Combining Differential Privacy and PIR for Efficient Strong Location Privacy

Eric Fung, Georgios Kellaris, and Dimitris Papadias

Hong Kong University of Science and Technology
{khfungac, gkellaris, dimitris}@cse.ust.hk

Abstract. Data privacy is a huge concern nowadays. In the context of location based services, a very important issue regards protecting the position of users issuing queries. Strong location privacy renders the user position indistinguishable from any other location. This necessitates that every query, independently of its location, should retrieve the same amount of information, determined by the query with the maximum requirements. Consequently, the processing cost and the response time are prohibitively high for datasets of realistic sizes. In this paper, we propose a novel solution that offers both strong location privacy and efficiency by adjusting the accuracy of the query results. Our framework seamlessly combines the concepts of ϵ-*differential privacy* and *private information retrieval* (PIR), exploiting query statistics to increase efficiency without sacrificing privacy. We experimentally show that the proposed approach outperforms the current state-of-the-art by orders of magnitude, while introducing only a small bounded error.

1 Introduction

Mobile devices enable the use of location based services (LBS) in order to facilitate everyday tasks. An LBS allows users to issue queries along with their locations to a server, which in turn replies with the results. For example, a user may ask for the closest gas station to his current location, the shortest path from his home to a shopping mall, real-time traffic condition in his area, and so on. Each of the queried locations, e.g., gas station, shopping mall, is called a Point of Interest (POI). However, location based queries raise privacy concerns, as they can reveal the sensitive location of the user. For example, a user may wish to find the nearest bar without revealing his presence in the specific area. In this work we focus on private k-Nearest Neighbor queries (kNN), which ask for the k nearest POIs to the user.

Numerous algorithms have been proposed for private kNN queries. Strong Location Privacy for kNN [23] is currently the only solution which renders the position of the user truly indistinguishable from all other possible locations. It leverages hardware PIR and a query plan. Hardware PIR ensures that the server is oblivious to the data acquired by the users, while the query plan requires that every user receives the same amount of information independent of the data size needed. By combining these two properties, the query process for any user from

any location appears exactly the same to the server. In order to guarantee that all users receive accurate answers, the algorithm of [23] sets the query plan as the maximum data size required by any possible query. Although this solution is viable for small databases, it becomes prohibitively expensive for a large number of POIs because the result size required to satisfy any query may be enormous.

In order to overcome this problem, we propose the *adaptive query plan*, which relaxes the need for answering all queries accurately. Instead, it computes a minimum data size, which guarantees that at least a predefined percentage of queries are answered correctly. The adaptive query plan depends on the actual user behavior and changes periodically based on statistics of previously issued queries. However, utilizing statistics on sensitive location data may reveal the whereabouts of a user [7]. In order to avoid this type of privacy breaches, we employ the notion of ϵ-differential privacy [8]. This concept offers theoretical privacy guarantees when publishing statistics on sensitive data. Our solution is applicable to [23], and in general to PIR techniques based on similar principles.

We demonstrate the efficiency and effectiveness of our approach by using rigorous secure hardware simulations on two real datasets consisting of millions of POIs. Compared to [23], it offers up to orders of magnitude better efficiency, while retaining high levels of accuracy, rendering it practical for large datasets.

2 Related Work

Section 2.1 describes the notion of ϵ-differential privacy. Section 2.2 reviews location privacy techniques in general and presents the implementation of the state-of-the-art method of [23].

2.1 ϵ-Differential Privacy

Differential privacy hides sensitive information about individual users when publishing statistics. Specifically, the published results are produced in a random way, so that the presence of any individual in the data has negligible impact. Let $\mathcal{D}$ be a set of finite databases with d attributes. Each $D \in \mathcal{D}$ is a set of rows. For example, each row of D represents a user, and each column a location. A cell $c_{i,j}$ of D is 1 if user i has visited location j, and 0 otherwise. Two databases $D, D' \in \mathcal{D}$ are considered ***neighboring*** if they differ in at most one row; essentially, D and D' differ in the locations of one user. A mechanism $\mathcal{M}$ is a randomized algorithm performed by the publisher; given a database D, $\mathcal{M}$ applies some functionality and outputs a *transcript* $\mathbf{o}$.

Definition 1. *A mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{O}$ satisfies* **ϵ-differential privacy** *if for all sets $O \subseteq \mathcal{O}$, and every pair $D, D' \in \mathcal{D}$ of neighboring databases*

$$\Pr[\mathcal{M}(D) \in O] \leq e^{\epsilon} \cdot \Pr[\mathcal{M}(D') \in O] \tag{1}$$

The smaller the value of ϵ, the stronger the privacy guarantees. Intuitively, $\mathcal{M}$ satisfies ϵ-differential privacy, if changing the attributes of one individual in the database has a negligible effect on the distribution of the output of $\mathcal{M}$.

A common differential privacy technique adds Laplace noise to the outputs using the Laplace Perturbation Algorithm (LPA [8, 7]). Before presenting LPA, we formulate the notion of *sensitivity*. We view the release of statistical information as a query performed on the data. For example the query asks for the counts on each column of D, i.e., the number of users who visited each location. We model the query as a function $\mathbf{Q} : \mathcal{D} \rightarrow \mathbb{N}^d$, where d is the number of elements in the output. For $D, D' \in \mathcal{D}$, $\mathbf{Q}(D), \mathbf{Q}(D')$ are two d-dimensional vectors. Let $\|\mathbf{Q}(D) - \mathbf{Q}(D')\|$ be the L_1 norm of $\mathbf{Q}(D), \mathbf{Q}(D')$. Then, the ***sensitivity*** of $\mathbf{Q}$ is $\Delta(\mathbf{Q}) = \max_{D,D' \in \mathcal{D}} \|\mathbf{Q}(D) - \mathbf{Q}(D')\|$ for all neighboring $D, D' \in \mathcal{D}$.

Let $Lap(\lambda)$ be a random variable drawn from Laplace distribution with mean zero and scale parameter λ. LPA achieves ϵ-differential privacy through the mechanism outlined in the following theorem [7].

Theorem 1. *Let $\mathbf{Q} : \mathcal{D} \rightarrow \mathbb{N}^d$, and define $\mathbf{c} \stackrel{\text{def}}{=} \mathbf{Q}(D)$. A mechanism $\mathcal{M}$ that adds independently generated noise from a zero-mean Laplace distribution with scale parameter $\lambda = \Delta(\mathbf{Q})/\epsilon$ to each of the d output values of $\mathbf{Q}$, i.e., which produces transcript*

$$\mathbf{o} = \mathbf{c} + \langle Lap(\Delta(\mathbf{Q})/\epsilon) \rangle^d$$

achieves ϵ-differential privacy. The error introduced in the i^{th} element of $\mathbf{o}$ by LPA *is*

$$error^i_{\mathsf{LPA}} = \mathbb{E}|\mathbf{o}[i] - \mathbf{c}[i]| = \mathbb{E}|Lap(\lambda)| = \sqrt{2}\lambda = \sqrt{2}\Delta(\mathbf{Q})/\epsilon$$

The higher the error the more the published results deviate from their actual values, reducing their *utility*. Next, we include a *composition* theorem [19] that is useful for our proofs. It concerns successive executions of differentially private mechanisms on the same input, and allows us to view ϵ as a *privacy budget*, distributed among these mechanisms.

Theorem 2. *Let $\mathcal{M}_1, \ldots, \mathcal{M}_r$ be a set of mechanisms, where each $\mathcal{M}_i$ provides ϵ_i-differential privacy. Let $\mathcal{M}$ be another mechanism that executes $\mathcal{M}_1(D), \ldots, \mathcal{M}_r(D)$ using independent randomness for each $\mathcal{M}_i$, and returns the vector of the outputs of these mechanisms. Then, $\mathcal{M}$ satisfies $(\sum_{i=1}^{r} \epsilon_i)$-differential privacy.*

An interesting problem concerns computing differentially private range-sums over sensitive histograms. Applying LPA in this scenario would result in high error due to the noise accumulation. [9, 3] reduce the error by building a full binary tree over the input values. Every node stores the sum of the values of its children, plus noise with logarithmic scale to the input size. Then, each sum of consecutive values is computed by summing the root values of the maximal subtrees covering the values. We exploit this technique in our solution in order to maximize utility.

2.2 Location Privacy for *k*NN queries

The setting of private kNN queries assumes a data owner, who provides the POIs to an LBS, and users, who issue queries to the LBS. The goal is to conceal the

user locations from the LBS. There exist two notions of privacy in the literature: (i) weak location privacy, where the LBS can derive that the query issuer lies in some general area, without, however, being able to pinpoint his exact position, and (ii) strong location privacy, where the LBS cannot infer anything about the position of the user issuing the query. Weak location privacy solutions adopt three general methodologies, namely K-anonymity, location obfuscation, and data transformation.

K-anonymity [20, 14] methods assume the existence of a trusted third party that receives and anonymizes the queries before sending them to the LBS. Specifically, a trusted *anonymizer* that has the locations of all users, generalizes each query so that the LBS cannot distinguish who among K users issued the query. Location obfuscation [16, 6, 5] substitutes the exact user location with a cloaking region, which is sent to the LBS instead of the exact location. In some obfuscation methods (e.g., [28]), the user sends a fake location and keeps obtaining results until it acquires all k nearest neighbors. In all the above techniques, the LBS can restrict the position of the querying users in some area within the data space, without however being able to pinpoint their exact location.

Data transformation techniques [16, 25] assume that the owner encodes the data before sending them to the LBS. Subsequently, the users send encoded queries to the server. The latter cannot determine either the queried data or the user query location. Data transformation methods conceal the user locations better than K-anonymity and obfuscation. However, they are expensive due to the encoding/decoding operations. Additionally, they are prone to *access pattern attacks* [27] because the same query always returns the same encoded results. For example, the LBS may use the query frequency and data density to infer the position of a user (e.g., queries at a city center are much more frequent than those from the suburbs).

Strong location privacy is based on private information retrieval (PIR) ([10, 11]), which allows the users to retrieve data from a database obliviously. There are three categories of PIR, namely information-theoretical, computational, and hardware-based. Information-theoretical PIR [4] offers privacy with theoretical guarantees, while computational PIR [18, 21, 24, 26, 13] assumes computationally bounded adversaries. They are both infeasible even for databases of moderate sizes [25]. Hardware-based PIR relies on a tamper-resistant CPU, trusted by the clients, attached to the server. This CPU receives client block requests, which are unreadable by the server, obliviously extracts the requested blocks from the server disk, and returns them to the client. Hardware-based methods are the only viable PIR solutions for large datasets.

[15] applies hardware-based PIR for kNN processing, allowing, however, a variable number of PIR requests for different queries. Consequently, although each PIR retrieval is private, the cardinality of these retrievals may allow access pattern attacks. On the other hand, the method of [12] achieves strong location privacy because every query involves a single PIR request and, hence, all queries are indistinguishable. Nevertheless, this scheme focuses on single NN processing ($k = 1$), and relies on a prohibitively expensive computational PIR protocol [18].

Currently, the only viable technique that guarantees strong location privacy for kNN queries is AHG [23]. AHG is a hardware-based PIR algorithm that utilizes a query plan to avoid access pattern attacks.

Specifically, the setting of [23] assumes that the LBS maintains the data as sequential blocks[1]. AHG initially imposes a Hilbert index grid G on the POIs, grouping them into cells. The Hilbert index grid is a mapping which defines an ordering among the cells according to their unique Hilbert values, e.g., Hilbert values of cell c_{21} and cell c_{11} are defined by $H(2,1) = 1$ and $H(1,1) = 0$, respectively. To preserve locality, the LBS stores the cells ordered by the Hilbert values along with their POI counts in multiple PIR blocks of a database $\mathcal{DB}_1$. It also keeps the blocks of individual POIs in two databases, namely $\mathcal{DB}_2$ and $\mathcal{DB}_3$. Essentially, $\mathcal{DB}_1$ maintains an index of the POIs stored in $\mathcal{DB}_2$. $\mathcal{DB}_2$ holds the actual locations of the POIs, i.e., the longitudes and latitudes, and pointers to $\mathcal{DB}_3$. Finally, $\mathcal{DB}_3$ stores the tail records of the POIs, i.e., other data related to the POIs, such as street addresses, phone numbers, and detailed information.

The users issue kNN queries to the trusted CPU (attached to the LBS), using a fixed query plan to ensure that each query retrieves the same number of blocks from each database, independent of the query location. The query plan is defined as $QP = ((\mathcal{DB}_1, cnt_1), (\mathcal{DB}_2, cnt_2), (\mathcal{DB}_3, k))$. Specifically, when a user asks a kNN query, he first obtains cnt_1 index blocks from $\mathcal{DB}_1$. Then, using the index, he retrieves cnt_2 blocks with POI coordinates from $\mathcal{DB}_2$, and locates the k nearest POIs using these coordinates. Finally, he issues a query to $\mathcal{DB}_3$ and obtains the k corresponding blocks from $\mathcal{DB}_3$. In order to guarantee that every user receives enough blocks for an accurate answer, cnt_1 and cnt_2 constitute upper bounds for the number of blocks needed by *any possible* query location[2].

3 Adaptive Query Plan

We assume the same setting as [23], where a curious, but not malicious LBS maintains the data as sequential blocks. Users issue kNN queries in the form of block requests to a trusted CPU attached to the LBS. This CPU obliviously extracts the requested blocks from the server, and returns them to the client. According to the fixed query plan of AHG, every user receives the maximum number of blocks required to accurately answer all possible queries. Consequently, most users obtain numerous *redundant* blocks since the vast majority of queries need relatively few blocks due to the fact that the query distribution usually follows the distribution of the POIs [2, 22, 17]. This has a negative impact on the LBS (in terms of processing cost) and the users (in terms of response time), rendering AHG too slow for large spatial datasets commonly found in practice.

To overcome this problem, we propose an adaptive query plan (AQP) that yields the exact kNN set for the majority of the queries, but may lead to inaccurate results for a pre-defined percentage $(1 - \alpha)$ of queries at sparse areas of the

[1] The size of each block depends on the PIR hardware.

[2] There is a distinct query plan for every allowed value of k. For ease of presentation, we focus on a single value of k.

data space. The value of α adjusts the trade-off between accuracy and efficiency. In order to derive the size of AQP, we utilize differentially private statistics about previous queries. Strong location privacy is always preserved independently of the value of α and the size of AQP.

Our framework involves two stages: (1) The *query stage* has a fixed period (e.g., a day), in which users issue kNN queries to the LBS. Every user can ask up to $q_{\max}$ private kNN queries, where $q_{\max}$ is a system parameter. For each query, he records the number of redundant blocks during the current period. (2) At the query plan *re-computation stage*, the LBS obtains the redundancy data from users in a differentially private manner, and computes the distribution of redundant blocks along with the number of blocks necessary to answer the issued queries. Finally, it generates an AQP for the next query stage, so that at least a percentage α of the queries receive enough PIR blocks for accurate results, according to the current statistics. Section 3.1 describes the query stage, Section 3.2 elaborates the re-computation stage, and Section 3.3 proves the correctness and analyzes the utility of our approach.

3.1 Query Stage

During this stage, each user u maintains a vector R_u of length $cnt_1 + 1$ that stores the number of redundant blocks received from $\mathcal{DB}_1$. Each element j of R_u holds the number of times that u received j redundant blocks. The last element $R_u[cnt_1]$ indicates the number of queries with insufficient blocks (i.e., those with potentially inaccurate results). For example, if $cnt_1 = 30$, $R_u[0] = 1$, $R_u[2] = 5$, and $R_u[30] = 4$, then u had 1 query for which he received the exact number of required blocks, 5 queries with 2 redundant blocks, and 4 queries without enough blocks. A similar vector S_u of length $cnt_2 + 1$ is maintained for database $\mathcal{DB}_2$.

Figure 1 depicts an example for 20 POIs (P_1 to P_{20}), a 6×6 grid, and a query location Q. Let c_{yx} be the cell of the y^{th} row and x^{th} column. All the c_{yx}'s are ordered according to their unique Hilbert values, and stored in $\mathcal{DB}_1$ as pairs of numbers. The first value of c_{yx} indicates the sum of the POIs contained in all preceding cells in the Hilbert order, while the second one is the number of POIs in c_{yx}. For example, (3,0) for c_{13} denotes 3 POIs lying in $c_{11}, c_{21}, c_{22}, c_{12}$, and no POIs in c_{13}. Each block in $\mathcal{DB}_1$ holds up to 8 cells, and it is denoted as $B_{1,id}$, where id is the block id. $\mathcal{DB}_2$ holds a tuple $< P.id, P.x, P.y, P.ptr >$ for each POI, where $P.id$ is the POI id, $P.x, P.y$ its coordinates, and $P.ptr$ a pointer to $\mathcal{DB}_3$. Each $\mathcal{DB}_2$ block, denoted as $B_{2,id}$, consists of up to 4 POIs, which are sorted on the Hilbert values of their cells. Finally, $\mathcal{DB}_3$ contains tuples of the form $< P.id, P.tail >$, where $P.id$ is the POI id, and $P.tail$ is the tail information of the POI. Each block in $\mathcal{DB}_3$ is denoted as $B_{3,id}$.

Let the query plan be $((\mathcal{DB}_1, 2), (\mathcal{DB}_2, 4), (\mathcal{DB}_3, 2))$. Assume that a user u in cell c_{45} issues a 2NN query from location Q. Initially, u discovers the block in $\mathcal{DB}_1$ that contains his residing cell as follows. He computes the Hilbert value $H(4, 5) = 29$ of c_{45}, and determines the required block as $(29+1)/8+1 = 4$. Then, he finds the position of c_{45} by computing $(29 + 1) \mod 8 = 6$. Consequently, u derives that the information of c_{45} is at position 6 of the 4^{th} block in $\mathcal{DB}_1$,

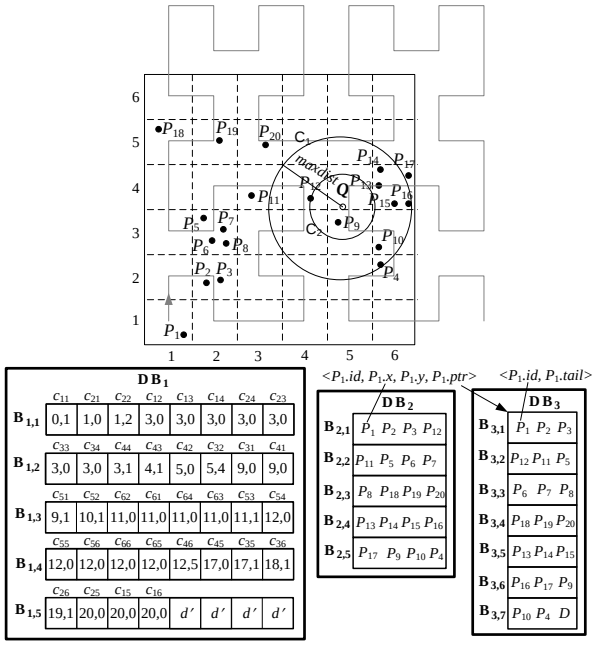


Fig. 1. Query stage example

and instructs the secure CPU to retrieve block $B_{1,4}$. The pair of values in c_{45} is $(17, 0)$, denoting that there are no POIs in u's cell. Thus, u proceeds with the next closest cell to Q, which is c_{35}. The latter belongs to the already retrieved block $B_{1,4}$ and hence, u does not need to obtain another block. Moreover, c_{35} contains only one POI and as such u needs to explore more cells by following the same procedure. After retrieving c_{44} from $B_{1,2}$, u has gathered 2 POIs, which he uses for pruning as follows. He computes the maximum possible distance $maxdist$ between Q and the retrieved cells, and draws a circle C_1 centered at Q, with radius of $maxdist$. All the cells that have no common area with the disk of C_1 cannot contain any POIs comprising the 2NN, and hence, they can be ignored. Therefore, u needs only the information of cells c_{34}, c_{46}, c_{36}, c_{55}, c_{25}, c_{54}, c_{24}, c_{43}, c_{33}, c_{56}, and c_{26}, which are distributed among all the 5 blocks of $\mathcal{DB}_1$. However, he cannot acquire the information for all these cells, since his $\mathcal{DB}_1$ block retrieval is limited to 2 blocks by the query plan. As such, he has insufficient blocks from $\mathcal{DB}_1$ in order to ensure an accurate 2NN, and hence, he updates the redundancy vector for $\mathcal{DB}_1$ by setting $R_u[2] := R_u[2] + 1$.

Next, u requests the coordinates of the POIs from $\mathcal{DB}_2$ in ascending order of the minimum distances between Q and each retrieved cell from $\mathcal{DB}_1$. Specifically, he obtained $(17, 1)$ for c_{35} from the previous step. He computes $(17+1)/4+1 = 5$, and $(17 + 1) \mod 4 = 2$, and derives that the POI is at the 2^{nd} position of the

5^{th} block of $\mathcal{DB}_2$. Thus, he requests $B_{2,5}$ from $\mathcal{DB}_2$, and repeats the process for all of the potential NNs. Each time he retrieves the coordinates of a POI, he updates the best 2NN so far, and further prunes the search space if the current distance of this 2NN is less than the minimum distance between Q and any other cell. For example, let the current 2NN be P_9 and P_{12}, which lets u draw circle C_2. Any cell with no common area with the disk of C_2 cannot contain a better NN, e.g., cell c_{43} is pruned, because the distance between Q and P_{12}, is smaller than the minimum distance between Q and c_{43}. Eventually, u receives 3 blocks, i.e., $B_{2,1}$, $B_{2,4}$, and $B_{2,5}$, and his 2NN consists of the POIs P_9 and P_{12}. However, the query plan requires that any user retrieves 4 blocks, so he sends a random request, retrieves one more block, and updates his redundant block vector S_u for $\mathcal{DB}_2$, to $S_u[1] := S_u[1] + 1$. Finally, he follows the pointers of P_9 and P_{12}, and retrieves blocks $B_{3,2}$ and $B_{3,6}$ from $\mathcal{DB}_3$ in order to acquire the tail information.

3.2 Re-computation Stage

During re-computation, the LBS aggregates the redundancy vectors of all active users, i.e., those that are on-line. The process is identical for both $\mathcal{DB}_1$ and $\mathcal{DB}_2$; in the following we focus on $\mathcal{DB}_1$. Each active user must specify the percentage ℓ of users that he trusts not to collude with the LBS. For instance, if $\ell = 10$, 10% of the users are considered trusted. The parameter adjusts the noise scale added by the differentially private mechanism. High values of ℓ, as well as a large number of active users, lead to more accurate statistics. For simplicity, we assume that every user chooses the same ℓ value.

When a user u registers with the service, he sends to the LBS a self-signed Diffie-Hellman (DH) component for computing pairwise keys, and a certificate to authenticate himself. At the beginning of re-computation, the LBS distributes the self-signed DH components and certificates to all active users. Every user computes the pairwise keys shared with the other users using the DH components. Then, each user u performs computations on R_u, before forwarding a noisy and encrypted version $\hat{R}_u$ of R_u to the LBS. Having collected $\hat{R}_u$ from the active users, the LBS derives a differentially private vector $\hat{R}$ of the aggregate statistics and uses it to generate the new AQP.

The process constitutes a combination of secure multiparty computation and distributed differential privacy, for which we adopt the method of [1], originally proposed for computing differentially private sums[3]. Let n be the number of active users, $K_{u,w}$ be the pairwise key shared by users u and w, r_1 and r_2 be two random numbers published by the LBS, and K_u a symmetric key between user u and the LBS. For every element $R_u[v]$, user u spends privacy budget $\lambda = \epsilon/(4 \cdot q_{\max})$ by drawing two noise values from the Gamma distribution, and computes $\hat{R}_u[v] = R_u[v] + G_{u,1}(n \cdot \ell, \lambda) - G_{u,2}(n \cdot \ell, \lambda)$. Subsequently, u selects approximately $n \cdot \ell$ other users randomly by using a secure pseudo random

[3] In this setting, there are several users, each holding a value, and they wish to publish the total sum, so that the value of any user is not revealed, even if an adversary has complete knowledge of all the remaining users.

function (PRF), so that if u selects w, then w selects u as well. The PRF works as follows: u chooses w, if $PRF(K_{u,w}, r_1) \leq n \cdot \ell/(n-1)$; for each chosen w, u computes $dkey_{u,w} = (u-w)/|u-w| \cdot PRF(K_{u,w}, r_2)$. Note that $dkey_{u,w} = -dkey_{w,u}$. User u then encrypts $\hat{R}_u[v]$ as $Enc(\hat{R}_u[v]) = \hat{R}_u[v] + K_u + \sum_w dkey_{u,w}$ and sends it to the server.

The LBS aggregates all the numbers sent from the users. By doing this, all the *dkey*s are canceled out, and the server decrypts the sum by $\hat{R}[v] = \sum_u Enc(\hat{R}_u[v]) - \sum_u K_u = \sum_u R_u[v] + \sum_u (G_{u,1}(n \cdot \ell, \lambda) - G_{u,2}(n \cdot \ell, \lambda))$. [1] shows that $\hat{R}[v]$ is equivalent to $\hat{R}[v] = \sum_u R_u[v] + Lap(\lambda/\ell)$ because the Laplace noise can be approximated from identically distributed (i.i.d.) gamma distributions. As such, after computing all the elements of $\hat{R}$, the final result satisfies differential privacy.

The LBS calculates cnt_1 by executing method $recomputeQP(\hat{R})$, shown in Figure 2. Lines 4−5 determine if the percentage of queries without enough blocks exceeds $(1-\alpha)$ by checking $ratio = \hat{R}[cnt_1]/\sum_{i=0}^{cnt_1} \hat{R}[i]$. In this case, the value of cnt_1 increases, so that queries receive more blocks. In case the percentage of queries without enough blocks does not exceed $(1-\alpha)$ (lines $6-11$), the server iteratively computes $ratio+ = \hat{R}[l]/\sum_{i=0}^{cnt_1} \hat{R}[i]$ by incrementing l until the ratio exceeds $(1-\alpha)$. Essentially, the ratio for a certain l value represents the percentage of queries that will not receive enough blocks if we decrease the previous query plan by l. Thus, the new query plan is computed as the previous one reduced by $(l-1)$.

Function recomputeQP($\hat{R}$)		
1.	**for** $i = 0$ to cnt_1 **do**	
2.	$sum+ = \hat{R}[i]$	//count the total number of queries
3.	$ratio = \hat{R}[cnt_1]/sum$	//ratio of queries without enough blocks
4.	**if** $ratio > (1-\alpha)$ **then**	//percentage of inaccurate queries exceeds $(1-\alpha)$
5.	$cnt_1 := cnt_1 + 1$	//increase AQP size by 1
6.	**else**	
7.	$l = 0$	
8.	**while** $ratio \leq (1-\alpha)$	//percentage of inaccurate queries below $(1-\alpha)$
9.	$ratio+ = \hat{R}[l]/sum$	//decrease AQP size
10.	$l = l + 1$	
11.	$cnt_1 := cnt_1 - (l-1)$	

Fig. 2. Pseudocode of *recomputeQP*

In lines 2 and 9 of *recomputeQP* (Figure 2), the LBS sums consecutive noisy values of $\hat{R}$ in order to compute differentially private range-sums over private data. As discussed in the related work session, this would result in high error due to the noise accumulation. Therefore we adapt the technique of [9, 3] as follows. Each user u creates a binary tree, such that the original redundancy vector R_u (without noise) represents the leaf nodes and each parent node is the sum of its direct children, i.e., the root node is the sum of all leaf nodes.

Moreover, the privacy budget spent by the users decreases from $\epsilon/(4q_{\max})$ to $\epsilon/(4q_{\max} \cdot \log_2 |\hat{R}|)$, in order to satisfy the same level of privacy as suggested by [9, 3]. Then, aggregation is executed for the tree nodes instead for the elements of R_u. Finally, the LBS acquires an aggregate tree with leaf nodes representing the noisy values of $\hat{R}$, and re-computes the query plan as before. The only difference is that, instead of adding the values of $\hat{R}$ one by one in order to compute *ratio*, the LBS utilizes the noisy tree structure, resulting in more accurate sums.

3.3 Correctness and Utility Analysis

The next theorem shows that our adaptive solution, when applied on both $\mathcal{DB}_1$ and $\mathcal{DB}_2$, satisfies ϵ-differential privacy.

Theorem 3. *The AQP algorithm satisfies ϵ-differential privacy for at most q_{max} queries per user.*

Proof. We refer to the query stage as mechanism M_1, and the re-computation stage as mechanism M_2. Due to the fact that the query stage satisfies strong location privacy [23], an adversary cannot distinguish if the user asks a query from a location j or any other location j'. In other words, the probability for the user to be at location j, and receive cnt_1 blocks from $\mathcal{DB}_1$ and cnt_2 blocks from $\mathcal{DB}_2$ is the same with the probability he is at j', and receives exactly the same number of blocks from the two databases. Thus, from Definition 1, M_1 satisfies 0-differential privacy.

In the case of M_2, we further split the mechanism into two mechanisms $M_{2.1}$ and $M_{2.2}$. $M_{2.1}$ computes cnt_1 for $\mathcal{DB}_1$ and $M_{2.2}$ computes cnt_2 for $\mathcal{DB}_2$. Thus, mechanism M_2 comprises of $cnt_1 + 1$ ($M_{2.1}$) and $cnt_2 + 1$ ($M_{2.2}$) mechanisms of [1], each utilizing privacy budget $\epsilon/(4 \cdot q_{\max})$.

Let D_1 be a table where each row represents a user, and each column the received redundant blocks from $\mathcal{DB}_1$, i.e., each row u of D_1 corresponds to R_u of user u for $\mathcal{DB}_1$. Each cell i, j of D_1 holds how many times user i received j redundant blocks from $\mathcal{DB}_1$ during the query stage. Similarly, we define a table D_2 for $\mathcal{DB}_2$, i.e., each row u of D_2 corresponds to S_u of user u for $\mathcal{DB}_2$.

Mechanism $M_{2.1}$ (resp. $M_{2.2}$) essentially executes the method of [1] on each column of D_1 (resp. D_2), and returns vector $\hat{R}$ (resp. $\hat{S}$), which holds the noisy sums on the columns. A neighboring database D_1' (resp. D_2') differs at most by $2 \cdot q_{\max}$ to D_1 (resp. D_2): A user issues at most $q_{\max}$ private queries in D_1 (resp. D_2), and there can be at most $q_{\max}$ different private queries in D_1' (resp. D_2'). As such, he can change at most $2 \cdot q_{\max}$ values of each database D_1 and D_2.

In order to determine the achieved privacy level we work as follows. For any D_1 (resp. D_2), w.r.t a D_1' (resp. D_2'), we create two databases; D_a which contains only the columns of D_1 (resp. D_2) that differ from D_1' (resp. D_2'), and D_b which contains the columns that are exactly the same as those of D_1' (resp. D_2'). Note that D_a has at most $2 \cdot q_{\max}$ columns due to the sensitivity of D_1 (resp. D_2). From D_a we compute the noisy sums of the columns $\hat{R}_a$, and from D_b we compute $\hat{R}_b$, where $\hat{R}_a \cup \hat{R}_b = \hat{R}_{D1}$ (resp. $\hat{R}_a \cup \hat{R}_b = \hat{R}_{D2}$) and $\hat{R}_a \cap \hat{R}_b = \varnothing$. Any mechanism

of [1] when applied on D_a with privacy budget $\epsilon/(4 \cdot q_{\max})$, it satisfies $\epsilon/(4 \cdot q_{\max})$-differential privacy, as shown in [1]. On the other hand, when it is applied on D_b, it satisfies 0-differential privacy, due to Definition 1, since $D_b = D'_b$. Moreover, we have at most $2 \cdot q_{\max}$ sub-mechanisms of $M_{2.1}$ (resp. $M_{2.2}$) applied on D_a, since D_a has at most $2 \cdot q_{\max}$ columns. Hence, from Theorem 2, $M_{2.1}$ satisfies $(2 \cdot q_{\max} \cdot \epsilon/(4 \cdot q_{\max}) = \epsilon/2)$-differential privacy, and equivalently $M_{2.2}$ satisfies $\epsilon/2$-differential privacy, while M_1 satisfies 0-differential privacy. Thus, the whole procedure satisfies $(0 + \epsilon/2 + \epsilon/2 = \epsilon)$-differential privacy due to Theorem 2.

Next, we quantify the expected error. We focus on $\mathcal{DB}_1$ since the analysis for $\mathcal{DB}_2$ is the same. Let q be the total number of queries performed at query stage, and R be the actual redundancy vector without the noise required for differential privacy. Due the the noise addition, we expect that the computed (at the LBS) vector $\hat{R}$ deviates from R, yielding an error during the query plan re-computation.

Let l_{real} be the position of R representing the α percentile, i.e., the minimum l_{real} value such that $R[cnt_1] + \sum_{i=0}^{l_{real}} R[i] > (1-\alpha) \cdot q$. Then, $cnt_1 - l_{real} - 1$ is the number of necessary blocks for accuracy α. On the other hand, the corresponding number in Figure 2 is computed as the minimum value of l for which it holds that $\hat{R}[cnt_1] + \sum_{i=0}^{l} \hat{R}[i] > (1-\alpha) \cdot q$. As such, *recomputeQP* may stop at an $l \leq l_{real}$ (or $l > l_{real}$), and return $|l_{real} - l|$ more blocks (resp. fewer blocks) for each query than required for accuracy α. We define $|l_{real} - l|$ as the error due to the noise perturbation.

As an example, let $n = 10$, $\alpha = 70\%$, $q = 10$, $cnt_1 = 3$, $|\hat{R}| = 4$, and R and $\hat{R}$ as shown in table 1. Then, $(1-\alpha) \cdot q = 3$, $R[3] + R[0] = 3$, and $l_{real} = 2$. Consequently, $cnt_1 - (l_{real} - 1) = 2$, and hence, $cnt_1 = 2$ for the next period. However, the server knows only the noisy $\hat{R}$. It computes $\hat{R}[3] + \hat{R}[0] + \hat{R}[1] > 3$, and sets $l = 3$. As a result, $cnt_1 - (l-1) = 1$, or $cnt_1 = 1$, and due to noise each query receives $|l_{real} - l| = 1$ block less than required for 70% accuracy.

Table 1. An example illustrating variables

Redundancy	0	1	2	insufficient
R	2	2	5	1
$\hat{R}$	1	3	5	1

In the worst case, R is highly skewed, and $(\alpha + \delta)q$ queries, for any small $\delta > 0$, receive $cnt_1 - 1$ redundant blocks (i.e., $R[cnt_1 - 1] = (\alpha + \delta)q$), while the rest receive insufficient blocks (i.e., $R[cnt_1] = (1-\alpha-\delta)q$). Therefore, l_{real} should be $|R| - 1$, i.e., the new query plan should be set as $cnt_1 = 1$. In this case, if the Laplace noise added while computing $\hat{R}[cnt_1]$ is positive, *recomputeQP* will compute the new query plan as $cnt_1 := cnt_1 + 1 = |R|$, resulting in the maximum error of $|R| - 1$. Due to the fact that the Laplace distribution is symmetric about its mean 0, the probability for *recomputeQP* to stop at position 0 is 50%, at position 1 is 25%, and so on. This is equivalent to multiple Bernoulli trials and

hence, the probability to stop at position i can be described with the Binomial distribution with $p = 0.5$. Thus, with probability 50%, we get the worst possible error $|R| - 1$, while the probability for the error to be reduced by l (i.e. to stop at position l) is equal to the probability we receive l heads in l coin flips.

In order to better quantify the expected error, we assume a uniform distribution of the queries in R, i.e., $q/|R|$ queries receive 0 redundant blocks, $q/|R|$ queries receive 1 redundant block, and so on. In this case, in order to achieve α accuracy, we need to set $l_{real} = (1-\alpha)|R|$. Then, it suffices to compute the expected value of l (returned by *recomputeQP*) in order to find the expected error $|l_{real} - l|$. Initially, we calculate the expected error of each sum $\sum_{i=0}^{i} \hat{R}[i]$, for any $0 \leq i \leq |R|$. The sum is computed utilizing the aggregate tree at the server side with noisy node values. The noise at each node is equivalent to Laplace noise with scale $\lambda = \frac{4q_{\max} \cdot \log |R|}{\epsilon \cdot \ell}$. In order to calculate each sum i we use the technique of [3], which results in error err less than $\lambda \cdot \sqrt{\log(i+1)} \cdot \log \frac{1}{\delta_i}$ with probability $(1 - \delta_i)$.

RecomputeQP checks the value of each sum for $i = 0$ to $|R|$, and returns the first i that results in a noisy sum which is higher to $(1-\alpha) \cdot q$. Thus, $i = l$ if it does not stop at $i = 0 \ldots l-1$, with probability higher than $1 - \delta$. The probability for the algorithm to stop at a position l is equal to the probability it does not stop until position $l-1$ or $\prod_{i=0}^{l} \left(1 - \Pr\left[(i+1)\frac{q}{|R|} + err \geq (1-\alpha) \cdot q\right]\right)$. Thus, the value of l can be computed as the first value for which the following inequality does not hold.

$$\prod_{i=0}^{l} \left(1 - \Pr\left[(i+1)\frac{q}{|R|} + err \geq (1-\alpha) \cdot q\right]\right) \geq (1-\delta)$$

$$\prod_{i=0}^{l} \left(1 - \Pr\left[err \geq (1-\alpha) \cdot q - (i+1)\frac{q}{|R|}\right]\right) \geq (1-\delta)$$

$$\prod_{i=0}^{l} (1-\delta_i) \geq (1-\delta)$$

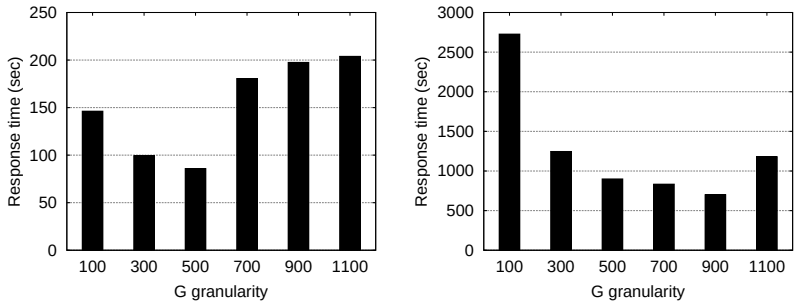
where $\delta_i = 0.5^{\frac{(1-\alpha)\cdot q - (i+1)\cdot q/|R|}{\lambda\sqrt{\log(i+1)}}}$.

4 Experimental Evaluation

In this section we compare our adaptive query plan AQP with the fixed query plan of AHG [23]. We implemented the methods in C++ on a Linux server with Intel Core i7-4770 and 32GB of RAM. Since the re-computation takes only a few seconds and is performed once after each query stage, it is excluded from the evaluation, and all experiments focus on query processing. In order to evaluate efficiency, we measure the query response time, which directly affects the user, and the number of block accesses, which determines the processing cost at the LBS. We used two datasets with real POIs from Germany (denoted as *Germany*)

Table 2. Parameter values

Parameter	Values	Default
# of active users n	2000, 4000, 6000, 8000, 10000	6000
% of trusted users ℓ	5, 10, 15, 20, 25	10
Accuracy α	0.75, 0.8, 0.85, 0.9, 0.95	0.95



(a) *Germany* (b) *USA*

Fig. 3. Response time vs. G granularity

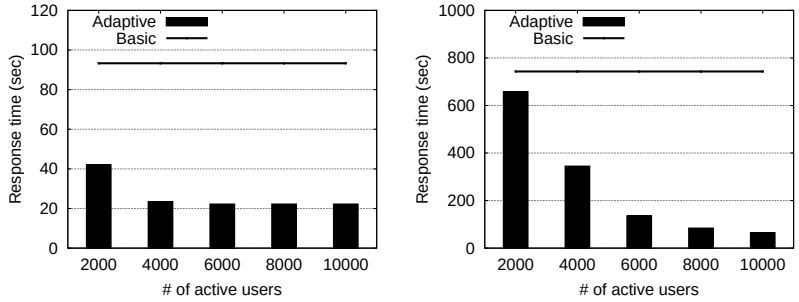
and the United States (denoted as *USA*)[4]. The former consists of 1.9 million POIs, while the latter has 12.8 million POIs.

Table 2 illustrates the examined parameters, along with their default values. The number n of active users denotes those participating in the re-computation stage. The percentage ℓ of trusted users corresponds to those trusted not to collude with others. Accuracy α is the percentage of queries that should be answered correctly. In all experiments, we set $\epsilon = 1$ for differential privacy, and $k = 10$ as the number of returned nearest neighbors. Every user issues $q_{\max} = 10$ private queries that follow the same distribution as the POIs. Since all queries retrieve the same number of blocks they incur the same cost and response time.

Following [23], we first fine-tune the granularity of the grid G, used by the basic AHG and AQP. Figure 3 shows the query response time, i.e. the total elapsed time until a user receives a query answer, as a function of the granularity of G, assuming fixed query plans (as in [23]). Coarse granularity leads to high cost because there are numerous POIs in each cell, leading to many $\mathcal{DB}_2$ PIR retrievals. The response time is also high when the granularity is too fine because there are numerous empty cells, yielding many $\mathcal{DB}_1$ PIR retrievals. In the remaining experiments, we set the grid granularity to the optimal configuration, which is 500×500 for *Germany*, and 900×900 for *USA*. Note that a single query by basic AHG requires more than 10 minutes in *USA*, even for the best granularity, motivating the need for AQP.

Figure 4 plots the response time versus the number of active users, setting $\alpha = 95\%$ and $\ell = 10\%$. The cost drops as the active users increase because the

[4] SimpleGeo's Places, available at `http://freegisdata.rtwilson.com/`.



(a) *Germany* (b) *USA*

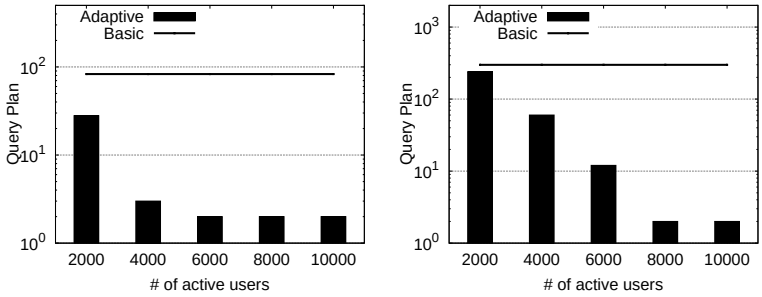
Fig. 4. Response time vs. number of active users

accuracy of statistics estimation improves, leading to a smaller query plan. The basic solution needs 93.3 seconds for *Germany*, and 743.1 seconds for *USA*, in order to answer a single query. For *Germany*, AQP reaches the lowest value (about 20 seconds) quickly, saving 78.6% compared to AHG. Concerning *USA*, the benefits of AQP are limited for a small number of users due to inaccurate query plan calculation by the LBS. However, as the number of active users increases, the cost drops quickly, reaching 65 seconds for 10,000 users and achieving savings of 91.3%. The larger benefits of AQP for *USA* are explained by the fact that the basic plan is very expensive leaving more space for optimization.

To better elaborate performance, we investigate the size (in blocks) of the query plans. The fixed query plan for *Germany* is $((\mathcal{DB}_1, 83), (\mathcal{DB}_2, 53), (\mathcal{DB}_3, 10))$ under all settings. This implies that any 10NN query retrieves $cnt_1 = 83$ blocks from $\mathcal{DB}_1$, $cnt_2 = 53$ from $\mathcal{DB}_2$, and $cnt_3 = 10$ from $\mathcal{DB}_3$. Recall that the cnt_1 blocks correspond to cell retrievals, whereas the cnt_2 blocks represent POI retrievals. The cnt_3 blocks refer to detail information about the 10NNs, and cannot be avoided by AQP or any other method. The fixed query plan for *USA* is $((\mathcal{DB}_1, 299), (\mathcal{DB}_2, 485), (\mathcal{DB}_3, 10))$. Note that cnt_1 (cnt_2) is higher for *USA* because of the finer grid granularity (larger number of POIs).

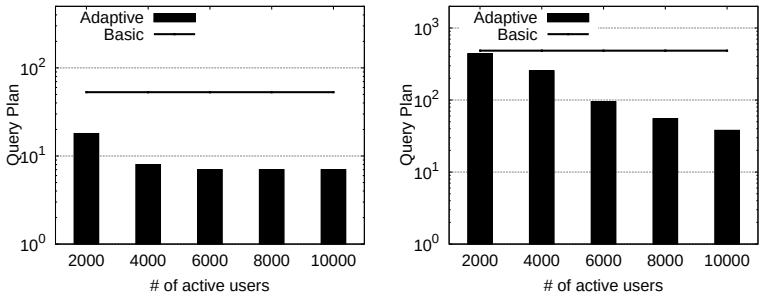
Figures 5 and 6 show the number of blocks cnt_1 and cnt_2, in both fixed and adaptive plans, as a function of the number of active users, setting $\alpha = 95\%$ and $\ell = 10\%$. Comparing Figures 5 and 6, the benefits of AQP are more pronounced in $\mathcal{DB}_1$. This is explained by the fact that the pruned cells are likely to contain few POIs; therefore the cell reduction in $\mathcal{DB}_1$ does not directly translate to an equivalent POI reduction in $\mathcal{DB}_2$. In general, the results are consistent with those on response time in Figure 4; i.e., for *Germany* a small number of active users suffices, while for *USA* more users are necessary to substantially reduce the number of block retrievals.

Figure 7 shows the response time as a function of the percentage of trusted users ℓ, fixing $\alpha = 95\%$ and $n = 6,000$. The time drops as ℓ increases because the resulting statistics have smaller noise scale. For *Germany*, even $\ell = 5\%$ (i.e., 300



(a) *Germany* (b) *USA*

Fig. 5. Query plan vs. number of active users for $\mathcal{DB}_1$



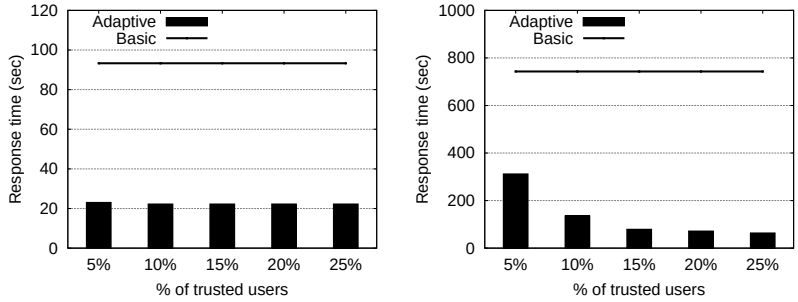
(a) *Germany* (b) *USA*

Fig. 6. Query plan vs. number of active users for $\mathcal{DB}_2$

trusted users) yields the lowest cost. Similar to Figure 4, for *USA* the number of users necessary for convergence is larger, but the savings with respect to the basic plan are more substantial.

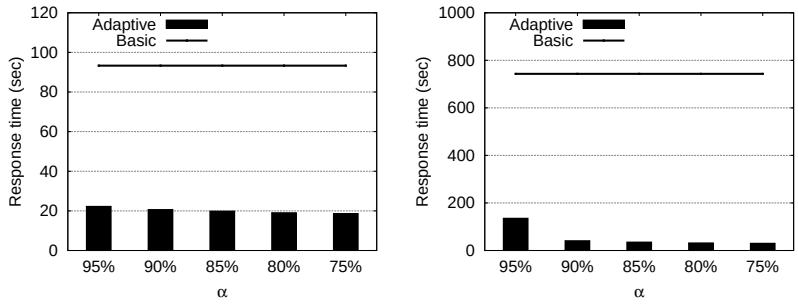
Figure 8 illustrates the response time as a function of the percentage of accuracy α, setting $\ell = 10\%$ and $n = 6,000$. As expected, the cost drops with the required accuracy, but the effect is more pronounced in *USA* where reducing the accuracy from 95% to 75% decreases the time from 135 to 30 seconds. The same reduction in *Germany* gains only about 3 seconds. It is worth pointing out that even for the inaccurate queries, the retrieved 10NN set is similar to the real one; i.e., 80%-90% of the actual nearest neighbors are in the query result.

In the next experiment, we evaluate the actual versus the expected accuracy of queries. Specifically, we first executed $n \cdot q_{\max} = 60,000$ queries based on which the LBS generated the AQP. Then, we performed another $100,000$ queries and measured the percentage for which the users obtain accurate results, i.e., the retrieved and the actual 10NN sets are identical. Note that the query distribu-



(a) *Germany* (b) *USA*

Fig. 7. Response time vs. percentage of trusted users



(a) *Germany* (b) *USA*

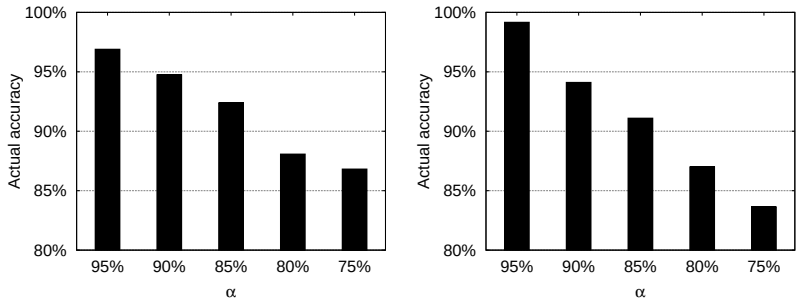
Fig. 8. Response time vs. accuracy α

tions in both cases are the same as that of the POIs. As shown in Figure 9, the accurate queries always exceed the desired accuracy level because the error incurred by the re-computation stage corresponds to a conservative estimation.

Summarizing the experimental evaluation, even the current state-of-the-art method may take several minutes (more than 10 for *USA*) to answer a nearest neighbor query. This implies that the results may be out-dated by the time they are received, especially for the case of mobile users. On the other hand, the proposed AQP approach achieves efficiency by sacrificing accuracy for a small percentage of queries (in our experiments, the default accuracy setting is 95%).

5 Conclusion

Strong location privacy requires that every query retrieves the same number of blocks in order to protect users from access pattern attacks. This has serious performance implications for both the LBS (in terms of processing cost) and



(a) *Germany* (b) *USA*

Fig. 9. Actual accuracy vs. α

the users (in terms of response time). To overcome the problem, we propose a novel approach that utilizes query statistics and ensures privacy by adopting the concept of ϵ-differential privacy. The trade-off is that accuracy is sacrificed for a small predefined percentage of queries. As shown in a comprehensive experimental evaluation with real POIs, ours is the first practical approach for strong location privacy in large datasets.

Acknowledgments. This work was supported by GRF grant 618011 from Hong Kong RGC.

References

1. G. Ács and C. Castelluccia. I have a dream!: Differentially private smart metering. In *IH*, 2011.
2. C. Böhm. A cost model for query processing in high dimensional data spaces. *TODS*, 25(2):129–178, 2000.
3. T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *TISSEC*, 14(3):26:1–26:24, 2011.
4. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.
5. M. Duckham and L. Kulik. A formal model of obfuscation and negotiation for location privacy. In *PERVASIVE*, 2005.
6. M. Duckham and L. Kulik. Simulation of obfuscation and negotiation for location privacy. In *COSIT*, 2005.
7. C. Dwork. A firm foundation for private data analysis. *CACM*, 54(1):86–95, 2011.
8. C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, 2006.
9. C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *STOC*, 2010.
10. W. Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

11. G. Ghinita. Privacy for location-based services. *Synthesis Lectures on Information Security, Privacy, & Trust*, 4(1):1–85, 2013.
12. G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: Anonymizers are not necessary. In *SIGMOD*, 2008.
13. G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: Anonymous location-based queries in distributed mobile systems. In *WWW*, 2007.
14. P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *Knowledge and Data Engineering, IEEE Transactions on*, 19(12):1719–1733, 2007.
15. A. Khoshgozaran, C. Shahabi, and H. Shirani-Mehr. Location privacy; moving beyond k-anonymity, cloaking and anonymizers. *KAIS*, 2010.
16. H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *ICPS*, 2005.
17. F. Korn, B.-U. Pagel, and C. Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *TKDE*, 13(1):96–111, 2001.
18. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 364–364. IEEE Computer Society, 1997.
19. F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.
20. M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The new casper: query processing for location services without compromising privacy. In *Proceedings of the 32nd international conference on Very large data bases*, pages 763–774. VLDB Endowment, 2006.
21. R. Ostrovsky and W. E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In *Public Key Cryptography–PKC 2007*, pages 393–411. Springer, 2007.
22. B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *ICDE*, 2000.
23. S. Papadopoulos, S. Bakiras, and D. Papadias. Nearest neighbor search with strong location privacy. *Proceedings of the VLDB Endowment*, 3(1-2):619–629, 2010.
24. N. Shang, G. Ghinita, Y. Zhou, and E. Bertino. Controlling data disclosure in computational pir protocols. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 310–313. ACM, 2010.
25. R. Sion and B. Carbunar. On the computational practicality of private information retrieval. In *In Proceedings of the Network and Distributed Systems Security Symposium*, 2007.
26. S. Wang, D. Agrawal, and A. El Abbadi. Generalizing pir for practical private retrieval of public data. In *Data and Applications Security and Privacy XXIV*, pages 1–16. Springer, 2010.
27. P. Williams and R. Sion. Usable pir. In *NDSS*, 2008.
28. M. L. Yiu, C. Jensen, X. Huang, and H. Lu. SpaceTwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile systems. In *ICDE*, 2008.