

Authenticating Location-Based Skyline Queries in Arbitrary Subspaces

Xin Lin, Jianliang Xu, *Senior Member, IEEE*, Haibo Hu, and Wang-Chien Lee

Abstract—With the ever-increasing use of smartphones and tablet devices, location-based services (LBSs) have experienced explosive growth in the past few years. To scale up services, there has been a rising trend of outsourcing data management to Cloud service providers, which provide query services to clients on behalf of data owners. However, in this data-outsourcing model, the service provider can be untrustworthy or compromised, thereby returning incorrect or incomplete query results to clients, intentionally or not. Therefore, empowering clients to authenticate query results is imperative for outsourced databases. In this paper, we study the authentication problem for location-based arbitrary-subspace skyline queries (LASQs), which represent an important class of LBS applications. We propose a basic Merkle Skyline R-tree method and a novel Partial S4-tree method to authenticate one-shot LASQs. For the authentication of continuous LASQs, we develop a prefetching-based approach that enables clients to compute new LASQ results locally during movement, without frequently contacting the server for query re-evaluation. Experimental results demonstrate the efficiency of our proposed methods and algorithms under various system settings.

Index Terms—Query authentication, skyline queries, location-based services

1 INTRODUCTION

WITH the rapid development of mobile handset devices (such as smartphones and tablet computers), wireless communication, and positioning technologies in the past decade, Location-based services (LBSs) have prospered. Users carrying location-aware mobile devices are able to query LBSs for surrounding points of interest (POIs) anywhere and at any time. Among the many types of location-based queries, one important class is *location-based skyline queries*. These queries take into account both the spatial and non-spatial attributes of the POIs [11], [20], [40]. A representative example is finding nearby restaurants with good food, where the distance to the querying user is a spatial attribute and the goodness of the food is a non-spatial attribute. The query returns a set of restaurants that are closer to the querying user and/or have better food than those not returned. In general, while spatial objects can have a long list of non-spatial attributes—such as food quality, service, hygiene, environment, and price—only a *small subset* of these attributes (termed a *subspace*) is of interest to a particular user in a single query. Moreover, different

users may have different preferences—e.g., Mary prefers taste, whereas Tom is concerned about hygiene, environment, and price. In this paper, we call these skyline queries *location-based arbitrary-subspace skyline queries* (LASQs).

To scale up LBSs along with their ever-growing popularity, a rising trend is to outsource data management and service provisioning to Cloud service providers (CSPs) such as Amazon EC2 and Google App Engine [1], [10]. More specifically, a data owner delegates its data to a CSP, which in turn provides query services to clients on behalf of the data owner. While such an outsourcing model is advantageous in terms of cost, performance, and flexibility in resource management, it brings a great challenge to query integrity assurance [17], [24]. If the CSP is untrustworthy or compromised, it may return incorrect or incomplete query results to clients (intentionally or not) for various reasons:

- The CSP may return incorrect results unintentionally because of bugs in the implementation of query processing algorithms.
- The CSP (or the adversary who compromised it) may intentionally tamper with the query results. For example, in the restaurant-finding scenario mentioned above, a restaurant may be ranked higher than other restaurants just because the CSP is sponsored by that restaurant.
- To cut costs or avoid performance bottlenecks in peak hours, the CSP may return incomplete results by carrying out the query evaluation process partially.

Therefore, in the data-outsourcing model, there is a need for clients to authenticate the *soundness* and *completeness* of query results,¹ where *soundness* means that the original

1. In this paper, we focus on static or infrequently updated data such as POIs. The issue of guaranteeing the freshness of query results is beyond the scope of this study.

- X. Lin is with the Department of Computer Science and Technology, East China Normal University, Shanghai 200241, China, and with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. E-mail: xlin@cs.ecnu.edu.cn.
- J. Xu and H. Hu are with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. E-mail: {xujl, haibo}@comp.hkbu.edu.hk.
- W.-C. Lee is with the Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802 USA. E-mail: wlee@cse.psu.edu.

Manuscript received 8 Dec. 2012; revised 14 May 2013; accepted 7 July 2013. Date of publication 4 Aug. 2013; date of current version 29 May 2014.

Recommended for acceptance by G. Miklau.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier 10.1109/TKDE.2013.137



Fig. 1. Authenticated query processing.

data is not tampered with by any third party (including the CSP), and *completeness* means that no valid result is missing. This leads to a problem known as *authenticated query processing* [17], [24], which has been studied for various spatial queries, including range queries [8], [9], [36], top- k queries [5], k NN queries [10], [39], and shortest-path queries [38].

Fig. 1 shows a general framework of authenticated query processing. The data owner obtains, through a certificate authority (e.g., VeriSign), a pair of private and public keys of digital signatures. Before delegating a spatial dataset to the CSP, the data owner builds an authenticated data structure (ADS) of the dataset. To support efficient query processing, the ADS is often a tree-like index structure, where the root is signed by the data owner using his/her private key. The CSP keeps the spatial dataset, as well as the ADS and its root signature. Upon receiving a query from the client, the CSP returns the query results, the root signature, and a verification object (VO), which is constructed based on the ADS. The client can authenticate the correctness of the query results using the returned VO, the root signature, and the data owner's public key.

In a preliminary study [21], we have investigated the authentication problem for location-based skyline queries in a fixed space of attributes. In this paper, we extend this study to the general problem of authenticating location-based skyline queries in arbitrary subspaces of attributes (*i.e.*, LASQs). Because a basic solution that returns all results in the full space is inefficient, we propose a new authentication method based on the notion of *signed subspace skyline scope* (S4). We devise a data structure, called *Partial-S4-tree*, which pre-computes, signs, and stores the skyline scopes of some subspaces, so that many redundant objects can be easily identified and safely removed from the VO, thereby minimizing its size and saving the server processing time. To improve the filtering effects, we further propose a storage-budget allocation policy to construct the *Partial-S4-tree* for each spatial object. For continuous LASQs, the concept of *clear area* is introduced to enable a moving client to re-evaluate new results locally. Moreover, we propose an approach to prolong the client's residence time inside a clear area.

In summary, our contributions in this paper are four-fold:

- We identify the problem of authenticating LASQs in outsourced databases. To the best of our knowledge, this study is the first attempt to investigate this problem.
- For a one-shot LASQ authentication, we propose a basic Merkle Skyline R-tree method and a *Partial-S4-tree* method, aiming to reduce the server processing time and minimize the VO size.

- We develop a prefetching-based approach for authenticating continuous LASQs. This approach enables the clients to re-evaluate new LASQ results locally during movement, thus reducing both communication and computation costs.
- We conduct extensive experiments to evaluate the performance of the proposed methods and algorithms. The results show that our proposed methods substantially outperform the basic authentication algorithm by up to 69% in terms of the overall query latency and up to 74% in terms of the VO size.

The rest of this paper is organized as follows. In Section 2, we present the background and related work. In Section 3, we give an overview of the LASQ problem to be studied. In Section 4, we extend our previous work on skyline authentication to a basic authentication method. To further optimize performance, we propose a *Partial-S4-tree* method in Section 5. In Section 6, we propose a prefetching-based approach for authenticating continuous LASQs. The experimental results are presented in Section 7. Finally, in Section 8, we conclude the paper.

2 RELATED WORK

In this section, we review the related work on query authentication and skyline query processing.

2.1 Query Authentication

Authenticated query processing has been studied extensively. Most studies on query authentication [17]–[19], [25]–[27], [37], [38] are based on an ADS called Merkle Hash Tree (MH-tree) [14]. In MH-tree, the digests of index nodes are recursively computed from the leaf nodes to the root. After that, the root digest is signed by the data owner's private key and stored on the outsourced database server. For each user query, this signature is returned to the client along with the query results and a VO for result verification. In contrast, an alternative method is to employ signature aggregation [24], which signs every object in the dataset and generates a VO by aggregating the signatures of the result objects along with some non-result objects (*e.g.*, the objects immediately beyond a query range). However, as the aggregate signature is generated on-the-fly, this method incurs high overhead in query processing and client-side verification. Therefore, in this paper, we focus on authentication methods based on MH-tree.

Following the concept of MH-tree, the query authentication problem has been studied for relational databases [17], [37], data streams [18], [27], and text search engines [25]. Yang *et al.* [36] first introduced this problem to the spatial database domain and studied the authentication of spatial range queries. They proposed an authenticated index structure called MR-tree, which combines the ideas of MB-tree [17] and R*-tree [2]. Yiu *et al.* investigated how to efficiently authenticate k NN queries [39] and shortest-path queries [38]. In [10], Hu *et al.* proposed a novel approach that authenticates spatial queries based on neighborhood information. More recently, in [5], [8], [9] we developed new schemes for

range and top- k query authentication that preserves the location privacy of queried objects.

In our preliminary studies [21], [22], we investigated the authentication of location-based skyline queries in *fixed* subspaces. A new authenticated index structure called MR-Sky-tree (or MSR-tree) was proposed in [21]. The main difference between MR-tree and MSR-tree is that the former indexes the spatial objects while the latter indexes the solution space of spatial objects (in form of a notion called skyline scope). In [23], Lo and Ghinita studied the authentication of group-based spatial skyline queries. They considered spatial attributes only and regarded the distances to query points as the dimensions of a skyline, which is different from our problem studied in this paper.

2.2 Skyline Query Processing

Skyline query processing was first introduced into the database community by Borzanyi *et al.* [4]. A number of algorithms have been developed since then. These algorithms can be divided into two categories. The first category is non-index-based algorithms. The representatives are Black-Nested-Loop (BNL) and Divide-and-Conquer (D&C) [4]. BNL scans the dataset sequentially and compares each new object to the skyline candidates obtained so far. D&C partitions the dataset into several parts, processes them part by part, and finally merges all partial skylines. SFS [6] improves BNL by pre-sorting the dataset. In the Bitmap approach [32], each data point is encoded in a bit string and the skyline is computed on the bit matrix of all data points.

The other category of skyline algorithms is index-based. In [32], a high-dimensional dataset is converted into a one-dimensional dataset and a B⁺-tree is built to accelerate query processing. In [13], an algorithm called NN was proposed based on depth-first nearest-neighbor search via R*-tree. Papadias *et al.* [28], [29] proposed an optimal algorithm, called Branch-and-Bound Skyline (BBS), which is based on best-first nearest-neighbor search. More recently, in [35], a subset of skyline points is collected to approximately represent the distribution of an entire set of skyline points. Lee *et al.* [16] proposed a new index structure called ZBtree to index data points based on a Z-order curve, and developed a novel algorithm ZSearch to process skyline queries.

Huang *et al.* [11] extended the skyline query problem to the context of LBSs and proposed a continuous location-based skyline processing algorithm for moving clients. Zheng *et al.* [40] presented a notion of valid scope for location-based skyline queries, which saves the recomputation if the next query point is within the valid scope. Sharifzadeh and Shahabi [31] defined a variant of skyline query by considering the distance between an object and a set of query points. Subspace skyline is an important application of the location-based skyline problem since the users may concern only a subset of object attributes in their queries. Tao *et al.* [34] has proposed an efficient processing algorithm for subspace skyline queries. Our work is inspired by these prior studies, but focus on the *authentication* of location-based skyline queries in arbitrary subspaces.

TABLE 1
Summary of Notations

Symbol	Meaning
Λ	Full set of non-spatial attributes
$DOM(o, \Lambda')$	Set of o 's non-spatial dominators in subspace Λ'
$S_{o, \Lambda'}$	o 's skyline scope in subspace Λ'
$n.S$	Subspace skyline scope of S4 node n
$n.\Lambda''$	Subspace of S4 node n
$GCE(n)$	Globally covering effect of S4 node n
$\tau(\Lambda')$	Power set of subspace Λ'
$P(o, q)$	Probability of o being a skyline result of q in full space
$BE(n)$	Beneficial effect of S4 node n

3 PROBLEM DEFINITION AND PRELIMINARIES

In this section, we give the formal problem definition and present some preliminaries on skyline query processing. Table 1 summarizes the notations used in this paper.

3.1 Problem Definition

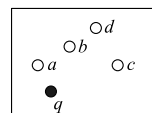
We consider a set of spatial objects \mathcal{O} . Each object $o \in \mathcal{O}$ is associated with one spatial attribute (*i.e.*, location, denoted by $o.l$) and several non-spatial attributes (*e.g.*, food quality and price, denoted by $o.x_i$ for the i -th non-spatial attribute). Assuming Λ is the full set of non-spatial attributes and Λ' is an arbitrary subset of Λ , we say Λ' is a *subspace* of the *full space* Λ . In this paper, we adopt the Euclidean distance metric to measure spatial proximity.

Definition 1 (Non-spatial Subspace Dominance). *Given two objects o and o' , o' non-spatially dominates o in a subspace Λ' (denoted as $o'_{\Lambda'} \triangleleft o$) iff $\forall x_i \in \Lambda'$, $o'.x_i$ is no worse than $o.x_i$. The set of o 's non-spatial subspace dominators is denoted as $DOM(o, \Lambda')$.*

Definition 2 (Subspace Dominance). *Given a query point q and two objects o and o' , if (1) $o'_{\Lambda'} \triangleleft o$ and (2) o' is closer to q than o (*i.e.*, o' also spatially dominates o), then we say o' dominates o in subspace Λ' w.r.t. the query point q . Formally, it is denoted as $o'_{\Lambda'} \triangleleft_q o$.*

Definition 3 (Location-based Arbitrary-subspace Skyline Query (LASQ)). *Given a dataset \mathcal{O} and a query point q , the query $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ returns a subset of objects in \mathcal{O} , each of which is not dominated by any other object in \mathcal{O} in the subspace Λ' w.r.t. q .*

Fig. 2 shows an example of LASQ in the aforementioned restaurant-finding scenario. Suppose we have four restaurants (*i.e.*, a , b , c , and d). The non-spatial attributes of the four restaurants are shown in the table, where high service level, good food quality, and low price are preferred. Assuming a user located at q concerns about the “service level” and “price,” the LASQ result is $\{a, b\}$ because c is dominated by b and d is dominated by both a and b in the attributes “service level” and “price”; on the other hand,



Restaurant	Service level	Food quality	Price
a	4	3	15
b	5	4	13
c	5	4	14
d	3	5	18

Fig. 2. Example of LASQ.

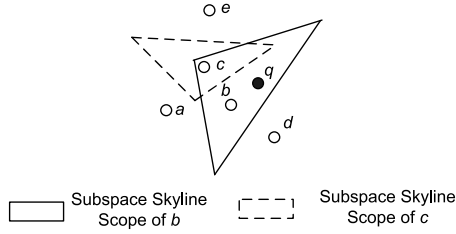


Fig. 3. Subspace skyline scopes.

if the user concerns about the “food quality” and “service level”, the LASQ result becomes $\{a, b, d\}$.

LASQ Authentication Problem. Given an LASQ query, the authentication problem studies how the client can verify the correctness of the query results returned by the CSP. It involves three correlated issues (see Fig. 1): i) ADS design and signature generation by the data owner; ii) online VO construction for each LASQ query on the CSP server; iii) result verification based on the received VO and signature(s) on the client.

3.2 Subspace Skyline Scope

Before we present the detailed authentication techniques, we introduce an important concept, *subspace skyline scope*, which facilitates subspace skyline query processing. Extended from the notion of *skyline scope* we proposed in [20], the subspace skyline scope of an object o is defined as the spatial area in which o is a skyline result in the designated subspace. It is formally defined as follows:

Definition 4 (Subspace Skyline Scope). Given a subspace Λ' and a 2D plane \mathbb{P} , the subspace skyline scope of an object $o \in \mathcal{O}$ is $S_{o, \Lambda'} = \{q \mid q \in \mathbb{P} \wedge o \in \text{LASQ}(\mathcal{O}, q, \Lambda')\}$.

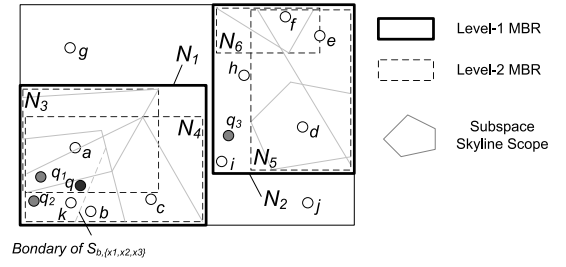
If o has no non-spatial dominators in subspace Λ' , i.e., $\text{DOM}(o, \Lambda')$ is empty, then o must be a skyline result of any query point q . That is, o 's subspace skyline scope is the entire plane \mathbb{P} . Otherwise, o will be a skyline result of a query point q only if it is closer to q than all of its non-spatial dominators in $\text{DOM}(o, \Lambda')$; hence, the corresponding skyline scope is a Voronoi cell of o in the object set $\{o\} \cup \text{DOM}(o, \Lambda')$, which can be computed in $O(|\mathcal{O}| \log |\mathcal{O}|)$ time [3].

With pre-computed subspace skyline scopes, the results of query q in subspace Λ' are those objects whose subspace skyline scopes cover the query point q . Fig. 3 shows an example, where a, d, e have no non-spatial dominators in the subspace, $a_{\Lambda'} \triangleleft b_{\Lambda'} \triangleleft c$, $d_{\Lambda'} \triangleleft b_{\Lambda'} \triangleleft c$, $e_{\Lambda'} \triangleleft b_{\Lambda'} \triangleleft c$. Hence, the subspace skyline scopes of a, d, e are the entire plane, and the subspace skyline scopes of b, c are the Voronoi cells shown in the figure. As such, the subspace skyline result is $\{a, b, d, e\}$ since q is covered by the subspace skyline scopes of these four objects.

It has been shown in [21] that skyline query authentication based on skyline scopes is much more efficient than leveraging a conventional spatial index. Therefore, in this paper we adopt the skyline scope-based method for subspace skyline authentication.

4 BASIC LASQ AUTHENTICATION METHOD

In this section, we propose a basic LASQ authentication method. We start with the authentication problem in a fixed subspace, and then extend it to arbitrary subspaces.



(a)

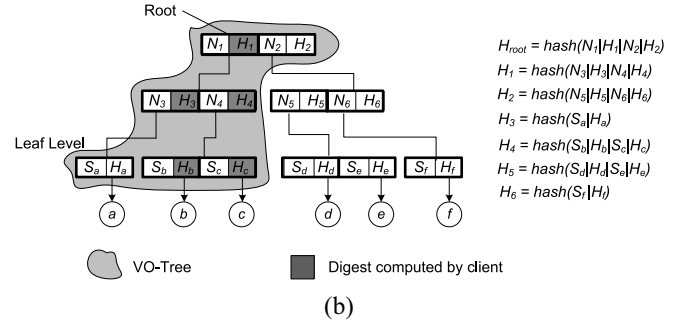


Fig. 4. MSR-tree example: (a) Subspace skyline scopes and MBRs. (b) MSR-tree structure and VO-tree.

4.1 LASQ Authentication in a Fixed Subspace

Design of Authenticated Index Structure. To expedite query processing, we index all the objects' subspace skyline scopes by an R*-tree [2], where the subspace skyline scopes are stored in the leaf nodes as data entries. Additionally, to support query authentication, we follow similar ideas of MB-tree [17] and MR-tree [36] to maintain a series of digests for all index nodes in the tree structure. More specifically, as shown in Fig. 4, each entry in a leaf node stores (1) a pointer to the actual object o , (2) the object's subspace skyline scope S_o , and (3) the object's digest H_o .² Formally,

$$H_o = \text{hash}(o),$$

where $\text{hash}(\cdot)$ is a one-way cryptographic hash function, such as SHA-1. Each entry in a non-leaf node is composed of a pointer to its child node, a minimum bounding rectangle (MBR) denoted by N_i , and a digest H_i of the child node. The digest is formally defined as:

$$H_i = \text{hash}(E_{c_1} | H_{c_1} | E_{c_2} | H_{c_2} | \dots | E_{c_n} | H_{c_n}),$$

where “|” is a concatenation operator, E_{c_k} ($k = 1, 2, \dots, n$) represents the k -th entry in the child node, and H_{c_k} represents the digest of the corresponding entry. Note that in a non-leaf node, E_{c_i} means an MBR, while in a leaf node, E_{c_i} is the subspace skyline scope associated with an object. Thus, the digest of each index node can be computed recursively in a bottom-up fashion. Finally, the digest of the root node is computed and signed by the data owner with his/her private key to generate the root signature, $\text{Sig}(H_{\text{root}})$. Hereafter, this authenticated index structure is called Merkle Skyline R-tree (or MSR-tree for short).

Fig. 4 shows an example of MSR-tree, where the non-spatial attributes of the eleven objects are given in Table 2 (low values are preferred). The subspace skyline scopes of

2. For simplicity, the subscript Λ' in $S_{o, \Lambda'}$ is omitted in this section.

TABLE 2
Non-Spatial Attribute Values of Objects

Object	x_1	x_2	x_3	x_4	Non-Spatial DOM Set
a	8	9	8	11	$\{b, c, g, i\}$
b	8	9	8	10	$\{c, g, i\}$
c	6	8	8	9	$\{g, i\}$
d	10	7	11	9	$\{e, h, i, j\}$
e	7	7	8	8	$\{h, i\}$
f	7	9	9	9	$\{c, e, g, h, i\}$
g	4	8	6	7	\emptyset
h	7	7	4	6	\emptyset
i	3	7	8	7	\emptyset
j	5	2	10	6	\emptyset
k	7	7	3	12	\emptyset

the objects in subspace $\{x_1, x_2, x_3, x_4\}$ are shown in Fig. 4(a). Since objects g through k are not non-spatially dominated by any other object, their subspace skyline scopes span the entire plane. Hence, they are called *non-spatial skyline objects* and should always be returned as results for any subspace skyline query. Since they can be pre-computed and signed by the data owner separately, we do not discuss them in the rest of this paper. Fig. 4(b) shows the structure of the MSR-tree built on the subspace skyline scopes of objects a through f .

Server Query Processing and VO Construction. With the help of MSR-tree, an LASQ is reduced to a point-location query on the indexed subspace skyline scopes. Specifically, starting from the root and going all the way down to the leaf nodes, the server checks whether any child of a node covers the query point. If it does, the node is *unfolded* and inserted into the VO for further checking; otherwise, the node is *pruned* and only its MBR and digest are inserted into the final VO. When visiting a leaf entry associated with an object o , if the corresponding S_o does not cover the query point, both S_o and H_o should be inserted into the VO; otherwise, o is an LASQ result and only S_o is inserted into the VO (H_o can be computed locally by the client based on the received result). It is noteworthy that as the nodes in the VO also form a tree structure, we call it a *VO-tree*.

In general, a VO-tree contains four types of data: 1) the subspace skyline scopes of all objects in the visited leaf nodes; 2) the digests of non-result objects in the visited leaf nodes; 3) the MBRs of all visited non-leaf entries; 4) the digests of the pruned index nodes. The pruned index nodes are also called *non-transparent nodes*, since the client does not know the details of their children nodes.

In the running example of Fig. 4, the query point q is covered by nodes N_1 , N_3 , and N_4 . Thus, these nodes are unfolded and inserted into the VO-tree. On the other hand, node N_2 does not cover q . Hence, only its MBR and digest are inserted into the VO-tree. The final VO-tree is shown as the shaded part in Fig. 4(b), where digests H_b , H_c , H_3 , H_4 , and H_1 are omitted to reduce the communication cost since they can be recursively computed by the client based on the query results $\{b, c\}$.

Client Result Verification. The VO-tree and the root signature ($Sig(H_{root})$), along with the skyline results, are sent to the client after query processing. To verify the correctness of the results, the client checks the following three conditions: 1) the subspace skyline scopes of all result objects should cover the query point q ; 2) no MBRs of the

pruned index nodes and no subspace skyline scopes of the non-result objects cover q ; 3) the root signature matches the root digest computed from the VO-tree. The third condition ensures the soundness of the results, while all the three conditions together guarantee the completeness of the results.

Again consider the running example in Fig. 4. Upon receiving the results $\{b, c\}$ and the VO-tree, the client first verifies conditions 1) and 2), by checking the spatial relationship between q and related subspace skyline scopes and MBRs. By these two conditions, it can ensure that: 1) q is covered by the subspace skyline scopes of result objects, i.e., S_b and S_c ; 2) q is not covered by the MBR of the pruned node (i.e., N_2) and the subspace skyline scope of the non-result object (i.e., S_a). Then the client computes the root digest from the bottom up to check condition 3): first $H_b = hash(b)$ and $H_c = hash(c)$, then $H_3 = hash(S_a|H_a)$, $H_4 = hash(S_b|H_b|S_c|H_c)$, and then $H_1 = hash(N_3|H_3|N_4|H_4)$, and finally $H_{root} = hash(N_1|H_1|N_2|H_2)$. By comparing this H_{root} with the signed root digest $Sig(H_{root})$ from the data owner, the client can verify both the soundness and completeness of the results.

4.2 LASQ Authentication in Arbitrary Subspaces

It is noted that the subspace skyline scopes for LASQs vary with the subspaces. Since it is unrealistic to build an MSR-tree for every possible subspace, the above method cannot be directly adapted to the arbitrary subspace setting.

To save the MSR-tree construction and storage costs, our basic method is to build a global MSR-tree in the full space Λ . When the server receives an LASQ query in subspace $\Lambda' \subseteq \Lambda$, it first computes the query results as if the query were in the full space and constructs the VO-tree based on the full-space MSR-tree. It then obtains the actual results, i.e., $LASQ(\mathcal{O}, q, \Lambda')$, by applying an existing skyline algorithm (e.g., BBS [28]) to the full-space results. Theorem 1 proves that the former must be a subset of the latter. Finally, the server returns the full-space results, the VO-tree, and the root signature, along with the actual results, to the client for result verification.

Theorem 1. *Given two subspaces Λ_1 and Λ_2 , $\Lambda_1 \subseteq \Lambda_2$, for any query point q , if an object o is a result of $LASQ(\mathcal{O}, q, \Lambda_1)$, o must be a result of $LASQ(\mathcal{O}, q, \Lambda_2)$.*

Proof. We prove this theorem by contradiction. Assume there is an object o that is a result of $LASQ(\mathcal{O}, q, \Lambda_1)$ but not an $LASQ(\mathcal{O}, q, \Lambda_2)$ result. There must be at least one object o' such that $o'_{\Lambda_2} \triangleleft_q o$. According to the definitions of dominance and subspace dominance, this implies $o'_{\Lambda_1} \triangleleft_q o$ since $\Lambda_1 \subseteq \Lambda_2$. This contradicts with the assumption that o is a result of $LASQ(q, \Lambda_1, \mathcal{O})$. Hence, the theorem follows. \square

After receiving the results and the VO-tree from the server, the client checks: 1) the full-space results are correct by the same verification method as in Section 4.1; 2) the actual results are sound, i.e., no object in $LASQ(\mathcal{O}, q, \Lambda')$ is dominated by other non-result objects in the full-space result set; 3) the actual results are complete, i.e., all non-result objects must be dominated by at least one result object in $LASQ(\mathcal{O}, q, \Lambda')$.

For the running example in Fig. 4, assume that $\Lambda = \{x_1, x_2, x_3, x_4\}$ and the client issues an $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$, where $\Lambda' = \{x_1, x_2, x_3\}$. Since object b is subspace dominated by object k , it is not an \mathcal{LASQ} result in subspace Λ' . Thus, besides the $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ result (i.e., $\{c\}$), the server will return the VO including: 1) $\mathcal{LASQ}(\mathcal{O}, q, \Lambda)$ results (i.e., $\{b, c\}$);³ and 2) the VO-tree as shown in Fig. 4(b). The client first checks the correctness of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda)$ results using the procedure described in Section 4.1. Based on these results, the client then verifies the correctness of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ results. Specifically, it should verify that object c is not dominated by any other result object of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ but object b is dominated by some object (i.e., k) in subspace Λ' .⁴

5 PARTIAL-S4-TREE METHOD

The basic method proposed in the previous section is easy to implement. However, the communication cost would be prohibitively high if the full space is large, because the VO must include all results for the full space, most of which however are not actual results (hereafter called *redundant objects*). In fact, our experimental results show that when $|\Lambda| = 8$ and $|\Lambda'| = 3$, such redundant objects account for nearly 95% of all returned objects. Moreover, the basic method computes the actual subspace skyline results by identifying the redundant objects on the fly, which incurs a high server processing cost. In this section, we propose a Partial-S4-tree scheme to more efficiently identify the redundant objects and filter out as many of them as possible from the VO.

5.1 Preliminary

We start by a preliminary solution. Similar to the basic authentication method, we assume that a global MSR-tree is built in the full space Λ . To avoid returning redundant objects while still supporting result verification, the data owner pre-computes and signs, for each object o , the skyline scopes of all possible subspaces (i.e., $S_{o,\Lambda'}$ and $\text{Sig}(S_{o,\Lambda'})$, $\forall \Lambda' \subseteq \Lambda$). When evaluating a query in subspace Λ' , the server first finds all the full-space results as in the basic method. Then, for each object r in the result set, the server checks if r is an actual result for Λ' by testing whether its corresponding subspace skyline scope (i.e., $S_{r,\Lambda'}$) covers q . If it does, r is returned to the client; otherwise, it is a redundant object and only its subspace skyline scope $S_{r,\Lambda'}$ and the signature $\text{Sig}(S_{r,\Lambda'})$ are returned as part of the VO. Note that the signatures of all redundant objects can be aggregated into a single signature in the VO (e.g., using the Condensed-RSA scheme [15]⁵). On the other hand, since the soundness of these redundant objects can be proved

3. In real implementation, there is no need to return the results that are already included in the $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ result set (e.g., c in this running example).

4. Object k is also an $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ result as its subspace skyline scope spans the entire space. As noted earlier, such objects are not shown in Fig. 4(b) and omitted for detailed discussions for ease of presentation.

5. Condensed-RSA is an extension of the standard RSA scheme [15]. Given t input messages and their signatures, a Condensed-RSA signature is given by the modular product of individual signatures. It can be verified by multiplying the hashes of all input messages.

by their signatures (or the aggregate signature), there is no need to return their full-space skyline scopes ($S_{o,\Lambda}$'s) in the original VO-tree. To enable this and reduce the VO size, the digest of each leaf node in the full-space VO-tree is re-defined as follows:

$$H_i = \text{hash}(\text{hash}(S_{o_1,\Lambda}|H_{o_1}) \mid \text{hash}(S_{o_2,\Lambda}|H_{o_2}) \cdots \text{hash}(S_{o_n,\Lambda}|H_{o_n})), \quad (1)$$

where o_1, o_2, \dots, o_n are the objects in the leaf node. For example, in Fig. 4, H_4 is re-defined as $H_4 = \text{hash}(\text{hash}(S_{b,\Lambda}|H_b) \mid \text{hash}(S_{c,\Lambda}|H_c))$. Thus, for a redundant object b , we can only return $\text{hash}(S_{b,\Lambda}|H_b)$, instead of $S_{b,\Lambda}$ and b , to verify the root signature of the VO-tree.

The final VO is composed of: 1) the VO-tree constructed based on the full-space MSR-tree (excluding the redundant objects and their full-space skyline scopes ($S_{o,\Lambda}$'s)); 2) the subspace skyline scopes ($S_{o,\Lambda'}$'s) of all redundant objects and their aggregate signature.

For result verification, the client first authenticates the soundness of the returned subspace skyline scopes ($S_{o,\Lambda'}$'s) by using the aggregate signature. It then checks the same three conditions as in the basic method. The only difference is that Condition 3) can now be directly verified by comparing the query point against the authenticated subspace skyline scopes of the redundant objects. In our running example in Fig. 4, as $\Lambda' = \{x_1, x_2, x_3\}$, the server will return $S_{b,\Lambda'}$ and $\text{Sig}(S_{b,\Lambda'})$ (possibly in an aggregate form) to the client.⁶ Since $q \notin S_{b,\Lambda'}$, the client can verify that b is not an \mathcal{LASQ} result.

Clearly, this preliminary solution can significantly improve the efficiency of query authentication, since the redundant objects are excluded from the VO. However, it is almost unrealistic to pre-compute, sign, and store, for each object, all the subspace skyline scopes. In the following, we present how to construct the VO using only a partial set of pre-computed subspace skyline scopes and how to select these subspace skyline scopes.

5.2 Partial-S4-Tree Overview and VO Construction

In this section, we give an overview of the Partial-S4-tree scheme. Given a list of subspace skyline scopes, Theorem 2 shows the conditions by which an object can be filtered out as a redundant object. In particular, given a query $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ and an object o , if q is outside o 's subspace skyline scope in some subspace $\Lambda'' \supseteq \Lambda'$, o must be a redundant object.

Theorem 2. *An object o is redundant and thus can be filtered out from the $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ results by a signed subspace skyline scope $S_{o,\Lambda''}$, if i) $q \notin S_{o,\Lambda''}$ and ii) $\Lambda' \subseteq \Lambda''$.*

Proof. By the first condition, if $q \notin S_{o,\Lambda''}$, o cannot be a result of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda'')$. Since $\Lambda' \subseteq \Lambda''$, according to Theorem 1, o cannot be a result of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ either. Thus, the theorem follows. \square

Following Theorem 2, if a signed subspace skyline scope $S_{o,\Lambda''}$ can facilitate the client to verify that o is not a result of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$, we say $S_{o,\Lambda''}$ *globally covers* the query.

6. In detail, the signature $\text{Sig}(S_{b,\Lambda'})$ should sign not only $S_{b,\Lambda'}$ but also $\text{hash}(S_{b,\Lambda}|H_b)$ to prove that $S_{b,\Lambda'}$ belongs to object b , so that we can use $\text{hash}(S_{b,\Lambda}|H_b)$ directly to compute the root digest of the VO-tree.

Algorithm 1 VO Construction based on Partial-S4-TreesINPUT: $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$, MSR-tree \mathcal{T} OUTPUT: VO \mathcal{VO} , LASQ result set R'

```

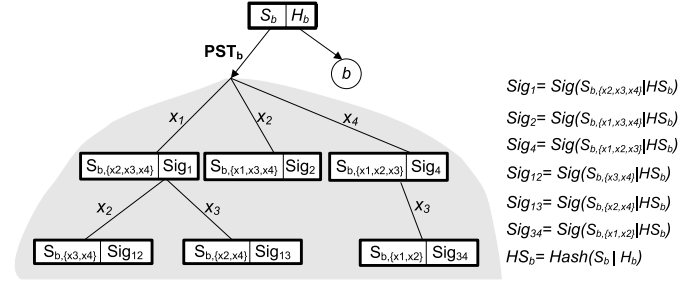
1: Evaluate  $\mathcal{LASQ}(\mathcal{O}, q, \Lambda)$  on  $\mathcal{T}$  using MSR-tree method
2:  $\mathcal{R} \leftarrow$  results of  $\mathcal{LASQ}(\mathcal{O}, q, \Lambda)$ 
3:  $\mathcal{VO}\text{-tree} \leftarrow$  VO-tree constructed by MSR-tree method
4: for each object  $o$  in  $R$  do
5:    $S4\_root \leftarrow$  the root of  $o$ 's Partial-S4-tree
6:    $\mathcal{Q} \leftarrow$  an empty queue
7:   push all children of  $S4\_root$  into  $\mathcal{Q}$ 
8:   while  $\mathcal{Q}$  is not empty do
9:     pop up the first element  $n$  from  $\mathcal{Q}$ 
10:    if  $n.\Lambda''$  contains  $\Lambda'$  then
11:      if  $n.S$  does not cover  $q$  then
12:        remove  $o$  from  $R$ 
13:        insert  $n.S$  into  $\mathcal{VO}$ 
14:        insert  $n.Sig$  into  $SigSet$ 
15:         $o$ 's leaf entry in  $\mathcal{VO}\text{-tree} \leftarrow hash(S_{o,\Lambda}|H_o)$ 
16:        break
17:      else if  $n$  has children nodes
18:        push these children into  $\mathcal{Q}$ 
19: insert  $\mathcal{VO}\text{-tree}$  into  $\mathcal{VO}$ 
20: insert the root digest of  $\mathcal{VO}\text{-tree}$  into  $SigSet$ 
21: insert aggregate signature computed from  $SigSet$  into  $\mathcal{VO}$ 
22:  $R' \leftarrow$  results of  $\mathcal{LASQ}(\mathcal{R}, q, \Lambda')$ 
23: remove all members of  $R'$  from  $R$ 
24: insert  $R$  into  $\mathcal{VO}$ 

```

To find such signed skyline scopes during VO construction, we organize the set of pre-computed subspace skyline scopes of an object into a Partial-Signed Subspace Skyline Scope-tree (*Partial-S4-tree* for short). Each *S4 node* in the tree is a signed subspace skyline scope (hereafter these two terms are used interchangeably).

In a Partial-S4-tree, let $n.\Lambda''$ and $n.S$ denote the subspace and subspace skyline scope of the node n . The root node represents the full space, and the subspace of a child node is fully contained by that of its parent node. We use the running example in Fig. 4 to further illustrate the idea. The authenticated index structure is still a full-space MSR-tree, but now each object in a leaf node is associated with a Partial-S4-tree. Consider object b . Suppose we have pre-computed six of its skyline scopes in the following subspaces: $\{x_2, x_3, x_4\}$, $\{x_1, x_3, x_4\}$, $\{x_1, x_2, x_3\}$, $\{x_3, x_4\}$, $\{x_2, x_4\}$, and $\{x_1, x_2\}$. The Partial-S4-tree associated with b is shown in Fig. 5. The root is b 's full-space skyline scope, which serves as a data entry in the original MSR-tree. Each edge in the Partial-S4-tree is labeled with the attribute(s) by which the subspaces of the parent and the child node differ.

Algorithm 1 shows how to construct the VO for an LSAQ query based on the MSR-tree and Partial-S4-trees. First, by the basic MSR-tree method, we get the full-space results (denoted by R) and construct the VO-tree, which is the base of the final VO (Lines 1-3). Then, for each result in R , we check whether it can be filtered out by its Partial-S4-tree (Lines 4-11). If an S4 node globally covers the point q , we remove the object from R (Line 12), insert the S4 node and its signature into the final VO (Lines 13-14), and update

Fig. 5. Example of Partial-S4-tree for object b .

the VO-tree (Line 15). After that, we compute the final $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ results from the remaining results in R and remove them from R (Lines 22-23). At last, all unfiltered results in R are inserted into the VO (Line 24).

In our running example in Fig. 5, given a query $\mathcal{LASQ}(\mathcal{O}, q, \{x_1, x_2, x_3\})$, object b is not a query result since it is dominated by object k , even though it is a result in the full space $\{x_1, x_2, x_3, x_4\}$. Therefore, the server will insert $S_{b,\{x_1,x_2,x_3\}}$ and Sig_4 (see Fig. 5) into the final VO as it is the first accessed S4 node that globally covers q . Supposing that only the object b is filtered out, the final VO will consist of: 1) the VO-tree as shown in Fig. 4 (excluding b and its full-space skyline scope); 2) unfiltered object c ; 3) $S_{b,\{x_1,x_2,x_3\}}$ and Sig_4 (possibly aggregated with the root signature of the VO-tree).

Upon receiving the VO and LASQ results, the client first checks the soundness of the VO-tree, using the basic MSR-tree authentication method described in Section 4.1. Second, it checks the soundness of filtered objects by verifying whether each filtered object is really globally covered by the associated S4 node. To do so, it hashes each of the S4 nodes of the filtered objects and computes an aggregate digest. By comparing the aggregate digest against that decrypted from the VO (using the data owner's public key), the soundness of filtered objects can be established. Lastly, the client checks the correctness of LASQ results. It should verify that each $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ result object is not dominated by the others in subspace Λ' and all unfiltered redundant objects are dominated by some result object in subspace Λ' .

In practice, although the exact subspace skyline scopes are polygons, the S4 nodes only store their MBRs to minimize the storage and communication costs. Thus, given an $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ query, an object o will be filtered out only if q is outside the MBR of some signed subspace skyline scope $S_{o,\Lambda''}$, where $\Lambda'' \supseteq \Lambda'$. While this slightly makes the filtering less effective, the size of each Partial-S4-tree can be greatly reduced.

5.3 Partial-S4-Tree Construction

As mentioned earlier, it is not feasible for the server to store S4 nodes for all subspaces in the Partial-S4-tree of an object. In this section, we discuss how to construct the Partial-S4-trees that optimize the storage and communication cost. First, we give an overview about how to span S4 nodes to

7. In fact, object c is a result of $\mathcal{LASQ}(\mathcal{O}, q, \{x_1, x_2, x_3\})$, and will be eventually excluded from the final VO. We show it here to exemplify the unfiltered objects.

Algorithm 2 Partial-S4-Tree ConstructionINPUT: Object set \mathcal{O} , storage budget A , full space Λ OUTPUT: Partial-S4-tree of each object o in \mathcal{O}

```

1:  $\mathcal{H} \leftarrow$  an empty max-heap ordered by  $BE$ 
2: for each object  $o$  in  $\mathcal{O}$  do
3:    $o.S4\_root \leftarrow$  new S4 node  $S_{o,\Lambda}$ 
4:   for each  $k$ -limited subset  $\Lambda''$  of  $\Lambda$ 
5:     compute subspace skyline scope  $S_{o,\Lambda''}$  for  $o$ 
6:      $c \leftarrow$  new candidate S4 node with  $S_{o,\Lambda''}$ 
7:     compute  $BE$  value of  $c$  and push  $c$  into  $\mathcal{H}$ 
8:   while  $A > 0$  do
9:      $A \leftarrow A - 1$ 
10:     $maxc \leftarrow$  top element of  $\mathcal{H}$ 
11:     $o \leftarrow$  the object associated with  $maxc$ 
12:    put  $maxc$  under its parent in  $o$ 's Partial-S4-tree
13:    sign  $maxc.S$  and store the signature
14:    re-compute  $BE$  values of  $o$ 's other nodes in  $\mathcal{H}$ 
15:    for each  $k$ -limited subset  $\Lambda'''$  of  $maxc.\Lambda''$ 
16:      compute subspace skyline scope  $S_{o,\Lambda'''}$  for  $o$ 
17:       $c \leftarrow$  a new candidate S4 node with  $S_{o,\Lambda'''}$ 
18:      compute  $BE$  value of  $c$  and push  $c$  into  $\mathcal{H}$ 

```

construct the Partial-S4-trees for all objects within a given storage budget. And then we investigate the spanning rules so that the VO size will be minimized.

Algorithm 2 gives an overview of the Partial-S4-tree construction. The main idea is gradually spanning the Partial-S4-trees from their roots. In each step we greedily select a candidate node which filters out the most number of redundant objects from the full-space result set. The candidate nodes to span are the children nodes of existing S4 nodes. Let BE (Beneficial Effect) denote the filtering effect of a candidate node, and \mathcal{H} a max-heap that sorts the candidates by their BE values. In the beginning, the root of each Partial-S4-tree is initialized to $S_{o,\Lambda}$ (Line 3). We compute the BE values for each root's children nodes and push them into \mathcal{H} (Lines 4-7). The details of BE computation will be discussed at the end of this section. We then iteratively pop up the first element $maxc$ from \mathcal{H} and make it to be an S4 node of o 's Partial-S4-tree until the storage budget is used up (Lines 8-13). In the meantime, the BE values of o 's other candidate nodes are updated with the new S4 node $maxc$ (Line 14), and $maxc$'s children are pushed into \mathcal{H} as new candidate nodes (Lines 15-18).

Recall that in a Partial-S4-tree, the subspace of a child node should be a subset of the parent node's. To limit the number of candidate nodes, we confine the difference between the subspaces of the parent and a child not to exceed k dimensions. We term the subspace of such a child node as a **k -limited subset**, and call the corresponding construction algorithm a greedy- k algorithm.

Nevertheless, even with the k -limited subsets, the overhead of maintaining a global heap \mathcal{H} is still high, especially when the dataset is huge. To alleviate such an overhead, we propose to pre-allocate the storage budget to each object. As such, we only need to maintain a small heap \mathcal{H}_o locally for each object o . The details of storage-budget allocation will be discussed in Section 5.4.

Now we present the details of computing BE values for candidate S4 nodes, with the aim of optimizing the VO size for query authentication. We illustrate the relationship between the VO size and the filtering effect of a candidate node, followed by a concrete analysis of the filtering effect.

The final VO is composed of the VO-tree, redundant objects, subspace skyline scopes of filtered objects, and an aggregated signature. As such, the VO size can be formally quantified as follows:

$$\|VO\| = \|VO_tree\| + \|O\| \cdot (|R| - |F|) + \|S\| \cdot |F| + \|SIG\|, \quad (2)$$

where $\|VO\|$, $\|VO_tree\|$, and $\|SIG\|$ represent the sizes of the VO, the VO-tree, and the aggregated signature; $\|O\|$ and $\|S\|$ are the average sizes of an object and a subspace skyline scope; $|R|$ and $|F|$ are the numbers of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda)$ results and filtered objects, respectively. Since all the above items except $|F|$ are fixed for a given query, $\|VO\|$ will monotonously decrease as $|F|$ grows. Hence, our aim is reduced to maximizing $|F|$ for each query. Consider a query $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$; $|F|$ is the sum for the probabilities for the full-space result objects to be filtered out by their respective Partial-S4-trees:

$$|F| = \sum_{o \in \mathcal{O}} P(o, q) \times PF(o, q, \Lambda'), \quad (3)$$

where $P(o, q)$ is the probability that o is a result of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda)$ and $PF(o, q, \Lambda')$ is the probability that o is filtered out by its Partial-S4-tree. For simplicity, we assume that query points are uniformly distributed. $P(o, q)$ is proportional to the area of $S_{o,\Lambda}$, because o is a full-space result if and only if q is located inside $S_{o,\Lambda}$. As for $PF(o, q, \Lambda')$, it is equal to the probability of a query being globally covered by some node in a Partial-S4-tree. We call this probability of an S4 node the *globally covering effect* of that node. It is formally defined as follows:

Definition 5 (Globally Covering Effect (GCE)). Given an \mathcal{LASQ} query q , the globally covering effect of an S4 node n is the probability that it globally covers q , denoted by $GCE(n)$. The globally covering effect of a Partial-S4-tree T (denoted by $GCE(T)$) is the probability that there is at least one node of T that globally covers q .

The S4 node n of an object o globally covers a query q if and only if q and n satisfy both conditions in Theorem 2. The first condition implies that q must be located inside $S_{o,\Lambda}$ but outside $n.S$, i.e., $q \in S_{o,\Lambda} - n.S$; the second condition requires $\Lambda' \subseteq n.\Lambda''$, i.e., $\Lambda' \in \tau(n.\Lambda'')$, where $\tau(n.\Lambda'')$ is the power set of $n.\Lambda''$. Since the spatial dimension and the non-spatial dimensions are independent, the probability of n covering q is the infinite cartesian product of these two dimensions, i.e., $(S_{o,\Lambda} - n.S) \times \tau(n.\Lambda'')$.

Supposing that the query point is uniformly distributed and the query subspace is randomly chosen, the globally covering effect of a node n can be expressed as:

$$GCE(n) = \frac{Area(S_{o,\Lambda} - n.S) \cdot |\tau(n.\Lambda'')|}{Area(\mathbb{P}) \cdot |\tau(\Lambda)|}, \quad (4)$$

where $S_{o,\Lambda}$ is the full-space skyline scope of object o and \mathbb{P} denotes the entire spatial space.

As such, the globally covering effect of a Partial-S4-tree T can be computed as follows:

$$\begin{aligned}
 GCE(T) = & \sum_{n \in T} (GCE(n)) - \sum_{(n_{i1}, n_{i2} \in T) \wedge (n_{i1} \neq n_{i2})} (GCE(n_{i1} \cap n_{i2})) \\
 & + \dots \\
 & + (-1)^{m+1} \cdot \sum_{(n_{i1}, \dots, n_{im}) \in T} (GCE(n_{i1} \cap \dots \cap n_{im})) \\
 & + \dots + (-1)^{|T|+1} \cdot GCE\left(\bigcap_{n \in T} n\right) l. \quad (5)
 \end{aligned}$$

According to the intersection rule of cartesian products, the GCE of intersected nodes can be defined as follows:

$$GCE\left(\bigcap_i n_i\right) = \frac{Area\left(\bigcap_i (S_{o,\Lambda} - n_i.S)\right) \cdot |\tau\left(\bigcap_i n_i.\Lambda''\right)|}{Area(\mathbb{P}) \cdot |\tau(\Lambda)|}. \quad (6)$$

Next, we show how to compute the BE value for a candidate node c given the existing Partial-S4-tree T of object o . It is defined as the benefit of the increased filtering effect by adding c to T :

$$BE(c) = P(o, q) \cdot (GCE(T \cup \{c\}) - GCE(T)). \quad (7)$$

We exemplify the greedy-1 Partial-S4-tree construction procedure for the running example in Fig. 5. First, four 1-limited subsets of the full space are selected as the initial candidate nodes, *i.e.*, $\{x_2, x_3, x_4\}$, $\{x_1, x_3, x_4\}$, $\{x_1, x_2, x_4\}$, and $\{x_1, x_2, x_3\}$. Then, their BE values are computed according to Equation (7). Since the initial Partial-S4-tree only has a root node, $GCE(T) = 0$. Thus, the BE value of each initial candidate node c_i can be computed as:⁸

$$BE(c_i) = P(o, q) \cdot Area(S_{o,\Lambda} - c_i.S) \cdot 2^3.$$

As the candidate node c_1 ($\{x_2, x_3, x_4\}$) has the largest BE value, it will be selected as the first S4 node. After that, the 1-limited subsets of its subspace are added as new candidate nodes, *i.e.*, $\{x_3, x_4\}$, $\{x_2, x_4\}$, and $\{x_2, x_3\}$, and the selection process continues until the allocated quota is exhausted.

5.4 Storage-Budget Allocation

In this section, we discuss how to allocate the storage budget to the Partial-S4-tree of each object. Algorithm 3 gives an overview of the budget allocation process. Let $BE(i, o)$ denote the beneficial effect of the i -th quota brought to object o . We prove below that $BE(i, o)$ is approximately equal to $\frac{BE(i-1, o)}{2}$. Meanwhile, $BE(1, o)$ is proportional to $Area(S_{o,\Lambda}) \cdot (Area(S_{o,\Lambda}) - Area(S_{o,\Lambda_1}))$, where Λ_1 denotes a 1-limited subset of the full space Λ , and $Area(S_{o,\Lambda_1})$ is the average skyline scope area of the candidate nodes with Λ_1 . In Algorithm 3, $o.BE$, the current beneficial effect of o , is maintained and initialized as $BE(1, o)$ (Line 4). We iteratively select the object with the maximum BE value and allocate a new quota to it. After each allocation, the BE of the selected object is halved (Line 9). The process is repeated until all the storage budget is consumed.

8. As the denominator in Equation (4) is a constant, it does not affect the comparison results and hence is omitted.

Algorithm 3 Partial-S4-Tree Quota Allocation

INPUT: Total storage budget A , Object set \mathcal{O}

OUTPUT: Storage budget for each object

```

1:  $\mathcal{H} \leftarrow$  an empty max-heap ordered by  $BE$  of objects
2: for each object  $o$  in  $\mathcal{O}$  do
3:    $o.quota \leftarrow 0$ 
4:    $o.BE \leftarrow Area(S_o) \cdot (Area(S_o) - Area(S_{o,\Lambda_1})) // BE(1, o)$ 
5:   insert  $o$  into  $\mathcal{H}$ 
6: while  $A > 0$  do
7:    $f \leftarrow$  top object in  $\mathcal{H}$ 
8:    $f.quota \leftarrow f.quota + 1$ 
9:    $f.BE \leftarrow f.BE/2$ 
10:  reinsert  $f$  into  $\mathcal{H}$ 
11:   $A \leftarrow A - 1$ 

```

In the following, we present the detailed proofs and analysis of this algorithm. First, we show that the S4 nodes with a larger subspace Λ'' are likely to achieve larger BE values and thus spanned earlier during the Partial-S4-tree construction. Second, we will show that $BE(i, o)$ is approximately equal to $\frac{BE(i-1, o)}{2}$ for each object o . Finally, the details of $BE(1, o)$ computation will be presented.

We start by showing that the S4 nodes with larger Λ'' are likely to have larger BE values. Assuming that the non-spatial attribute values of each object is uniformly distributed, we obtain the expected size of a non-spatial dominator set as:

$$|\mathcal{DOM}(o, \Lambda')| = \frac{|\mathcal{O}|}{2^{|\Lambda'|}}.$$

We further assume that the objects in $\mathcal{DOM}(o, \Lambda')$ are uniformly distributed in the spatial space and each Voronoi cell for $\mathcal{DOM}(o, \Lambda') \cup \{o\}$ has the same area. Thus, we have:

$$Area(S_{o,\Lambda'}) = \frac{Area(\mathbb{P})}{|\mathcal{DOM}(o, \Lambda')| + 1} = \frac{Area(\mathbb{P})}{\frac{|\mathcal{O}|}{2^{|\Lambda'|}} + 1} \quad (8)$$

$$= \frac{Area(\mathbb{P}) \cdot 2^{|\Lambda'|}}{|\mathcal{O}| + 2^{|\Lambda'|}} \quad (9)$$

Theorem 3. Assuming $|\mathcal{O}|$ is much larger than $2^{|\Lambda|}$, for two S4 nodes n_1 and n_2 , if $|n_1.\Lambda''| > |n_2.\Lambda''|$, $GCE(n_1)$ is expected to be larger than $GCE(n_2)$ without considering the overlapping effect of GCE.

Proof sketch. Since $GCE(n_1)$ and $GCE(n_2)$ are both positive, we can just prove that $\frac{GCE(n_1)}{GCE(n_2)}$ is larger than 1. The detailed proof is given in Appendix B, which is available in the Computer Society Digital Library at <http://doi.ieeecomputersociety.org/10.1109/137>. \square Based on the above claim, we obtain Theorem 4.

Theorem 4. $BE(i, o)$ is approximately equal to $\frac{BE(i-1, o)}{2}$ for each object o .

Proof sketch. According to Equations (5) and (7), the BE of a candidate node c can be expressed by c 's GCE excluding the overlapped part with the GCEs of the nodes that have been selected. The theorem can then be proved by analyzing the remaining GCE. The detailed proof is given in Appendix C, available online. \square

To simplify the budget allocation, we estimate the benefit of an S4 node with a k -limited subset using the same

approximation, since the error is expected to be slim and the probability of selecting a candidate node with a k -limited subset ($k > 1$) is generally low.

In summary, the benefit of selecting a candidate node c of object o can be expressed by the following equations:

$$BE(c) = \left(\frac{1}{2}\right)^{|\mathcal{N}|} BE(1, o), \quad (10)$$

where \mathcal{N} is the set of S4 nodes that have been selected and

$$\begin{aligned} BE(1, o) &= P(o, q) \cdot Area(S_{o,\Lambda} - S_{o,\Lambda_1}) \cdot |\tau(\Lambda_1)| \\ &\propto Area(S_{o,\Lambda}) \cdot Area(S_{o,\Lambda} - S_{o,\Lambda_1}) \\ &= Area(S_{o,\Lambda}) \cdot (Area(S_{o,\Lambda}) - Area(S_{o,\Lambda_1})), \end{aligned} \quad (11)$$

where $Area(S_{o,\Lambda})$ can be obtained from the MSR-Tree, $|\tau(\Lambda_1)|$ is omitted since it is a constant and does not affect the comparison results, and $Area(S_{o,\Lambda_1})$ can be computed based on the object set.

6 AUTHENTICATION OF CONTINUOUS LASQS

Thus far, we have studied the authentication of one-shot LASQs. However, in location-based services, a user may sometimes prefer issuing a query once and monitoring its results continuously. For example, a driver may issue an LASQ query “finding nearby gas stations with cheap gas prices” on his/her driving route. In this section, we study the authentication problem for continuous LASQs.

A naive method to solve the authentication of continuous LASQs is to periodically repeat the one-shot LASQ processing and VO construction procedure presented in Section 5.2. However, this method is extremely inefficient in terms of both the computation and communication costs. Therefore, a better solution to processing and authenticating LASQs is desired. Traditionally, maintaining incremental results and prefetching are two basic methods to process continuous queries. However, the former requires the server to maintain all the results and VOs sent to the clients, which is impractical for a large population of clients. Thus, we adopt the prefetching method in this paper. The idea is to return some additional information to the client as an attachment to the VO so that it can evaluate new results locally. Since the data in the VO or result set can be reused, the attachment only incurs minimal communication overhead.

Let us revisit the running example in Fig. 4. Assume that the client has received the result of $\mathcal{LASQ}(\mathcal{O}, q, \Lambda')$ (where $\Lambda' = \{x_1, x_2, x_3\}$) and its VO from the server. If the query point moves from q to q_1 , the client cannot locally evaluate the new LASQ based on the previously returned information. Even though q_1 is covered by $S_{a,\Lambda}$, since it does not know a 's exact subspace skyline scope of Λ' or its attribute values (spatial or non-spatial), it is unclear whether a is a result of $\mathcal{LASQ}(\mathcal{O}, q_1, \Lambda')$. Obviously, this issue occurs for any query point located inside $S_{a,\Lambda}$. Such an area is thus called *unclear area*. In addition to q_1 , q_3 is also located in the unclear area since previously the client only got the digest and MBR of N_2 in Fig. 4, and does not know the distribution of its children. In contrast, if the next query point is q_2 , the client can decide locally that the new result is still $\{c\}$. We term such an area where the client can perform local evaluation as *clear area*.

Algorithm 4 Greedy Algorithm to Attach Information for Continuous LASQ Authentication

INPUT: The size limitation of attached information: *quota*

- 1: $\mathcal{VO} \leftarrow$ VO constructed by Algorithm 1
- 2: insert all non-transparent nodes in \mathcal{VO} into set \mathcal{CS}
- 3: insert all subspace skyline scopes of non-result objects in \mathcal{VO} into set \mathcal{CS} .
- 4: compute ΔVO and initial Δt of each obstacle shape in \mathcal{CS}
- 5: **while** *quota* > 0 **do**
- 6: select the obstacle shape $s \in \mathcal{CS}$ with maximum $\frac{\Delta t}{\Delta VO}$
- 7: *quota* \leftarrow *quota* - $s \cdot \Delta VO$
- 8: **if** s is a (subspace) skyline scope **then**
- 9: $o \leftarrow$ the object associated with s
- 10: insert o into \mathcal{VO}
- 11: **else** // s is a non-transparent node
- 12: insert the children of s into \mathcal{VO}
- 13: insert the children of s into \mathcal{CS}
- 14: **for** each obstacle shape e in \mathcal{CS} that overlap with s **do**
- 15: update Δt of e

Our idea is to attach some additional information to the client, through which more unclear area can be converted to clear area. The following theorem analyzes the characteristics of the unclear area and clear area.

Theorem 5. *A new query point q_i is located inside the unclear area, if and only if it is covered by at least one obstacle shape in the previously returned information, where an obstacle shape is defined as: 1) the MBR of a non-transparent node of the VO-tree or 2) the subspace skyline scope of a filtered object.*

Proof sketch. The proof is divided into two directions, i.e., “if” and “only if” directions. For “if” direction, we prove the theorem by analyzing which objects cannot be decided as results. For “only if” direction, we prove it by contradiction. The detailed proof is given in Appendix D, available online. \square

Theorem 5 gives us some clue on how to select the additional information attached to the VO. Specifically, if q_i is covered by the subspace skyline scope of some object o in the VO, the server can attach o to the VO; or if q_i is covered by a non-transparent node N , the server can attach N 's children nodes. Ideally, such attaching operations should be carried out iteratively until no obstacle shape covers q_i . However, this results in a huge VO which is useless if the client never moves to q_i . In what follows, we propose a greedy algorithm that attempts to maximize the stay period of the client in the clear area, given a VO size limit. In this algorithm, the objects or the children information of a non-transparent node are attached step by step according to their utility. Each step *discloses* some area (denoted as *disclosed area*) from unclear area into clear area, and prolongs the client's stay in the clear area. Let Δt denote this prolonged time and ΔVO denote the size of attached information in each step. We define the utility metric in our greedy algorithm as $\frac{\Delta t}{\Delta VO}$. Algorithm 4 shows the details of the greedy algorithm. The derivation of Δt can be found in Appendix E, available online.

The final VO of the continuous LASQ comprises three parts: 1) the VO-tree, which may be different from that of the one-shot LASQ because some non-transparent nodes are unfolded; 2) an object set O_T ; 3) some S4 nodes and their signatures. When the client receives the VO, it first verifies the correctness of $\mathcal{LASQ}(\mathcal{O}, q, \Delta')$ results by the same verification procedure as in Section 5.2. Afterwards it periodically checks whether any non-transparent node or the subspace skyline scope of any filtered object covers its current location. If so, it means that the client has moved out of the clear area and should re-issue a new LASQ query to the server. Otherwise, it can still make use of the objects in O_T to determine the new results locally.

7 PERFORMANCE EVALUATION

7.1 Experiment Setup

In this section, we evaluate the performance of our proposed methods and algorithms through simulations. Two real datasets, Qunar and HOU, are used. Qunar contains the spatial information and seven non-spatial attributes of 25,000 hotels (crawled from www.qunar.com). The non-spatial attributes include the price, traffic condition, area, equipment, food, service, and environment. HOU combines the spatial information of 123,000 postal addresses in three metropolitan areas (New York, Philadelphia and Boston) and six non-spatial attributes of 123,000 peoples in USA (available at <http://www.rtreeportal.org> and <http://www.ipums.org>). To evaluate the scalability of the algorithms, we also created a synthetic dataset of 1,000,000 objects from the street segments in California [30]. The non-spatial attributes are synthesized by following the *independent* distribution [4]. We assume that each object has a size of 64 bytes. Since we use MBRs to approximate the real subspace skyline scopes in Partial-S4-trees, the size of a subspace skyline scope is 16 bytes. The spatial space is normalized to a 100,000 unit \times 100,000 unit square, where 1 unit represents 1 meter.

For one-shot query experiments, we evaluate three algorithms: the basic MSR-tree authentication method (denoted by “Basic”, discussed in Section 4), the Partial-S4-tree method with a uniform storage-budget allocation policy (denoted by “Uniform”), and the Partial-S4-tree method with our proposed storage-budget allocation policy (denoted by “Skewed”, discussed in Section 5.4). We set k at 1 for the greedy- k construction algorithm in the Partial-S4-tree method. For continuous query experiments, we simulate the client to move by the random waypoint mobility model [7]. That is, the client randomly selects a point in the plane as the destination and moves towards it at a speed randomly chosen from the range $(0, v)$; upon arrival or expiration of a constant movement period (randomly selected from the range $[0, t_v]$), it selects a new destination and repeats the above process.

We conduct our server-end experiments on a workstation (Xeon X5570, 2.93GHz CPU) running CentOS 5.5 Linux operating system, and simulate the client on a low-end desktop (Intel 1.0GHz CPU). We assume that the client connects to the server through a 3G-equivalent network: 384 Kbps downlink speed and 128 Kbps uplink speed. We employ SHA-1 as the secure hash function, and

TABLE 3
Parameter Settings

Parameter	Value Range	Default
Cardinality of synthetic dataset	10K, 100K, 1,000K	100K
Full-space dimensionality of synthetic dataset	8, 12, 16	8
S4 node storage budget (ratio to # objects)	[30%, 1000%]	300%
Subspace dimensionality in LASQ queries	[2, 5]	3
Max movement speed (v)	–	40 km/h
Max movement period (t_v)	[6s, 20,000s]	–
Buffer size (ratio to whole dataset)	–	5%

Condensed-RSA as the signature function [15]. We adopt a number of metrics, including server elapsed time (for query processing and VO construction), VO size, client verification time, and overall query latency (including server processing, VO transmission, and client verification). To measure the server elapsed time, all objects and indexes are stored on the disk and the maximum storage space for the Partial-S4-trees is 140 MB. The disk page size is 8 KB and the average page access latency is 5 ms. The buffer size is set at 5% of the whole dataset. For Skewed and Uniform algorithms, they access the Partial-S4-trees after traversing the MSR-tree, whereas for the Basic algorithm, it loads the redundant objects directly from the disk. Each measurement is averaged over 100 randomly generated queries. The default settings and value ranges of the system parameters are summarized in Table 3.

7.2 One-Shot LASQ Authentication

7.2.1 Overall Performance Comparison

This subsection compares the overall performance of the three authentication algorithms. As shown in Fig. 6, the Skewed algorithm performs the best in both datasets, whereas the Basic algorithm is the worst. In addition, the performance gaps in HOU are much larger than that in Qunar. The overall query latency is reduced by up to 26% and the VO size is reduced by up to 63%. This is because the dataset cardinality of HOU is much larger so that it has more redundant objects in the full-space LASQ results. As such, our proposed Partial-S4-tree method can filter out more objects from the VO to save the communication cost. The Basic algorithm needs to extract the final LASQ results from a much larger candidate set, *i.e.*, the full-space skyline set. Hence, it also costs more server time than Uniform and Skewed. Regarding the

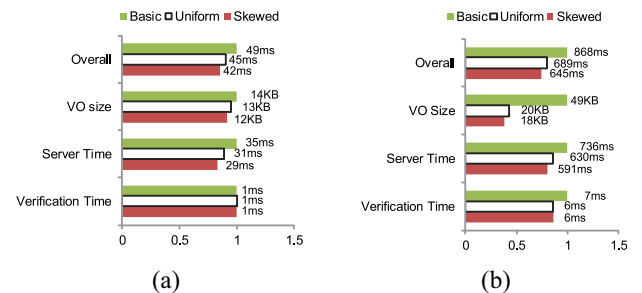


Fig. 6. Overall performance comparison (default settings). (a) Qunar. (b) HOU.

TABLE 4
Index Pre-Computation Costs (HOU)

	MSR-Tree	Partial-S4-Trees
Index construction time	2.3 hours	4.2 hours
Index storage cost	34 MB	130 MB

two Partial-S4-tree-based algorithms, Skewed outperforms Uniform, since it takes objects' filtering effects into consideration when allocating the storage budget for the Partial-S4-trees. As expected, Skewed filters out more redundant objects and beats Uniform in terms of both the VO size and server time. Table 4 shows the pre-computation costs of the MSR-tree and Partial-S4-trees on the larger dataset HOU, which appear acceptable as the pre-computation is an offline and one-time operation. It is further noted that the construction process of the MSR-tree and Partial-S4-trees is fully parallelizable; the pre-computation costs can thus be linearly reduced by using more servers.

7.2.2 Impact of Subspace Dimensionality of Queries

In this subsection, we vary the dimensionality of subspaces in LASQ queries and plot the performance results in Figs. 7(a) and 7(b). As the subspace dimensionality increases, the overall query latencies of all three algorithms are slightly increased. This can be explained by the performance breakdown of VO size and skyline result size in Figs. 7(c) and (d). As shown in Fig. 7(c), the VO sizes of all algorithms decrease when the number of subspace dimensions increases. This is mainly due to the increased size of LASQ results, which are excluded from the VO. As the number of full-space LASQ results is fixed, the redundant objects in the VO are reduced. On the other hand, when the subspace dimensionality increases, the filtering effect of the Uniform and Skewed algorithms degrades as S4 nodes are less likely to globally cover the queries in higher dimensionality. For Uniform and Skewed in dataset HOU, such performance degradation dominates the performance gain by the reduced redundant objects. As a result, the VO sizes of these two algorithms, while slightly growing, are much smaller than that of Basic (see Fig. 7(d)). With higher subspace dimensionality, these algorithms spend more time extracting the final LASQ results from the candidate set, so the server elapsed time increases (see Figs. 7(e) and 7(f)). This is also the main reason why their overall query latency is enlarged with increasing subspace dimensionality. In Figs. 7(g) and 7(h), more verification time is needed for all three algorithms when the subspace dimensionality increases. This is because the client needs more time to check the dominance relationship among a larger set of result objects in higher dimensions.

7.2.3 Scalability Evaluation

To evaluate the scalability of our proposed algorithms, we generate synthetic datasets of large cardinality and high non-spatial dimensionality. As shown in Fig. 8, the Skewed algorithm outperforms the Basic algorithm by up to 69% in terms of the overall query latency and up to 74% in terms of the VO size. A large dataset

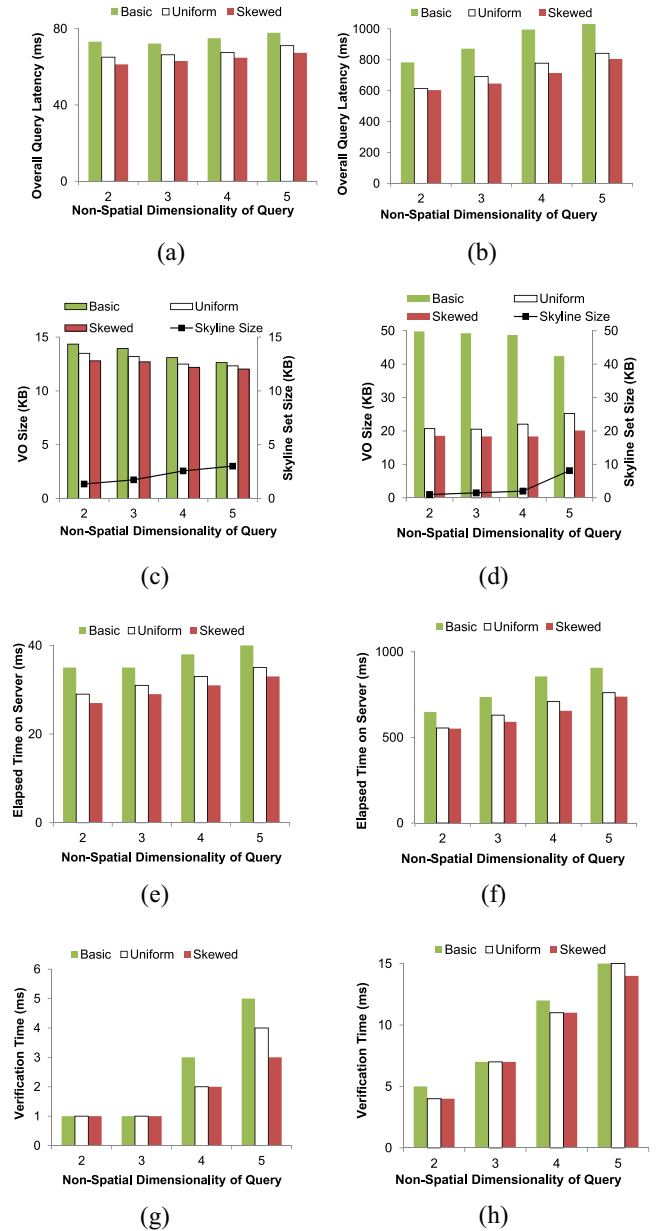


Fig. 7. Impact of subspace dimensionality of queries: (a) Overall (Qunar). (b) Overall (HOU). (c) VO size (Qunar). (d) VO size (HOU). (e) Server time (Qunar). (f) Server time (HOU).

degrades the performance of all algorithms due to more skyline results and a larger VO. Nevertheless, even if the dataset cardinality goes up to 1,000K, the Skewed algorithm can still save nearly 59% of the overall query latency against the Basic algorithm. In Fig. 9, we vary the non-spatial dimensionality of the datasets. As the non-spatial dimensionality increases, the performance of all algorithms deteriorates drastically, especially for the Basic algorithm. In fact, when the non-spatial dimensionality is 16, the cardinality of the full-space LASQ result set is as large as 87,000. This means the Basic algorithm needs to return to the client almost all objects in the dataset. In contrast, the Skewed algorithm can still filter out about 56% of such full-space LASQ results and greatly improve the overall performance.

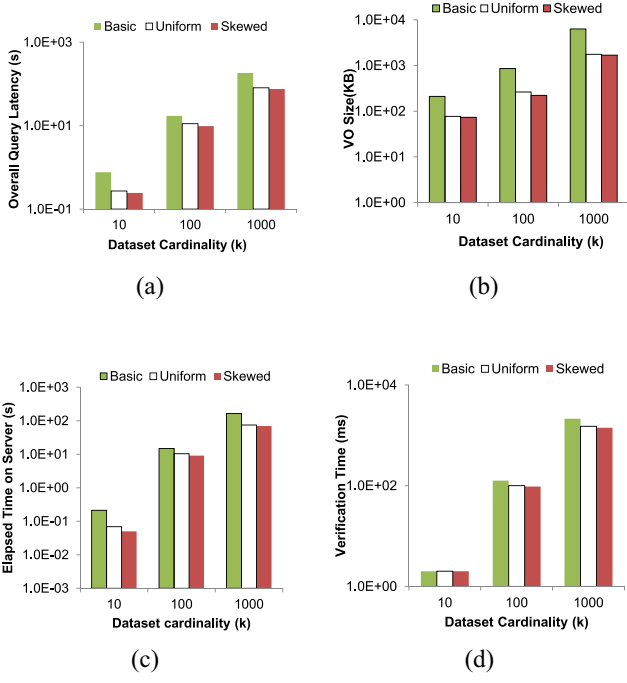


Fig. 8. Effect of dataset cardinality: (a) overall query latency (Logscale). (b) VO size (Logscale). (c) Server elapsed time (Logscale). (d) Client verification time (Logscale).

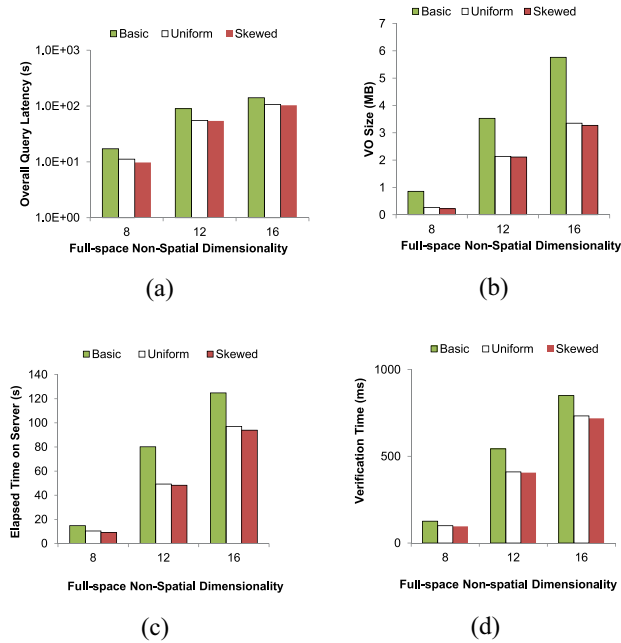


Fig. 9. Effect of non-spatial dimensionality: (a) Overall query latency (Logscale). (b) VO size. (c) Server elapsed time. (d) Client verification time.

7.3 Performance of Authenticating Continuous LASQs

In this subsection, to evaluate the performance of authenticating continuous LASQs, we compare our prefetching-based algorithm (denoted by “Pref”) against the naive algorithm that does not attach any extra information in the VO (denoted by “Non_pref”), using the real dataset Qunar. In “Non_pref”, we also integrate the two techniques (*i.e.*, valid scope and visible region) proposed

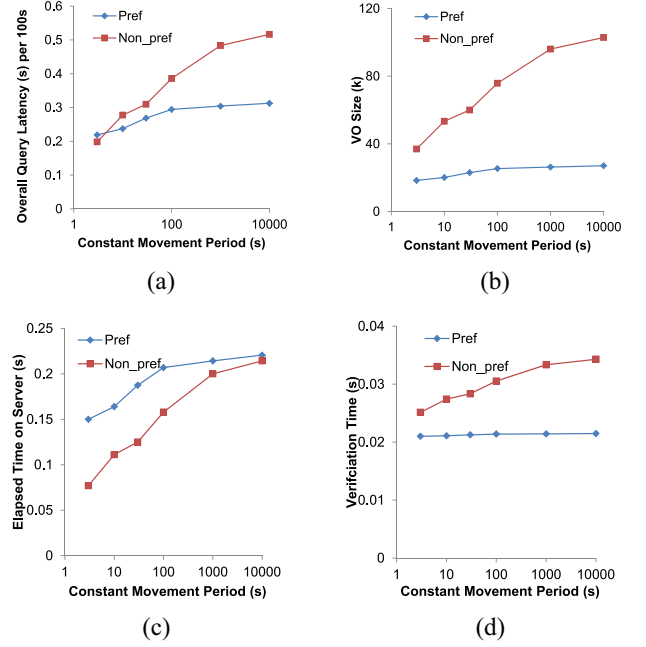


Fig. 10. Effect of constant movement period: (a) Overall time. (b) VO size. (c) Server CPU time. (d) Client verification time.

in [22]. The average moving velocity is set to 20 km/h and the average query duration of continuous LASQs is set to 100 seconds. We vary the average constant movement period t_v in the random way point model, and measure the accumulated communication and computation costs. A smaller t_v makes the movement trajectory of the client more centered around the origin query point. As such, with a smaller t_v , the stay period of the client in the clear area is longer, which is consistent with the result shown in Fig. 10(a). As t_v grows, the query point leaves the clear area more quickly in the Non_pref algorithm, so the client has to re-issue the query more frequently. As shown in Figs. 10(b) and 10(d), the Pref algorithm performs better than the Non_pref algorithm in terms of the VO size and verification time, because the client is more likely to compute the new LASQ results locally with the presence of a larger clear area in Pref. Nonetheless, Pref is at the cost of more expensive server time to determine which obstacle shapes should be attached (see Fig. 10(c)).

8 CONCLUSION AND FUTURE WORK

In this paper, we have studied the problem of authenticating location-based skyline queries in arbitrary subspaces (LASQs). We have proposed a basic MSR-tree authentication method by extending our previous work on skyline query authentication. To enable authentication for large-scale datasets and subspaces, we have further proposed a Partial-S4-tree method, in which most of the redundant objects can be easily identified and filtered out from the VO. For authenticating continuous LASQs, we have proposed a prefetching-based solution to avoid frequent query issuances and VO transmissions. Extensive experimental results demonstrate the efficiency of our proposed methods and algorithms under various system settings. In particular, our proposed Partial-S4-tree method

outperforms the basic authentication method by up to 69% in terms of the overall query latency and up to 74% in terms of the VO size.

As for the future work, we will extend this work to road network environments. Since the query distance is defined by network distance in a road network, the skyline scope defined in this paper no longer works, which calls for new authentication methods. Moreover, we are also interested in studying the authentication problem for dynamic objects, where how to guarantee the freshness of query results is a very challenging issue.

ACKNOWLEDGMENTS

The authors are grateful to the editor and the anonymous reviewers for their constructive comments that significantly improved the quality of this paper. This work was supported in part by the HK RGC Grants 210811 and 210612, NSFC Grant 60903169, and in part by the Hong Kong Scholar Program (Grant XJ2011008).

REFERENCES

- [1] (2011) *AT&T to Launch Cloud-Based LBS Mobility Data Offering* [Online]. Available: <http://www.mobilecommercedaily.com/2011/01/06/att-to-launch-cloud-based-lbs-mobility-data-offering>
- [2] 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*, Atlantic City, NJ, USA, 1990, pp. 322–331.
- [3] M. Berg, O. Cheong, and M. Kreveld, *Computational Geometry: Algorithms and Applications*, 3rd ed., Berlin, Germany: Springer, 2008, ch. 7.
- [4] S. Borzsonyi, D. Kossmann, and K. Stocker, "The Skyline operator," in *Proc. ICDE*, Heidelberg, Germany, 2001, pp. 421–430.
- [5] Q. Chen, H. Hu, and J. Xu, "Authenticating Top-k queries in location-based services with confidentiality," in *PVLDB*, Hangzhou, China, 2014.
- [6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with pre-sorting," in *Proc. ICDE*, 2003.
- [7] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *SIGMOD*, Baltimore, MD, USA, 2005.
- [8] H. Hu, J. Xu, Q. Chen, and Z. Yang, "Authenticating location-based services without compromising location privacy," in *SIGMOD*, 2012.
- [9] H. Hu, Q. Chen, and J. Xu, "VERDICT: Privacy-preserving authentication of range queries in location-based services," in *ICDE*, Brisbane, QLD, Australia, 2013 (Demo).
- [10] L. Hu, W.-S. Ku, S. Bakiras, and C. Shahabi, "Spatial query integrity with voronoi neighbors," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 863–876, Apr. 2013.
- [11] Z. Huang, H. Lu, B. C. Ooi, and K. H. Tong, "Continuous skyline queries for moving objects," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 12, pp. 1645–1658, Dec. 2006.
- [12] Y. Gao and B. Zheng, "Continuous obstructed nearest neighbor queries in spatial databases," in *Proc. ACM SIGMOD*, Providence, RI, USA, 2009, pp. 557–590.
- [13] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *VLDB*, Hong Kong, China, 2002.
- [14] R. C. Merkle, "A certified digital signature," in *CRYPTO*, Santa Barbara, CA, USA, 1989.
- [15] E. Mykletun, M. Narasimha, and G. Tsudik, "Signature bouquets: Immutability for aggregated/condensed signatures," in *ESORICS*, Sophia Antipolis, France, 2004.
- [16] C. K. Lee, W.-C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: An efficient skyline query processing framework based on Z-order," *VLDB J.*, vol. 19, no. 3, pp. 333–362, 2010.
- [17] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proc. SIGMOD*, Chicago, IL, USA, 2006.
- [18] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios, "Proof-infused streams: Enabling authentication of sliding window queries on streams," in *VLDB*, 2007.
- [19] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Authenticated index structures for aggregation queries," in *TISSEC*, vol. 13, no. 4, 2010.
- [20] X. Lin, J. Xu, and H. Hu, "Range-based skyline queries in mobile environments," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 835–849, Apr. 2013.
- [21] X. Lin, J. Xu, and H. Hu, "Authentication of location-based skyline queries," in *CIKM*, Glasgow, Scotland, U.K., 2011.
- [22] X. Lin, J. Xu, and J. Gu, "Continuous skyline queries with integrity assurance in outsourced spatial databases," in *WAIM*, Harbin, China, 2012.
- [23] H. Lo, and G. Ghinita, "Authenticating spatial skyline queries with low communication overhead," in *CODASPY*, 2013.
- [24] H. Pang, A. Jain, K. Ramamritham, and K. Tan, "Verifying completeness of relational query results in data publishing," in *SIGMOD*, Baltimore, MD, USA, 2005.
- [25] H. Pang and K. Mouratidis, "Authenticating the query results of text search engines," in *PVLDB*, Auckland, New Zealand, 2008.
- [26] S. Papadopoulos, Y. Yang, S. Bakiras, and D. Papadias, "Continuous spatial authentication," in *SSTD*, Aalborg, Denmark, 2009, pp. 62–79.
- [27] S. Papadopoulos, Y. Yang, and D. Papadias, "CADS: Continuous authentication on data streams," in *VLDB*, Vienna, Austria, 2007, pp. 135–146.
- [28] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. SIGMOD*, San Diego, CA, USA, 2003.
- [29] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM TODS*, vol. 30, no. 1, pp. 41–82, 2005.
- [30] *R-tree Portal* [Online]. Available: <http://www.rtreeportal.org/>
- [31] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *Proc. VLDB*, Seoul, Korea, 2006.
- [32] K.-L. Tan, P. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. VLDB*, Roma, Italy, 2001.
- [33] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *Proc. VLDB*, Hong Kong, China, 2002.
- [34] Y. Tao, X. Xiao, and J. Pei, "Efficient skyline and Top-k retrieval in subspaces," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, pp. 1072–1088, 2007.
- [35] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *Proc. ICDE*, Shanghai, China, 2009.
- [36] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *VLDB J.*, vol. 18, no. 3, 2009.
- [37] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis, "Authenticated join processing in outsourced databases," in *SIGMOD*, Providence, RI, USA, 2009.
- [38] M. L. Yiu, Y. Lin, and K. Mouratidis, "Efficient verification of shortest path search via authenticated hints," in *ICDE*, Long Beach, CA, USA, 2010, pp. 237–248.
- [39] M. L. Yiu, E. Lo, and D. Yung, "Authentication of moving kNN queries," in *ICDE*, 2011.
- [40] B. Zheng, C. K. Lee, and W.-C. Lee, "Location-dependent skyline query," in *MDM*, Beijing, China, 2008.



Xin Lin received the BE degree and the PhD degree, both in computer science and engineering, from Zhejiang University, Hangzhou, China. He is currently an Associate Professor at the Department of Computer Science, East China Normal University, Shanghai, China. He is also a Visiting Scholar at the Database Group, Hong Kong Baptist University, Kowloon Tong, Hong Kong (<http://www.comp.hkbu.edu.hk/~db/>). His current research interests include location-based services, spatial databases, and privacy-aware computing.



Jianliang Xu is an Associate Professor in the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. He received the BE degree in computer science and engineering from Zhejiang University, Hangzhou, China, and the PhD degree in computer science from Hong Kong University of Science and Technology, Hong Kong. He has held visiting positions at Pennsylvania State University, State College, PA, USA and Fudan University, Shanghai, China. His current research interests

include data management, mobile/pervasive computing, and networked and distributed systems. He has published more than 110 technical papers in these areas. He has served as Vice Chairman of the ACM Hong Kong chapter. He is a senior member of the IEEE.



Wang-Chien Lee received the BS degree from the Information Science Department, National Chiao Tung University, Hsinchu, Taiwan, the MS degree from the Computer Science Department, Indiana University, Bloomington, Indiana, and the PhD degree from the Computer and Information Science Department, Ohio State University, Columbus, OH, USA. He is an Associate Professor of Computer Science and Engineering at Pennsylvania State University, University Park, PA, USA. Currently, he leads the Pervasive Data

Access (PDA) Research Group at Penn State University to pursue cross-area research in data management, pervasive/mobile computing, and networking.



Haibo Hu received the BE degree in computer science and engineering from Shanghai Jiaotong University, Shanghai, China, in 2001, and the PhD degree in computer science from Hong Kong University of Science and Technology, Hong Kong, in 2005. He is a Research Assistant Professor in the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong. Prior to that, he has held several research and teaching posts at HKUST and HKBU. His current research interests include

mobile and wireless data management, location-based services, and privacy-aware computing. He has published more than 30 research papers in leading conferences and journals. He is also the recipient of many awards, including the ACM-HK Best PhD Paper Award and the Microsoft Imagine Cup.

▷ **For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.**