Efficient Authentication of Continuously Moving kNN Queries

Duncan Yung Yu Li Eric Lo Man Lung Yiu

Abstract—A moving kNN query continuously reports the k results (restaurants) nearest to a moving query point (tourist). In addition to the query results, a service provider often returns to mobile client a safe region that bounds the validity of query results in order to minimize the communication cost between the service provider and that mobile client. However, when a service provider is not trustworthy, it may send inaccurate query results or incorrect safe regions to mobile clients. In this paper, we present a framework for authenticating both the query results and the safe regions of moving kNN queries. We theoretically proved that our methods for authenticating moving kNN queries minimize the data sent between the service provider and the mobile clients. Extensive experiments are carried out using both real and synthetic datasets and results show that our methods can perform moving kNN query authentication with small communication costs and overhead.

Index Terms—H.2.4.h Query processing; H.2.7.d Security, integrity, and protection

1 INTRODUCTION

Location-based service providers (LBS) offer remote mobile clients with querying services on points-of-interest (e.g., restaurants, cafes, gas stations). A mobile client q issues a moving k nearest neighbor (kNN) query [33], [14] in order to find k points-of-interest closest to q continuously while traveling. Such queries have numerous mobile applications. For example, a tourist may issue a moving kNN query to obtain k nearest restaurants continuously when walking in a city. A driver issues a moving kNN query to find k nearest gas stations continuously while driving.

LBS that offer kNN querying services often return mobile clients a *safe region* [33], [14] in addition to the query results. Given a moving client q, its safe region contains all possible query locations that have the same results as q. In other words, the client only issues a new query to the LBS (for the latest results) when she leaves the safe region. This optimization significantly reduces the communication frequency between the service provider and the clients.

Unfortunately, the query results and safe regions returned by LBS may not always be accurate. For instance, a hacker may have infiltrated the LBS's servers [24] 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. The LBS may also return an overly large safe region to the clients for the sake of saving computing resources and communication bandwidth [21], [17], [29]. On the other hand, the LBS may opt to return overly small safe regions so that the clients have to request new safe regions more frequently, if the LBS charges fee for each request, or if the LBS wishes to boost its request rate — a figure that could influence its advertisement revenue.

Recently, techniques for authenticating query results have received a lot of attentions [16], [9], [18], [26], [10], [20], [17], [27], [19], [29]. Most authentication techniques are based on *Merkle tree* [13], which is an *authenticated data structure* (ADS) for ensuring the correctness of query results on a dataset. Recently, Yang et al. [27] developed an ADS called Merkle R-tree (MR-tree) for authenticating queries on a spatial dataset, and also an improved tree called MR*-tree. Upon receiving a query issued by a mobile client, the LBS not only retrieves the query results but also computes a *verification object* $\mathcal{VO}$ from the tree. Specifically, the $\mathcal{VO}$ consists of certain tree entries that can be later utilized by the client to verify the correctness of results.

The issue of authenticating *moving kNN queries*, however, has not been addressed yet. Existing authentication techniques for static spatial queries [27], [19], [5] have their authentication target as the *query results*, being a subset of the dataset. In contrast, the authentication target of moving queries includes the *safe region*, which is a geometric shape computed by the LBS at runtime but **not** part of the dataset. Since a safe region is defined based on both query results as well as points not in the query results, the missing of a non-result point in the $\mathcal{VO}$ may also fail the authentication of the safe region. Thus, the above techniques cannot help in authenticating moving kNN queries.

This paper is devoted to addressing this challenging issue of authenticating moving kNN queries. Our preliminary work [30] has developed two methods for authenticating moving kNN queries. In this paper, we improve the best authentication method (Section 4.2) and prove that it achieves $\mathcal{VO}$-optimality (Section 4.3). This optimality notion guarantees that the $\mathcal{VO}$ contains the minimum data points and tree entries (with respect to the given tree) [32].

- *D. Yung is with the Department of Computer Science, University of Pittsburgh, USA.*
  E-mail: duncanyung@cs.pitt.edu
- *Y. Li, E. Lo, and M. L. Yiu are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong.*
  E-mail: {csyuli, ericlo, csmlyiu}@comp.polyu.edu.hk

We also present new optimization techniques for reducing the computation cost (Section 4.4) and the communication cost of our authentication method (Section 4.5). It is especially important to minimize the mobile client's total communication cost as it translates to the client's money (paid to the mobile network provider). For example, the typical data rate of GO-SIM is US $0.49/MB.[1]

In addition, we extend our authentication method to handle moving kNN queries on multiple datasets (Section 5). In summary, the technical contributions of this paper include:

1) The design of *verification objects* $\mathcal{VO}$ specific for authenticating the safe regions (and also the results) of moving kNN queries.
2) An improved moving kNN authentication method that is $\mathcal{VO}$*-optimal* [32], i.e., its $\mathcal{VO}$ contains the minimum data points and tree entries (with respect to the given tree).
3) Techniques for optimizing the computation and communication costs of our authentication method.
4) Extension of our authentication method for moving kNN queries on multiple datasets.
5) A thorough experimental study on the efficiency of our proposed method on real data and synthetic data.

The rest of this paper is organized as follows. We discuss related work in Section 2. Our moving query processing and authenticating framework is presented in Section 3. In Section 4, we present our $\mathcal{VO}$-optimal method for authenticating moving kNN queries. We then extend our method for query authentication on multiple datasets in Section 5. The experimental study is presented in Section 6. Finally, we conclude this paper in Section 7.

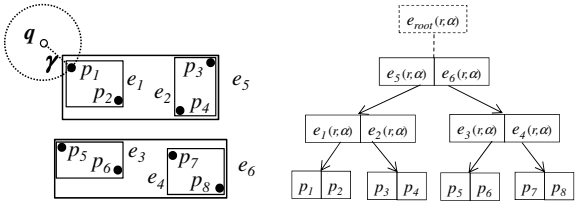
2 Related Work

2.1 Query authentication

In the literature, most authentication techniques [9], [26], [10], [20], [27], [19], [17], [29], [11] are based on *Merkle tree* [13]. Merkle tree 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.[2] Then, the signature of the dataset is obtained by signing the root digest using the data owner's private key. *Signature aggregation* [16], [18], [4] is an alternative authentication technique, which builds a signature per data object (or per grid cell). The advantage of this approach is that it avoids revealing non-result data objects to the client. Recently, Yi et al. [28] propose a probabilistic approach for authenticating aggregation queries; and Kundu et al. study the authentication of trees and graphs [7], [8].

The Merkle R-tree (MR-tree) is an ADS for authenticating spatial queries, including range queries [27], kNN queries [27], and location-based skyline queries [11]. MR-tree is developed based on R*-tree [1] and Merkle tree [13]. 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 verify the correctness of the NN results, the server 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.



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

Fig. 1. Authentication of nearest neighbor queries

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 $e.b \cap \odot(q, \gamma') = \emptyset$. If so, the client can assure that the computed NN result is correct. If the server omits some point (e.g., p_1) in the above $\mathcal{VO}$, then the client cannot reconstruct the correct root signature in the above verification process.

MR*-tree is an extension of MR-tree [27], where each node is embedded with a conceptual Merkle KD-tree defined on entries in the node. It achieves a smaller $\mathcal{VO}$ than MR-tree by replacing non-result entries in $\mathcal{VO}$ with digests. Our proposed solutions for moving query authentication are applicable on both MR-tree and MR*-tree.

Existing spatial authentication techniques [27], [19], [5]

1. http://www.gosim.ekit.com/ekit/MobileInfo/popup_data_services
2. A secure-hash function is often used to compute a fix-length digest.

cannot help in authenticating moving queries because they focus on *static* queries, whose authentication targets (i.e., 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).

2.2 Moving query processing

In moving query processing, Tao et al. [23] compute the nearest neighbors for each possible query point on a given line segment, whereas Iwerks et al. [6] study how to maintain the client's future kNN upon a change of the client's velocity. Both work [23], [6] model the client's movement as a linear function. When the client's future movement is unknown, the buffering approach [22] and the safe region approach [33], [14] are more appropriate for efficient moving query processing.

In the buffering approach [22], the LBS returns to the client kNN points and the next Δk nearest points, where the parameter Δk controls the number of additional points. Specifically, let q_{last} be the last query client location and d_i be the i-th nearest neighbor distance for q_{last}. The *buffer region* is defined as a circle $\odot(q_{last}, \epsilon)$ with center q_{last} and radius $\epsilon = (d_{k+\Delta k} - d_k)/2$. Song et al. [22] has proven that, while the client moves within the buffer region, her latest result can be recomputed locally from those buffered $k+\Delta k$ points. Only when the client moves outside $\odot(q_{last}, \epsilon)$, it needs to issue a new $(k+\Delta k)$NN query to the LBS. However, it is not easy to tune Δk in practice, which heavily influences the communication frequency and the number of objects transferred per communication. Furthermore, the client incurs computation overhead on recomputing result.

In the safe region approach, the LBS reports a safe region [33], [14], [3] for the query result, such that the result remains unchanged as long as the client 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 overhead at the client.

We illustrate the buffering approach and the safe region approach in Figure 2. Assume $k = 1$ and all locations are on the x axis. Let q_t be the location of client q at timestamp t. At time $t = 0$, the NN result of q is p_1. For the safe region approach, the safe region of NN (p_1) is the interval $[1, \infty]$. When q travels towards the right-side, q still falls into the safe region and does not need further communication with the LBS. On the other hand, the buffering approach may incur frequent communication if Δk is not carefully tuned. Suppose that $\Delta k = 1$. At $t = 0$, the client (at q_0) retrieves $(k + \Delta k)$NN: p_1 and p_2. The buffer region is the circle $\odot(q_0, \epsilon)$ where $\epsilon = (d_{k+\Delta k} - d_k)/2 = (4 - 2)/2 = 1$. This region is the interval $[3, 5]$. At $t = 1$, the client (at q_1) leaves the buffer region. Thus, it requests the LBS for the latest $(k+\Delta k)$NN and the buffer region $\odot(q_1, \epsilon)$, where $\epsilon = (5-3)/2 = 1$. This region is the interval $[4, 6]$. Similarly, in each subsequent timestamp, the client has to communicate with the LBS upon reaching $q_2, q_3, \cdots$. In summary, the communication cost of the buffering approach can be much higher than the safe region approach.

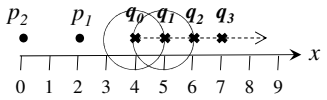


Fig. 2. Example on buffering and safe region, $k = 1$

3 The Framework

Following previous work [33], [14], [3], in this paper, we use a general problem setting in which future locations of a moving query client cannot be predicted in advance.

Figure 3 illustrates our framework for answering moving kNN queries with query correctness verification. A map provider (e.g., the government's land department, NAVTEQ[3] and TeleAtlas[4]) collects (public) points-of-interests into a spatial dataset D. Each point-of-interest contains a lat-long coordinate and auxiliary attributes (e.g., name, description, phone). Then, the map provider builds a MR-tree/MR*-tree [27] T_D on the dataset and signs the digest of the root node, before distributing the tree to a service provider (i.e., LBS). Our adversary model and security guarantee are the same as in Yang et al. [27]. Adversaries know all information (e.g., the map provider's public key, the secure-hash function, the tree, and our authentication algorithms), except the map provider's private key.

Initially, a mobile client q downloads the root signature from the LBS and the map provider's public key from a certificate authority (e.g., VeriSign). Afterwards, the client sends its location to the LBS, and obtains the query result, the safe region, and the $\mathcal{VO}$. The correctness of the query result 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.

However, the LBS may return incorrect safe regions, rendering the client's future result incorrect. For example, the LBS may return overly large safe regions for the sake of saving computing resources and communication bandwidth [21], [17], [29]. On the other hand, the LBS may opt to return overly small safe regions so that the clients have to request new safe regions more frequently, if the LBS charges fee for each request or wishes to boost its request rate. *Our goal is thus to devise an effective method for the client to verify the correctness of the safe region returned by LBS.* We aim at minimizing the client's total communication cost as it translates to the client's money to pay to the mobile network provider (cf. Introduction).

As a remark, the spatial dataset is expected to have infrequent updates (e.g., monthly map updates). In case the client requires *fresh* results (i.e., obtained from the latest

3. NAVTEQ Maps and Traffic. http://www.navteq.com
4. TeleAtlas Digital Mapping. http://www.teleatlas.com

data), the map provider could follow Li et al. [9] to include a timestamp in the root signature of the tree.

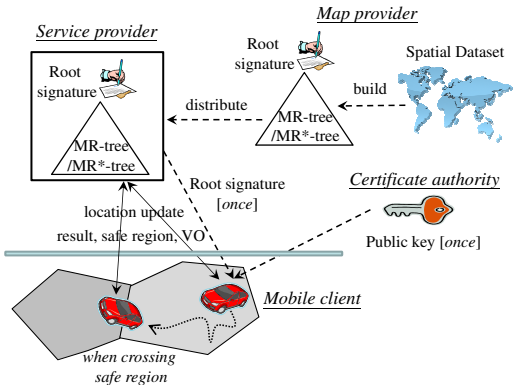


Fig. 3. Moving kNN Authentication

3.1 Baseline: Buffering Approach

In a moving query environment, the question is when and where the client should (re-)issue query in order to get the most updated query range result as q moves.

A baseline authentication method for moving kNN queries is to apply the buffering approach [22] as follows. The client requests $(k + \Delta k)$NN from the server, as discussed in Section 2.2. These $(k + \Delta k)$NN points can then be authenticated by using an MR-tree/MR*-tree [27]. After verifying their correctness, the client can derive its latest kNN result locally from those retrieved points, while it moves within the buffer region.

Although simple, the buffering approach requires finding the optimal value of Δk for different datasets. A small Δk leads to more frequent communication, thereby increasing the communication cost. A large Δk, unfortunately, also increases the communication cost because the size of the $\mathcal{VO}$ increases. Furthermore, even when staying within the buffer region, the client still incurs overhead on recomputing the result locally.

In the next section, we present our safe region approach for authenticating kNN queries. The query result remains unchanged as long as the client's location q stays within the safe region. This approach not only reduces the communication frequency between the server and the client, but also the computational overhead at the client.

4 Authenticating Moving kNN Queries

First, we examine the safe regions of kNN queries and the challenge on authenticating them (Section 4.1). Then, we present our *Vertex-Based* (VB) method for constructing $\mathcal{VO}$ for moving kNN query authentication (Section 4.2). Our methods are applicable on both MR-tree and MR*-tree. We improve the method presented in our preliminary work [30] and now prove that this (new) VB method is $\mathcal{VO}$-optimal [32], i.e., it puts the minimum data points and tree entries into the $\mathcal{VO}$, with respect to the given tree (Section 4.3). Then, we present techniques for optimizing the computation cost and the communication cost, in Sections 4.4 and 4.5 respectively. Finally, we discuss how to support selection predicates in the query in Section 4.6.

4.1 Preliminaries and Authentication Challenges

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. A summary of notation used in this paper is given in Table 1.

TABLE 1
Summary of Notations

Symbol	Meaning
D	the dataset of points
q	the (current) query point
k	number of nearest neighbors
S	kNN result
γ	the distance between q and its k-th nearest neighbor
$\odot(q, r)$	circular region with center q and radius r
$dist(p, p')$	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 (the safe region)
$G(S, D)$	the generator set of $\mathcal{V}(S, D)$
Ψ	vertices of $\mathcal{V}_k(S, D)$
e	a non-leaf entry in an MR-tree/MR*-tree
$mindist(e, q)$	the minimum distance between q and the extent of e
KVR	kNN verification region
$SRVR$	safe region verification region
Γ	verification region

We adopt the order-k Voronoi cell [33] as the safe region for moving kNN queries. We are aware of another safe region technique called V*-diagram [14], 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.

The basic definitions for Voronoi cells are as follows.

Definition 1. Voronoi cell [15].
Given a point p from a point set D, the Voronoi cell $\mathcal{V}(p, D)$ *of p with respect to D is defined as:*

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

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

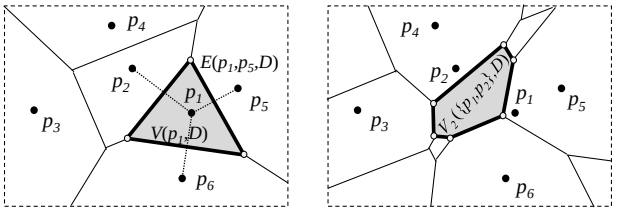
Definition 2. Voronoi edge.
For two adjacent Voronoi cells $\mathcal{V}(p, D)$ and $\mathcal{V}(p', D)$, their Voronoi edge is defined as their shared line segment:

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

Definition 3. 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 4a 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 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 $\mathcal{V}(p_1, D)$.

Definition 4. Order-k Voronoi cell [15].
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:

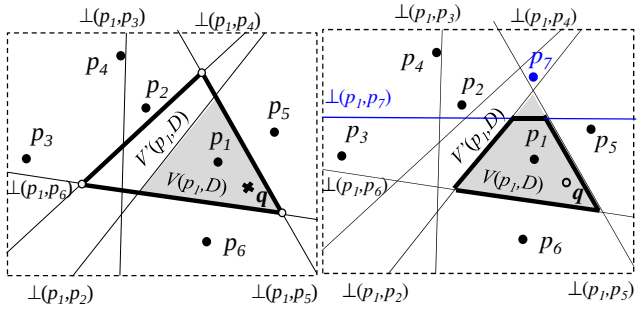


(a) Voronoi cells (b) order-k Voronoi cells, at $k = 2$

Fig. 4. Example of Voronoi cells

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

Figure 4b depicts the order-2 Voronoi cells on the dataset D by using bold line segments. 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.



(a) a larger safe region (b) a smaller safe region

Fig. 5. Challenge of safe region authentication

The challenge of verifying the correctness of a Voronoi cell (safe region) is illustrated in Figure 5. The server 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 grey triangle). Through Definition 3, the server 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 in Figure 5a the server 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 overly large safe region using an incorrect Voronoi cell $\mathcal{V}'(p_1,D)$, as indicated by the bold triangle in the figure. Since all points of $G'(p_1,D)$ also originate from the dataset D, they pass the data correctness checking that verifies the root signature. The problem here is that the client cannot verify whether the points in $G'(p_1,D)$ are the same as the actual generator set $G(p_1,D)$.

The server may also produce a smaller fake safe region, as shown in Figure 5b, by using a fake generator set $G'(p_1,D) = \{p_2,p_5,p_6,p_7\}$. Observe that the fake safe region can only become smaller than the actual safe region when the generator set contains fake points (e.g., p_7). However, such fake points can be easily detected by the client because the reconstructed $\mathcal{VO}$ will have a root signature different from the original root signature.

By Definition 1, $\mathcal{V}(p_1,D)$ is constructed by *all points* in D. So, a brute-force solution is to return the whole dataset D to the client so that she is guaranteed to compute $\mathcal{V}(p_1,D)$ correctly. Our goal is to design an algorithm to construct verification object $\mathcal{VO}$ for verifying the correctness of the Voronoi cell, yet the size of $\mathcal{VO}$ should be as small as possible.

4.2 Vertex-Based Method

Our Vertex-Based (VB) method exploits the vertices of an order-k Voronoi cell in order to construct a compact $\mathcal{VO}$ for authenticating a moving kNN query. VB consists of a server algorithm (for computing result, safe region, and $\mathcal{VO}$) and a client algorithm (for verifying result and safe region by using $\mathcal{VO}$).

Algorithm 1 Vertex-Based Method (Server)

Receive from client: (Query point q, Integer k)
Using MR-tree/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 method [33]);
3: $\gamma := \max_{p\in S} dist(q,p)$; ▷ authenticate kNN
4: $KVR := \odot(q,\gamma)$;
5: Ψ := set of vertices of $\mathcal{V}_k(S,D)$; ▷ authenticate safe region
6: $SRVR := \bigcup_{\psi\in\Psi} \odot(\psi, \max_{p\in S} dist(\psi,p)) - \circlearrowleft + \circledast$;
7: $\Gamma := KVR \cup SRVR$;
8: $\mathcal{VO}$:= DepthFirstRangeSearch(T_D.root, Γ);
9: **send** $\mathcal{VO}$ to the client;

4.2.1 Server Algorithm

Algorithm 1 is the pseudo-code of the server algorithm. Upon receiving the client location q and the number k of required NNs, it computes the kNN result S from an MR-tree/MR*-tree T_D (Line 1). Then, it computes the safe region as the order-k Voronoi cell $\mathcal{V}_k(S,D)$ (Line 2). Next, it defines verification region Γ so as to identify useful points for verifying the kNN result and the safe region, and puts these points into the $\mathcal{VO}$. Specifically, the verification region Γ is defined as the union of (i) the *kNN query result verification region* (KVR) and (ii) the *kNN safe region verification region* ($SRVR$).

Definition 5. kNN query result verification region (KVR).
The kNN query result verification region *(KVR) is defined as the circular region $\odot(q,\gamma)$, where γ is the distance between q's k-th nearest neighbor and q.*

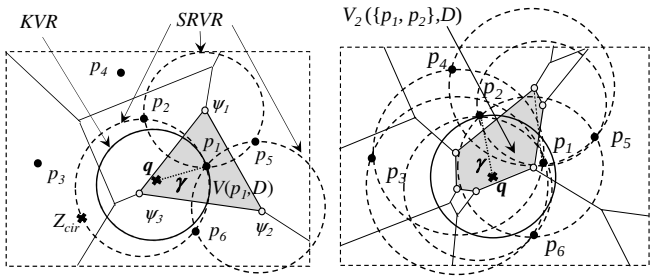
Definition 6. kNN safe region verification region ($SRVR$).
The kNN safe region verification region *($SRVR$) is defined as the union of circular regions at each vertex ψ of the Voronoi cell to its farthest point in the kNN result*

S. i.e., $\bigcup_{\psi\in\Psi} \odot(\psi, \max_{p\in S} dist(\psi,p))$*, but excluding the circumference* $\circlearrowleft$ *except the intersections* $\circledast$*. i.e., we have:*

$$SRVR = \bigcup_{\psi\in\Psi} \odot(\psi, \max_{p\in S} dist(\psi,p)) - \circlearrowleft + \circledast$$

where Ψ *denotes the set of vertices of the Voronoi cell* $\mathcal{V}_k(S,D)$*,* $\circlearrowleft = \{z \in \mathbb{R}^2 \mid \exists \psi_i \in \Psi,\ dist(\psi_i, z) = \gamma_i, \forall \psi_j \in \Psi,\ dist(z,\psi_j) \geq \gamma_j\}$*, and* $\circledast = \{z \in \mathbb{R}^2 \mid \exists \psi_i, \psi_j \in \Psi,\ dist(z,\psi_i) = \gamma_i,\ dist(z,\psi_j) = \gamma_j,\ i \neq j\}$.

Unlike our earlier definition in [30], this definition of $SRVR$ excludes the circumference $\circlearrowleft$ except circle intersections $\circledast$. With this improved definition, we will show that VB method is $\mathcal{VO}$-optimal in Section 4.3.



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

Fig. 6. Verification region of order-k Voronoi cell

Figure 6a depicts the verification region Γ for the case $k = 1$. The NN of q is point p_1 and its safe region is the Voronoi cell $\mathcal{V}(p_1, D)$ (in grey triangle). The KVR is the solid circle centered at q. $SRVR$ is the union of the dotted circles centered at vertices ψ_1, ψ_2, ψ_3. Note that $SRVR$ does not include the circumference of those dotted circles, except the intersections of the circles (p_2, p_5, and p_6).

The $\mathcal{VO}$ is computed by a depth-first traversal of the MR-tree/MR*-tree (Line 8). Finally, the server sends the $\mathcal{VO}$ to the client (Line 9), who will extract the kNN result and the safe region from $\mathcal{VO}$ and authenticate them (see Section 4.2.2).

Specifically, at Line 8, the tree is traversed with the following two conditions: (i) if a non-leaf entry e does not intersect Γ, e is added to the $\mathcal{VO}$ and its subtree will not be visited; (ii) data points in any visited node (i.e., intersecting Γ) are put into the $\mathcal{VO}$. Thus, in our example (Figure 6a), the $\mathcal{VO}$ contains the points p_1, p_2, p_5 and p_6 and other non-leaf entries in the tree. Note that z_{cir} is a data point on $\circlearrowleft$, so it is not part of $SRVR$ and would not be included in the $\mathcal{VO}$. Figure 6b illustrates the verification region Γ for the case $k = 2$. The points p_1 and p_2 are 2NN of q and the order-2 Voronoi cell $\mathcal{V}_2(\{p_1, p_2\}, D)$ (the grey polygon) is the safe region. In this example, the verification region Γ 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 verification region Γ *cannot* alter the kNN results and the corresponding safe region $\mathcal{V}_k(S, D)$ and thus it is safe to not include them in the $\mathcal{VO}$.

Theorem 1. [Points p^* outside Γ *cannot* alter the kNN results and the corresponding safe region $\mathcal{V}_k(S, D)$]

Proof: See the proof in the Appendix. □

For example, in Figure 6a, points z_{cir}, p_3 and p_4 fall outside $SRVR$. By Lemma 4, 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 put only those points within Γ into 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 result).

Note that, for efficient implementation, the server does not need to render the complex shape of the verification region Γ (Line 7). Instead, we check whether the following condition holds during the tree traversal (Line 8):

$$\begin{aligned}(mindist(e,q) \leq \gamma_q) \vee \bigvee_{p\in G(S,D)} (mindist(e,p) = 0) \\ \vee \bigvee_{\psi\in\Psi} \left(mindist(e,\psi) < \max_{p\in S} dist(\psi,p) \right)\end{aligned} \quad (1)$$

where the first term checks whether a non-leaf entry e intersects KVR and the second and third term check whether e intersects $SRVR$. If the condition is false, we add e into the $\mathcal{VO}$. Using this condition, the $\mathcal{VO}$ can be constructed efficiently.

The time complexity of the server algorithm is analyzed as follows. First, it is dominated by the number of node accesses for the verification region Γ (at Line 8). Yang et al. [27] has a formula of computing the node access cost for a range query. In the following, we attempt to approximate the verification region Γ as a circular region and then apply the formula in [27] to estimate the node access cost.

Since $\Gamma = KVR \cup SRVR$ and $SRVR$ always covers KVR, it suffices to estimate the area of $SRVR$. For simplicity, we assume that data points are uniformly distributed in the unit space $[0, 1]^2$. With this assumption, we can approximate the shape of an order-k Voronoi cell $\mathcal{V}_k(S, D)$ by a circle with radius λ. We have $\lambda = \sqrt{\frac{1}{\pi\cdot(2k-1)n}}$ because the average area of $\mathcal{V}_k(S, D)$ is $\approx \frac{1}{(2k-1)n}$, according to [15]. Next, we approximate the verification region $\Gamma = SRVR$ as a circle with the radius $\lambda + \gamma$, where $\gamma = \sqrt{\frac{k}{\pi n}}$ is the average k-th NN distance. Finally, we estimate the node access cost by substituting the circle radius $\gamma + \lambda$ into the cost formula in [27].

The above analysis is based on uniform data distribution. In case of skewed data distributions, histograms may be used to estimate the local density, γ and λ.

4.2.2 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 tree root signature signed by the map provider

(Lines 1–2). If the verification is successful, the $\mathcal{VO}$ is guaranteed to contain only entries from the original 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).

Algorithm 2 Vertex-Based Method (Client)

Receive from server: (Verification object $\mathcal{VO}$)
1: h'_{root} := reconstruct the root digest from $\mathcal{VO}$;
2: verify h'_{root} against the 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 $\mathcal{V}_k(S', D')$; ▷ authenticate safe region
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**
13: **return** kNN result S' and safe region $\mathcal{V}$;
14: **return** authentication failed;

As described in Section 2, the kNN set S' is correct if every non-leaf entry of R' does not intersect $\odot(q,\gamma')$, where γ' is the distance between q and its k-th NN (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 $SRVR$ by the set of vertices Ψ in $\mathcal{V}$ (Lines 10–11). By Lemma 2, the client can ensure that the cell $\mathcal{V}$ is correct if every non-leaf entry of R' does not intersect $SRVR$ (Line 12). And if so, the client can regard the computed kNN result and the safe region $\mathcal{V}$ as correct.

Lemma 1. [kNN result correctness verifiable at client]

Proof: Direct results from Yang et al. [27]. □

Lemma 2. [safe region correctness verifiable at client] *Following the Vertex-Based (VB) method, a client can verify that the constructed safe region $V_k(S', D')$ equals to the correct safe region $V_k(S, D)$, i.e., $V_k(S', D') = V_k(S, D)$.*

Proof: See the proof in the Appendix. □

4.3 $\mathcal{VO}$-optimality

In this section, we prove that the Vertex-Based (VB) method is $\mathcal{VO}$-optimal [32], i.e., it puts the minimum data points and tree entries into the $\mathcal{VO}$, with respect to the given tree.

Theorem 2. [The VB method is $\mathcal{VO}$-optimal, with respect to the given tree T_D]

Proof: See the proof in the Appendix. □

4.4 Computation Optimization

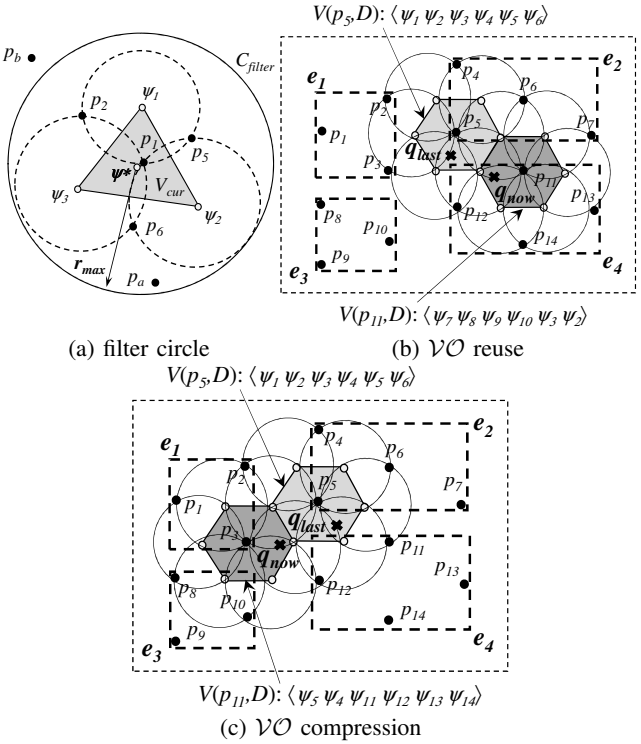
This section presents an optimization to boost the efficiency of a frequently-used operation in our authentication algorithms. In both the server and client algorithms of VB, a frequently-used operation is to check whether a point p' (or an entry e') may refine the safe region. It is used at Lines 2, 8 of Algorithm 1, and Lines 9, 12 of Algorithm 2.

Let $\mathcal{V}_{cur}$ be the safe region found so far (with respect to examined points), for the kNN result set S. To determine whether a point p' can refine $\mathcal{V}_{cur}$, we check the condition:

$$\vee_{\psi_i \in \Psi_{cur}} \quad p' \in \odot(\psi_i, r_i) \tag{2}$$

where ψ_i is a vertex of $\mathcal{V}_{cur}$ and $r_i = \max_{p \in S} dist(\psi_i, p)$. In Figure 7a, $\mathcal{V}_{cur}$ is the safe region found so far for the result set $S = \{p_1\}$. It is formed after examining points p_2, p_5, p_6. Next, we examine points p_a, p_b and check the above condition. Since they are outside all dotted circles in Figure 7a, they cannot refine $\mathcal{V}_{cur}$.

We now present an early termination technique to avoid examining unpromising points that cannot help refine $\mathcal{V}_{cur}$. The idea is to represent the above dotted circles by a large circle $C_{filter} = \odot(\psi^*, r_{max})$. For this, we first compute the centroid ψ^* from the vertices of $\mathcal{V}_{cur}$ and then derive the covering radius $r_{max} = \max_{\psi_i \in \Psi} dist(\psi^*, \psi_i) + r_i$. It guarantees that, if a point p' is outside C_{filter}, then it can be pruned safely. This checking condition is more efficient than the previous one (Equation 2), as it involves only one circle instead of multiple circles. In Figure 7a, point p_b can be pruned by using C_{filter}. By applying the above checking condition in the tree traversal, we are able to establish an early termination condition.



(a) filter circle (b) $\mathcal{VO}$ reuse

(c) $\mathcal{VO}$ compression

Fig. 7. Optimization techniques for computation and communication costs

4.5 Communication Optimization

Upon leaving the safe region, the client would request the new safe region (and its verification object) from the server. In our VB method, the safe region is an order-k Voronoi cells, whose area becomes small at a large k. In this case, the previous and the current verification objects, say $\mathcal{VO}_{last}$ and $\mathcal{VO}_{now}$, may (i) have significant overlap or (ii) even intersect the same tree nodes. Section 4.5.1 proposes a client-side technique to check case (ii) and avoid communication in such case. Section 4.5.2 addresses case (i) and proposes a server-side technique to compress $\mathcal{VO}_{now}$ and thus reduce its communication cost.

4.5.1 Client-Side: Reuse of $\mathcal{VO}$

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.

Let us consider the example of Figure 7b. Assume the client was previously located at q_{last}, which had its NN as p_1 and its safe region as $\mathcal{V}(p_5, D)$ (the light-grey region), formed by the vertices $\psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6$ in the clockwise order. Its corresponding verification region Γ_{last} includes $\bigcup_{i=1,2,3,4,5,6} \odot(\psi_i, dist(\psi_i, p_5))$. Let $\mathcal{VO}_{last}$ be the resulting verification object reported by the server. Since Γ_{last} intersects the non-leaf entries e_1, e_2, e_4, they are visited and their data points $p_1 \cdots p_3$, $p_4 \cdots p_7$, $p_{11} \cdots p_{14}$ are inserted into $\mathcal{VO}_{last}$. As Γ_{last} does not intersect e_3, so e_3 is inserted into $\mathcal{VO}_{last}$, and the subtree of e_3 is not visited. Thus, the client received: $\mathcal{VO}_{last} = \{\{p_1 \cdots p_3\}, \{p_4 \cdots p_7\}, e_3, \{p_{11} \cdots p_{14}\}\}$.

Later on, the client moves to a new location q_{now} outside $\mathcal{V}(p_5, D)$. Before sending a new query to the server, the client can run Algorithm 2 again using the previous $\mathcal{VO}_{last}$ but with the current client location q_{now}. First, the client attempts to find, from $\mathcal{VO}_{last}$, the NN result p_{11} and safe region $\mathcal{V}(p_{11}, D)$ (the dark-grey region) of q_{now}. The vertices of $\mathcal{V}(p_{11}, D)$ are: $\psi_7, \psi_8, \psi_9, \psi_{10}, \psi_3, \psi_2$. Its corresponding verification region Γ_{now} includes $\bigcup_{i=7,8,9,10,3,2} \odot(\psi_i, dist(\psi_i, p_{11}))$. Observe that Γ_{now} does not intersect with any non-leaf entries in $\mathcal{VO}_{last}$ (e.g., e_3) as well. Thus, the correctness of $\mathcal{VO}_{last}$ is maintained even for the new query location q_{now}. 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 sends a new query to the server only when the new verification region Γ_{now} intersects some non-leaf entry in $\mathcal{VO}_{last}$.

4.5.2 Server-Side: Compression of $\mathcal{VO}$

This section presents two optimizations for the server to save the communication cost per response.

The first optimization is to remove irrelevant attributes from the $\mathcal{VO}$. Recall from Section 3 that a data point p has a lat-long coordinate $p.loc$ and auxiliary attributes $p.aux$ (e.g., name, description, phone). Among data points in $\mathcal{VO}$, we must keep $p.aux$ for each result p (as the client needs) and discard $p'.aux$ for each non-result p'. To ensure that the client can compute the digests of points correctly, we redefine the digest of point p as: $h^*(p) = h(p.loc|h(p.aux))$, where h is the secure-hash function. Then, in the $\mathcal{VO}$, we only include $p'.loc$ and $h(p'.aux)$ for each non-result p'. Note that $h(p'.aux)$ occupies less space than $p'.aux$.

We proceed to elaborate the second optimization. Let $\mathcal{VO}_{last}$ and $\mathcal{VO}_{now}$ be the previous and current verification objects for the client respectively. A certain tree leaf node, consisting of points $\{p_i \cdots p_j\}$, may appear in both $\mathcal{VO}_{last}$ and $\mathcal{VO}_{now}$. Our main idea is to let the server detect all such shared leaf nodes between $\mathcal{VO}_{last}$ and $\mathcal{VO}_{now}$. Then, the server computes a compressed verification object $\Delta\mathcal{VO}_{now}$, whose shared leaf nodes are replaced by (compact) tokens.

Let us consider the example of Figure 7c. When the client was located at q_{last}, the NN was p_5, the safe region was $\mathcal{V}(p_5, D)$ (the light-grey region). As discussed in Section 4.5.1, the client received: $\mathcal{VO}_{last} = \{\{p_1 \cdots p_3\}, \{p_4 \cdots p_7\}, e_3, \{p_{11} \cdots p_{14}\}\}$. Now, the client moves to a new location q_{now} outside $\mathcal{V}(p_5, D)$, and thus requests the server for the updated result, safe region, and verification object. The server computes the NN as p_3 and the safe region as $\mathcal{V}(p_3, D)$ (the dark-grey region). Its corresponding verification region Γ_{now} intersects e_1, e_2, e_3, e_4. Thus, the current verification object is: $\mathcal{VO}_{now} = \{\{p_1 \cdots p_3\}, \{p_4 \cdots p_7\}, \{p_8 \cdots p_{10}\}, \{p_{11} \cdots p_{14}\}\}$.

Observe that there is a significant overlap between $\mathcal{VO}_{last}$ and $\mathcal{VO}_{now}$. For instance, both of them contain the tree leaf nodes: $\{p_1 \cdots p_3\}$, $\{p_4 \cdots p_7\}$, $\{p_{11} \cdots p_{14}\}$. By replacing these nodes with the tokens $\#1, \#2, \#4$ respectively, the server can derive the compressed verification object: $\Delta\mathcal{VO}_{now} = \{\#1, \#2, \{p_8 \cdots p_{10}\}, \#4\}$. Clearly, the size of $\Delta\mathcal{VO}_{now}$ is much smaller than $\mathcal{VO}_{now}$.

Upon receiving $\Delta\mathcal{VO}_{now}$, the client substitutes the tokens in $\Delta\mathcal{VO}_{now}$ with the corresponding leaf nodes (that appeared in $\mathcal{VO}_{last}$) in order to reconstruct the original verification object $\mathcal{VO}_{now}$. Then, the client proceeds with the verification procedure on $\mathcal{VO}_{now}$. As a remark, to support efficient lookup of tokens, the client may employ a hash table for storing the leaf nodes of $\mathcal{VO}_{last}$.

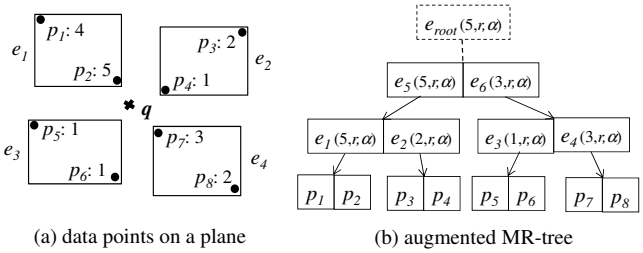
It remains to clarify how to compute the token of a leaf node. Let B be the maximum capacity of nodes in the tree. Consider the traversal path from the root to a leaf node: $(a_1, a_2, a_3, \cdots, a_h)$, where a_i is the branch visited at level i. The server defines the token of the node as: $\sum_{i=1}^{h} a_i \cdot B^{h-1-i}$. Note that the client is also able to compute the token of leaf nodes because it can infer their traversal paths from $\Delta\mathcal{VO}_{now}$ directly.

4.6 Discussion for Selection Predicates

In some applications, the client wishes to specify selection predicates in the kNN query. An example query is: "find my 2 nearest restaurants with 4 stars or above". In Figure 8a, the results are p_1 and p_2 as their ratings are at least 4.

To enable processing such queries efficiently at the server, we recommend the data owner to construct an

augmented MR-tree as shown in Figure 8b. In this tree, each non-leaf entry e_i is augmented with the max rating of its children. For example, the entry e_2 is augmented with the rating 4 (i.e., max rating among p_3, p_4). At query time, the $\mathcal{VO}$ would also contain non-leaf entries e_i whose max ratings are below the query's rating (e.g., 4). In the above example, the entries e_2, e_6 are included into the $\mathcal{VO}$.



(a) data points on a plane (b) augmented MR-tree

Fig. 8. Augmented MR-tree (with rating information)

The above technique can be extended to queries with multiple selection predicates, e.g., "find my 2 nearest asian restaurants with 4 stars or above". In such case, we augment each non-leaf entry in the tree with a Boolean flag for the condition "asian". Data compression techniques may be applied to reduce the space of augmented information.

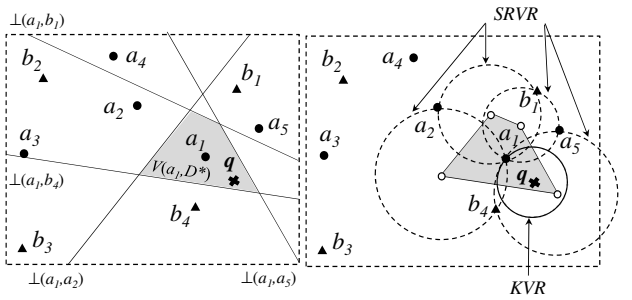
5 Querying on Multiple Datasets

Our discussion so far assumes that mobile clients always issue queries on the same dataset. In practice, clients may issue queries on multiple types of datasets that match with their interests. For instance, a client may issue the query "find my nearest restaurant or gas station", which involves two datasets. These datasets originate from different dataset owners. For example, in USA, a restaurant association (http://www.restaurant.org/) provides the locations of restaurants, and a gas station directory (http://gasbuddy.com/) maintains the locations of gas stations.

At query time, a mobile client specifies her location q and a subset of m datasets that match with her interests. Without loss of generality, we assume that these queried datasets are $D_1, D_2, \cdots, D_m$ respectively and denote their union as D^*. The kNN result set of the query is then defined as the set S in Definition 7. The safe region of S is the order-k Voronoi cell $\mathcal{V}_k(S, D^*)$, according to Definition 4.

Definition 7. Multi-dataset kNN query result. *Let D^* be the union of m datasets $D_1, D_2, \cdots, D_m$. Given a query location q, its multi-dataset kNN result set is a size-k subset $S \subseteq D^*$ such that: $\forall p \in S, \forall p' \in D^* - S, dist(q,p) \leq dist(q,p')$.*

For example, suppose that a client q issues the query "find my nearest restaurant or gas station". Datasets D_1 (restaurants) and D_2 (gas stations) match with her query interests. Figure 9a shows the location of q and points in $D_1 = \{a_1, a_2, a_3, a_4, a_5\}$ and $D_2 = \{b_1, b_2, b_3, b_4\}$. The NN of q is point a_1. This figure also illustrates the safe region of the result $\mathcal{V}(a_1, D^*)$, which is formed by the half-planes of q with a_2, a_5, b_1, b_4.



(a) safe region (b) VB verification region

Fig. 9. Multi-dataset kNN example, with $k = 1, m = 2$, and $D_1 = \{a_1, a_2, a_3, a_4, a_5\}, D_2 = \{b_1, b_2, b_3, b_4\}$

First, we study the centralized server scenario where a single server hosts all datasets. Then, we examine the scenario for distributed servers where different servers host different datasets separately.

5.1 The Centralized Server Scenario

We illustrate the centralized server scenario in Figure 10. Each data owner O_i publishes its dataset D_i as a MR-tree/MR*-tree T_i and signs its tree root signature separately. For example, T_1, T_2, T_3, T_4, T_5 are the trees built on restaurants, gas stations, shops, bars, and car parks, respectively. The LBS offers mobile clients a one-stop querying service on these trees. Observe that, since the LBS does not have the private keys of data owners, it cannot merge all trees T_i into a single tree T^* and provide query authentication on T^* directly. Thus, we need to develop specific query authentication techniques for multi-datasets.

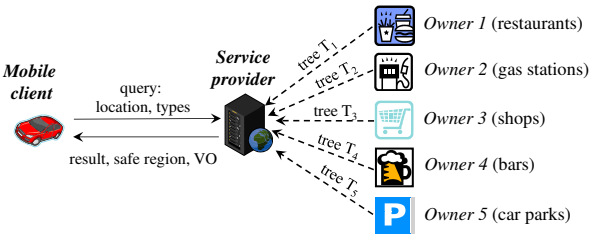


Fig. 10. A Centralized Server (Service Provider)

At query time, the mobile client can download the public key of each data owner from a certificate authority (e.g., VeriSign). Upon receiving the client's request, the LBS computes the query result, the safe region, and the $\mathcal{VO}$. Similar to Section 3, the client can verify the correctness of the query result and the safe region via the $\mathcal{VO}$.

Our goal is to develop a communication-efficient method for the client to verify the correctness of the multi-dataset query result and the safe region returned by the LBS.

It is straightforward to adapt the buffering approach for multi-dataset kNN queries, thus we do not present it here.

Extension of Vertex-Based Method.
We proceed to extend our VB method for processing

moving multi-dataset kNN queries.

Our server algorithm aims to construct m compact $\mathcal{VO}$ for authenticating a moving multi-dataset kNN query. It takes as input the client location q, the number k of results, and m types of datasets specified by the client. First, it computes the kNN result set S and the safe region $\mathcal{V}_k(S, D^*)$ from m trees $T_1, \cdots, T_m$. The efficiency of this step can be optimized by applying a synchronous traversal on all m trees. As an example in Figure 9b, suppose that $k = 1$ and the queried datasets be $D_1 = \{a_i\}$ and $D_2 = \{b_i\}$. The result is point a_1 and the safe region is shown as the grey region. Regarding the verification region Γ, Theorem 1 is also applicable to the multi-dataset kNN result and the corresponding safe region. The only difference from Algorithm 1 is that, the algorithm traverses each tree T_i independently to construct a verification object $\mathcal{VO}_i$ with respect to the verification region Γ. In the example, the verification region Γ (the union of circles) covers points a_1, a_2, a_5 from tree T_1 and points b_1, b_4 from tree T_2.

We proceed to describe our client algorithm. Upon receiving m verification objects $\mathcal{VO}_1, \cdots, \mathcal{VO}_m$ from the server, it first reconstructs m root digests from each $\mathcal{VO}_i$ and verifies them against the corresponding tree root signatures signed by data owners. If the verification is successful, then those $\mathcal{VO}$s are guaranteed to contain only entries from the original tree(s) (i.e., no fake entries). Next, it extracts from each $\mathcal{VO}_i$ (i) a set D'_i of data points, and (ii) a set R'_i of non-leaf entries. After that, the client computes the kNN result S' and the safe region $\mathcal{V}$ from the extracted data points. Finally, the client computes the KVR and the SRVR to verify the correctness of the kNN result and the safe region.

5.2 The Scenario for Distributed Servers

We sketch an approach for authenticating kNN queries in the distributed scenario.

Initial $\mathcal{VO}$ computation.
In this approach, the client issues its location q to all servers in parallel. Each server SP_i uses its local dataset D_i (stored in tree T_i) to compute a local kNN result set S_i, a safe region $\mathcal{V}_k(S_i, D_i)$, a verification region Γ_i, and a verification object $\mathcal{VO}_i$. Then, every server sends its $\mathcal{VO}_i$ to the client in parallel.

When compared to the centralized scenario, a server SP_i cannot use other types of data points to tighten S_i, $\mathcal{V}_k(S_i, D_i)$, and Γ_i. Thus, each individual verification region Γ_i in the distributed scenario covers the verification region (say Γ) in the centralized scenario. This guarantees that the client obtains sufficient information for verification. Finally, the client applies the client-side verification algorithm as stated in Section 5.1.

$\mathcal{VO}$ maintenance.
When a client leaves its current safe region (say $\mathcal{V}_{cur}$), it computes the new safe region (say $\mathcal{V}_{new}$) and the new verification region (say Γ_{new}) by using the previous verification objects $\{\mathcal{VO}_i\}$. If Γ_{new} can be covered by all $\mathcal{VO}_i$, then the client needs not communicate with any server.

Otherwise, let $\{SP_x\}$ be the set of servers whose verification objects do not cover Γ_{new}. It suffices for the client to request the updated verification objects from the servers in the set $\{SP_x\}$.

6 Experimental Study

We evaluate all methods on two large real datasets[5]: CN (China, 0.64M points) and US (United States, 2.09M points). Each point-of-interest stores a lat-long coordinate (16 bytes) and a full geographic name (250 bytes). CN and US cover the domain areas 9.736 million km^2 and 9.826 million km^2, respectively. We also test the scalability of all methods on two types of synthetic datasets: UNI (uniform) and GAU (Gaussian). We followed the literature [2], [31], [25] to generate each GAU dataset such that it contains 100 Gaussian bells of equal size and every Gaussian bell has a standard deviation as 2.5% of the domain space length.

We obtained an implementation of MR*-tree from Yang et al. [27]. For each dataset, we build a MR*-tree for it with the page size as 4 Kbytes. As a remark, for real datasets CN and US, their MR*-trees occupy 263 Mbytes and 844 Mbytes disk space respectively, and their tree building times are 6.5 and 22 minutes respectively. For synthetic datasets, the building times of their MR*-trees scale well with the data size.

The query workload contains trajectories of 100 moving clients generated by trajectory generator in Nutanong et al. [14]. Clients issue moving kNN queries with the default value of k as 10. Each trajectory simulates a client running in Euclidean space and has a location measurement record at every timestamp (1 second); there are 10,000 timestamps in total. Therefore, each client's journey is about 10,000 seconds (i.e., about 2.7 hours). We simulate the scenario of a car (default speed 50 km/hr) moving at the country level.

All experiments were run on a 2.5 GHz Intel PC running Ubuntu with 8 GB of RAM. In each experiment, we report the average performance measure (e.g., communication cost, communication frequency, server and client CPU time) per client journey per timestamp. The *communication cost* is measured as the total size of network packets (including HTTP and TCP headers [12]) per client journey per timestamp. This is the most important measure as we aim to minimize the client's money to pay to the mobile network provider (cf. Section 3).

We implemented all methods in C++ with the cryptographic functions in the Crypto++ library[6]. **VB** denotes our proposed Vertex-Based method (cf. Section 4.2) whereas **BUF** is the baseline buffering method (cf. Section 3.1). The default Δk value used in BUF is 100. For the sake of comparison, in each experiment, we exhaustively try every possible Δk value and use **BUF*** to denote the instance of

5. Real datasets obtained from National Geospatial-Intelligence Agency (http://earth-info.nga.mil/gns/html/namefiles.htm) and U.S. Board on Geographic Names (geonames.usgs.gov)

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

BUF with the lowest communication cost (using the optimal Δk value). We use the same method names (VB, BUF, BUF*) in experiments on a single dataset and on multiple datasets.

6.1 Moving kNN Query

This section focuses on moving kNN queries on a single dataset. We first study the efficiency of our proposed optimizations on VB. Then, we proceed to compare the performance of all methods (VB, BUF, BUF*) with respect to various parameters.

6.1.1 Benefit of optimizations

First, we investigate the benefit of optimizations on the performance of VB. Compared to our preliminary work [30], the new optimizations for VB in this paper are:

1) computation optimization (Section 4.4) that reduces the server and the client CPU time, and
2) $\mathcal{VO}$ compression (Section 4.5.2) that reduces the size of each $\mathcal{VO}$.

By default, all our proposed optimizations have been incorporated into VB. We use NO-CPU to denote the version of VB without computation optimization, and use NO-COMPRESS to denote the version of VB without $\mathcal{VO}$ compression.

We measure the performance of the above methods on real datasets, with all parameters at their default values. Table 2 shows (a) the communication cost, (b) the communication frequency, (c) the server CPU time, and (d) the client CPU time, per each client journey per timestamp. VB incurs lower communication cost than NO-COMPRESS. Since VB applies $\mathcal{VO}$ compression, the server reports to the client a compact $\mathcal{VO}$, in which nodes that appeared in previous $\mathcal{VO}$ are now represented by tiny tokens. All versions of VB have the same communication frequency because they employ the same safe region (i.e., the order-k Voronoi cell of results).

Observe that VB is more efficient than NO-CPU in terms of both the server and the client CPU cost. This is because the computation optimization (used in VB) is able to reduce the computation cost of the order-k Voronoi cell (and subsequent condition checking) significantly.

We also conduct experiments with varying parameters (k, query speed, data size, and data distribution), and find that VB incurs much lower CPU time and communication cost than NO-CPU and NO-COMPRESS respectively. Thus, we only use VB in subsequent experiments.

TABLE 2
Effect of optimizations on VB, parameters at default

Performance per client per timestamp	CN dataset			US dataset		
	VB	NO-COMPRESS	NO-CPU	VB	NO-COMPRESS	NO-CPU
comm. cost (Kbytes)	0.5516	2.2706	0.5516	3.1846	15.9386	3.1846
comm. frequency	0.0093	0.0093	0.0093	0.0252	0.0252	0.0252
server CPU (ms)	0.040	0.040	0.074	0.25	0.25	0.43
client CPU (ms)	0.017	0.018	0.064	0.10	0.10	0.35

6.1.2 Effect of query speed

We study the effect of query speed on the performance of VB, BUF and BUF*. Figure 11 shows (a) the communication cost, (b) the communication frequency, (c) the server CPU time, and (d) the client CPU time, per each client journey per timestamp, with respect to different query speeds, on the CN dataset.

The communication cost of VB is lower than that of BUF* (see Figure 11a) because VB constructs compact $\mathcal{VO}$ and also applies our communication optimizations. Clearly, the communication cost of BUF (without the optimal Δk) is worser than VB. In Figure 11b, when the query speed increases, a client leaves its buffer region / safe region sooner, so its communication frequency increases. Observe that lower communication frequencies (in BUF/BUF*) do not necessarily imply a lower communication cost than our method VB.

Since the communication frequency increases with the client's speed, the server and the client CPU time also rise accordingly (see Figures 11c and d). The server and client CPU times of all methods remain low. At the client side, BUF incurs less CPU time than BUF*. This is because the optimal Δk (used in BUF*) is larger than the default $\Delta k = 100$ (used in BUF).

The experimental results of varying query speed on the US dataset are presented in Figure 12. The trends on US are similar to those observed on CN.

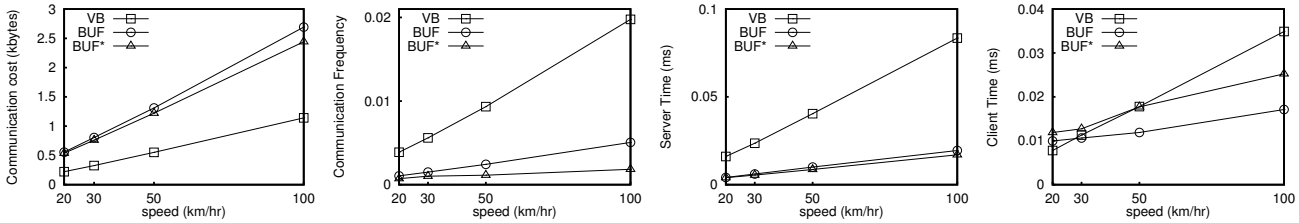
6.1.3 Effect of k

Figure 13 shows the effect of k on the performance of the methods, on the CN dataset. The communication cost of VB is still lower than BUF / BUF* (see Figure 13a), thanks to our $\mathcal{VO}$ reuse and compression techniques. When k increases, the communication frequency of BUF and BUF* stays low (see Figure 13b) because k is small compared to the value of Δk. However, since they incur high communication cost in each communication, their total communication costs are higher than VB. Figures 13c,d show that the server and client CPU times of all methods remain low even when k increases.

The experimental results of varying k on the US dataset are presented in Figure 14. Results similar to the CN dataset are observed.

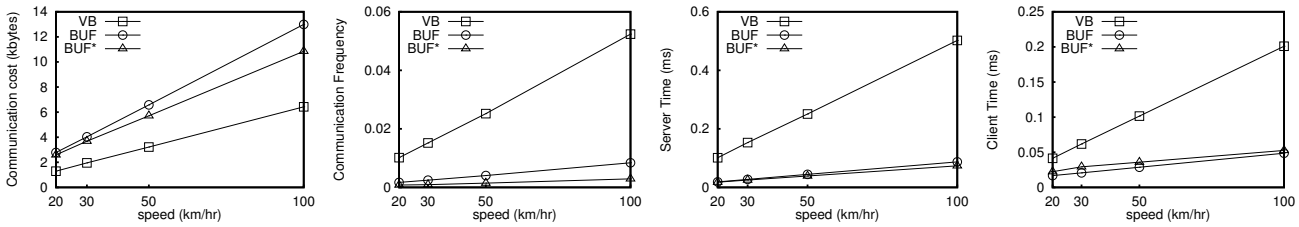
6.1.4 Effect of data size

We also study the scalability of VB, by generating synthetic datasets of different sizes. Figure 15 shows the performance of the methods in different uniform data sizes. The communication cost rises (Figure 15a) because of the increase in the communication frequency between the server and the client (Figure 15b). When the data size increases, the data density increases and thus the buffer region / safe region in the methods shrink. Similarly, the server CPU time (Figure 15c), and the client CPU time (Figure 15d) of all methods. In terms of the most important measure — the communication cost, VB consistently outperforms the baseline methods and the gap becomes more pronounced as the data size increases.



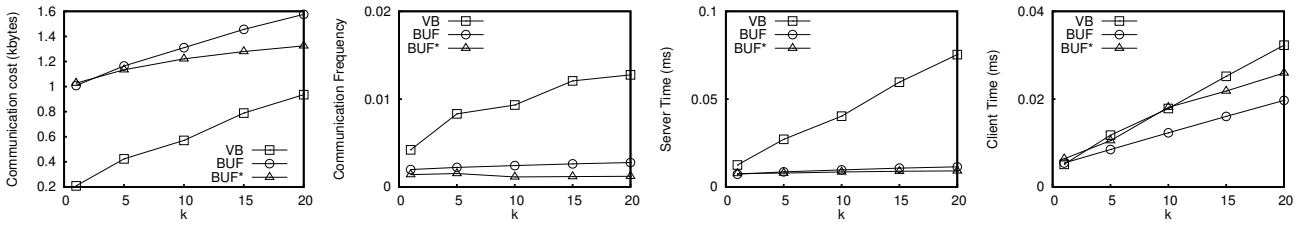
(a) comm. cost (Kbytes) (b) comm. frequency (c) server CPU time (ms) (d) client CPU time (ms)

Fig. 11. Effect of query speed (CN dataset)



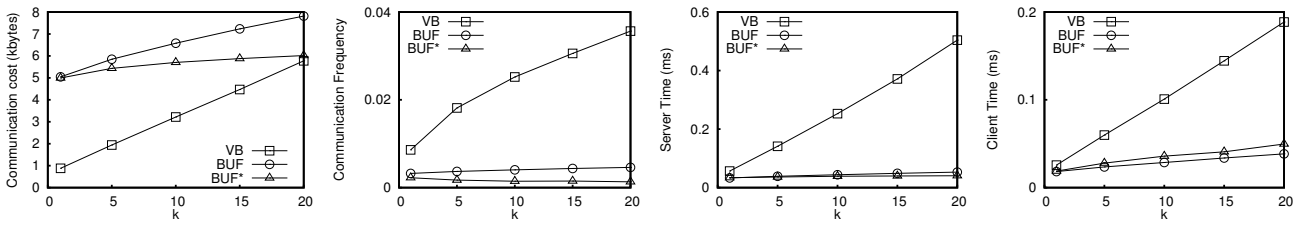
(a) comm. cost (Kbytes) (b) comm. frequency (c) server CPU time (ms) (d) client CPU time (ms)

Fig. 12. Effect of query speed (US dataset)



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

Fig. 13. Effect of k (CN dataset)



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

Fig. 14. Effect of k (US dataset)

Figure 16 plots the performance of the methods on Gaussian datasets with different data sizes. Similar to the results on the uniform datasets, our method has better communication cost than the baseline methods.

6.1.5 Comparison on the client's money to pay

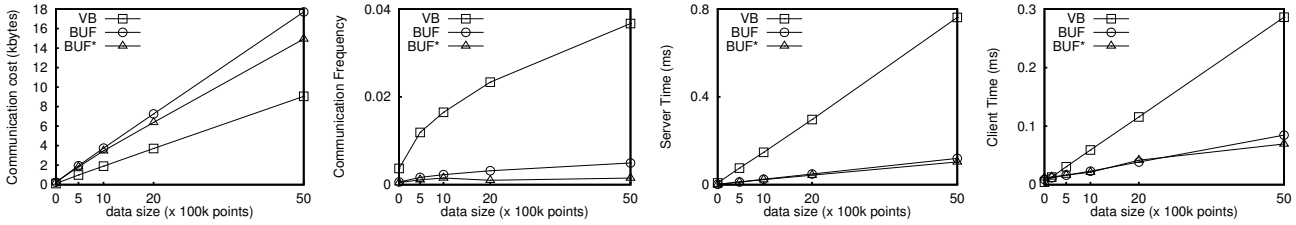
Table 3 shows the client's money (paid to the mobile service provider) using the implemented methods on real datasets, with all parameters at their default values. We calculate the client's money (per hour) by using the communication cost and the rate \$0.49/MB (cf. Section 1). Observe that VB incurs much less client's money than BUF and BUF* because of the low communication cost in VB.

TABLE 3
The client's money to pay, parameters at default

	CN dataset			US dataset		
	VB	BUF	BUF*	VB	BUF	BUF*
comm. cost (Kbytes) per client per timestamp	0.5516	1.31	1.22	3.1846	6.57	5.70
money (\$ per hour)	\$0.95	\$2.26	\$2.10	\$5.48	\$11.32	\$9.82

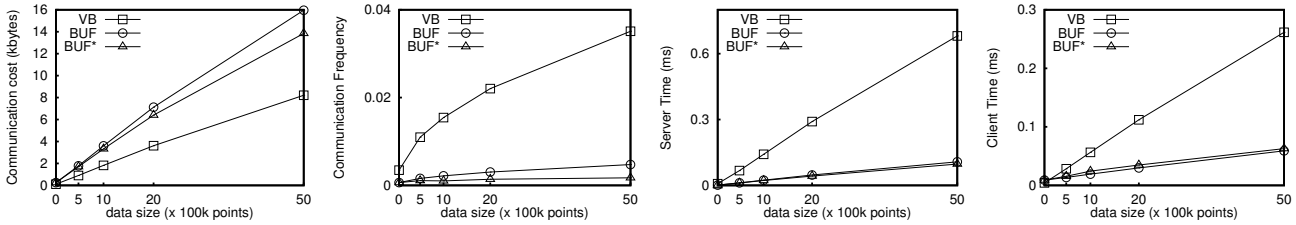
6.2 Moving Multi-Dataset kNN Query

We then evaluate the performance of the methods on (moving) multi-dataset kNN queries. We obtained five representative datasets from U.S. Board on Geographic Names (geonames.usgs.gov): D_1 (22,956 airports), D_2 (32,264 post offices), D_3 (67,756 parks), D_4 (205,847 schools), D_5 (227,611 churches). We assume that a client queries m datasets ($D_1, D_2, \cdots, D_m$), where $m \in [1, 5]$.



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

Fig. 15. Effect of data size (UNIFORM)



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

Fig. 16. Effect of data size (GAUSSIAN)

Again, the communication cost of BUF* rises with m (see Figure 17a). Figure 17b plots the communication frequency of the methods with respect to m. When m increases, the density of queried data points increases and so does the communication frequency. The communication frequency of BUF* rises at a slower rate just because its optimal Δk value increases with m. VB achieves the lowest communication cost, due to our proposed communication optimizations. All methods incur low server and client CPU times (see Figure 17c,d).

7 Conclusion

In this paper, we presented an efficient method to authenticate moving kNN queries. We proved that our method is $\mathcal{VO}$-optimal, i.e., the verification object ($\mathcal{VO}$) has the minimal size with respect to the given tree. We developed optimization techniques that can further reduce the computation cost, communication frequency and cost between a moving client and the LBS. Furthermore, we extended our solution to handle moving kNN queries that involve multiple datasets. Experimental results show that our authentication method achieves low communication cost and CPU overhead.

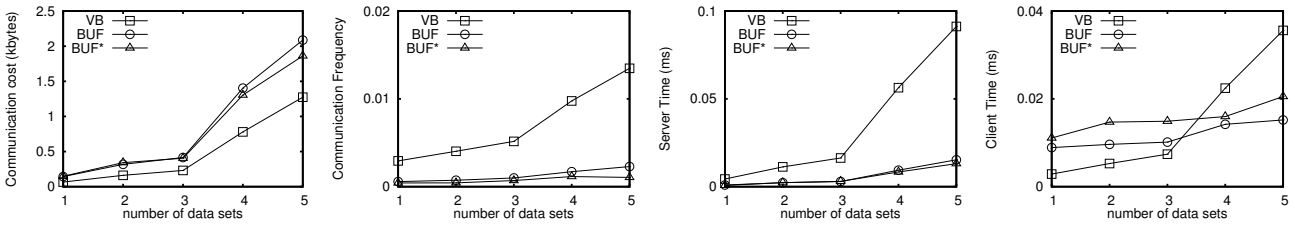
An interesting future work is to develop moving kNN query authentication technique for a location registry [4] that manages mobile users' private locations. In this scenario, an additional requirement is to avoid disclosing non-result points to the query client [4]. It is challenging to authenticate a safe region (i.e., order-k Voronoi cell) as its vertices and edges may allow an adversary to infer some non-result points.

Acknowledgments

This work was supported by grant PolyU 5302/12E 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] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *SIGMOD*, pages 211–222, 2001.
[3] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang. Continuous monitoring of distance based range queries. *TKDE*, 2010.
[4] H. Hu, J. Xu, Q. Chen, and Z. Yang. Authenticating location-based services without compromising location privacy. In *SIGMOD*, 2012.
[5] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi. Verifying spatial queries using voronoi neighbors. In *GIS*. ACM, 2010.
[6] 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.
[7] A. Kundu and E. Bertino. Structural Signatures for Tree Data Structures. *PVLDB*, 1(1):138–150, 2008.
[8] A. Kundu and E. Bertino. How to Authenticate Graphs without Leaking. In *EDBT*, pages 609–620, 2010.
[9] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *SIGMOD*, pages 121–132, 2006.
[10] 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.
[11] X. Lin, J. Xu, and H. Hu. Authentication of location-based skyline queries. In *CIKM*, 2011.
[12] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, 2011.
[13] R. C. Merkle. A Certified Digital Signature. In *CRYPTO*, 1989.
[14] 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.
[15] A. Okabe, B. Boots, K. Sugihara, and S. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, second edition, 2000.
[16] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *SIGMOD*, 2005.



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

Fig. 17. Effect of m, for multi-dataset queries

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

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

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

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

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

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

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

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

[25] X. Xiong, M. F. Mokbel, and W. G. Aref. Lugrid: Update-tolerant grid-based indexing for moving objects. In *MDM*, page 13, 2006.

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

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

[28] 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.

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

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

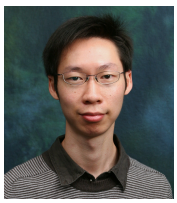
[31] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, pages 443–454, 2004.

[32] D. Yung, E. Lo, and M. L. Yiu. Authentication of Moving Range Queries. In *CIKM*, 2012.

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

Duncan Yung received his Bachelor and Master degree in Computer Science in 2009 and 2012 from the University of Hong Kong and Hong Kong Polytechnic University respectively. His research focuses on different kinds of spatial data management and authentication.

Yu Li received the bachelor's degree in 2010 from Northwestern Polytechnical University, China. She is currently a PhD student in Hong Kong Polytechnic University, under the supervision of Dr. Man Lung Yiu.



Eric Lo received his PhD degree in 2007 from ETH Zurich. He is currently an assistant professor in the Department of Computing, Hong Kong Polytechnic University. He research interests include query processing, query optimization, and large-scale data analysis.

Man Lung Yiu received the bachelor's degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an assistant professor in the Department of Computing, Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular query processing topics on spatiotemporal data and multidimensional data.

APPENDIX

PROOF of Theorem 1: To prove the theorem, we prove that (i) any point p^* outside KVR *cannot* be kNN (Lemma 3), and (ii) any points p^* outside $SRVR$ *cannot* alter the safe region (Lemma 4). Since the verification region Γ is the union of KVR and $SRVR$, so the theorem is proved if Lemma 3 and Lemma 4 hold. □

Lemma 3. [Points outside KVR are not kNN]
We have $\forall p^*$ outside KVR, $p^* \notin S$, where S is the kNN result set.

Proof: KVR is a circle with radius γ, where γ is the distance between query point q and its k-th nearest neighbor. There are k points in KVR such that their distance from $q \leq \gamma$. However, $\forall p^*$ outside KVR, $dist(p^*, q) > \gamma$. Therefore, this lemma holds. □

Lemma 4. [Points outside $SRVR$ 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 dataset D. Let p^* be a point in $D \backslash S$. If p^* is outside $SRVR$ then $\mathcal{V}_k(S, D) = \mathcal{V}_k(S, D \backslash \{p^*\})$.

Proof: To begin, we first have to state two facts. (i) A half-plane is convex. (ii) The intersection of half-planes is also convex [15]. $---(\blacklozenge)$

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

Since all generators of Voronoi edges are inside $SRVR$, p^* is not a generator.$---(\spadesuit)$

By $(\bigstar)$ and $(\spadesuit)$, $\mathcal{V}_k(S, D)$ is inside $\bigcap_{p \in S} \perp(p, p^*)$ and p^* cannot contribute to a Voronoi edge of $\mathcal{V}_k(S, D)$. Hence, $\mathcal{V}_k(S, D \backslash \{p^*\}) \cap \bigcap_{p \in S} \perp(p, p^*)$ is still $\mathcal{V}_k(S, D \backslash \{p^*\})$. By definition of Voronoi cell, $\mathcal{V}_k(S, D \backslash \{p^*\}) \cap \bigcap_{p \in S} \perp(p, p^*)$ = $\mathcal{V}_k(S, D)$. Therefore, $\mathcal{V}_k(S, D)$=$\mathcal{V}_k(S, D \backslash \{p^*\})$. □

Proof of Lemma 2: First, from Lemma 1, we know $S' = S$. $---(\blacklozenge)$

Next, Lines 1–2 of Algorithm 2 ensure that all data points $p \in D'$ and all non-leaf entries $e \in R'$ in $\mathcal{VO}$ are originated from the map provider [27]. Then, the client algorithm checks whether any non-leaf entry $e \in R'$ intersect $SRVR$, and if no, it regards the safe region $V_k(S', D')$ as correct, i.e., $V_k(S', D') = V_k(S, D)$. We prove its correctness by contradiction.

Assume the client regards $V_k(S', D') = V_k(S, D)$ even when a non-leaf entry $e \in R'$ in $\mathcal{VO}$ intersects $SRVR$. When e intersects $SRVR$, it is possible that $\exists p'$ in entry e inside $SRVR$.

Let P' be the set of p' in entry e that is inside $SRVR$ and P^* be the set of p^* in entry e that is outside $SRVR$.

By Lemma 4, $p' \in P'$ may (or may not) alter the safe region. Since e covers points in $P' \cup P^*$, a point p in e may (or may not) alter the safe region, i.e., it is possible that $V_k(S, D \backslash \{P' \cup P^*\}) \neq V_k(S, D)$. $---(\bigstar)$

In our VB method, the client computes the safe region from D' (the set of data points in $\mathcal{VO}$), as points in $P' \cup P^*$ are represented by a non-leaf entry e, they are not in D', therefore, we have $D' = D \backslash \{P' \cup P^*\}$. Hence, we have $V_k(S', D') = V_k(S', D \backslash \{P' \cup P^*\})$.

By $(\blacklozenge)$, we get $V_k(S', D') = V_k(S', D \backslash \{P' \cup P^*\}) = V_k(S, D \backslash (P' \cup P^*)$. Combining this with $(\bigstar)$, we obtain that $V_k(S', D')$ may not equal to $V_k(S, D)$, i.e., it possible that $V_k(S', D') \neq V_k(S, D)$, which contradicts with the assumption. □

PROOF of Theorem 2: To prove the theorem, we need to prove that (i) Points p inside Γ are sufficient for client to verify the correctness of the query result and the safe region (Lemma 5). (ii) All points p inside Γ are necessary for the client to verify the correctness of the query result and the safe region (Lemma 6). Since the establishment of (i) shows that all points in Γ are sufficient and the establishment of (ii) shows that all points in Γ are necessary, the $\mathcal{VO}$ contains the minimum data points. Furthermore, since we apply depth-first-search on T_D with the search range as Γ, the number of non-leaf entries inserted into $\mathcal{VO}$ is also minimum. □

Lemma 5. [Points p inside Γ are sufficient for client to verify the correctness of kNN result and its safe region]

Proof: Lemma 1 implies that points p inside KVR are sufficient for the client to verify the correctness of the query result. Lemma 2 implies that points p inside $SRVR$ are sufficient for the client to verify the correctness of the safe region. Since Γ is the union of KVR and $SRVR$, points p inside Γ are sufficient for client to verify the correctness of query result and safe region. □

Lemma 6. [Points p inside Γ are necessary for client to verify the correctness of kNN result and its safe region]

Proof: First, we prove that all points p inside KVR are necessary for the client to verify the correctness of the query result.$---(\blacklozenge)$ Recall that KVR is a circle with radius γ, where γ is the distance of k-th nearest neighbor of q. For all p inside KVR, $dist(p, q) \leq \gamma$, so p is kNN result. Since the query result must be returned to the client, all p inside KVR are thus necessary to be included in the $\mathcal{VO}$,

Next, we prove that all points p inside $SRVR - KVR$ are necessary for the client to verify the correctness of the safe region.$---(\bigstar)$

First, points inside $\circledast$ are generators so they are all necessary. We continue to prove that points inside $SRVR-KVR-\circledast$ are necessary. We do the proof by contradiction. Suppose $\exists p'$ inside $SRVR-KVR-\circledast$ that is not necessary for client to verify the correctness of $V_k(S', D')$. This implies that p' cannot alter the safe region and $V_k(S', D') = V_k(S', D'\backslash\{p'\})$.

By definition of $SRVR$, if p' inside $SRVR - KVR - \circledast$, $\exists\psi$ inside $V_k(S', D'\backslash\{p'\})$ such that $dist(\psi, p') < \max_{p\in S'} dist(\psi, p)$. (Note that p' does not belong to $\circlearrowleft$.) Therefore p' is kNN query result at $\psi.---(\clubsuit)$ Since p' is outside KVR, by Lemma 3, p' is not kNN query result.$---(\spadesuit)$

By $(\clubsuit)$ and $(\spadesuit)$, we derive that kNN result at query point q and query point ψ are not the same. By definition of safe region, $\forall q$ inside $V_k(S', D')$, they should have the same query result set S'. Since kNN query result at query point q and query point ψ are not the same, ψ is outside $V_k(S', D')$.

Since ψ is outside $V_k(S', D')$ and ψ is inside $V_k(S', D'\backslash\{p'\})$, we know that $V_k(S', D') \neq V_k(S', D'\backslash\{p'\})$. This implies that p' can alter the safe region and p' is necessary for client to verify the correctness of the safe region. This leads to a contradiction.

Finally, since $KVR \cup (SRVR-KVR) = KVR \cup SRVR$, and Γ is the union of KVR and $SRVR$, by $(\blacklozenge)$ and $(\bigstar)$, all points p' inside Γ are necessary for client to verify the correctness of query result and safe region. $\square$