updates degrade the performance of the R-tree index and create a performance bottleneck at the root node. Our approach overcomes this problem by separating the authentication information from the spatial indexes. Verification information is attached only to the data objects, and updates affect only a specific neighborhood in the dataset. In addition, as neighboring objects are close to each other, they are often stored on the same or nearby pages on the disk. Therefore, updates in the VN-Auth framework are expected to be more efficient than MR-tree updates.

5 AUTHENTICATING SPATIAL QUERIES

In this section, we introduce the verification algorithms for typical location-based queries. Section 5.1 describes the query processing mechanism at the SP. Section 5.2 introduces the signature verification algorithm, which is common for all spatial queries, followed by the geometric verification algorithms for different types of spatial queries in Section 5.3.

5.1 Query Processing at the SP

Service providers process queries on the outsourced database (the database stored in the cloud) on behalf of the data owner. For verifiable queries, returning the query result to the clients is no longer sufficient. Instead, the SP is required to return a verification object ($\mathcal{VO}$) that contains 1) a condensed signature $\mathbb{AS}$ that verifies the authenticity of all objects in the $\mathcal{VO}$ and 2) the result set of the query with some additional objects that are necessary for the geometric verification process.

Depending on the spatial query type, the SP employs different query processing algorithms to retrieve the result. For kNN queries and queries that can be converted into kNN queries (e.g., range queries), query processing can follow any state-of-the-art algorithm that exists in the literature. The resulting $\mathcal{VO}$ contains the objects in the result set and their Voronoi neighbor information, which are sufficient for the verification process. For more sophisticated spatial queries, existing query processing algorithms still apply. However, the server needs to return some additional objects so that the client can perform the geometric verification. For example, in a reverse k nearest neighbor query, a few candidate objects (which might not be reverse kNNs of the query point) are returned in the $\mathcal{VO}$ to prove there are no objects missing from the result set (see details in Section 5.3.2).

5.2 Signature Verification

Two sequential steps are generally taken in a spatial query verification process. First, the aggregate signature of the $\mathcal{VO}$ is examined by the client to ensure that all returned objects originated at the DO. Next, all objects in the result set are evaluated to ensure that the geometric properties are satisfied and no legitimate objects are eliminated. Since signature verification is common in all query types, it is briefly discussed here and is omitted from the geometric verification process (Section 5.3).

When a client receives the $\mathcal{VO}$ from the SP, it verifies the aggregate signature using the public key of the DO. Specifically, the client simply performs a modular multiplication of the hash digests of all objects included in the $\mathcal{VO}$, and verifies that the result matches the plaintext that is derived by decrypting the aggregate signature with the DO's public key (see details in [19]). Note that, forging or altering a signature is computationally intractable for a polynomial-time adversary. If the $\mathcal{VO}$ fails the signature verification process, the client considers the result as corrupted and the verification process terminates. Otherwise, it continues with the geometric verification.

5.3 Geometric Verification

Geometric verification algorithms are specific to the geometric properties of the spatial query. Since different spatial queries exhibit different geometric characteristics, we discuss next how to proceed with the verification process for each individual query type.

5.3.1 kNN and Range Query Verification

Nearest neighbor (NN) queries are the fundamental building blocks in location-based services. In particular, kNN queries allow mobile users to retrieve the k closest POIs from the database, i.e., they may issue queries such as *"find the 10 nearest restaurants to my location"*. For the geometric verification, VN-Auth employs an incremental verification process that is based on Properties 3 and 4. Specifically, according to Property 3, p_i is the 1^{st} NN of the query point q, if and only if q lies inside the Voronoi cell of p_i. Once this geometric test is verified, Property 4 states that the 2^{nd} NN of q must be one of the Voronoi neighbors of the 1^{st} NN (p_i). In the general case, the k^{th} NN of a query point q exists in the union of the Voronoi neighbors of the first $(k-1)$ NNs of q.

The kNN 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 kNN result returned by the server is retrieved by calling $\mathcal{VO}$.result() on line 2. H is a min-heap which sorts points according to their distances to query q, and *Visited* is a set that retains already accessed objects. First (lines 3–8), the algorithm constructs the Voronoi cell of the first object p_1 and checks whether q falls inside $VC(p_1)$. If not, p_1 is not the 1st NN and the verification process fails. Otherwise, p_1 is verified as the 1st NN, and is added to the *Visited* set. Recall that each object is augmented with its Voronoi neighbors in a clockwise (or counter-clockwise) order. Therefore, the Voronoi cell can be computed on-the-fly by finding the circumcenters of the surrounding Delaunay triangles of p_1. This is not an expensive computation and can be performed efficiently on a mobile device.

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

```
1. $H \leftarrow \emptyset$; Visited $\leftarrow \emptyset$;
2. $L \leftarrow \mathcal{VO}.result()$; $p_1 = L[1]$;
3. VCP $\leftarrow$ computeVC($p_1$);
4. if ($q \notin$ VCP) then
5.     return false;{the 1st NN fails}
6. else
7.     Visited.add($p_1$);
8. end if
9. for $i = 1$ to $k - 1$ do
10.    for all ($n \in L[i].neighbors$) do
11.        if ($n \notin$ Visited) then
12.            Visited.add($n$);
13.            $H \leftarrow n$;
14.        end if
15.    end for
16.    if ($L[i+1].location \neq H.pop()$) then
17.        return false; {the $(i+1)^{th}$ NN fails}
18.    end if
19. end for
20. return true;
```

The subsequent *for* loop (lines 9–19) iterates through all objects in L (kNNs from the $\mathcal{VO}$) and performs the following operations: 1) if the Voronoi neighbor of the last verified object ($L[i]$) has not been visited yet, it is inserted into the min-heap H and the *Visited* set (lines 10–15), and 2) it compares the next object in the result set ($L[i+1]$) with the top of H (lines 16–18). If they are identical, $L[i+1]$ is verified as the next NN. Otherwise, verification fails and the program returns false. Note that the capacity of the min-heap is initially set to $(k-1)$, i.e., it will only hold the first $(k-1)$ objects that are closest to q (we are not interested in objects further away). Furthermore, the capacity can be decreased by one after each iteration in order to minimize the computational and storage cost of the client.

To illustrate the kNN verification algorithm, consider the 3NN query q in Fig. 4(a). First, suppose that the SP returns the correct result $\{p_1, p_4, p_2\}$. Once the aggregate signature is verified, the client computes the Voronoi cell of p_1 and checks whether q lies inside $VC(p_1)$. Since this is true, p_1 is proven to be the 1st NN of q. Consequently, the algorithm goes through the Voronoi neighbors of p_1 (p_2, p_3, p_4, p_5, p_8) and inserts the two closest objects to q into the min-heap. Therefore, the min-heap is now equal to $\{p_4, p_2\}$. Next, the second object in the result set (p_4) is compared with the top of the heap and, as they are identical, p_4 is verified as the 2nd NN of q. In the next iteration, every neighbor of p_4 is examined, but nothing is inserted into the heap, because p_2 is still the closest point to q (note that the min-heap only contains p_2 now). Finally, p_2 is compared against the 3rd NN reported in $\mathcal{VO}$, and the 3NNs are verified successfully at the client. Suppose now that the SP returns the incorrect result $\{p_1, p_4, p_3\}$ to the client. The aggregate signature is still valid, but the client will discover the error on line 18, as the 3rd NN derived from the first two neighbors should be p_2 rather than p_3.

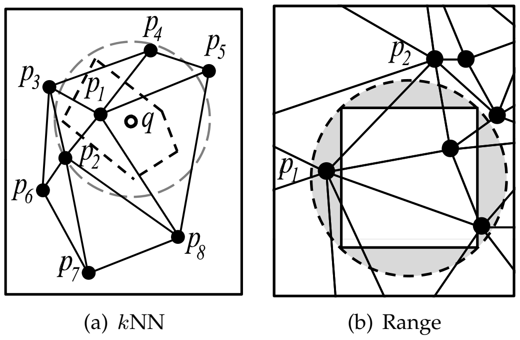


Fig. 4. Query verification examples.

Lemma 1: Algorithm Verify$k$NN can verify that the kNN result set returned to the client is correct and complete.

Proof: The proof follows directly from Properties 3 and 4 of the Voronoi diagrams (Section 3.1). □

VN-Auth also handles range query verification without difficulty. Specifically, any arbitrary range query can be transformed into one or more kNN queries, and the kNN query verification technique discussed above can be applied directly. Fig. 4(b) shows an example of a range query verification. The discussion on verifying arbitrary range queries is omitted here due to space limitations and interested readers can refer to [10] for more details.

5.3.2 *RkNN Query Verification*

Given a query point q and a set of objects P, the reverse kNN (RkNN) query of q is to find all objects in P such that q is one of the k nearest neighbors of p ($p \in P$) [30]. Let RkNN(q) be the result set of q. A verifiable reverse kNN query requests a verification object ($\mathcal{VO}$) that contains not only RkNN(q), but also the proof that the RkNN(q) set is correct and complete. Reverse kNN query verification is a challenging problem, especially for verifying that there are no false negatives. This is due to the fact that the cardinality of the result set for a RkNN query is not predetermined by the query.

Fig. 5 shows an example of a reverse k nearest neighbor query where q is the query point and k is 2. The objects are numbered in ascending order of their distance to the query point. The reverse two nearest neighbors of q are p_1, p_2, p_3, and p_4 (shown as black dots), i.e., R2NN(q)= $\{p_1, p_2, p_3, p_4\}$. RkNN(q) is called a correct and complete result if and only if it satisfies the following two conditions: 1) q is one of the k nearest neighbors of every object p in RkNN(q) (no false positives) and 2) all qualified reverse k nearest neighbors from P are in RkNN(q). In other words, no legitimate result object is eliminated by the server or a malicious attacker (no false negatives).

No false positives: To prove that object $p \in$ RkNN(q) is one of the reverse k nearest neighbors, the server needs to return up to k nearest neighbors of p. For example, in Fig. 5, p_4 is the R2NN of query q. Therefore, the server must return the 2NNs of p_4 (2NN(p_4)=$\{p_6, p_7\}$) for a client to verify the 2NNs and prove that q is closer to p_4 than p_7 is to p_4. If the server had returned p_{12} in RkNN(q), it should also have returned its first and second nearest neighbors p_{13} and p_{15} (2NN(p_{12})=$\{p_{13}, p_{15}\}$). Incorrect or incomplete 2NNs of p_{12} will fail the kNN verification algorithm and therefore be rejected by the client. Next, the client can prove p_{12} wrong by comparing the distance from p_{12} to q with the distance from p_{12} to p_{15}. kNN query verification is discussed in Section 5.3.1; we can use the Verify$k$NN algorithm (Algorithm 1) as a component for the RkNN verification. Note that the server does not always need to return k objects for each object in RkNN(q). In general, i additional objects are returned for the i^{th} reverse nearest neighbor of q. As shown in Fig. 5, there is a large overlap between the kNN sets of neighboring objects, and only one copy of each object is returned.

***Lemma** 2:* If every object p in the RkNN result set satisfies $D(p, q) \leq D(p, p_k)$ and p_k is the verified k^{th} nearest neighbor of p, then there are no false positives in RkNN(q).

Proof: The proof of the lemma is straightforward. By definition, if p is a reverse kNN of q, p is closer to q than to its k^{th} nearest neighbor p_k in P. The k^{th} nearest neighbor $p_k \in P$ of p is provable using Lemma 1. Therefore, comparing the distances $D(p, q)$ and $D(p, p_k)$ proves whether p is a true reverse k nearest neighbor of q. □

No false negatives: To prove that there are no false negatives or false misses, the server should return a verifiable superset of the result which guarantees that there are no more objects in RkNN(q) that are not returned by the server. There are two tasks to solve here.

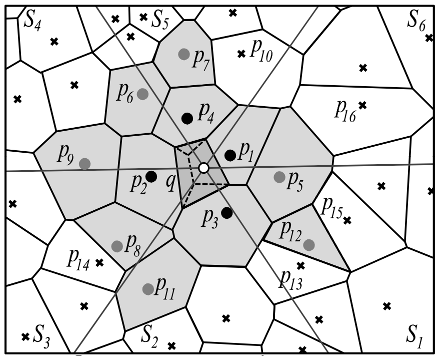


Fig. 5. Reverse kNN query verification.

First, we should find the proper superset of RkNN(q) and, second, the superset should be verifiable in the $\mathcal{VO}$.

To identify the superset of RkNN(q), we adopt the filtering technique of [27, 30]. In particular, if p is one of the reverse k nearest neighbors of query q, p satisfies the following two conditions: 1) the Delaunay distance between p and q in $DG(P \cup \{q\})$ is no more than k and 2) p is one of the k nearest neighbors of q in a partition S_i ($1 \leq i \leq 6$), where S_i is one of the six equal sectors partitioned by three lines that meet at the query point q, as shown in Fig. 5. Every object in P that qualifies for both conditions is a *candidate* and should be returned to ensure that there are no false negatives in RkNN(q). The filtering step prunes the number of candidates down to no more than $6k$ objects, which reside in the Voronoi cells local to q. For example, in Fig. 5, all objects in the grey area are candidates of R2NN(q). p_{16} is not a candidate because the Delaunay distance $DL(q, p_{16})$ is 3, and p_{10} is not a candidate because it is the 3^{rd} nearest neighbor of q in partition S_5.

***Lemma** 3:* If all candidates that satisfy the two aforementioned conditions are returned, then there are no false negatives in RkNN(q).

Proof: The proof can be found in [27, 30]. □

Although the partitioning method does not change the RkNN result, it does affect the candidate set. A client should use the same partitions as the server in order to verify the candidates. Note that the six partitions cover the sectors around the query point q. If the angle α of one of the three lines is fixed, the partitions are also fixed because the other two lines pass through the same point q and form $60°$ and $120°$ angles with the first line, respectively. Therefore, we can either set α to 0 (a horizontal line parallel to the x-axis), or let α be one of the query parameters submitted along with the RkNN query.

The verification algorithm first checks whether all candidates in P for the RkNN query are returned in the $\mathcal{VO}$. Candidate objects are verified in ascending order of their distance to q. Recall that each object is augmented with its Voronoi neighbors in a clockwise (or counter-clockwise) order. To compute the Delaunay distance from q to each $p \in P$, we need to verify the first NN p_1 of q by checking whether $q \in VC(p_1)$ (Property 3). Next, the neighbors of q in $DG(P \cup \{q\})$ are computed based on p_1 and its neighbors, and the result contains all objects with $DL(q, p) = 1$ in $DG(P \cup \{q\})$. In the example of Fig. 5, p_1–p_4 are identified as the Voronoi neighbors of q and are pushed into a min-heap H where points are sorted according to their distance to q. Meanwhile, a list is used to keep track of the verified candidate objects in each partition. Let L_i be the candidate list for partition S_i. L_i contains no more than k objects which are also sorted by their distance to q. Next, we iterate over the top entry p of H (lines 11–30). If partition S_i

to which p belongs already contains k objects, p is dropped. Otherwise, p is one of the kNNs in partition S_i, and we will check whether p is returned in the $\mathcal{VO}$. If $p \in \mathcal{VO}$, it is inserted into L_i ($p \in S_i$) and all the "not visited" neighbors of p are pushed into H if $DL(p,q) < k$ (lines 21–29). This ensures that every object in H has a DL distance of no more than k. The loop stops when H is empty. If all objects inserted into L_i are included in the $\mathcal{VO}$, all candidates for the RkNN query are verified. Finally, for each object p in the candidate set, the Verify$k$NN algorithm is called to check whether p is a reverse k nearest neighbor of q (lines 31–37).

Algorithm 2 VerifyRkNN(q,$\mathcal{VO}$,k)

```
1.  $H \leftarrow \emptyset$;$Visited \leftarrow \emptyset$; $R \leftarrow \mathcal{VO}$.result();
2.  if (!Verify1NN($p_1$, $q$)) then
3.      return false;
4.  end if
5.  VNQ $\leftarrow$ computeVN($q$, $p_1$, $p_1.neighbors$);
6.  for all ($p \in$ VNQ) do
7.      $p.dl = 1$;
8.      $Visited$.add($p$);
9.      $H \leftarrow p$;
10. end for
11. while (!H.isEmpty()) do
12.     $p \leftarrow H$.pop();
13.     $i \leftarrow$ getPartition($p$);
14.     if ($L[i]$.size() $==$ $k$) then
15.         continue;
16.     end if
17.     if ($p \notin \mathcal{VO}$) then
18.         return false;
19.     end if
20.     $L[i]$.add($p$);
21.     if ($p.dl < k$) then
22.         for all ($n \in p.neighbors$) do
23.             if ($n \notin Visited$) then
24.                 $n.dl = p.dl + 1$;
25.                 $H \leftarrow n$;
26.                 $Visited$.add($n$);
27.             end if
28.         end for
29.     end if
30. end while
31. for all ($p \in L$) do
32.     if (Verify$k$NN($p$, $q$, $\mathcal{VO}$)) then
33.         if ($p \notin$ R) then
34.             return false;
35.         end if
36.     end if
37. end for
```

The number of candidates returned for a RkNN query is no more than $6k$ (6 partitions with at most k objects in each partition). For each candidate p, up to k objects are returned to prove whether p is one of the RkNNs of q (i objects returned for the i^{th} RNN of q). Therefore, the size of the $\mathcal{VO}$ is $O(6k^2)$. However, observe that the neighbors overlap heavily around q, and only one copy of each object is returned. Therefore, the actual size of the $\mathcal{VO}$ is in general smaller than $6k^2$. The $\mathcal{VO}$ size is experimentally studied in Section 6.3.

5.3.3 kANN Query Verification

Given a set of query points $Q = \{q_1, q_2, \ldots, q_m\}$, a k aggregate nearest neighbor (kANN) query finds k objects in P which have a smaller aggregate distance to all points in Q than any other object in P [25]. The aggregate distance is defined by a monotonically increasing function f. An example of the kANN query is to find a meeting point p for a group of people (Q) with the smallest overall travel distance ($f = sum$). The first ANN p in P has the minimum sum distance to Q. Let kANN(Q) be the result set of the kANN query. A verifiable kANN query requires a verification object ($\mathcal{VO}$) which can prove that kANN(Q) is correct and complete.

Suppose we have a kANN query Q with the aggregate function $f = sum$, where the server returns p as the 1ANN of Q. To verify whether p is the 1ANN of Q, it is sufficient to show that there are no objects in P that have a smaller sum distance to Q than p. To this end, one can construct an area satisfying the equation $f_{sum}(o,Q) \leq f_{sum}(p,Q)$. This equation defines a safe region (*SR*) such that the distance from every point inside (or on the boundary of) *SR* to Q is smaller than (or equal to) $f_{sum}(p,Q)$. In other words, if there exists an object o in P with a smaller sum distance to Q, o must lie in *SR*. Therefore, a kANN verification algorithm can prove p to be the 1ANN by showing that p is the only object in *SR*. The *SR* of the 1ANN of query $Q = \{q_1, q_2, q_3\}$ in Fig. 6 is shown as the light grey region. In general, the kANN query verification algorithm can prove kANN(Q) correct and complete by showing that the *SR* corresponding to the k^{th} ANN (p_k) in kANN(Q) contains only the objects in kANN(Q). An example of a 3ANN query is shown in Fig. 6 as the grey area inside the contour passing through p_3. To prove that there are no false negatives, the server returns all objects whose Voronoi cells intersect with *SR*. The same safe region holds for weighted sum aggregate functions as well.

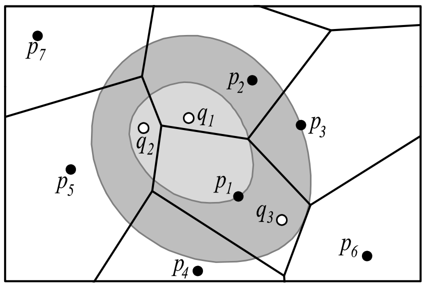


Fig. 6. Safe Regions of 1ANN and 3ANN queries ($f = sum$).

The verification algorithm first constructs the safe region *SR* from the kANN result and verifies that all result objects lie inside *SR*. A heap H is maintained and the first ANN object p_1 is pushed into H. Next, the algorithm iterates through the first entry of H to visit all objects whose Voronoi cells intersect with *SR*.

The top entry p_1 is removed from H and the neighbors p_i of p_1 are pushed into H if and only if the corresponding Voronoi edge intersects *SR* and p_i has not been visited so far. The algorithm stops when H is empty. The kANN verification algorithm is shown in Algorithm 3.

Algorithm 3 Verify$k$ANN(Q, $\mathcal{VO}$, k)

```
1. $H \leftarrow \emptyset$;$Visited \leftarrow \emptyset$; $R \leftarrow \mathcal{VO}.result()$;
2. sortByAggregateDistance($R$, $Q$);
3. $SR \leftarrow safeRegion(R[k], Q)$;
4. for ($i$ from 1 to $k$) do
5.     $p = R[i]$;
6.     if ($p \notin SR$) then
7.         return false;
8.     end if
9. end for
10. $p_1 = R[1]$;
11. H $\leftarrow p_1$; $Visited.add(p_1)$;
12. while (!$H.isEmpty()$) do
13.     $p \leftarrow H.pop()$;
14.     if ( ($p \in SR$) && ($p \notin R$)) then
15.         return false;
16.     end if
17.     $VCP \leftarrow computeVC(p)$;
18.     for all ($n$ in $p.neighbors$) do
19.         if (($n \notin Visited$) && VE($p$, $n$).$intersects(SR)$) then
20.             $H \leftarrow n$;
21.             $Visited.add(n)$;
22.         end if
23.     end for
24. end while
```

For the aggregate functions $f = max$ and $f = min$, we can construct appropriate safe regions as well. Specifically, the safe region for $f = max$ is the intersection of the circular areas centered at each query point q_i with radius $D(q_i, p_k)$, where p_k is the k^{th} object in the kANN result. Similarly, in the case of $f = min$, the safe region is the union of all circular areas centered at q_i with radius $D(q_i, p_k)$, where p_k is the k^{th} object returned by the server.

5.3.4 *Spatial Skyline Query Verification*

Given a set of data objects $P = \{p_1, \ldots, p_n\}$ and a set of query points $Q = \{q_1, \ldots, q_m\}$, a Spatial Skyline Query (SSQ) retrieves those objects in P that are not spatially dominated by any other object in P [28]. A point p_x spatially dominates another point p_y, with respect to Q, if and only if $D(p_x, q_i) \leq D(p_y, q_i)$ for all $q_i \in Q$ and $D(p_x, q_j) < D(p_y, q_j)$ for at least one object $q_j \in Q$. Let SSQ(Q) be the result of the SSQ query. A verifiable spatial skyline query requests a verification object ($\mathcal{VO}$) which can certify that the SSQ(Q) of a given set of query points Q is correct and complete.

When a client receives the $\mathcal{VO}$ from the SP, it uses the neighborhood information and the geometric properties of Spatial Skylines to verify the results. In particular, it first generates the Voronoi cells of all objects in R = SSQ(Q) and combines these Voronoi cells into a polygon V_p. According to Theorems 4.2 and 4.4 in [28], the convex hull of query Q ($CH(Q)$) must be covered by V_p. Otherwise, there must be at least one missing result object in R. If V_p contains $CH(Q)$, the client subsequently computes the *dominance region* (the intersection of the areas that lie outside of all circles $C(q_i, p)$, $\forall q_i \in Q$) of each point p in R (see Figure 7). Furthermore, for each Voronoi neighbor p' of p, if p' is not a spatial skyline point (i.e., $p' \notin R$), p' is inserted into a set N. Then, the client conducts a dominance check to test whether every point in N is in the dominance region of a spatial skyline point reported in R. Meanwhile, the client also checks to verify that every point in R is not dominated by any other point in R.

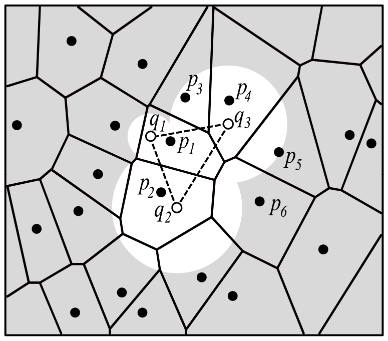


Fig. 7. The dominance region of p_1 (shaded area outside the union of three circles).

Algorithm 4 VerifySSQ(Q, $\mathcal{VO}$)

```
1. $N \leftarrow \emptyset$; $R \leftarrow \mathcal{VO}.result()$;
2. for ($i$ from 1 to $|R|$) do
3.     $p_i = R[i]$;
4.     $V_p \cup$ computeVC($p_i$);
5. end for
6. compute the convex hull $CH(Q)$;
7. if $CH(Q) \not\subset V_p$ then
8.     return false;
9. end if
10. for ($i$ from 1 to $|R|$) do
11.     $p_i = R[i]$;
12.     compute the dominance region $D_R$ of $p_i$;
13.     for all ($p'$ in $p_i.neighbors$) do
14.         if $p' \notin R$ then
15.             $N \leftarrow p'$;
16.         end if
17.     end for
18. end for
19. for ($i$ from 1 to $|N|$) do
20.     if $N[i] \not\subset D_R$ of any points in $R$ then
21.         return false;
22.     end if
23. end for
24. for ($i$ from 1 to $|R|$) do
25.     if $R[i] \subset D_R$ of any points in $R$ then
26.         return false;
27.     end if
28. end for
```

If a spatial skyline object s is missing from R (false negative), it can be detected by the geometric coverage check (lines 7–9), because if s lies inside $CH(Q)$ or $VC(s)$ intersects with $CH(Q)$, V_p cannot cover $CH(Q)$. The dominance check (lines 10–18) is capable of detecting any false positives, if they exist, since s is not dominated by any other point in R. Similarly, any fake results (false positives) can be discovered by the dominance check as well. The SSQ verification algorithm is summarized in Algorithm 4.

6 Experiments

6.1 Experimental Settings

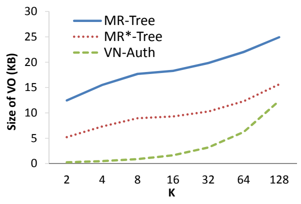
In this section, we evaluate the performance of VN-Auth experimentally and compare it against the MR-tree variants. Our implementation is in Java, with the client-side application running on a Google Android mobile device, and the server-side (SP) service hosted on a Windows Server PC with an Intel Core2 Duo 3GHz CPU and 4GB memory. The DO also runs on a PC with the same configuration. To implement the cryptographic operations, we used the Java cryptography extension packages [11]. Our experiments were performed on two real-world datasets obtained from the U.S. Census Bureau [31]: 1) **CA** which contains 62k data points from California, and 2) **NA** which consists of 556k POIs taken from North America. The RSA and SHA algorithms used in our experiments are both adopted from [11]. Signature generation/aggregation is performed on a PC, and the verification is done on an Android phone.

The VN-Auth framework consists of an offline database transformation part and an online query evaluation part. In the offline phase, we executed a Delaunay triangulation algorithm provided by Matlab to compute the Voronoi neighbors, and then incorporated the verification information into each object and generate a signature for each object. Compared with MR- and MR*-trees, VN-Auth incurs more storage overhead and more initialization cost. Specifically, VN-Auth data structure consumes about 3.5 to 4 times more space than a MR- or MR*-tree. And for initialization cost, it takes about 63 (70) seconds to build a MR-tree (MR*-tree) while it takes about 50 minutes to generate all data structures for VN-Auth on the NA dataset. Therefore, VN-Auth is clearly more expensive with regard to the offline cost.

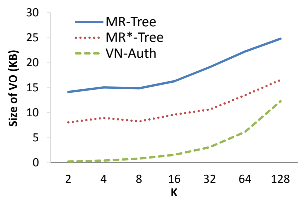
The SP provides online query evaluation to clients after indexing the spatial attributes of the objects with an R*-tree. We implemented the MR- and MR*-trees, based on the algorithms described in [35], with the R*-tree serving as the basis of both index structures. The page size was set to 4KB for all trees. For both the MR- and MR*-trees, index nodes and leaf nodes had a fan-out of 64 and 256, respectively.

6.2 VN-Auth vs. MR- and MR*-trees

We first evaluate the communication cost for all methods under kNN query processing. In particular, Fig. 8 shows the total size of the $\mathcal{VO}$ as a function of k. We assume that the outsourced spatial database contains a set of POIs, each with size of 16 bytes (spatial coordinates). An MBR is represented by two data points, and thus, consumes 32 bytes. Every point/MBR in the MR- and MR*-trees contains a 32-byte digest, which is computed on a leaf-level object or an MBR. Digests are computed using the SHA-256 algorithm and signatures are generated according to the RSA-1024 algorithm. Both algorithms are provided in [11]. In the MR*-tree, all the entries of an internal or a leaf node form a KD-tree, and the digest of each entry is computed in a similar fashion. When a query intersects with a node, the MR-tree returns all non-overlapping entries in the $\mathcal{VO}$, while the MR*-tree only returns the entries that are necessary for constructing the root digest of the KD-tree. Therefore, the MR*-tree is more communication-efficient than the MR-tree but at the same time more computationally expensive during query processing and database updates. Fig. 8 shows that VN-Auth outperforms both MR-tree variants in terms of communication cost, and is considerably better for queries with low selectivity (i.e., for $k \leq 32$). As an example, for an 8NN query, the MR-tree returns a $\mathcal{VO}$ of 22KB while the MR*-tree returns a $\mathcal{VO}$ of 14KB. In contrast, VN-Auth only returns a $\mathcal{VO}$ of 1.3KB, including signatures. This is due to the fact that the SP only returns the database objects that belong to the result set. In other words, the size of the $\mathcal{VO}$ in this approach is linear to k. Note that the values of k on the x-axis increase exponentially.

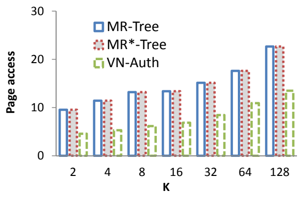


(a) CA

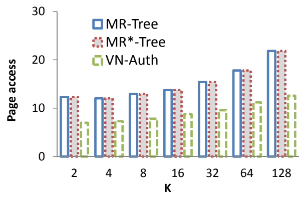


(b) NA

Fig. 8. $\mathcal{VO}$ size over k.



(a) CA



(b) NA

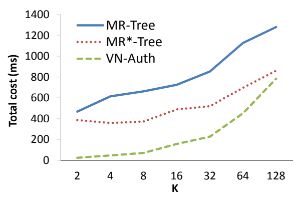
Fig. 9. Page accesses.

In our next experiment, we evaluate the query processing cost at the SP in terms of I/O accesses. For VN-Auth, we perform a slight optimization on the R*-tree index, based on the neighborhood information available at the SP. Specifically, we keep the internal nodes intact and modify the leaf nodes by storing pointers to each Voronoi neighbor of an object. The pointers facilitate easy navigation to the neighboring objects, and thus, achieve better I/O efficiency than the standard BFS algorithm. Note that such improvement is only possible for VN-Auth, as this information is incorporated in all database objects. Fig. 9 depicts the I/O cost at the SP as a function of k. VN-Auth results in fewer I/O accesses than both MR-tree variants in all cases. The page access cost is the same for the MR- and MR*-trees, because the MR*-tree does not store any additional information in the nodes. The idea of incorporating Voronoi diagrams into R*-trees was originally proposed in [27], where each object stored information about both its Voronoi neighbors and the corresponding cells. However, in VN-Auth, we only store Voronoi neighbors, in order to increase the fan-out of the leaf nodes.

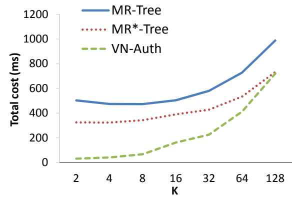
Next, we investigate the feasibility of implementing complex verification algorithms on mobile devices. The cost of cryptographic primitives has become less expensive as computer hardware gets more advanced and the corresponding algorithms become more efficient. Signing an individual message (on a PC) with the RSA algorithm costs $4.12ms$, while verifying it takes $0.32ms$. On mobile devices, verifying one signature costs $2.12ms$, while verifying a condensed-RSA signature with 20 individual signatures takes additional $0.32ms$ on modular multiplications. Clearly, state-of-the-art mobile devices are more than capable of performing complex cryptographic operations, thus allowing the implementation of efficient spatial query verification techniques. Note that compared with cryptographic operators, the cost of hash operations is much less. With the SHA-256 hashing algorithm, a PC can process 63MB per second while a mobile phone can process around 8.72MB per second, which is 7-8 times slower than a PC. Although the hashing operator is very efficient, the actually verification cost on large messages is not negligible due to the cost of converting point (MBR) objects to byte arrays (the input that hash functions take) and the concatenation of entries (index/leaf nodes) on the MR- and MR*-trees. Therefore, computing the root digest may become more expensive than verifying a 1024-bit condensed-RSA signature on the mobile device (which takes less than $3ms$).

The next set of experiments investigates the query cost at the mobile clients. In particular, Fig. 10 depicts the total amount of time required for receiving and verifying a kNN query result as a function of k. Our experiments are conducted on Android phones that are connected to the Internet through their built-in Wi-Fi. The total query cost in Fig. 10 consists of two parts: the data transfer cost (the transmission time of the $\mathcal{VO}$), and the computational cost at the client, i.e., for result checking, digest computations, and signature verification. In other words, the query cost is measured from the moment the mobile client receives the first byte of the $\mathcal{VO}$ from the SP until the verification process is finalized. Again, VN-Auth is superior to the MR-tree based methods, and its verification cost is significantly lower for queries that return fewer results. Fig. 11 shows the data transfer cost for the same set of experiments. Comparing it with Fig. 10, we conclude that the data transfer cost is a dominant factor in the overall query verification cost. Consequently, minimizing the $\mathcal{VO}$ size is an important factor in designing efficient query verification algorithms.

Our next set of experiments studies the cost of database updates. Fig. 12 shows the response time to complete a set of updates as a function of the number of objects being updated. The update cost comprises of the time for locating the object, and the time for recomputing digests and signatures as dictated by the corresponding authentication mechanisms. for the MR-tree based approaches, a database update affects all the nodes from the leaf to the root, because it triggers a series of hash digest re-computations in a bottom-up fashion. In VN-Auth, however, only neighboring objects are modified by the DO during an update. Consequently, VN-Auth completes the update process 2 times faster than the MR-tree, and 3 times faster than the enhanced MR*-tree approach. The time cost includes both CPU and I/O cost.

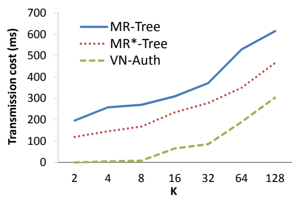


(a) CA

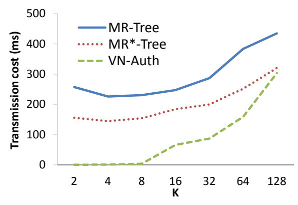


(b) NA

Fig. 10. Total query cost over k.

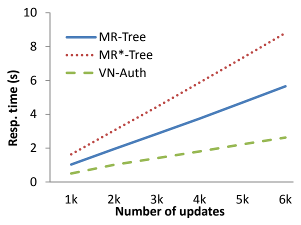


(a) CA

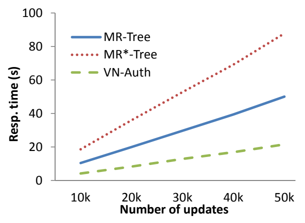


(b) NA

Fig. 11. Data transfer cost over k.

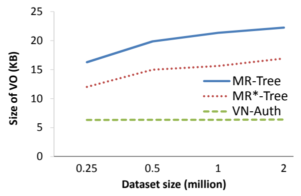


(a) CA

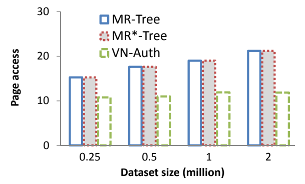


(b) NA

Fig. 12. Response time over database updates.



(a) VO size



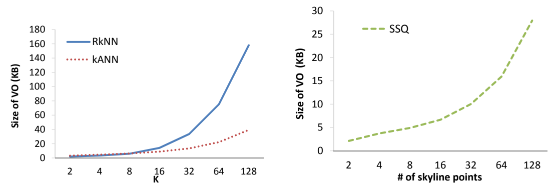
(b) Page access

Fig. 13. Effects of database sizes.

In the last set of experiments of this section, we investigate how the dataset size affects the performance of all three approaches. Our NA dataset contains about half a million data points. We select a subset of the NA dataset as the quarter million dataset. In addition, we generated two synthetic datasets with 1 million and 2 million data points. Fig. 13 shows the $\mathcal{VO}$ size (a) and the number of page access (b) over different datasets for kNN queries where k is set to 64. The experiments shows that, for MR- and MR*-tree, both the $\mathcal{VO}$ size and the number of page access increase as the size of dataset increases. However, for VN-Auth, the size of $\mathcal{VO}$ and page access depend on the query size k, therefore they remain about the same on different datasets.

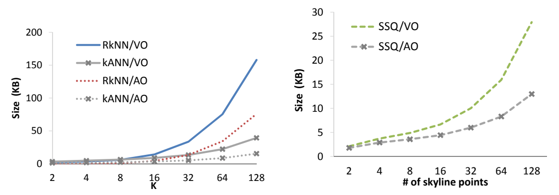
6.3 Advanced Spatial Queries

Query verification for advanced spatial queries such as reverse k nearest neighbor (RkNN), k aggregate nearest neighbor (kANN) and spatial skyline query (SSQ) is very challenging, due to the algorithms' complexity. In this section, we investigate the efficiency of the VN-Auth verification algorithms on mobile clients for handling these query types. To the best of our knowledge, there are no existing verification approaches that can handle such queries, and therefore, we cannot compare our results against other methods. Instead, we report experimental results on various performance metrics for all our algorithms. Additionally, we compared the query cost on the SP based on the VN-Auth approach with the best algorithms using MR- and MR*-tree for all three query types. Since the performance trends for each metric are very similar in both datasets (**CA** and **NA**), we only report the results obtained from the **NA** dataset.



(a) RkNN and kANN (b) SSQ

Fig. 14. Size of $\mathcal{VO}$.

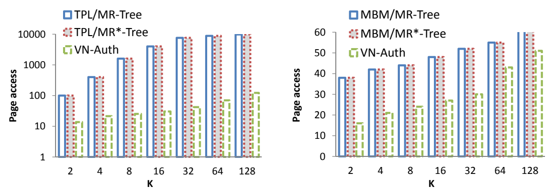


(a) RkNN and kANN (b) SSQ

Fig. 15. Size of $\mathcal{VO}$ and $\mathcal{AO}$.

In the first set of experiments, we investigate the size of the $\mathcal{VO}$ for RkNN, kANN, and SSQ queries. Particularly, Fig. 14(a) shows the $\mathcal{VO}$ size as a function of k (for RkNN and kANN queries), while Fig. 14(b) demonstrates the $\mathcal{VO}$ size as a function of the number of query points (for SSQ queries). For RkNN queries, the $\mathcal{VO}$ size grows more rapidly, compared to kANN and SSQ queries, as the number of result objects increases. This is due to two reasons. First, the cardinality of the result set for an RkNN query is not determined by the query parameter k, but depends on the actual data distribution. In general, an RkNN query returns more than k results and, as k increases, the number of results grows significantly. Second, to enable client verification, the $\mathcal{VO}$ contains not only the query results but also the objects necessary for verifying those results (see Section 5.3.2). For ease of presentation, we term these objects auxiliary objects ($\mathcal{AO}$). Consequently, as the value of k increases, the size of the $\mathcal{AO}$ increases as well, resulting in larger $\mathcal{VO}$ sizes. Fig. 15(a) and Fig. 15(b) show the corresponding $\mathcal{AO}$ and $\mathcal{VO}$ sizes for all three query types.

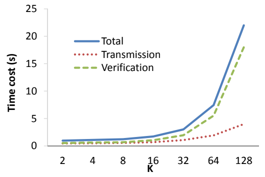
The second set of experiments evaluates the query processing cost at the SP in terms of I/O accesses. As VN-Auth maintains neighborhood information for each object on the leaf nodes, processing RkNN, kANN and SSQ queries is more efficient. Specifically, after the first nearest neighbor of the query point (or one of the query points) is discovered, the neighborhood pointers embedded on the leaf nodes navigate the query process within adjacent areas and retrieve objects that are most relevant to the query. As a result, the I/O cost for the three query types is lower than their counterpart algorithms (TPL [30] for RkNN, MBM [25] for kANN and B^2S^2 [24] for SSQ) when implemented on an MR- or MR*-tree. As shown in Fig. 16 and Fig. 18(a), for all three query types, the VN-Auth-based approaches achieve much better I/O efficiency. Note that the I/O cost is identical for the MR- and MR*-trees, as the MR*-tree does not maintain additional information in the nodes. Our experimental results are also consistent with the results reported in [27].



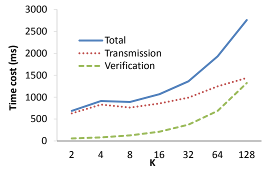
(a) RkNN (b) kANN

Fig. 16. Page accesses.

Our final experiment studies the communication and verification cost on a mobile client for all three query types. Specifically, Fig. 17(a) and Fig. 17(b) depict the communication and verification cost (for RkNN and kANN queries, respectively) as a function of k, while Fig. 18(b) shows the corresponding cost (for SSQ queries) as a function of the number of skyline points. The RkNN query cost is dominated by the verification process when the value of k grows larger than 32. Conversely, the cost for kANN and SSQ queries is dominated by the transmission cost when k or the number of skyline points. This is due to the fact that the geometric verification process for RkNN queries is very expensive. In particular, for each candidate object returned by the server, the mobile client runs a kNN verification algorithm to check whether the candidate belongs to the RkNN result. With increasing k, the number of kNN verification computations grow exponentially, resulting in a large overhead on the verification cost.

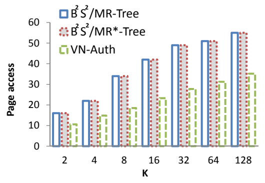


(a) RkNN

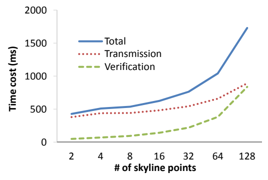


(b) kANN

Fig. 17. Client query cost.



(a) Page access



(b) Client query cost

Fig. 18. SSQ queries.

7 Discussion

For wireless applications, due to limitations such as bandwidth constraints, unstable connections, and restricted power supply, the amount of data communicated between client and server should be kept as small as possible. Therefore, the size of the $\mathcal{VO}$ plays an important role in designing efficient query verification algorithms for mobile access to outsourced databases. kNN queries from mobile clients usually have low selectivity (i.e., the user is interested in very few results) but require fast response time. VN-Auth is a novel approach that successfully meets these requirements in the location-based services model. However, we are aware of the fact that, when the query selectivity is high, the Merkle Hash Tree (MHT) based approaches are more efficient. In the extreme case where the client retrieves the whole database, the MHT approach would only return the root signature, while VN-Auth would need to return $6 \cdot n$ neighbors, where n is the database cardinality. Consequently, it is important to first identify the query types and user profiles in the system and then choose a solution that accommodates the most important requirements of the application. For location-based services on mobile devices, VN-Auth is clearly a better solution.

Another important issue is providing the user with the ability to verify the query results in a progressive fashion. This is very useful in cases where decision making is based on the first few objects in the result. To ensure query integrity, the application program needs to verify the overall result before showing any part of it to the end user. For MHT based approaches, result reporting is blocked until the root digest is computed and the signature is verified. Therefore, it is not possible to pre-qualify any object in the result set, due to the hierarchical structure of the authenticated index. However, progressive result reporting is feasible under the VN-Auth framework. The result can be returned in batches, each of which can be verified independently, based on information that arrived before and not on information that is expected to arrive later. For example, the SP can divide the result set of a 20NN query into 4 batches with 5 objects each. The first 5NNs in the first batch contain the aggregate signature of the five objects. After the first batch is verified, the five objects can be reported to the end user before the verification of the second batch starts. The batches that arrive later depend on preceding ones, but not vise versa. Therefore, query verification can be performed in an incremental fashion, and results can be shown to the end user progressively. In addition, the application may allow the user to examine each batch before receiving the next one. In this way, if the user is satisfied with the current result set and does not wish to retrieve more objects, query processing may be terminated early.

8 Conclusions

In this paper, we introduced the VN-Auth query integrity assurance framework for outsourced spatial databases. Our approach separates the authentication information from the spatial index, thus allowing efficient query processing at the service provider. Additionally, since the verification information depends only on the object and its Voronoi neighbors, database updates can be disseminated quickly to their local regions and be performed independently of all other updates in the database. VN-Auth handles not only kNN and range queries, but also more advanced query types, such as reverse kNNs, k aggregate NNs and spatial skylines. More importantly, VN-Auth produces compact verification objects, which enables fast query verification on mobile devices with limited capabilities. Finally, we showed that our approach facilitates progressive result verification, which allows a user to retrieve objects in an incremental fashion until the results are deemed satisfactory. Our experiments with real-world datasets and on mobile platforms confirm that, compared to the MR-tree variants, VN-Auth produces significantly smaller verification objects, and incurs lower query verification cost, especially for queries with low selectivity.

9 Acknowledgements

This research has been funded in part by NSF grants CNS-0831505 (CT), CNS-0831502 (CT), IIS-0845262 (CAREER), the NSF Integrated Media Systems Center (IMSC), and unrestricted cash and equipment gift from Google and Microsoft.

References

[1] M. Bellare and A. Palacio. Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In *CRYPTO*, pages 162–177, 2002.

[2] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *J. Cryptology*, 17(4):297–319, 2004.

[3] W. Cheng, H. Pang, and K.-L. Tan. Authenticating Multi-dimensional Query Results in Data Publishing. In *DBSec*, pages 60–73, 2006.

[4] W. Cheng and K.-L. Tan. Authenticating *k*NN Query Results in Data Publishing. In *Secure Data Management*, pages 47–63, 2007.

[5] W. Cheng and K.-L. Tan. Query assurance verification for outsourced multi-dimensional databases. *Journal of Computer Security*, 17(1):101–126, 2009.

[6] A. Fiat. Batch RSA. *J. Cryptology*, 10(2):75–88, 1997.

[7] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD Conference*, pages 47–57, 1984.

[8] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database-service-provider Model. In *SIGMOD Conference*, pages 216–227, 2002.

[9] H. Hacigümüs, S. Mehrotra, and B. R. Iyer. Providing Database as a Service. In *ICDE*, page 29, 2002.

[10] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi. Verifying spatial queries using Voronoi neighbors. In *GIS*, pages 350–359, 2010.

[11] Java SE Security. http://java.sun.com/javase/technologies/security.

[12] M. R. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *VLDB*, pages 840–851, 2004.

[13] W.-S. Ku, L. Hu, C. Shahabi, and H. Wang. Query integrity assurance of location-based services accessing outsourced spatial databases. In *SSTD*, pages 80–97, 2009.

[14] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *SIGMOD Conference*, pages 121–132, 2006.

[15] X. Lin, J. Xu, and H. Hu. Authentication of location-based skyline queries. In *CIKM*, pages 1583–1588, 2011.

[16] K. Mouratidis, D. Sacharidis, and H. Pang. Partially Materialized Digest Scheme: An Efficient Verification Method for Outsourced Databases. *VLDB J.*, 18(1):363–381, 2009.

[17] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and Integrity in Outsourced Databases. In *NDSS*, 2004.

[18] E. Mykletun, M. Narasimha, and G. Tsudik. Signature Bouquets: Immutability for Aggregated/Condensed Signatures. In *ESORICS*, pages 160–176, 2004.

[19] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and Integrity in Outsourced Databases. *TOS*, 2(2):107–138, 2006.

[20] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000.

[21] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *SIGMOD Conference*, pages 407–418, 2005.

[22] H. Pang and K.-L. Tan. Authenticating Query Results in Edge Computing. In *ICDE*, pages 560–571, 2004.

[23] H. Pang, J. Zhang, and K. Mouratidis. Scalable Verification for Outsourced Dynamic Databases. *PVLDB*, 2(1):802–813, 2009.

[24] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[25] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Trans. Database Syst.*, 30(2):529–576, 2005.

[26] S. Papadopoulos, Y. Yang, S. Bakiras, and D. Papadias. Continuous Spatial Authentication. In *SSTD*, pages 62–79, 2009.

[27] M. Sharifzadeh and C. Shahabi. VoR-Tree: Incorporating Voronoi Diagrams into R-Trees for I/O-efficient Processing of Spatial Nearest Neighbor Queries. *VLDB J.*, in press, 2010.

[28] M. Sharifzadeh, C. Shahabi, and L. Kazemi. Processing spatial skyline queries in both vector spaces and spatial network databases. *ACM Trans. Database Syst.*, 34(3), 2009.

[29] R. Sion. Query Execution Assurance for Outsourced Databases. In *VLDB*, pages 601–612, 2005.

[30] Y. Tao, D. Papadias, and X. Lian. Reverse *k*NN search in arbitrary dimensionality. In *VLDB*, pages 744–755, 2004.

[31] U.S. Census Bureau. http://www.census.gov/geo/www/tiger/.

[32] H. Wang, J. Yin, C.-S. Perng, and P. S. Yu. Dual Encryption for Query Integrity Assurance. In *CIKM*, pages 863–872, 2008.

[33] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity Auditing of Outsourced Data. In *VLDB*, pages 782–793, 2007.

[34] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Spatial Outsourcing for Location-based Services. In *ICDE*, pages 1082–1091, 2008.

[35] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Authenticated Indexing for Outsourced Spatial Databases. *VLDB J.*, 18(3):631–648, 2009.

[36] M. L. Yiu, E. Lo, and D. Yang. Authentication of Moving kNN Queries. In *ICDE*, 2011.

Ling Hu is a Ph.D. student in the Computer Science department at the University of Southern California. Her research interests include spatial and temporal data management, geographical information systems, query integrity and query optimization, and data warehousing. She has an M.S. degree in Computer Science from Northeastern University. She is a student member of the ACM and the IEEE.

Wei-Shinn Ku received his Ph.D. degree in Computer Science from the University of Southern California (USC) in 2007. He also obtained both the M.S. degree in Computer Science and the M.S. degree in Electrical Engineering from USC in 2003 and 2006, respectively. He is an Assistant Professor with the Department of Computer Science and Software Engineering at Auburn University. His research interests include spatial and temporal data management, mobile data management, geographic information systems, and security and privacy. He has published more than 50 research papers in refereed international journals and conference proceedings. He is a member of the ACM and the IEEE.

Spiridon Bakiras received the BS degree in Electrical and Computer Engineering from the National Technical University of Athens in 1993, the MS degree in Telematics from the University of Surrey in 1994, and the PhD degree in Electrical Engineering from the University of Southern California in 2000. Currently, he is an associate professor in the Department of Mathematics and Computer Science at John Jay College, City University of New York. Before that, he held teaching and research positions at the University of Hong Kong and the Hong Kong University of Science and Technology. His current research interests include database security and privacy, mobile computing, and spatiotemporal databases. He is a member of the ACM and a recipient of the US National Science Foundation (NSF) CAREER award.

Cyrus Shahabi is a Professor and the Director of the Information Laboratory (InfoLAB) at the Computer Science Department and also the Director of the NSF's Integrated Media Systems Center (IMSC) at the University of Southern California. He is also the CTO and co-founder of a USC spin-off, Geosemble Technologies. He received his B.S. in Computer Engineering from Sharif University of Technology in 1989 and then his M.S. and 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 hundred-fifty research papers in the areas of databases, GIS and multimedia. Dr. Shahabi was an Associate Editor of IEEE Transactions on Parallel and Distributed Systems (TPDS) from 2004 to 2009. He is currently on the editorial board of the VLDB Journal, IEEE Transactions on Knowledge and Data Engineering (TKDE), ACM Computers in Entertainment and Journal of Spatial Information Science. He is the founding chair of IEEE NetDB workshop and also the general co-chair of ACM GIS 2007, 2008 and 2009. He chaired the nomination committee of ACM SIGSPATIAL for the 2011-2014 terms. He regularly serves on the program committee of major conferences such as VLDB, ACM SIGMOD, IEEE ICDE, ACM SIGKDD, and ACM Multimedia. Dr. Shahabi is a recipient of the ACM Distinguished Scientist award in 2009, the 2003 U.S. Presidential Early Career Awards for Scientists and Engineers (PECASE) and the NSF CAREER award in 2002.