Authentication of Moving kNN Queries

Man Lung Yiu, Eric Lo, Duncan Yung

Department of Computing, Hong Kong Polytechnic University
{csmlyiu, ericlo, cskwyung}@comp.polyu.edu.hk

***Abstract*— A moving kNN query continuously reports the k nearest neighbors of a moving query point. In addition to the query result, a service provider that evaluates moving queries often returns mobile clients a safe region that bounds the validity of query results to minimize the communication cost between the two parties. However, when a service provider is not trustworthy, it may send inaccurate query results or incorrect safe regions to clients. In this paper, we present a framework and algorithms to authenticate results and safe regions of moving kNN queries. Extensive experiments on both real and synthetic datasets show that our methods are efficient in terms of both computation time and communication costs.**

I. Introduction

A moving kNN query [9, 25] continuously reports the k nearest neighbors of a moving query point q and it has numerous mobile applications, for example, finding the k nearest gas stations when a car moves or finding the k nearest restaurants when a tourist walks.

Location-based service providers (LBS) that offer remote kNN querying services often return mobile users a *safe region* [9, 25] in addition to the query results. A safe region $SR(q)$ of a moving query point q is a region in the dataset D where the results of query q remain unchanged as long as q moves within it. Safe region is a powerful optimization in moving query processing because it allows a mobile user to issue a new query to the LBS (to get the latest query results) only when the user leaves the safe region, thereby significantly reduces the communication frequency between the user and the service provider.

Query results and safe regions returned by LBS, however, may not be always accurate. For instance, a hacker may have infiltrated the LBS's servers [20] so that results of kNN queries all include a particular location (e.g., the White House). Furthermore, it is possible that the LBS is self-compromised, and thus ranks sponsored facilities higher in its query results or returns overly large safe region to clients so as to save computation resources [12, 17, 24].

Recently, techniques for authenticating query results have received a lot of attention. For example, [6, 11, 21] study the authentication of relational queries; [7, 15] study the authentication of sliding window (data stream) queries; [12] addresses this issue on text similarity queries; [14, 22] study the authentication of static spatial queries; and [24] addresses the authentication of shortest path queries. The issue of authenticating moving queries, however, has not been addressed yet. Existing techniques for authenticating static spatial queries such as [14, 22] cannot help in authenticating moving queries because their authentication target is the query result, which is a subset of the dataset, whereas the authentication targets of moving queries include both the query result and the *safe region*—the latter is not a subset of the dataset but dynamically computed by the LBS at run-time.

To the best of our knowledge, this paper is the first to address this challenging issue of authenticating moving kNN queries. Our contributions include:

1) A framework to support the evaluation and authentication of moving kNN queries.
2) The design of *verification objects* $\mathcal{VO}$ [6, 12, 22, 24] for authenticating the safe regions (and also the results) of moving kNN queries.
3) Efficient algorithms to construct the verification objects. First, we present a solution that focuses on *communication-efficiency* (i.e., minimizing the total communication cost between a mobile client and the LBS) for better network bandwidth utilization and also for mobile users who are charged by mobile services on a per-data basis. Unlike traditional moving query processing, the total communication cost in our problem setting depends not only on (i) the frequency of communication between the LBS and the client, (ii) the size of query results, but also (iii) the size of the verification object $\mathcal{VO}$ transferred in each communication. Second, we present another solution that focuses on *computation-efficiency*. Such a solution is important to mobile devices that have limited computation power and/or battery life.
4) A method for a mobile client to refresh its safe region without issuing a new query to the server even when the client leaves its current safe region. This technique further helps to reduce the communication frequency as well as the computation effort.
5) A thorough experimental study on real data and synthetic data to study the communication and computation efficiency of our proposed methods.

The rest of this paper is organized as follows. We discuss related work in Section II. Our framework is presented in Section III. In Section IV, we present two methods for authenticating safe region of moving kNN queries that fit our framework. The experimental study is presented in Section V. Finally, we conclude this paper in Section VI.

II. Related Work and Background

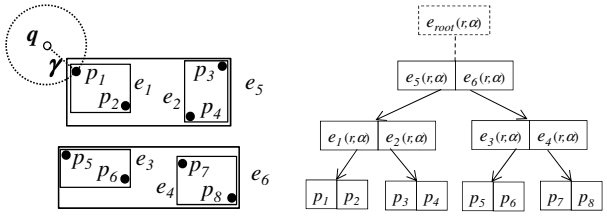
Query authentication In the literature, most authentication techniques [6, 7, 12, 14, 15, 21, 22, 24] are based on *Merkle tree* [8], which is an *authenticated data structure* (ADS) that is built on the dataset. Digests of nodes in the tree are first

recursively computed from the leaf level to the root level (a secure-hash h function is often used to compute a fixed-length digest). Then, the signature of the dataset is obtained by signing the root digest using the data owner's private key. *Signature aggregation* [11, 13] is an alternative of Merkle tree. It works by having a signature for each tuple in the dataset. When compared to Merkle tree based methods, signature aggregation achieves better concurrency and generates smaller verification objects. However, it incurs larger update overhead at the data owner and higher verification cost at the client. Recently, Yi et al. [23] propose a probabilistic approach for authenticating aggregation queries; and Kundu et al. study the authentication of trees [4] and graphs [5] without revealing any data objects beyond the user's access rights. In this paper, we adopt the Merkle tree approach because of its wide popularity/applicability.

To authenticate spatial queries, Yang et al. [22] develops an authenticated data structure (ADS) called Merkle R-tree (MR-tree) based on R*-tree [1] and Merkle tree [8]. Figure 1b shows an MR-tree for the dataset shown in Figure 1a. A leaf entry p_i stores a data point. A non-leaf entry e_i stores a rectangle $e_i.r$ and a digest $e_i.\alpha$, where $e_i.r$ is the minimum bounding rectangle of its child node, and $e_i.\alpha$ is the digest of the concatenation (denoted by $|$) of binary representation of all entries in its child node. For instance, $e_1.\alpha = h(p_1|p_2)$, $e_5.\alpha = h(e_1|e_2)$, and $e_{root}.\alpha = h(e_5|e_6)$. The *root signature* is generated by signing the digest of the root node $e_{root}.\alpha$ using the data owner's private key.

Consider the nearest neighbor (NN) query q in Figure 1. Its NN is p_1 and its NN distance is denoted by $\gamma = dist(q, p_1)$. In order to let the client to verify the correctness of the NN results, the LBS utilizes the MR-tree (provided by the data owner) to generate a verification object $\mathcal{VO}$. For kNN queries, the $\mathcal{VO}$ is computed by a depth-first traversal of the MR-tree. First, a circular verification region $\odot(q, \gamma)$ with center q and radius γ is defined. Then, the tree is traversed with the following conditions: (i) if a non-leaf entry e does not intersect $\odot(q, \gamma)$, e is added to the $\mathcal{VO}$ (e.g., e_2, e_6) and its subtree will not be visited; (ii) data points in any visited leaf node are added into the $\mathcal{VO}$ (e.g., p_1, p_2). In this example, we have $\mathcal{VO} = \{\{\{p_1, p_2\}, e_2\}, e_6\}$, where $\{$ and $\}$ are tokens for marking the start and end of a node.

Upon receiving the $\mathcal{VO}$, the client first checks the correctness of the $\mathcal{VO}$ by reconstructing the digest of root of the MR-tree from the $\mathcal{VO}$ and then verifying it against the root signature using the data owner's public key. If the verification is successful, the client next finds the NN result (and the NN distance γ') directly from the data points extracted from the $\mathcal{VO}$ (ignoring the non-leaf entries). Then, the client re-defines his own verification region $\odot(q, \gamma')$. Note that if a non-leaf entry e in the $\mathcal{VO}$ does not intersect $\odot(q, \gamma')$, that means all the points in e are farther than the computed NN result with respect to q. Thus, the client verification step is to check whether every non-leaf entry in the $\mathcal{VO}$ satisfies $e.r \cap \odot(q, \gamma') = \emptyset$. If so, the client can assure that the computed NN result is correct.



(a) data points on a plane (b) Merkle R-tree (T_D)

Fig. 1. Authentication of nearest neighbor queries

To the best of our knowledge, we are the first to study the authentication of moving queries. Existing spatial authentication techniques [14, 22] cannot help in authenticating moving queries because they focus on *static* queries, so that the authentication targets (i.e., the query results) are part of the dataset. In contrast, authentication of moving queries require authenticating both the query results (part of the dataset) and the safe regions (*not* part of the dataset).

Moving query processing In moving query processing, [19] computes the nearest neighbors for each possible query point on a given line segment, whereas [3] studies how to maintain the user's future kNN upon a change of the user's velocity. Both [19] and [3] model the user's movement as a linear function. When the user's future movement is unknown, the buffering approach [18] or the safe region approach [9, 25] are more appropriate for efficient moving query processing.

In the buffering approach [18], the LBS retrieves the user's $(k + \Delta k)$NN and uses them to derive a buffer region for the user. While moving within the buffer region, the client's latest kNN result can be recomputed locally from the $(k + \Delta k)$NN result of the previous query. However, it is not easy to find the optimal value of Δk in practice, and that value actually strongly influences the communication frequency and the number of objects transferred per communication.

In the safe region approach, the LBS reports a safe region [9, 25] for the query result, such that the result remains unchanged as long as the user moves within the safe region. Unlike the buffering approach, this approach does not require the client to compute result locally. This approach not only reduces the communication cost between the LBS and the client, but also the computational cost. In this paper, we adopt the order-k Voronoi cell [25] as the safe region for moving kNN queries. We are aware of another safe region technique called V*-diagram [9], which formulates an advanced safe region by fetching $(k + \Delta k)$ nearest neighbors. However, the optimal value for Δk is not easy to tune because it depends on the data distribution and the query parameters.

III. The Framework

Following [9, 25], we use a more general problem setting in which q is a moving object whose future locations cannot be predicted in advance. In a moving query environment, the question is when and where the client should (re-)issue query in order to get the most updated kNN result as q moves. A brute-force method is by sampling, in which the client

periodically issues a kNN query to the server for every T time units. The correctness of the results can then be authenticated using an MR-tree. However, the problem of this method is that a high T makes the result stale while a small T leads to very high communication cost. More problematically, there is no way to guarantee that result is always up-to-date even when a very small T is used, rendering this method impractical.

Baseline Method A baseline method is to use the *buffering technique* [18] to compute kNN queries and then authenticate the results by using an MR-tree [22]. Specifically, a client issues $(k + \Delta k)$NN queries to the LBS. Let q_{last} be the last location sent to server and d_i be the distance between q_{last} and its i-th nearest neighbor. The *buffer region* is a circle $\odot(q_{last}, \epsilon)$ with q_{last} as the center and $\epsilon = (d_{k+\Delta k} - d_k)/2$ as the radius. It is proven in [18] that, as long as the current location of q moves within the buffer region $\odot(q_{last}, \epsilon)$, the latest kNN result can be derived from the $(k + \Delta k)$NN result of the last query. The client is required to issue a new $(k+\Delta k)$NN query only when it moves outside $\odot(q_{last}, \epsilon)$. Although simple, finding the optimal value of Δk for this method is very difficult in practice. A small Δk leads to more frequent communication, thereby increasing the communication cost. A large Δk, unfortunately, also increases the communication cost because of the size of the $\mathcal{VO}$ increases.

Safe Region Approach The buffering technique in the baseline method is not a truly safe region approach because even when staying within the buffer region the client is still required to recompute result locally. Our approach is to return a safe region [9, 25] for the query result, such that the result remains unchanged as long as the user's location q stays within the safe region. This approach helps reducing the communication frequency between the LBS and the client, as well as their computation cost. However, for the sake of saving computing resources and communication bandwidth [12, 17, 24], the LBS may simply return, in our problem context, an overly large safe region, rendering the user's future result incorrect. Our goal is thus to devise methods for the client to verify the correctness of the safe region returned by LBS.

Figure 2 illustrates our framework for answering moving kNN that supports query correctness verification. A map provider (e.g., the government's land department, NAVTEQ[1], and TeleAtlas[2]) collects points-of-interests into a spatial dataset. It builds the ADS[3] of the dataset and signs the digest of the root node, before distributing/selling it to a service provider (i.e., LBS). Initially, a user downloads the root signature from the LBS and the map provider's public key from a certificate authority (e.g., VeriSign). Afterwards, the user sends its location to the LBS, and obtains as the results the kNN, safe region, and the $\mathcal{VO}$. The correctness of the kNN and the safe region can be verified at the client by using the received $\mathcal{VO}$, the root signature and the map provider's public key. The client needs to issue a query to the LBS again only when it leaves its safe region.

The spatial dataset is expected to have infrequent updates (e.g., monthly map updates). In case the user wants to ensure the results obtained are *fresh* (i.e., obtained from the latest datasets), the map provider could follow [6] to include a timestamp in the root signature of the tree.

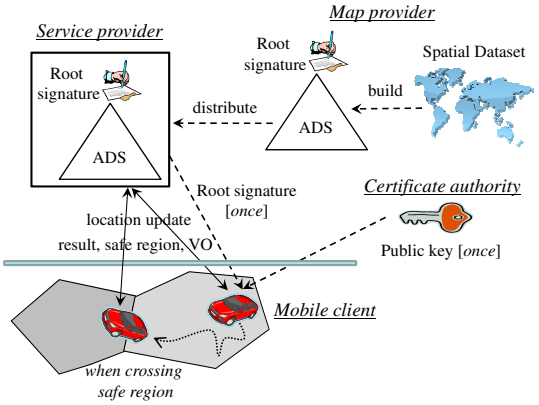


Fig. 2. Moving kNN authentication

IV. Safe Region Authentication

In this paper, we focus on the 2-dimensional space $\mathbb{R}^2$ because most spatial data reside in such a space. Nevertheless, our result can be extended to the 3-dimensional space. We abbreviate the minimum bounding rectangle $e.r$ by its non-leaf entry e. A summary of notation used in this paper is given in Table I.

TABLE I
Summary of Frequently Used Symbols

Symbol	Meaning
D	the dataset of points
p	a data point of D
q	the (current) query point
k	number of nearest neighbors
S	a subset of D with k points / kNN result
γ	the kNN distance
$\odot(q, a)$	circular region with center q and radius a
$dist(p, p')$	distance between points p and p'
$\perp(p, p')$	the half-plane closer to point p than point p'
$\mathcal{V}(p, D)$	Voronoi cell of p with respect to D
$G(p, D)$	the generator set of $\mathcal{V}(p, D)$
$\mathcal{V}_k(S, D)$	order-k Voronoi cell of S with respect to D
Ψ	vertices of $\mathcal{V}_k(S, D)$
e	a non-leaf entry in an MR-tree
$mindist(e, q)$	the minimum distance between q and the extent of e
KVR	the kNN verification region
SRVR	the safe region verification region

We adopt the (order-k) Voronoi cell as the *safe region* for moving kNN queries as in [25]. The basic definitions for Voronoi cells are as follows.

Definition 1: **Half-plane [10].**
Given points p and p', the *half-plane* $\perp(p, p')$ denotes the set of locations that are closer to point p than point p':

$$\perp(p, p') = \{z \in \mathbb{R}^2 \mid dist(z, p) \leq dist(z, p')\}$$

Definition 2: **Voronoi cell [10].**
Given a point p from a point set D, the *Voronoi cell* $\mathcal{V}(p, D)$

[1]NAVTEQ Maps and Traffic. http://www.navteq.com
[2]TeleAtlas Digital Mapping. http://www.teleatlas.com
[3]We use MR-tree [22] and a new ADS called Voronoi MR-tree (Section IV-B) in this paper.

of p with respect to D is defined as:

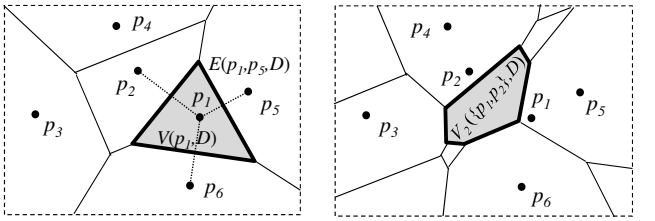
$$\mathcal{V}(p, D) = \bigcap_{p' \in D \setminus \{p\}} \perp(p, p')$$

Definition 3: **Voronoi edge.**
For two adjacent Voronoi cells $\mathcal{V}(p, D)$ and $\mathcal{V}(p', D)$, their Voronoi edge is defined as their shared line segment:

$$E(p, p', D) = \{z \in \mathbb{R}^2 \mid z \in \mathcal{V}(p, D) \wedge z \in \mathcal{V}(p', D)\}$$

Definition 4: **Generator and symmetric property.**
A point p' is a *generator* of $\mathcal{V}(p, D)$ if $E(p, p', D)$ is non-empty. Let $G(p, D)$ be the set of all generators of $\mathcal{V}(p, D)$. Since $E(p, p', D) = E(p', p, D)$, we obtain: $p' \in G(p, D)$ iff $p \in G(p', D)$.

Figure 3a illustrates the Voronoi cell of each point from the dataset $D = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. The Voronoi cell $\mathcal{V}(p_1, D)$ of p_1 is shown as the triangle with three bold edges, which can be represented by point p_1 and its generator set $G(p_1, D) = \{p_2, p_5, p_6\}$. Observe that Voronoi cells of points in $G(p_1, D)$ share common edges with $\mathcal{V}(p_1, D)$.



(a) Voronoi cells (b) order-k Voronoi cells, at $k = 2$

Fig. 3. Example of Voronoi cells

Definition 5: **Order-k Voronoi cell [10].**
Given a subset $S \subset D$ such that $|S| = k$, the order-k Voronoi cell $\mathcal{V}_k(S, D)$ of S with respect to D is defined as:

$$\begin{aligned} \mathcal{V}_k(S, D) &= \{z \in \mathbb{R}^2 \mid \max_{p \in S} dist(z, p) \leq \min_{p' \in D \setminus S} dist(z, p')\} \\ &= \bigcap_{p \in S} \Big(\bigcap_{p' \in D \setminus S} \perp(p, p') \Big) \end{aligned}$$

Figure 3b depicts the order-2 Voronoi cells on the dataset D. For instance, any location within the cell $V_2(\{p_1, p_2\}, D)$ regard p_1 and p_2 as its 2 nearest neighbors.

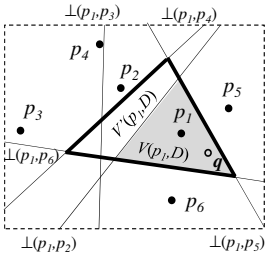


Fig. 4. Challenge of safe region authentication

The challenge of verifying the correctness of a safe region (a Voronoi cell) is illustrated in Figure 4. Suppose that the LBS needs to compute the 1NN of q (i.e., point p_1) and also the safe region (i.e., the Voronoi cell $\mathcal{V}(p_1, D)$; the gray triangle). Through Definition 4, the LBS can represent the Voronoi cell $\mathcal{V}(p_1, D)$ by point p_1 and its generator set: $G(p_1, D) = \{p_2, p_5, p_6\}$. The client can reconstruct $\mathcal{V}(p_1, D)$ by using p_1 and $G(p_1, D)$.

Suppose that the LBS intentionally represents $\mathcal{V}(p_1, D)$ using a fake generator set $G'(p_1, D) = \{p_4, p_5, p_6\}$. In this case, the client will reconstruct an incorrect Voronoi cell $\mathcal{V}'(p_1, D)$, as indicated by the bold triangle in the figure. The problem here is that the client cannot verify the correctness of the fake generator set $G'(p_1, D)$. Note that all points of $G'(p_1, D)$ also originate from the dataset D (thus it passes the data correctness checking that verifies the root signature). However, the client cannot determine whether those points belong to the actual generator set $G(p_1, D)$.

By Definition 2, $\mathcal{V}(p_1, D)$ is constructed by *all points* in D. So, a brute-force solution is to return the whole dataset D to the client so that he is guaranteed to compute $\mathcal{V}(p_1, D)$ correctly. Our goal is to design algorithms to construct verification objects $\mathcal{VO}$ for verifying the correctness of the Voronoi cell, yet the size of $\mathcal{VO}$ should be as small as possible.

In the following, we first present an approach that exploits the relationship between the vertices of an order-k Voronoi cell and the kNN to construct the $\mathcal{VO}$ (Section IV-A). The $\mathcal{VO}$s constructed by this approach are very compact and thus the approach is more communication efficient. Afterwards, we present another method that materializes order-1 Voronoi cells in the MR-tree off-line and constructs the $\mathcal{VO}$ on-the-fly (Section IV-B). The $\mathcal{VO}$ constructed by this approach may be larger than the ones constructed by the first approach; however, this approach is more computation efficient and thus more suitable for mobile devices with limited computational power and battery. In the end of this Section, we present a method to re-use the $\mathcal{VO}$ to reduce the communication cost when the client crosses a safe region (Section IV-C).

A. Vertex-based Approach (VA)

Our *Vertex-based Approach* (VA) constructs a compact $\mathcal{VO}$ for authenticating a moving kNN query. Its idea is to exploit the relationship between the vertices of an order-k Voronoi cell and the kNN result. It consists of a server algorithm and a client algorithm.

Algorithm 1 Vertex-based Approach (Server)

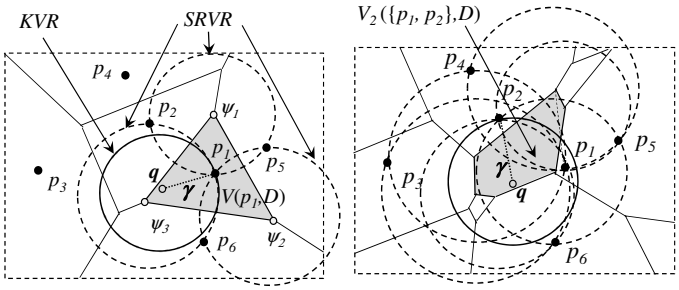
Receive from client: (Query point q, Integer k)
Using MR-tree T_D (on dataset D)
1: S:=compute the kNN of q from the tree T_D;
2: compute $\mathcal{V}_k(S, D)$ from the tree T_D (using [25]);
3: γ:=$\max_{p \in S} dist(q, p)$; ▷ authenticating kNN
4: KVR:=$\odot(q, \gamma)$;
5: Ψ := set of vertices of $\mathcal{V}_k(S, D)$; ▷ authenticating safe region
6: SRVR := $\bigcup_{\psi \in \Psi} \odot(\psi, \max_{p \in S} dist(\psi, p))$;
7: $\mathcal{C}$:= KVR $\cup$ SRVR;
8: $\mathcal{VO}$:=DepthFirstRangeSearch(T_D.root, $\mathcal{C}$);
9: **send** $\mathcal{VO}$ to the client;

Server Algorithm

Algorithm 1 is the pseudo-code of the server algorithm. Upon receiving the user location q and the number k of required

NNs, it computes the kNN result S from an MR-tree T_D (line 1).[4] Then, it computes the safe region as the order-k Voronoi cell $\mathcal{V}_k(S, D)$ (line 2). Next, it defines a verification region $\mathcal{C}$ so as to identify points that are directly useful for verifying the kNN results and the safe region and put them into the $\mathcal{VO}$. More specifically, the verification region $\mathcal{C}$ is the union of:

- The *kNN verification region* (KVR): the circular region $\odot(q, \gamma)$, where γ is the k-th nearest neighbor distance of q (lines 3–4).
- The *safe region verification region* (SRVR): the union of circular regions defined using each vertex ψ of the Voronoi cell and the farthest point in the kNN result S from ψ, i.e., $\bigcup_{\psi \in \Psi} \odot(\psi, \max_{p \in S} dist(\psi, p))$, where Ψ denotes the set of vertices of the Voronoi cell $\mathcal{V}_k(S, D)$ (lines 5–6).



(a) order-1 cell $\mathcal{V}(p_1, D)$ (b) order-2 cell $\mathcal{V}_2(\{p_1, p_2\}, D)$

Fig. 5. Verification region of order-k Voronoi cell

Figure 5a illustrates the verification region $\mathcal{C}$ for the case $k = 1$. The point p_1 is the NN of q and the Voronoi cell $\mathcal{V}(p_1, D)$ (in gray color) is the safe region. The KVR is the solid circle centered at q. The SRVR is the region formed by the union of the dotted circles centered at vertices ψ_1, ψ_2, ψ_3. The $\mathcal{VO}$ is then computed by a depth-first traversal of the MR-tree (line 8). That is, the tree is traversed with the following two conditions: (i) if a non-leaf entry e does not intersect $\mathcal{C}$, e is added to the $\mathcal{VO}$ and its subtree will not be visited; (ii) data points in any visited node (i.e., intersecting $\mathcal{C}$) are put into the $\mathcal{VO}$. Thus, in our example, the $\mathcal{VO}$ contains the points p_1, p_2, p_5 and p_6 and other non-leaf entries in the tree. Figure 5b illustrates the verification region $\mathcal{C}$ for the case $k = 2$. The points p_1 and p_2 are 2NN of q and the order-2 Voronoi cell $\mathcal{V}_2(\{p_1, p_2\}, D)$ (in gray color) is the safe region. In this example, the verification region $\mathcal{C}$ contains the points p_1, p_2, p_3, p_4, p_5, p_6 and other non-leaf entries in the tree.

We now prove that any point p^* outside the SRVR *cannot* alter the safe region $\mathcal{V}_k(S, D)$.

Theorem 1: **[Points outside the SRVR cannot alter safe region $\mathcal{V}_k(S, D)$]** Let Ψ be the set of vertices of the order-k Voronoi cell $\mathcal{V}_k(S, D)$, where S is a set of k points from the point set D. Let p^* be a point of $D \backslash S$. If $p^* \notin \bigcup_{\psi \in \Psi} \odot(\psi, \max_{p \in S} dist(\psi, p))$, then $\mathcal{V}_k(S, D) = \mathcal{V}_k(S, D \backslash \{p^*\})$. □

[4]Other tree-based spatial ADS, such as MR*-tree [22], are also applicable here.

Proof: To begin, we first have to state down two preliminary lemmas.

Lemma 1: Let S be a set of k points from D. Let p^* be a point of $D \backslash S$. We have: $\mathcal{V}_k(S, D) = \mathcal{V}_k(S, D \backslash \{p^*\}) \cap \bigcap_{p \in S} \perp(p, p^*)$. □

Proof:

$$\begin{aligned}
&\mathcal{V}_k(S, D) \\
&= \bigcap_{p \in S} \Big(\bigcap_{p' \in D \backslash S} \perp(p, p') \Big) \\
&= \bigcap_{p \in S} \Big(\Big(\bigcap_{p' \in (D \backslash \{p^*\}) \backslash S} \perp(p, p') \Big) \cap \perp(p, p^*) \Big) \\
&= \bigcap_{p \in S} \Big(\bigcap_{p' \in (D \backslash \{p^*\}) \backslash S} \perp(p, p') \Big) \cap \bigcap_{p \in S} \perp(p, p^*) \\
&= \mathcal{V}_k(S, D \backslash \{p^*\}) \cap \bigcap_{p \in S} \perp(p, p^*)
\end{aligned}$$

■

Lemma 2: A half-plane $\perp(\cdot, \cdot)$ is convex. The intersection of half-planes is also convex [10].□

We now show that, with the given if-condition in Theorem 1, both $\mathcal{V}_k(S, D)$ and $\mathcal{V}_k(S, D \backslash \{p^*\})$ are strictly inside $\bigcap_{p \in S} \perp(p, p^*)$, and thus they are equal.

Let's consider the region $\bigcap_{p \in S} \perp(p, p^*)$. Any location $z \in \mathbb{R}^2$ can be classified into three types:

- *z_{in}, which is strictly inside $\bigcap_{p \in S} \perp(p, p^*)$, i.e., $\forall p \in S, dist(z_{in}, p) < dist(z_{in}, p^*)$*
- *z_{bor}, which is on the border of $\bigcap_{p \in S} \perp(p, p^*)$, i.e., $\forall p \in S, dist(z_{bor}, p) \leq dist(z_{bor}, p^*)$ and $\exists p \in S, dist(z_{bor}, p) = dist(z_{bor}, p^*)$*
- *z_{out}, which is outside $\bigcap_{p \in S} \perp(p, p^*)$, i.e., $\exists p \in S, dist(z_{out}, p) > dist(z_{out}, p^*)$*

Note that $z_{in}, z_{bor} \in \bigcap_{p \in S} \perp(p, p^*)$, but $z_{out} \notin \bigcap_{p \in S} \perp(p, p^*)$.

From the given if-condition, we obtain: $\forall \psi \in \Psi, dist(\psi, p^*) > \max_{p \in S} dist(\psi, p)$. By rearranging it, we have: $\forall \psi \in \Psi, \forall p \in S, dist(\psi, p) < dist(\psi, p^*)$. Thus, each vertex ψ (of the cell $\mathcal{V}_k(S, D)$) is strictly inside $\bigcap_{p \in S} \perp(p, p^*)$. Combining this with the fact that both $\mathcal{V}_k(S, D)$ and $\bigcap_{p \in S} \perp(p, p^*)$ are convex regions (refer to Lemma 2), we infer that $\mathcal{V}_k(S, D)$ is strictly inside $\bigcap_{p \in S} \perp(p, p^*)$. $-(\bigstar)$

By Lemma 1, $\mathcal{V}_k(S, D) = \mathcal{V}_k(S, D \backslash \{p^*\}) \cap \bigcap_{p \in S} \perp(p, p^*)$. Clearly, any z_{in} that belongs to $\mathcal{V}_k(S, D \backslash \{p^*\})$ must also belong to $\mathcal{V}_k(S, D)$. To prove $\mathcal{V}_k(S, D) = \mathcal{V}_k(S, D \backslash \{p^*\})$, we proceed to show that z_{bor} and z_{out} do not exist in $\mathcal{V}_k(S, D \backslash \{p^*\})$.

Suppose that there exists a location z_{bor} in $\mathcal{V}_k(S, D \backslash \{p^*\})$. Thus, z_{bor} must also belong to $\mathcal{V}_k(S, D)$, by Lemma 1. This contradicts with the property $(\bigstar)$ that $\mathcal{V}_k(S, D)$ is strictly inside $\bigcap_{p \in S} \perp(p, p^*)$.

Suppose that there exists a location z_{out} in $\mathcal{V}_k(S, D \backslash \{p^*\})$. We construct a line segment from z_{out} to a location z_{in} in $\mathcal{V}_k(S, D \backslash \{p^*\})$. This line segment must pass through a location z_{bor}, which falls in $\mathcal{V}_k(S, D \backslash \{p^*\})$ because it is

convex. This will then lead to contradiction, as stated above. Therefore, Theorem 1 is proved. ■

For example, in Figure 5a, points p_3 and p_4 fall outside SRVR. By Theorem 1, they cannot alter $\mathcal{V}(p_1, D)$ and so they are not included in the $\mathcal{VO}$. On the other hand, points p_2, p_5 and p_6 are the generators of the order-1 Voronoi cell $\mathcal{V}(p_1, D)$. They are essential for constructing $\mathcal{V}(p_1, D)$, so they are in SRVR and included in the $\mathcal{VO}$. By Theorem 1, it is safe to include only those points within $\mathcal{C}$ in the $\mathcal{VO}$ (and other points are represented by a few non-leaf entries). Therefore, while the $\mathcal{VO}$ constructed by this method is very compact, it also provides all the information the client needs to verify the correctness of the safe region (and the kNN results).

Note that, for efficient implementation, the server does not need to render the complex shape of the verification region $\mathcal{C}$ (line 7). Instead, we can check whether the following condition holds during the tree traversal (line 8):

$$(mindist(e, q) \leq \gamma) \vee \bigvee_{\psi \in \Psi} \left(mindist(e, \psi) \leq \max_{p \in S} dist(\psi, p) \right) \quad (1)$$

where the first term checks whether a non-leaf entry e intersects KVR and the second term checks whether e intersects SRVR. If the condition holds, we add e into the $\mathcal{VO}$. Using this condition, the $\mathcal{VO}$ can be constructed efficiently. In the final step, the server sends the $\mathcal{VO}$ to the client (line 9) and the client will obtain the kNN result and the safe region from the $\mathcal{VO}$ and authenticate them.

Client Algorithm

Algorithm 2 is the pseudo-code of the client algorithm. Upon receiving the verification object $\mathcal{VO}$ from the server, it first reconstructs the root digest from the $\mathcal{VO}$ and verifies it against the MR-tree root signature signed by the data owner (lines 1–2). If the verification is successful, then the $\mathcal{VO}$ is guaranteed to contain only entries from the original MR-tree (i.e., no fake entries). Next, it proceeds to verify the correctness of the kNN result and the safe region provided by the $\mathcal{VO}$. It extracts from the $\mathcal{VO}$ (i) a set D' of data points, and (ii) a set R' of non-leaf entries, and then computes the kNN result S' from D' (lines 4–6).

As described in Section II, the kNN set S' is correct if every non-leaf entry of R' does not intersect $\odot(q, \gamma')$, where γ' is the kNN distance (lines 7–8). If the kNN result is authenticated, the client next computes the order-k Voronoi cell $\mathcal{V} = \mathcal{V}_k(S', D')$ from D' as the safe region for the kNN result S' (line 9). It then constructs the SRVR by the set of vertices Ψ in $\mathcal{V}$ (lines 10–11). By Theorem 1, the client can ensure that the cell $\mathcal{V}$ is correct if every non-leaf entry of R' does not intersect the SRVR (line 12). And if so, the client can regard the computed kNN result and the safe region $\mathcal{V}$ as correct.

$\mathcal{VO}$ Size Analysis

We estimate the size of the $\mathcal{VO}$ by estimating the number of data points in the verification region $\mathcal{C} = \odot(q, \gamma) \cup \bigcup_{\psi \in \Psi} \odot(\psi, \max_{p \in S} dist(\psi, p))$. We will ignore $\odot(q, \gamma)$ because it is always covered by $\bigcup_{\psi \in \Psi} \odot(\psi, \max_{p \in S} dist(\psi, p))$.

Algorithm 2 Vertex-based Approach (Client)

Receive from server: (Verification Object $\mathcal{VO}$)
1: h'_{root}:=reconstruct the root digest from $\mathcal{VO}$;
2: verify h'_{root} against the MR-tree root signature;
3: **if** h'_{root} is correct **then**
4: D':=the set of data points extracted from $\mathcal{VO}$;
5: R':=the set of non-leaf entries extracted from $\mathcal{VO}$;
6: S':=compute the kNN of q from D';
7: γ':=$\max_{p \in S'} dist(q, p)$;
8: **if** $\forall e \in R',\ e \cap \odot(q, \gamma') = \emptyset$ **then** ▷ authenticate kNN
9: $\mathcal{V}$:=compute $\mathcal{V}_k(S', D')$;
10: Ψ := set vertices of $\mathcal{V}$;
11: SRVR := $\bigcup_{\psi \in \Psi} \odot(\psi, \max_{p \in S'} dist(\psi, p))$;
12: **if** $\forall e \in R', e \cap$ SRVR = $\emptyset$ **then** ▷ authenticate safe region
13: **return** kNN result S' and safe region $\mathcal{V}$;
14: **return** authentication failed;

For simplicity, we assume that the data points are uniformly distributed in the square domain space $[0, 1]^2$. Let n be the number of points in the dataset D. The average kNN distance is: $\gamma = \sqrt{\frac{k}{\pi n}}$. According to [10], the number of order-k Voronoi cells is: $F(k, n) = (2k - 1)n - (k^2 - 1)$. Thus, the average area of an order-k Voronoi cell is: $\frac{1}{F(k,n)}$.

Under uniform distribution, we approximate the shape of an order-k Voronoi cell $\mathcal{V}_k(S, D)$ by a circle with radius λ. Thus, we have: $\lambda = \sqrt{\frac{1}{\pi F(k,n)}}$. Recall that the verification region $\mathcal{C}$ is formed by a union of circles, in which each circle has its radius as γ and its center as a vertex of $\mathcal{V}_k(S, D)$.

The verification region $\mathcal{C}$ can be approximated as a circle with the radius $\lambda + \gamma$. Therefore, the number of data points in $\mathcal{C}$ is estimated as: $n \cdot \pi(\lambda + \gamma)^2 = \left(\sqrt{\frac{n}{F(k,n)}} + \sqrt{k} \right)^2 \approx \left(\sqrt{\frac{1}{2k-1}} + \sqrt{k} \right)^2 \leq k + 3$. The number of non-leaf entries in the $\mathcal{VO}$ can be estimated by substituting the above circle into the MR-tree analysis in [22].

Observe that the above estimated value is robust for uniform data distribution. In case of other data distributions, different vertices of $\mathcal{V}_k(S, D)$ could be located in different parts of the space, with different densities. We leave the analysis of $\mathcal{VO}$ size under other distributions as our future work.

Discussion

An advantage of this Vertex-based Approach (VA) is that it exploits the property between the vertices of order-k Voronoi cell and the kNN results such that it only adds very few data points to the verification object. That leads to a low communication cost per query while the use of safe region reduces the number of queries issued. Also, this approach does not require the building of a new authenticated data structure (ADS) to authenticate the Voronoi cells; rather, it reuses the existing MR-tree, which is useful for authenticating other types of spatial queries. However, this method requires the online computation of an order-k Voronoi cell on both the server-

side and the client-side at run-time.

B. Materialization Approach (MA)

We now present an alternate approach that materializes some Voronoi cell information in the ADS so as to save computational cost at the server and mobile clients. The challenge is that the value of k is a user parameter and it cannot be known in advance. Therefore, pre-computing order-k Voronoi diagrams for all possible values of k and materializing them in the MR-tree is not practical—that places a very heavy burden on the data owner and makes the MR-tree bulky and inefficient.

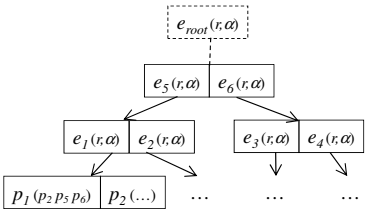
Instead of using full materialization, our *Materialization Approach* (MA) is a client-server method that operates on a simple extension of VoR-tree [16], which we call Voronoi MR-tree. We will show that by using the Voronoi MR-tree, the safe region verification region (SRVR) is identical to the kNN verification region (KVR). Therefore, the clients and the LBS need not explicitly authenticate the safe region (order-k Voronoi cells). Also, we will prove that points added to the $\mathcal{VO}$ based on the Voronoi MR-tree are sufficient for the client to derive the safe region from the $\mathcal{VO}$ directly. In other words, the server needs not even compute the safe region. All these enhancements dramatically speed-up the server and client algorithms.

Voronoi MR-tree

The Voronoi MR-tree is a simple "Merkle extension" of VoR-tree [16]. VoR-tree is an R-tree like index structure. In a VoR-tree, each leaf entry stores not only a point p_i but also the generator set $G(p_i, D)$ of its order-1 Voronoi cell. However, the MBR of a leaf node is solely constituted by the entries' data points. The Voronoi MR-tree is simply a VoR-tree with digests added to each node recursively like MR-tree (see Section II). The signature of the dataset is obtained by signing the root digest with the map provider's private key. For instance, Figure 6a shows a Voronoi MR-tree ($T_D^\sharp$) built from the data points shown in Figure 3a. The leaf entry for point p_1 stores the binary string of p_1 and its generators $(p_2\ p_5\ p_6)$.

Maintaining (i.e., insert/delete/update of data points) a Voronoi MR-tree is the same as maintaining a VoR-tree (see [16]) followed by maintaining the digests as in MR-tree (see [22]), so we do not discuss it in detail. Maintaining a Voronoi MR-tree is asymptotically as efficient as maintaining an R-tree because the average number of generators of each Voronoi cell (in $\mathbb{R}^2$) is at most six [16], which is a constant. The average capacity of the leaf nodes of VoR-tree and Voronoi MR-tree is seven (1+6) times lower than that of the MR-tree on average but the capacity of non-leaf nodes in the VoR-tree and Voronoi MR-tree is still the same as in the MR-tree. As the capacity of non-leaf nodes is usually in the order of hundred, the height of Voronoi MR-tree is usually the same or one level higher than that of MR-tree.

The Voronoi MR-tree is simply a blend of MR-tree and VoR-tree. Actually, we put our focus on *how to compute and authenticate order-k Voronoi cells using Voronoi MR-tree, which contains only order-1 Voronoi cells.*[5]



(a) Voronoi MR-tree $T_D^\sharp$ (b) Compression dictionary

Fig. 6. Voronoi MR-tree

Server Algorithm

Algorithm 3 is the pseudo-code of the server algorithm. Upon receiving the user location q and the number k of required NNs, it computes the kNN result from a Voronoi MR-tree $T_D^\sharp$ (line 1). Next, it defines a verification region $\mathcal{C}$ so as to identify points that are directly useful for the client to verify the kNN results and the safe regions. Differ from the vertex-based approach, the verification region $\mathcal{C}$ consists of only one part, which is surprisingly the usual kNN verification region (KVR), i.e., $\mathcal{C} = \odot(q, \gamma)$, where γ is the k-th nearest neighbor distance of q (lines 2–3).

With a Voronoi MR-tree, we now prove that by putting all points within the verification region $\mathcal{C}$ (and their associated order-1 Voronoi cell generators) into the $\mathcal{VO}$, the client can compute the safe region, i.e., order-k Voronoi cell for the kNN result, merely from the points in the $\mathcal{VO}$:

Theorem 2: **[Correctness of the verification region computed from Voronoi MR-tree]**

Let S be a set of k points from the point set D, q be a query point, and $\gamma = \max_{p\in S} dist(q,p)$. The order-$k$ Voronoi cell $\mathcal{V}_k(S, D)$ can be computed from points in Voronoi MR-tree that fall into $\odot(q, \gamma)$. □

Proof: We first establish the following lemma.

Lemma 3: Let S be a subset of the point set D. Let p^* be a point in the set D but not in the set union $D_S^* = \cup_{p\in S} G(p, D)$. We have: $\mathcal{V}_k(S, D) = \mathcal{V}_k(S, D\backslash\{p^*\})$. □

Proof: We first attempt to prove: $G(p^*, D) \cap S = \emptyset$. For the sake of contradiction, assume that there exists a point $p \in S$ such that $p \in G(p^*, D)$. By the symmetry property in Definition 4, we derive: $p^* \in G(p, D)$, and thus: $p^* \in \cup_{p\in S} G(p, D)$. This contradicts with the initial condition of p^*. Thus, we obtain: $G(p^*, D) \cap S = \emptyset$.
Since $G(p^*, D) \cap S = \emptyset$, we obtain: $G(p^*, D) = G(p^*, D\backslash S)$. Let p be any point of S. We have $p \notin G(p^*, D\backslash S)$. By the symmetric property in Definition 4, we have: $p^* \notin G(p, D\backslash S)$. Therefore, we obtain: $\mathcal{V}(p, D\backslash S) = \mathcal{V}(p, (D\backslash\{p^*\})\backslash S)$. Thus, we derive: $\bigcap_{p\in S} \mathcal{V}(p, D\backslash S) = \bigcap_{p\in S} \mathcal{V}(p, (D\backslash\{p^*\})\backslash S)$. —(★1)

(Continued on next page)

[5] We remark that [16] has not studied the computation nor the authentication of order-k Voronoi cells from order-1 Voronoi cells.

Finally, we attempt to prove: $\mathcal{V}_k(S,D) = \mathcal{V}_k(S,D\backslash\{p^*\})$. *By Definition 5, we have:* $\mathcal{V}_k(S,D) = \bigcap_{p\in S}\,(\bigcap_{p'\in D\backslash S} \perp(p,p'))$, *which is equal to* $\bigcap_{p\in S}\mathcal{V}(p,D\backslash S)$, *by using Definition 2. Therefore, we obtain:* $\mathcal{V}_k(S,D) = \bigcap_{p\in S}\mathcal{V}(p,D\backslash S)$ —(★2).
Then, by replacing D *with* $(D\backslash\{p^*\})$ *in equation* (★2), *we obtain:* $\mathcal{V}_k(S,(D\backslash\{p^*\})) = \bigcap_{p\in S}\mathcal{V}(p,(D\backslash\{p^*\})\backslash S)$. —(★3).
By combining the equations (★1), (★2), *and* (★3), *we have:* $\mathcal{V}_k(S,D) = \mathcal{V}_k(S,D\backslash\{p^*\})$.

We then establish the proof of Theorem 2 as follows. The circular region $\odot(q,\gamma)$ *contains each point satisfying* $dist(q,p) \le \gamma$. *Thus, it contains each point from* S. *From the Voronoi MR-tree, we can fetch the generator set* $G(p,D)$ *for each point in* S. *These generator sets are correct, as they were pre-computed by the data owner, by definition. Thus, the union set* $D^*_S = \cup_{p\in S}\, G(p,D)$ *can be obtained correctly. Let* p^* *be a point in the set* D *but not in the set union* D^*_S. *By repeating Lemma 3 and exhausting each point* p^* *in* D *but not in* D^*_S, *we have:* $\mathcal{V}_k(S,D) = \mathcal{V}_k(S,D^*_S)$, *which can be computed correctly as we have the correct* D^*_S. ■

By applying Theorem 2, the remaining part of the server algorithm is pretty simple. It simply traverses the Voronoi MR-tree as what the vertex-based approach does and puts (i) non-leaf entries that do not intersect $\mathcal{C}$ and (ii) points in visited leaf entries into the $\mathcal{VO}$ (line 4). As a final step, the server sends the $\mathcal{VO}$ to the client (line 5) and the client will be able to derive the kNN results and the safe region from the $\mathcal{VO}$ and authenticate them.

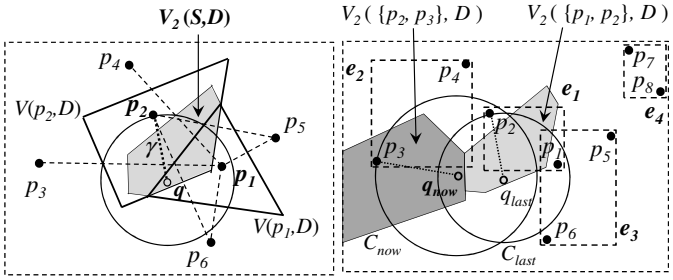
Figure 7a illustrates an example for the case $k = 2$. The query point q has its 2NN set as $S = \{p_1, p_2\}$. The verification region $\mathcal{C}$ is simply the circular region $\odot(q,\gamma)$, which contains the points p_1 and p_2. In the tree $T^\sharp_D$, the leaf entries of p_1 and p_2 store their corresponding generator sets: $G(p_1,D) = \{p_2,p_5,p_6\}$ and $G(p_2,D) = \{p_1,p_3,p_4,p_5,p_6\}$. They are also added into the $\mathcal{VO}$ when the leaf entries of p_1 and p_2 are inserted into the $\mathcal{VO}$.

Algorithm 3 Materialization Approach (Server)

Receive from client: (Query point q, Integer k)
Using Voronoi MR-tree $T^\sharp_D$ (on dataset D)
1: S:=compute the kNN of q from the tree T^*_D;
2: γ:=$\max_{p\in S}\, dist(q,p)$; ▷ start preparing the $\mathcal{VO}$
3: $\mathcal{C}$:=$\odot(q,\gamma)$;
4: $\mathcal{VO}$:=DepthFirstRangeSearch(T^*_D.root, $\mathcal{C}$);
5: **send** $\mathcal{VO}$ to the client;

Client Algorithm

Algorithm 4 is the pseudo-code of the client algorithm. The first few lines (lines 1–3) are identical to the client algorithm of the vertex-based approach (Algorithm 2). However, as this approach is based on a Voronoi MR-tree, the successful verification of the root digest implies both the data points and the order-1 Voronoi cells are originated from the data owner's Voronoi MR-tree (i.e., no fake entries). Next, it proceeds to verify the correctness of the kNN result and the safe region provided by the $\mathcal{VO}$. Same as Algorithm 2, it extracts from the $\mathcal{VO}$ (i) a set D' of data points and (ii) a set R' of non-leaf entries, then computes the kNN result S' and the kNN distance γ' from D' (lines 4–7).



(a) $\mathcal{VO}$ derivation (b) $\mathcal{VO}$ reuse

Fig. 7. Deriving an order-k cell from order-1 cells, $k = 2$

Recall from Theorem 2 that, given a query point q and a set S of k points, an order-k Voronoi cell can be computed from points in a Voronoi MR-tree that intersect $\odot(q,\gamma)$, where γ=$\max_{p\in S}\ dist(q,p)$. Thus, the client first defines its own circular verification region $\odot(q,\gamma')$. Next, it checks whether all non-leaf entries in R' do not intersect $\odot(q,\gamma')$ (line 8). If yes, it immediately satisfies the kNN authentication requirement (that we have encountered before; e.g., Section II, or line 8 of Algorithm 2). More interestingly, Theorem 2 is applicable to the client after it defines its own circular verification region $\odot(q,\gamma')$. Specifically, by ensuring that no non-leaf entry in R' intersects $\odot(q,\gamma')$, Theorem 2 guarantees that we can compute the order-k Voronoi cell using only the points in the $\mathcal{VO}$. The client then exploits the following lemma to collect points that are required to construct the order-k Voronoi cell for the kNN result S':

Lemma 4: Let S be a subset of the point set D. Let $D^*_S = \cup_{p\in S}\, G(p,D)$. We have: $\mathcal{V}_k(S,D) = \mathcal{V}_k(S,D^*_S)$. □

Proof: Let p^* *be a point in the set* D *but not in the set union* D^*_S. *By repeating Lemma 3 and exhausting each point* p^* *in* D *but not in* D^*_S, *we have:* $\mathcal{V}_k(S,D) = \mathcal{V}_k(S,D^*_S)$, *which can be computed correctly as we have the correct* D^*_S. ■

This lemma states that an order-k Voronoi cell $V_k(S,D)$ of a set S of points can be computed from their order-1 Voronoi cells if they are represented as generator points. Therefore, the client collects a small set of points $D^*_{S'}$ by union the generator sets for each nearest neighbor in S' (line 9) and computes the order-k Voronoi cell from $D^*_{S'}$ as the safe region for the current kNN result (line 10).

Continuing with the example of Figure 7a (for the case $k = 2$), the client first computes the kNN set as $S = \{p_1, p_2\}$ and verifies its correctness. The client then extracts their generator sets: $G(p_1,D) = \{p_2,p_5,p_6\}$ and $G(p_2,D) = \{p_1,p_3,p_4,p_5,p_6\}$. The union set D^*_S of these generator sets is used to compute the order-2 Voronoi cell $\mathcal{V}_2(S,D)$ (the gray region; the dotted lines illustrate the formation of edges

of $\mathcal{V}_2(S, D)$).

Algorithm 4 Materialization Approach (Client)

Receive from server: (Verification Object $\mathcal{VO}$)
h'_{root}:=reconstruct the root digest from $\mathcal{VO}$;
verify h'_{root} against the MR-tree root signature;
if h'_{root} is correct **then**
  D':=the set of data points extracted from $\mathcal{VO}$;
  R':=the set of non-leaf entries extracted from $\mathcal{VO}$;
  S':=compute the kNN of q from D';
  γ':=$\max_{p \in S'} dist(q, p)$;
  if $\forall e \in R',\ e \cap \odot(q, \gamma') = \emptyset$ **then** ▷ authenticate both kNN and safe region
    $D^*_{S'}$:=$\bigcup_{p \in S'} G(p, D)$; ▷ extract generators from $\mathcal{VO}$
    $\mathcal{V}$:=compute $\mathcal{V}_k(S', D^*_{S'})$;
    return kNN result S' and safe region $\mathcal{V}$;
return authentication failed;

Optimizing the $\mathcal{VO}$ Size using Space-Efficient Tokens
When the verification object $\mathcal{VO}$ is computed by using the Voronoi MR-tree, the $\mathcal{VO}$ may contain duplicate points because a data point p can appear in the generator sets of different points. For instance, in Figure 3a, the point p_1 belongs to the generator sets of the points p_2, p_3, and p_5. Thus, p_1 may be stored multiple times in the $\mathcal{VO}$.

To reduce the size of the $\mathcal{VO}$, we propose to represent the data points using some space-efficient tokens. While the server is constructing the $\mathcal{VO}$, it computes a dictionary (see Figure 6b) to represent the data points it encounters. Each occurrence of a point (e.g., p_1) in the $\mathcal{VO}$ will be replaced by its corresponding token (e.g., j_1). The dictionary will be appended to the end of the $\mathcal{VO}$. When the client receives the $\mathcal{VO}$, those tokens will be replaced with the corresponding points from the dictionary in order to reconstruct the original $\mathcal{VO}$. The resulting space reduction can be quite significant because the number of data points in a $\mathcal{VO}$ is usually much smaller than the dataset cardinality. For a $\mathcal{VO}$ with B data points, each token can be represented by $\lceil \log_2 B \rceil$ bits. Therefore, a $\mathcal{VO}$ with 100 data points requires only 7 bits per token, which is much smaller than using 2×8 bytes = 128 bits per (2-dimensional) data point.

$\mathcal{VO}$ Size Analysis
We estimate the size of the $\mathcal{VO}$ by analyzing the number of points in the verification region $\mathcal{C} = \odot(q, \gamma)$. Observe that the number of points intersecting $\odot(q, \gamma)$ is exactly k, which is independent of the extent and vertices of $\mathcal{V}_k(S, D)$. In addition, the generator set of each such point will also be included into $\mathcal{VO}$. According to [10], the Voronoi cell of a point has 6 edges on average, and thus its generator set contains 6 points on average, regardless of data distribution. In summary, the total number of points (including duplicates) is estimated to be $(6 + 1)k = 7k$. The number of non-leaf entries in the $\mathcal{VO}$ can be estimated by substituting the above circle into the MR-tree analysis in [22].

Discussion
One advantage of this Materialization Approach (MA) is that, at query time, the server saves significant computational cost by skipping the order-k Voronoi cell computation. Furthermore, the client also saves computational cost because it computes the high order Voronoi cells by using only the generator sets of points in the kNN results but not other irrelevant points.

C. Handling Client Location Updates

Both VA and MA use order-k Voronoi cells $\mathcal{V}_k(S, D)$ as the safe region. As long as the client travels within the Voronoi cell, the kNN result set S remains valid so that there is no need to submit a new query (with updated client location q) to the server. When the client leaves the region $\mathcal{V}_k(S, D)$, the kNN result set changes so that the client needs to submit a new query to the server.

In this section, we discuss how the client can reuse the previously received $\mathcal{VO}$ to reduce the communication frequency between the server and the client even when the client leaves the safe region. This optimization technique is applicable to both VA and MA. For ease of discussion, we only show its example for MA.

Let's consider the example of Figure 7b, at $k = 2$. Assume the client is now located at q_{now} and its previous location is at q_{last}, which has its 2NN set as $\{p_1, p_2\}$ and its safe region as $\mathcal{V}_2(\{p_1, p_2\}, D)$ (the light-gray region). Note that, for q_{last}, its verification region is defined as the circle $\mathcal{C}_{last}$ (with center at q_{last}). Since $\mathcal{C}_{last}$ intersects the non-leaf entries e_1, e_2, e_3, their leaf nodes are visited and their data points are inserted into the verification object $\mathcal{VO}_{last}$. $\mathcal{C}_{last}$ does not intersect e_4, so e_4 is inserted into $\mathcal{VO}_{last}$, and the subtree of e_4 is not visited. Thus, $\mathcal{VO}_{last} = \{p_1, p_2, p_3, p_4, p_5, p_6, e_4\}$.

Later on, the client moves to a new location q_{now} outside the region $\mathcal{V}_2(\{p_1, p_2\}, D)$. Before sending a new query to the server, the client can run Algorithm 4 again using the previous $\mathcal{VO}_{last}$ but with the current client location q_{now}. First, the client attempts to find the kNN result set of q_{now} from $\mathcal{VO}_{last}$. The 2NN answer is: $\{p_2, p_3\}$ and its corresponding verification region is defined as the circle $\mathcal{C}_{now}$ (with center at q_{now}). Observe that $\mathcal{C}_{now}$ does not intersect with any non-leaf entries in $\mathcal{VO}_{last}$ (e.g., e_4) as well. Thus, the correctness of $\mathcal{VO}_{last}$ is maintained even for the new query location q_{now}. The client can then extract the generator sets of p_2 and p_3, and compute the new safe region as $\mathcal{V}_2(\{p_2, p_3\}, D)$ (the dark-gray region). This example illustrates how the client is able to refresh the safe region without issuing a new query to the server. By using this technique, the client needs to send a new query to the server only when the new verification region $\mathcal{C}_{now}$ intersects some non-leaf entry in $\mathcal{VO}_{last}$.

V. Experimental Study

We evaluate our methods using both synthetic and real datasets. We implemented all methods in C++ and used cryptographic functions in the Crypto++ library[6]. Experiments were run on a PC with Intel 2.83GHz Quad CPU and 4

[6]Crypto++ library: http://www.cryptopp.com/

Gbytes RAM. The page size of MR-trees and Voronoi MR-trees is set to 4 Kbytes. The two real datasets are NA (North America, 175K points)[7] and SF (San Francisco, 174K points) [2]. We also generated SYN (synthetic uniform) datasets for the scalability experiments. In our experiments, we normalize the datasets to the domain space $[0, 10000]^2$. By default, the number k of requested NNs is 10. Each query workload contains the trajectories of 500 objects generated by Brinkhoff's trajectory generator [2]. Each trajectory simulates an object running on a road network and has a location measurement record for its object at every timestamp; there are 100 timestamps in total. The default speed of an object along the route is 50 distance units per timestamp. A distance unit denotes 1 meter and the time between adjacent timestamps is 1 second.

Table II shows the building time of MR-trees and Voronoi MR-trees on the real datasets and synthetic datasets. For the Voronoi MR-tree, we also show the breakdown of its building time into: (i) the time for computing generators of order-1 Voronoi cell for each data point, and (ii) the time for constructing the tree structure. The building time of both trees scales well with the data size. The building time of Voronoi MR-trees is not sensitive to the data distribution and is within a small factor of 2 from that of MR-trees. The Voronoi MR-trees have the same tree height as the MR-trees in most cases and are one level higher than the MR-trees in only a few cases.

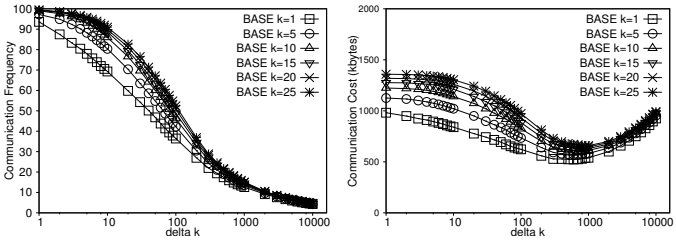
TABLE II
TREE BUILDING TIME

Dataset	**Building Time (seconds)**			
	MR-tree	Voronoi MR-tree		
		Total	*Generators*	*Tree*
SF	115.38	208.8	91.69	117.11
NA	124.05	214.45	89.44	125.01
SYN–50,000	30.11	49.94	19.51	30.43
SYN–100,000	63.11	104.43	40.32	64.11
SYN–200,000	128.80	215.36	84.74	130.62
SYN–500,000	334.30	572.04	234.13	337.91
SYN–1,000,000	732.92	1300.98	558.42	742.56

Following [9], we measure the cumulative total cost for each object's trajectory (which has 100 timestamps). In our experiments, we measure the *communication cost* (in kilobytes) as the total size of $\mathcal{VO}$, per object. We measure the *communication frequency* as the total number of clients' queries received by the server, per object. We also report the total *server* and *client CPU time* (in seconds), per object. We evaluate the performance of the vertex-based approach (VA) and the materialization approach (MA) based on above measures.

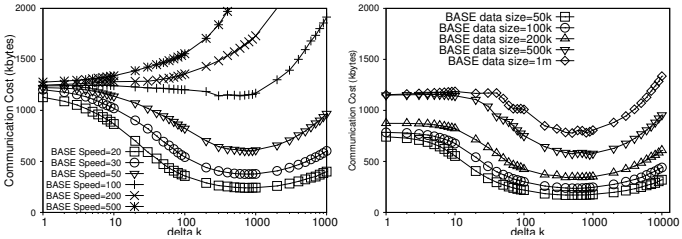
For comparison, we also implemented the baseline method (BASE) mentioned in Section III. We remark that BASE is an impractical method because its performance heavily depends on the Δk value, which in turns heavily depends on various factors such as k, query speed, data size, and data distribution. The optimal Δk for a particular setting cannot be found unless we exhaustively try every possible Δk value.

[7]www.maproom.psu.edu/dcw/



(a) total comm. frequency (b) total comm. cost (Kbytes)

Fig. 8. Tuning Δk for BASE, varying k, on SF data



(a) Varying query speed (SF) (b) Varying data size (SYN)

Fig. 9. Tuning Δk for BASE

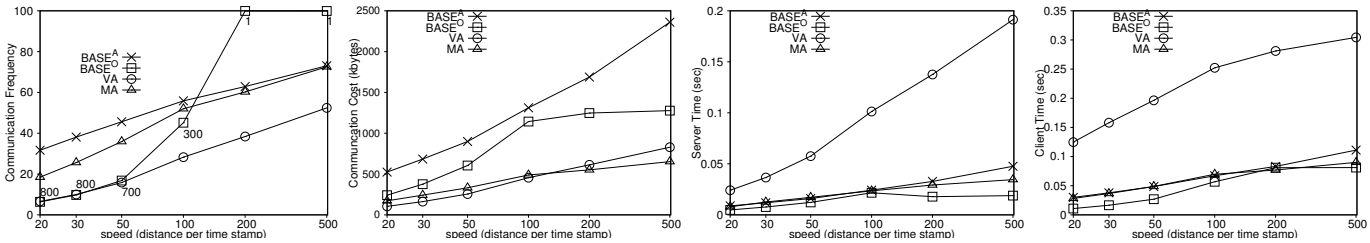
This tedious tuning process is impractical for typical mobile users. However, for comparison purpose, we do carry out such a tuning process for BASE in all our experiments. We use BASE^O to denote the best performance of BASE that uses the optimal Δk value. We use BASE^A to represent the average performance of BASE over every Δk value.

The following is an example of how we find the optimal Δk value for each given k value on the SF dataset. Figure 8a shows the communication frequency of BASE as a function of Δk, for six different values of k. As Δk increases, the buffer region of BASE becomes larger, and thus the communication frequency between the server and the client reduces. However, low communication frequency does not necessarily imply low communication cost. Figure 8b shows the communication cost of BASE with respect to Δk. When Δk is high, the size of $\mathcal{VO}$ (in bytes) per communication trip is high and thus the communication cost of BASE rises. When Δk is low, the communication frequency is high and so the communication cost of BASE also rises. Observe that different values of k lead to different optimal values of Δk, rendering it impossible to tune Δk in advance. For instance, the optimal Δk is 600 at $k = 1$ but the optimal Δk is 900 at $k = 25$. The tuning of Δk with respect to other parameters and datasets are done similarly. Due to limited space, we do not present all figures related to the tuning process. The tuning of Δk with respect to different query speeds on SF dataset, and the tuning of Δk with respect to different synthetic data sizes, are presented in Figure 9 for readers' reference.

Effect of query speed on real datasets

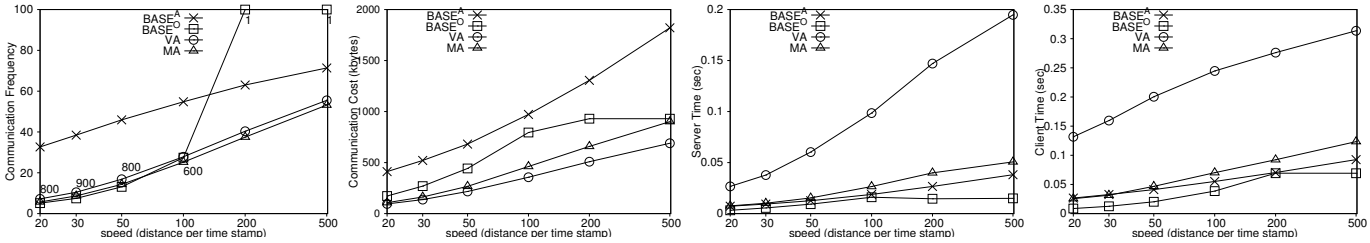
We study the effect of query speed on the performance of VA, MA, BASE^A, and BASE^O. Figures 10a,b,c,d show the communication frequency, communication cost, server CPU time, and client CPU time, with respect to different query speeds, per object, on the SF dataset.

In Figure 10a, the communication frequency of BASE^O is



(a) total comm. frequency (b) total comm. cost (Kbytes) (c) total server CPU time (sec) (d) total client CPU time (sec)

Fig. 10. Effect of query speed, on SF data



(a) total comm. frequency (b) total comm. cost (Kbytes) (c) total server CPU time (sec) (d) total client CPU time (sec)

Fig. 11. Effect of query speed, on NA data

annotated with the corresponding optimal Δk value found by our manual tuning. When the query speed increases, a BASE^O client leaves its buffer region sooner so its communication frequency increases and eventually reaches 100 (i.e., one query per timestamp). The communication frequency of VA and MA rises slower than that of BASE^O because they apply the safe region and the $\mathcal{VO}$ reusing technique to reduce the communication frequency. Thus, VA and MA incur lower communication cost than BASE^O (see Figure 10b). Observe that, without knowing the optimal Δk, the average performance of BASE^A is much worse than VA and MA. VA has slightly better communication cost than MA in most cases. MA's server and client CPU costs are lower than VA (see Figures 10c,d) because VA has to compute the order-k Voronoi cell $\mathcal{V}_k(S, D)$ at the server-side but MA does not. The client CPU cost depends on the number of comparisons for checking whether a non-leaf entry e in $\mathcal{VO}$ intersects the verification region $\mathcal{C}$. MA has its $\mathcal{C}$ as a circle, so it needs only one comparison to check if e intersects $\mathcal{C}$. In contrast, VA has its $\mathcal{C}$ as the union of $1 + |\Psi|$ circles, by Equation 1, where Ψ is the number of vertices of $\mathcal{V}_k(S, D)$. It requires at most $1 + |\Psi|$ distance comparisons.

The experimental results of varying query speed on NA dataset are presented in Figure 11; results similar to the SF data are observed.

Effect of k on real datasets

Figure 12 shows the effect of k on the performance of the methods. When k increases, the communication frequency of BASE^O appears to be constant, but that is contributed by giving it an increasing optimal Δk found by our manually-tuning. Since Δk increases, the $\mathcal{VO}$ size increases, and thus the communication cost of BASE^O increases. The average performance of BASE^A is much worse than BASE^O.

The communication cost of all methods rises when k increases, as more points are included into $\mathcal{VO}$ when k increases. Both VA and MA incur much lower communication cost than BASE^A and BASE^O. According to our analysis, the verification region $\mathcal{C}$ of VA and MA contains $k + 3$ and $7k$ points, respectively. It justifies why the communication cost of VA increases at a slower rate than MA. As explained before, MA has its $\mathcal{C}$ as a circle but VA has its $\mathcal{C}$ as the union of multiple circles. Therefore, the server and client CPU time of VA rise at a faster rate than MA.

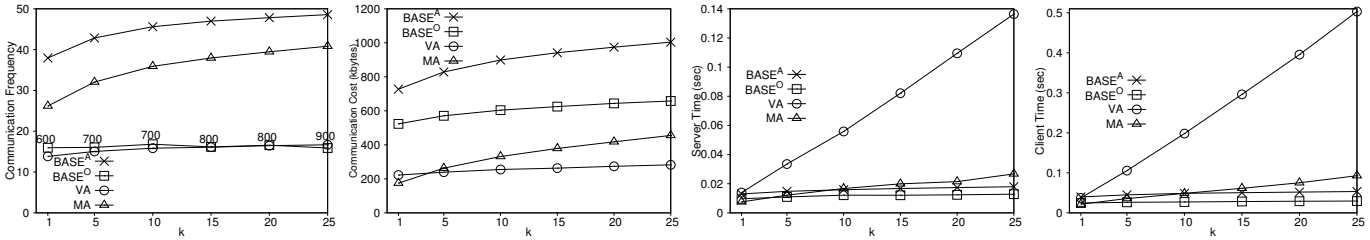
The experimental results of varying k on NA dataset are presented in Figure 13; results similar to the SF data are observed.

Effect of data size on (synthetic) SYN data

We then study the scalability of our methods, by using synthetic uniform datasets. Figure 14 shows the performance of the methods as a function of data size. When the data size increases, the data density rises. Thus, the buffering region in BASE^A/BASE^O and the area of a safe region in VA/MA shrink. Therefore, the communication frequency between the server and the client rises, causing increase in the communication cost, server CPU time, and client CPU time of all methods. Furthermore, the average performance of BASE^A is higher than MA and VA by 2.5 times and 4 times respectively, in terms of the communication cost.

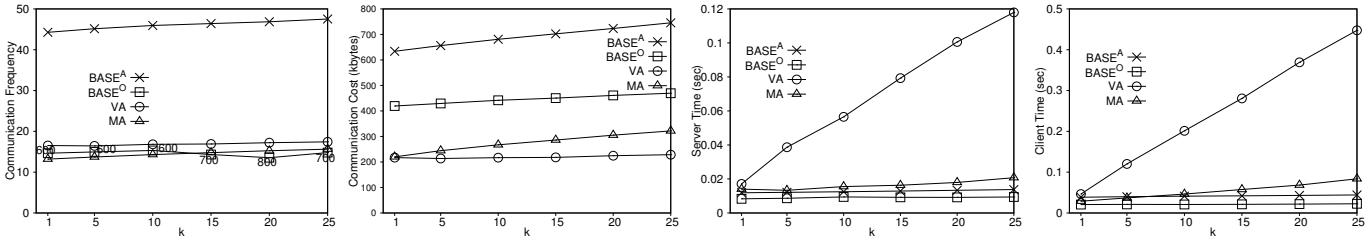
VI. Conclusion

In this paper, we have presented a framework with two efficient methods (VA and MA) for authenticating moving kNN queries using the safe region approach. We also develop an optimization technique for reducing the communication frequency when the client leaves a safe region. Experimental results show that our methods efficiently authenticate moving kNN queries using a small communication cost and outperform the baseline method by a wide margin. In general, VA has a better communication efficiency and MA has a better computation efficiency. Our future work is to study the authentication of other moving queries such as moving range queries.



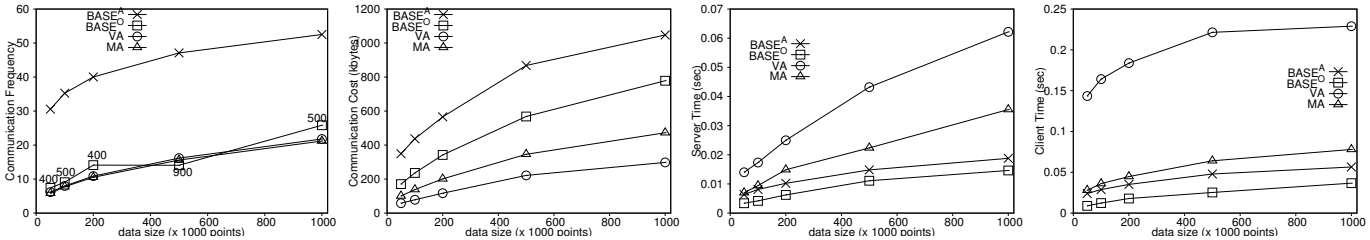
(a) total comm. frequency (b) total comm. cost (Kbytes) (c) total server CPU time (sec) (d) total client CPU time (sec)

Fig. 12. Effect of k, on SF data



(a) total comm. frequency (b) total comm. cost (Kbytes) (c) total server CPU time (sec) (d) total client CPU time (sec)

Fig. 13. Effect of k, on NA data



(a) total comm. frequency (b) total comm. cost (Kbytes) (c) total server CPU time (sec) (d) total client CPU time (sec)

Fig. 14. Effect of data size, on synthetic data

ACKNOWLEDGMENT

This work was supported by grant PolyU 5333/10E from Hong Kong RGC.

REFERENCES

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, 1990.

[2] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[3] G. S. Iwerks, H. Samet, and K. P. Smith. Maintenance of K-nn and Spatial Join Queries on Continuously Moving Points. *ACM TODS*, 31(2):485–536, 2006.

[4] A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. *PVLDB*, 1(1):138–150, 2008.

[5] A. Kundu and E. Bertino. How to Authenticate Graphs without Leaking. In *EDBT*, pages 609–620, 2010.

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

[7] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-Infused Streams: Enabling Authentication of Sliding Window Queries On Streams. In *VLDB*, pages 147–158, 2007.

[8] R. C. Merkle. A Certified Digital Signature. In *CRYPTO*, 1989.

[9] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. The V*-Diagram: A Query-dependent Approach to Moving KNN Queries. *PVLDB*, 1(1):1095–1106, 2008.

[10] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, second edition, 2000.

[11] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *SIGMOD*, 2005.

[12] H. Pang and K. Mouratidis. Authenticating the Query Results of Text Search Engines. *PVLDB*, 1(1):126–137, 2008.

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

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

[15] S. Papadopoulos, Y. Yang, and D. Papadias. CADS: Continuous Authentication on Data Streams. In *VLDB*, pages 135–146, 2007.

[16] M. Sharifzadeh and C. Shahabi. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *PVLDB*, 3(1):1231–1242, 2010.

[17] R. Sion. Query execution assurance for outsourced databases. In *VLDB*, pages 601–612, 2005.

[18] Z. Song and N. Roussopoulos. K-Nearest Neighbor Search for Moving Query Point. In *SSTD*, pages 79–96, 2001.

[19] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, pages 287–298, 2002.

[20] L. Wang, S. Noel, and S. Jajodia. Minimum-Cost Network Hardening using Attack Graphs. *Computer Communications*, 29(18):3812–3824, 2006.

[21] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated Join Processing in Outsourced Databases. In *SIGMOD*, 2009.

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

[23] K. Yi, F. Li, G. Cormode, M. Hadjieleftheriou, G. Kollios, and D. Srivastava. Small Synopses for Group-by Query Verification on Outsourced Data Streams. *ACM TODS*, 34(3), 2009.

[24] M. L. Yiu, Y. Lin, and K. Mouratidis. Efficient Verification of Shortest Path Search via Authenticated Hints. In *ICDE*, pages 237–248, 2010.

[25] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based Spatial Queries. In *SIGMOD*, pages 443–454, 2003.