Protecting User Preference in Ranked Queries

Rong Tang
College of Cyber Security
Jinan University, Guangzhou China

Xinyu Yu
College of Cyber Security
Jinan University, Guangzhou China

Abstract—**Protecting data privacy is of extremely importance for users who outsource their data to a third-party in cloud computing. Although there exist plenty of research work on data privacy protection, the problem of protecting user's preference information has received less attention. In this paper, we consider the problem of how to prevent user preference information leakage from the third-party when processing ranked queries. We propose two algorithms to solve the problem, where the first one is based on distortion of preference vector and the second one is based on homomorphic encryption. We conduct extensive experiments to verify effectiveness of our approaches.**

Keywords—Preference privacy; distortion; homomorphic encryption

I. Introduction

Ranked queries are useful in many real world applications for users to express their tastes in queries, such that the returned query results are the most preferable ones than the other records in database. Currently, there are two kinds of approaches to capturing user's preferences, that is, the quantitative approach [1] and qualitative approach [2]. In the paper, we focus on quantitative approach for modeling user preference.

Given a relational database D = $\{A_1, A_2,..., A_d\}$ with numerical attribute A_i, i = 1,..., d, i.e., a linear function $q = \omega_1 * A_1 + \omega_2 * A_2 + ... + \omega_d * A_d$, where ω_1, ω_2,..., ω_d are non-negative weights given by the user as preferences toward the corresponding attributes, and $\omega_1 + \omega_2 + ... + \omega_d = 1$. Upon receiving a user ranked query q, for each record $r_i \in D$ the server computes its preference score as $Score(i) = \sum_{j=1}^{d} \omega_j * r_i.A_j$, j = 1,..., d, and returns to the user the top k records with the highest scores.

Although there are extensive research work on processing ranked queries, the privacy issue of user preferences so far has not received attention. There is no doubt that user's preference is a strong and direct link to his/her identity, which the user wants to keep in private, otherwise the information may be utilized by an adversary to against he/she. Thus, one may want to hide his/her preference embedded in the ranked query in order to protect privacy. To illustrate, we give two examples below.

Example 1. Consider a user looking for a second-hand car at a website running by a car dealer. The dealer maintains a used car database, recording for all the cars the features such as Make, Model, Year, MPG (miles per gallon), Mileage, and Price, etc. The user may care more about some of the features, such as MPG, mileage, and Year, and want to find a suitable car for him with the lowest price. By collecting and observing user's queries with preferred features, the dealer knows that the user strongly favors MPG. So the dealer may want to increase a bit the prices of the cars with favorable MPG, such that he is expected to profit more from the user. To have the edge over the car dealer while making a deal, the user wishes to hide his preferences during the querying process.

Example 2. Consider a customer searching for financial information of NYSE-listed companies, through websites such as Yahoo! finance or Google finance. By giving preferences on attributes such as cash flow, P/E (price-to-earnings) ratio, ROA (return on assets) ratio, and debt ratio, the customer intends to search through query interface the favorable companies, based on which he/she may make buy or sale decision for his/her investment portfolio management. On the other hand, a curious adversary may sniff the search results of the customer at the server side and extract the preferable attributes, which may be used to infer the possible investment of the customer. Since customer's portfolio information is critical for him/her to stand up to the fierce business competition, he/she may strongly oppose the exposure of his/her preferences to anyone else.

Problem Statement: Our model includes the user and the service provider (the server). The server maintains a database D and processes users' queries with difference preference on the attributes of D. We assume that the server is semi-honest, that is, the server correctly performs the query processing and returns the results, but he is curious and tries to find out the user's preferences. Each user composes his ranked query as a weight vector f = $\{\omega_1, \omega_2,..., \omega_d\}$ on attributes of D, and submits it to the server for processing. Our objective is to prevent the server from knowing the exact preferences, i.e., the weight vector of the user, without deteriorating the query processing efficiency and accuracy.

The contributions of this paper are as follows:

- We consider the problem of protecting user's preferences in ranked queries, which is of great importance in many real world applications.
- We propose a simple strategy to distort user's true preferences, and give a test to the strategy.
- To strengthen the privacy level, we devise another algorithm based on homomorphic encryption to protect user's preference.
- We conduct extensive experiments to verify effectiveness of the proposed approaches.

II. Background

A. Preliminaries

Ranked query. Ranked query [3] is very useful in many real-world applications. It is a powerful technology to simulate users' personalized information needs, which has attracted great attention of researchers all over the world. It allows users to express their preferences for queries and specify the specific weight of each query limit, so that the returned query results meet the needs of users better than other data records in the database. At present, there are two methods to describe user preferences, namely qualitative method and quantitative method.

The qualitative method [2] usually uses the preference formula to evaluate which data tuple is more favorable over the others, i.e. determining the partial order relationship between tuples. The quantitative method [1] defines a preference function to express the user's preference for different data attributes, and then finds the data records that best meet the user's needs from the database according to the preference function, that is, measured by specific values. For example, given a hotel dataset containing three dimensions $D = \{d_1, d_2, d_3\}$, where d_1 represents price, d_2 represents score, d_3 represents distance, preference function is defined as $f = \omega_1 * d_1 + \omega_2 * d_2 + \omega_3 * d_3$ according to user preference $f = \{\omega_1, \omega_2, \omega_3\}$. The highest score is the most desirable.

In practical applications, ranked query is often directly related to other problems, such as skyline computing [4] and top-k query [5], [6]. Combining multiple queries can usually get the results you want most quickly.

Top-k query. Top-k query [7], [8] refers to returning the best k data records according to an objective function. It is widely used in many fields, such as e-commerce, recommendation system, search engine and so on. When the Top-k query is associated with the preference problem, it will calculate the score of each data record according to the preference function given by the user, and then return the k objects with the smallest (or largest) scores.

The concept of Top-k query has been around for many years and is widely used in real life. So far, many algorithms for Top-k query have been proposed, which can be roughly divided into three categories. The first is index based algorithm [9], [10]. Its main idea is to divide the whole data set into multiple layers according to the division rules, then index and mark each layer, and finally retrieve layer by layer according to the index order to return the best k results, such as Onion algorithm [11]. The second is the Top-k query algorithm based on view [12]. This kind of algorithm first calculates the scores corresponding to each tuple according to the preference function provided by the user and arranges them in order. The view contains the identifier and score of tuples. After these preparations, it finally returns the Top-k query results. The third is the Top-k query algorithm based on ordered list [13], [14], which is realized by using multiple column files.

B. Related Work

Skyline query. Skyline query problem [4], [15], [16] is a popular technology for processing user preferences and Top-k query. It is used to select a series of objects that meet user preferences and are not dominated by other objects. Given a dataset D with d-dimensions $\{d_1, d_2,..., d_d\}$ and n $objects\{A_1, A_2,..., A_d\}$, where $A_i.d_j$ denotes the j-th dimension value of object A_i. The definition of dominance and skyline are as follows:

Dominance: An object $A_i \in D$ is said to dominate another object $A_j \in D$, denoted as $A_i \prec A_j$, if $A_i.d_r \leq A_j.d_r (1 \leq r \leq d)$ for all d dimensions and $A_i.d_t < A_j.d_t (1 \leq t \leq d)$ for at least one attribute. We call such A_i as *dominant object* and such A_j as *dominated object* between A_i and A_j.

Skyline: An object A_i is said to be a skyline object of D, if and only if there is no such object $A_j \in D (j \neq i)$ that dominates A_i. The skyline of D is the set of skyline objects in D.

At present, there are mainly two kinds of methods for skyline query processing. The first kind of methods do not need to preprocess the data set, but retrieve the query by scanning the whole database at least once, such as block nested loop (BNL), divide and conquer, etc. BNL algorithm [17] adopts the most straightforward method, that is, a point p is compared with each other point to decide whether it is dominated by other points, so as to determine whether the point is part of the skyline. Divide and conquer algorithm [17] divides the universe into several regions, calculates the skyline in each region, and produces the final skyline from the regional skylines. Therefore, the performance of this kind of methods is low because of scanning the whole database.

The second [18] is to reduce the query cost by using the index structure, such as nearest neighbor(NN), branch and bound skyline(BBS), etc. NN and BBS find the skyline by using an R-tree. NN algorithm [19] divides the data space iteratively based on the nearest object in the space, and prunes the dominant object quickly and effectively. However, BBS algorithm [20] uses heap to realize progressive search without redundant query in subspace. Obviously, the difference is that NN issues multiple NN queries [21], whereas BBS performs only a single traversal of the tree. It has been proved [22] that BBS is I/O optimal; that is, it accesses the least number of disk pages among all algorithms based on R-trees (including NN).

Homomorphic encryption(HE). Homomorphic cryptographic system [23], [24] is a public-key cryptosystem that can provide user with the ability to directly perform algebraic operations on ciphertext without decrypting the ciphertext.

Given two messages m_1 and m_2, suppose a homomorphic cryptographic system encrypts them, using public key $\mathcal{PK}$, to ciphertexts $C_1 = E(\mathcal{PK}, m_1)$ and $C_2 = E(\mathcal{PK}, m_2)$. Without knowing the corresponding secret key, one can compute $E(\mathcal{PK}, m_1 + m_2)$, i.e., the ciphertext of the addition of m_1 and m_2, by simply multiplying the two ciphertexts $C_1 \times C_2$. This property is called additive homomorphicsm. Similarly, a crytographic system is multiplicatively homomorphic if one can derive $E(\mathcal{PK}, m_1 \times m_2)$ from C_1 and C_2 directly.

The concept of homomorphic encryption [25], [26] is proposed to directly perform operations in the encryption domain, that is, the results obtained by decrypting the operations performed in the ciphertext domain are consistent with those obtained by performing the same operations in the plaintext domain. However, most existing homomorphic encryption schemes only support accurate computing operations in some discrete spaces, so these schemes can not be applied to tasks requiring floating-point or real number computing. For example, in the bit-wise encryption scheme, the integer is first converted into binary, and then encrypted by bit. The addition and multiplication operations are also based on bits. This scheme cannot be applied to floating-point numbers. For the word-wise encryption scheme, multiple numbers can be encrypted in a single ciphertext, but the rounding operation is difficult to evaluate because it is not expressed as a decimal polynomial.

After Regev [27] introduced the learning with errors (LWE) problem, approximate homomorphic encryption was proposed one after another. Since the key of the method based on LWE problem is realized through matrix, its efficiency will decrease rapidly with the increase of security parameters. Then the ring-LWE problem [28] is proposed. The key of the encryption scheme based on this problem is expressed by several polynomials, which greatly reduces the size of the key and speeds up the encryption and decryption operations. The scheme based on RLWE problem include BFV [26], BGV [29] and HEAAN [30]. BFV and BGV encryption scheme only support accurate computing operations on integers.

However, HEAAN scheme can encrypt floating-point numbers. And the goal of this scheme is efficient approximate calculation on HE. Its main idea is to add a noise to the plaintext number that can reflect important information, so that the addition and multiplication operations of encrypted messages can be approximately calculated. In HEAAN encryption scheme, its decryption structure of the form $\langle \mathbf{c},\ sk \rangle = m + e$ (mod q) where e is a small error inserted to guarantee the security. In addition, HEAAN also provides a rescaling operation to remove the error of the least significant bit, which ensures that the length of the error bit increases linearly in proportion to the number of levels consumed, rather than exponentially. The efficiency of HEAAN [30] has been proved in many practical applications, and is still being improved by better bootstrapping algorithms [31], [32]. Therefore, considering that the data used in this paper are floating-point numbers, HEAAN homomorphic encryption scheme will be adopted.

Approximate Algorithm for Comparison Function. Logical operation has always been a difficult point in HE. Bit-wise FHEs encrypt data in a bit-by-bit manner. They support fast logical operations, such as comparison. But they can not support floating point encryption. On the contrary, word-wise FHEs, which store messages as their word-sized numbers, support high-speed arithmetic operations between messages. Therefore, in order to calculate the comparison function, an approximate form of the comparison function is proposed by using polynomials.

Cheon et al. [33] first proposed a new identity $comp(a, b) = \lim_{k\to\infty} a^k/(a^k + b^k)$, and showed that the identity can be computed by an iterative algorithm. Because of the iterative feature, this algorithm is slow in HE implementation. Then, they proposed a new comparison methods SIMD [34], using composite polynomial approximation on the sign function, which is equivalent to the comparison function. That is, repeated compositions of ($2n$+1)-degree polynomial $f_n(x)$ and $g_n(x)$ output the approximate value of the sign function.

We denote the approximate comparison for two inputs x,y by $(x > y)$ or $(y < x)$. According to the conclusion of [34], given iteration numbers d_f and d_g, $(x > y)$ is computed as follows

$$(x > y) := (f_n^{d_f} \circ g_n^{d_g}(x - y) + 1)/2 \qquad (1)$$

TABLE I. An Example Database

Record	A_1	A_2
r_1	1.4	4.4
r_2	2.5	2.9
r_3	3.5	1.0
r_4	3.3	4.1
r_5	4.2	2.8
r_6	1.9	2.3

Here f_d means $f \circ f \circ \cdots \circ f$, i.e., the operation is performed d times.

III. Our Approaches for Preference Protection

Ranked queries are complementary to traditional SQL query semantics. The preferences expressed by the user are considered as *soft constraints* of the queries, whereas the hit-or-miss query conditions, e.g., $\leq$, $>$, and $\neq$, are known as *hard constraints* [1], [2], [35]. To evaluate a ranked query q, the server computes the sum of the linear combination of attributes of the records, based on the user preference vector $f = \{\omega_1, \omega_2,.., \omega_d\}$(note we use vector and weight interchangeably), then returns to the user the top-k objects with the highest sum.

Table I gives an example of a toy database with 2 attributes A_1 and A_2. Suppose the user expresses in a query his preference as $f = \{0.4, 0.6\}$, and wants to pull top 3 favorable records from the server. Starting from the first record r_1, the server computes its preference score *score*(r_1) = 0.4 $*$ 1.4 + 0.6 $*$ 4.4 = 3.2, and the scores of the rest records. Among the computed scores {3.2, 2.7, 2, 3.8, 3.4, 2.1}, r_1, r_4, and r_5 are the top 3 records with highest scores and they are returned as result to the user. As this example shows, there is no protection for user queries, which means that the server can easily obtain the preference information of a user. In this section, we propose two approaches to prevent leakage of user preference, which are described below.

A. The First Approach

Our first approach is called PD, which is based on preference distortion. So far in the model we assume that there is no encryption involved, i.e., the user queries and database content are all in plaintext. We defer discussion of the case in which encryption is employed in the next section.

Our strategy to protect user's true preference is simple, and goes as follows. Instead of directly submitting to the server the true preference vector f, the user distorts f by randomly adding to or subtracting from the components of f a small number, then sends the modified preference vector to the server. Specifically, given a preference vector $f = \{\omega_1, \omega_2,.., \omega_d\}$, we transform f into a new vector $f' = \{\omega_1', \omega_2',.., \omega_d'\}$, such that $\omega_i' \in [0, 1]$ and $\sum_i^d \omega_i' = 1$. Suppose the increment/decrement added to component ω_i of f is δ_i, based on the relation $\omega_i' = \omega_i + \delta_i$ we can easily verify the following equation

$$\delta_1 + \delta_2 + ... + \delta_d = 0 \qquad (2)$$

where $-\omega_i \leq \delta_i \leq 1 - \omega_i$.

We outline the preference distortion in Algorithm 1. We generate d random numbers, and normalize them to maintain the convex property. These random numbers are sorted in descending order, such that the relative preference relation among attributes is maintained. For example, we are given a vector *f* = {0.3, 0.5, 0.2}, and the generated random numbers are *R* = {0.4, 0.2, 0.8}. After normalization and sorting, we get *R* = {0.6, 0.3, 0.1}, from which we get the distorted vector *f'* = {0.3, 0.6, 0.1}, and the increment/decrement vector Δ = {0, 0.1, -0.1}. It is obvious that after distortion, the new vector still preserves the relative preference relation among attributes as in the original vector, but the actual value of preference weights have been changed.

Algorithm 1: Preference Distortion

```
    input : Preference vector $f = \{\omega_1, \omega_2, .., \omega_d\}$
    output: Distorted preference vector $f'$
1   $R' = \Delta = f' = \emptyset$;
2   Generate a set $R$ of $d$ random numbers;
3   for $\forall rand_i \in$ R do
4       $rand'_i = \frac{rand_i}{\sum_j rand_j}$;
5       Insert $rand'_i$ into $R'$;
6   end
7   Sort $R'$ in descending order;
8   count = len($f$);
9   j = 1;
10  while $count > 0$ do
11      i = the index of the greatest weight in f;
12      $\omega'_i = rand'_j$;
13      $\delta_i = \omega'_i - \omega_i$;
14      $f$[index] = 0;
15      Insert $\omega'_i$ and $\delta_i$ into $f'$ and $\Delta$,respectively;
16      count = count - 1;
17      j = j + 1;
18  end
19  Return $f'$;
```

Having obtained the distorted vector *f'*, the user sends it to the server for processing, where at the server side the query processing is just as the same as before and the query results and scores are sent back to the user. Since the scores do not reflect the true values for the original preference vector, the user has to revert the scores returned by the server. Consider the score of a record r_i with respect to *f'*, $score(i) = f' \cdot r_i$, which can be represented as

$$(\omega_1 + \delta_1 \times r_i.a_1 + \omega_2 + \delta_2 \times r_i.a_2 + ... + \omega_d + \delta_d \times r_i.a_d)$$

After re-arrangement of the above formula we have

$$(\omega_1 \times r_i.a_1 + ... + \omega_d \times r_i.a_d) + (\delta_1 \times r_i.a_1 + ... + \delta_d \times r_i.a_d)$$

It is clear that the first parenthesized part is the correct score with respect to *f*, and the second part is the noise artificially added by the distorted vector *f'*. Thus, the user computes the noise for each r_i of the return top k records by multiplying Δ with r_i, and then subtracting the noise from the returned score. After the reverting procedure, the user re-orders the resulting records according to the restored scores. We summarize the reverting procedure in Algorithm 2.

The algorithm of preference distortion is simple and efficient for protecting user preferences, however it may introduce false negative and false positive query result, as we will show in the next section.

Algorithm 2: Score Reverting

```
    input : Query result set $S$, and the set $Score$
    output: The restored score set $Score'$
1   $Score = \emptyset$;
2   for Score$(i) \in$ Score do
3       $Noise = \delta_1 \times r_i.a_1 + ... + \delta_d \times r_i.a_d$;
4       $Score'(i) = Score(i) - Noise$;
5       Insert $Score'$(i) into $Score'$;
6   end
7   Sort the records in $S$ according to $Score'$;
8   Return $Score'$;
```

Security Discussion. When the user sends the preference vector to the server for query, even if the preference vector is disturbed, the server may predict the user's real vector according to the returned top k result, that is, the higher the precision, the higher the probability of being predicted. It is known that the weight of the user preference vector is a decimal between [0,1], and the server cannot know the specific decimal places of each weight in the real vector. Obviously, the top k result corresponding to the vector in a small range is the same. Therefore, when the server predicts the user preference vector according to the top k query results, there are countless possibilities in the case of uncertain decimal places of each weight in the vector. Especially when the dimension is higher and the number of decimal places is more, there is a lower probability of being predicted by the server.

B. The Second Approach

To further strengthen the privacy level of user's preferences, in this section we propose the second approach called HE, which is based on homomorphic encryption.

Considering that the attributes of a record are floating-point numbers, and HEAAN [30] scheme is selected as the encryption scheme. Assume the secrete/public key pair of HEAAN homomorphic encryption system is $\langle \mathcal{SK}, \mathcal{PK} \rangle$. The user encrypts his plaintext preference weights *f* = {ω_1, ω_2,..., ω_d} by using the public key $\mathcal{PK}$, which gives the encrypted weights *E(f)* = {$E(\omega_1)$, $E(\omega_2)$,..., $E(\omega_d)$}(see Algorithm 3).

Algorithm 3: Preference Encryption

```
    input : Preference vector $f$,the public key of
            CKKS encryption system $\mathcal{PK}$
    output: Encrypted preference vector $E(f)$
1   $E(f) = \emptyset$;
2   for $\forall \omega_i \in$ f do
3       $E(\omega_i) = \langle \omega_i, \mathcal{PK} \rangle$;
4       Insert $E(\omega_i)$ into $E(f)$;
5   end
6   Return $E(f)$;
```

After the preference vector encryption operation is completed, the next step is to perform the linear weighted summation operation on the data records in the data set, and select the k results with the top scores through comparison.

The comparison operation of ciphertext is realized by SIMD [34] scheme. In order to reduce the computational consumption of encryption, we consider that the server first performs K-Skyband operation on the dataset *D*, and then only calculates and compares the data records in the K-Skyband dataset. Similar to K nearest-neighbor queries, a K-Skyband [22] query reports the set of points which are dominated by at most $K-1$ points.The definition of K-Skyband is as follows.

K-Skyband: An object $A_i \in D$ is said to be a K-Skyband object of *D*, if A_i is dominated by at most $K-1$ objects in *D*.

As can be seen from the definition, K-Skyband [22] is a variant of skyline, in which K can be regarded as the thickness of skyline; the case $K = 0$ corresponds to a conventional skyline. Thus, we implement K-Skyband by modifying the BBS algorithm. Based on this, the specific process of query is shown in algorithm 4.

```
Algorithm 4: Query Evaluation
   input : Encrypted preference vector $E(f)$
   output: Result records and their encrypted scores
1  $Score = \emptyset$;
2  Compute the set $Band_k$ of k-skyband points of $D$;
3  for $\forall r_i \in \text{Band}_k$ do
4      $E(score(i)) = \sum_{\forall a_j \in r_i} E(\omega_j) * a_j$;
5      Insert $E(score(i))$ into $Score$,and compare;
6  end
7  Send $Score$ and $Band_k$ to the user;
```

Security Discussion. This scheme encrypts the preference vector. The IND-CPA security nature of FHE scheme ensures that any opponent with the result homomorphic operation between ciphertext and ciphertext cannot extract any information of the message in the ciphertext, our HE method is secure based on the security of FHE, because the server can only access the ciphertext. There is no information leak of user preference.

IV. Experimental Results

In this section we conduct extensive experiments to evaluate performance of the proposed two approaches, and all experiments are conducted on a PC running Ubuntu 18.04.2 LTS. We discuss experiment settings below.

Dataset. We use synthetic datasets of three data distributions, namely, independent (uniform), correlation and anti-correlation, with dimensionality *d* varying in the range [2,5] and cardinality *N* in the range [1k,20k], respectively.

Performance Indicators. For our first approach PD, since it is an approximate method, we focus on precision and recall indicator, i.e., how many real results can be found for PD. On the other hand, for our second approach HE, since it is an exact method, i.e., HE can find all the correct results, we employ running time as the performance indicator for the HE approach. The precision and recall indicator is defined as

$$precision = \frac{TP}{TP + FP} \tag{3}$$

$$recall = \frac{TP}{TP + FN} \tag{4}$$

where TP indicates that the number of positive query results retrieved, FP indicates the number of points which the retrieved positive ones are actually negative, FN indicates the number of points which it has not been retrieved but they are actually positive. Each precision and each recall result is the average of 10 trails.

Suppose $Q(x)$ is defined as the number of data points in area x. Based on the premise of Top-k preference query in this paper, it is easy to obtain

$$Q(TP) + Q(FP) = k$$

and

$$Q(TP) + Q(FN) = k$$

then $Q(FP) = Q(FN)$. Therefore, the results of precision and recall are the same.

A. Effect of Dimensionality d

We vary data dimensionality *d* from 2 to 5. As shown in Fig. 1, the cardinality N is $10k$ and K is 50. And it can be seen that the average precision and recall results of datasets with independent (uniform), correlation and anti-correlation distribution are basically stable and were not be affected by the dimensionality. It is obvious that the average precision and recall results of correlation data set are the highest, while that of anti-correlation are poor.

B. Effect of K

In order to study the effect of K we carried out experiments when $K = 10, 20, 50, 100$. Fig. 2 shows the precision and recall corresponding to each case. Obviously, the query results are not affected by the K value.

C. Effect of Cardinality N

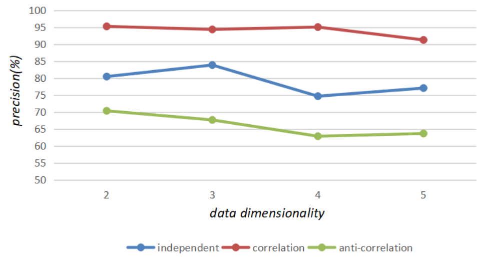
Fig. 3 shows the precision and recall results when the cardinality *N* in the range [1k,20k]. It can be clearly seen that the precision and recall values of the three types of datasets tend to be stable.

In conclusion, our proposed distortion algorithm has little relationship with the size of K, data dimension and data set cardinality. Thus, the disturbance scheme is practical and will not be affected by other factors. In addition, it can be clearly seen that it has a good precision and recall result in correlation datasets but poor in anti-correlation datasets.

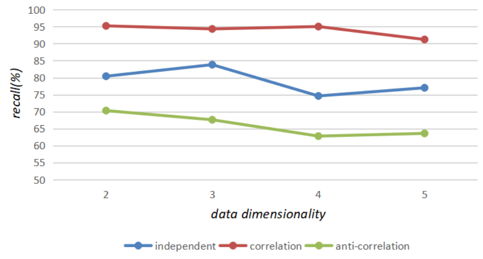
D. Discussion

From the above experimental results, it can be obviously seen that precisions are float up and down. That is due to the existence of false negative and false positive points, and the number of them is uncertain.

For ease of discussion, we use the example database in Table I, and assume the data space range for attribute A_1, A_2 are $x = [0, U_x]$, $y = [0, U_y]$, respectively, and $k = 3$ for the returned top k results. Now consider an original vector *f* = $\{\omega_1, \omega_2\}$ and the corresponding transformed vector *f'* = $\{\omega'_1, \omega'_2\}$. After examining each vector against the records in **D**, the qualified records (represented as data points in 2-D space) are highlighted in Fig. 4(A). Specifically, the top 3 records with

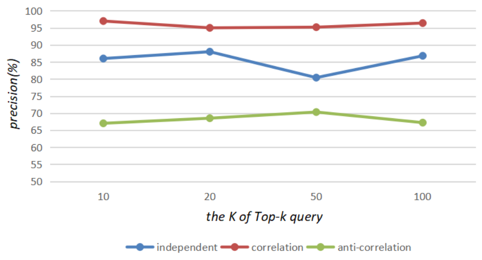


(a) precision

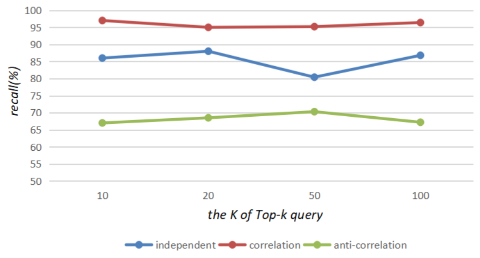


(b) recall

Fig. 1. Precision & Recall on a Dataset with Top-50 Query, $10k$ Records and Varying Dimensionality d.

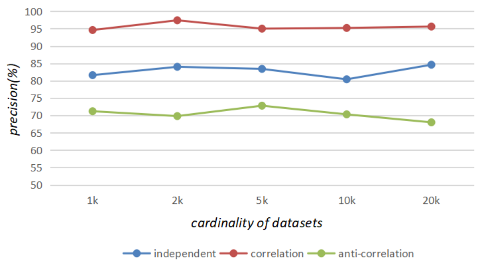


(a) precision

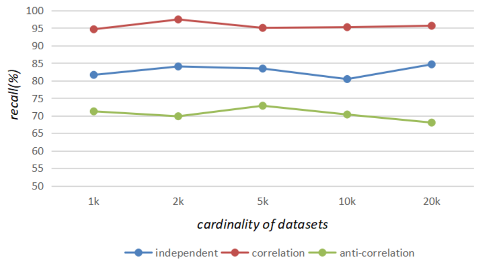


(b) recall

Fig. 2. Precision & Recall on a Dataset of $10k$ Records in $3D$ Versus K.



(a) precision



(b) recall

Fig. 3. Precision & Recall on a Dataset with Top-50 Query and Varying Cardinality N.

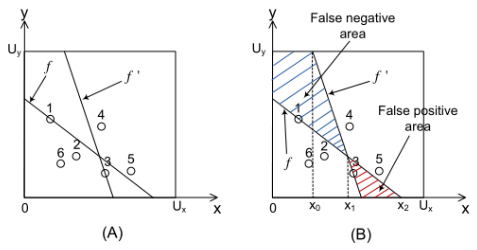


Fig. 4. Original and Distorted Ranked Queries.

respect to the original vector f include r_4, r_5, r_1, whereas the answers for distorted vector f' are r_4, r_5 and r_3. There is a false positive result, r_3, and a false negative r_1, due to the weight distortion.

To evaluate the original and distorted preference vector, one has to quantify the number of false positive and negative results. As depicted in Fig.4(B), it is clear that the larger striped area (denoted by FN) contains false negative points, and the smaller striped area (denoted by FP) false positive points. The area of FN and FP can be easily calculated by computing the following integrals.

$$FN = \int_0^{x_0} (U_y - f)dx + \int_{x_0}^{x_1} (f' - f)dx \tag{5}$$

$$FP = \int_{x_1}^{x_2} (f - f')dx \tag{6}$$

Under the assumption of uniform distribution of the data records in D, we can estimate the number of false negative and false positive results as $N*FN$ and $N*FP$ respectively, where N is the cardinality of D. On the other hand, the computation of the FP and FN becomes complicated in the case of high

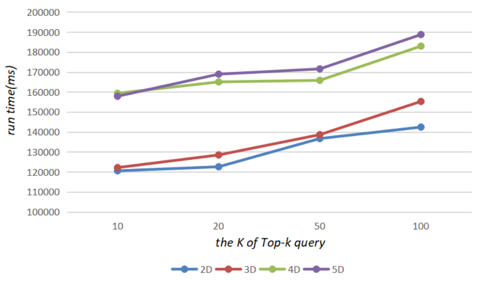


Fig. 5. Running Time on a Dataset by Varying K.

dimensional data. However, one may resort to techniques such as Monte Carlo method to approximate the integral in high-dimensional case. This is a future direction of our work.

E. Performance of HE Approach

For HEAAN, we fix the dimension $N = 2^{17}$, $\Delta = 2^{40}$, and hamming weight $h = 128$ and the secret key is chosen from the ring with ternary secret distribution, i.e., all nonzero coefficients of secret key are ± 1. For SIMD, we employs the approximate comparison method where f_n and g_n are chosen by $n = 3$, where

$$f_3(x) = (35x - 35x^3 + 21x^5 - 5x^7)/2^4 \quad (7)$$

$$g_3(x) = (4589x - 16577x^3 + 25614x^5 - 12860x^7)/2^{10} \quad (8)$$

We conduct experiments with different k and dimensionality d varying in the range [2,5], and report the corresponding running time. Fig. 5 shows the running time when k = 10, 20, 50, 100 with $10k$ objects.

When K in K-Skyband is large, the data to be calculated decreases relatively, but the running time still increases steadily with the increase of dimensionality.

V. Conclusion

In order to protect user preference privacy information, we propose two approaches to solve the problem of user preference privacy protection in ranked queries. The first method, called PD, hides the user's real preference information by introducing perturbation to the user ranked query vector. This scheme will lead to a slight degradation in precision in the query results. Moreover, the experiment results also show that PD achieves the best performance on dataset with correlation distribution, whereas it performs relatively poor on dataset with anti-correlation distribution. Therefore, PD is suitable for real-time ranked query processing scenarios with less accurate requirement for the query result. In order to get exact query result, a homomorphic encryption-based method called HE is proposed to encrypt the preference vector, which enables the third-party server to process ranked query by calculating the ciphertext, so as to fully protect the privacy of user preference.

Acknowledgment

We would like to thank the anonymous reviewers for reviewing this paper. This work is supported in part by National Natural Science Foundation of China (Grant No. 61972177), Science Plan of Yunfu city (Grant No. 2021020302), Guangdong China. Rong Tang and Xinyu Yu contribute equally to this paper.

References

[1] R. Agrawal and E. L. Wimmers, "A framework for expressing and combining preferences," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 297–306.

[2] J. Chomicki, "Preference formulas in relational queries," *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, pp. 427–466, 2003.

[3] Y. Gao and Q. Liu, "Introduction to preference query analysis and optimization," in *Preference Query Analysis and Optimization*. Springer, 2017, pp. 1–7.

[4] Y. Wang, Z. Shi, J. Wang, L. Sun, and B. Song, "Skyline preference query based on massive and incomplete dataset," *IEEE Access*, vol. 5, pp. 3183–3192, 2017.

[5] P. K. Agarwal, S. Sintos, and A. Steiger, "Efficient indexes for diverse top-k range queries," in *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2020, pp. 213–227.

[6] T. M. N. Le, M. L. T. Be, and D. H. T. Ngoc, "A processing of top-k aggregate queries on distributed data," in *IOT with Smart Systems*. Springer, 2022, pp. 815–826.

[7] A. Shanbhag, H. Pirk, and S. Madden, "Efficient top-k query processing on massively parallel hardware," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1557–1570.

[8] W. Ren, X. Lian, and K. Ghazinour, "Effective and efficient top-k query processing over incomplete data streams," *Information Sciences*, vol. 544, pp. 343–371, 2021.

[9] S.-Y. Ihm, K.-E. Lee, A. Nasridinov, J.-S. Heo, and Y.-H. Park, "Approximate convex skyline: a partitioned layer-based index for efficient processing top-k queries," *Knowledge-Based Systems*, vol. 61, pp. 13–28, 2014.

[10] D. Xin, C. Chen, and J. Han, "Towards robust indexing for ranked queries," in *VLDB*. Citeseer, 2006, pp. 235–246.

[11] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, "The onion technique: Indexing for linear optimization queries," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of data*, 2000, pp. 391–402.

[12] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering top-k queries using views," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 451–462.

[13] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of computer and system sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[14] Z. Gong, G.-Z. Sun, J. Yuan, and Y. Zhong, "Efficient top-k query algorithms using k-skyband partition," in *International Conference on Scalable Information Systems*. Springer, 2009, pp. 288–305.

[15] K. Mouratidis, K. Li, and B. Tang, "Marrying top-k with skyline queries: Relaxing the preference input while producing output of controllable size," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1317–1330.

[16] Z. Zheng, M. Zhang, M. Yu, D. Li, and X. Zhang, "User preference-based data partitioning top-k skyline query processing algorithm," in *2021 IEEE International Conference on Industrial Application of Artificial Intelligence (IAAI)*. IEEE, 2021, p. 436—444.

[17] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings 17th international conference on data engineering*. IEEE, 2001, pp. 421–430.

[18] N. Sukhwani, V. R. Kagita, V. Kumar, and S. K. Panda, "Efficient computation of top-k skyline objects in data set with uncertain preferences," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 17, no. 3, pp. 68–80, 2021.

[19] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 275–286.

[20] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “An optimal and progressive algorithm for skyline queries,” in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 467–478.

[21] G. R. Hjaltason and H. Samet, “Distance browsing in spatial databases,” *ACM Transactions on Database Systems (TODS)*, vol. 24, no. 2, pp. 265–318, 1999.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “Progressive skyline computation in database systems,” *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 41–82, 2005.

[23] J. H. Cheon and D. Stehlé, “Fully homomophic encryption over the integers revisited,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 513–536.

[24] Y. Doröz, Y. Hu, and B. Sunar, “Homomorphic aes evaluation using the modified ltv scheme,” *Designs, Codes and Cryptography*, vol. 80, no. 2, pp. 333–358, 2016.

[25] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, “Improved security for a ring-based fully homomorphic encryption scheme,” in *IMA International Conference on Cryptography and Coding*. Springer, 2013, pp. 45–64.

[26] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical gapsvp,” in *Annual Cryptology Conference*. Springer, 2012, pp. 868–886.

[27] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[28] V. Lyubashevsky, C. Peikert, and O. Regev, “On ideal lattices and learning with errors over rings,” in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2010, pp. 1–23.

[29] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[30] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[31] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, “Bootstrapping for approximate homomorphic encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 360–384.

[32] J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, “Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation,” *Cryptology ePrint Archive*, 2022.

[33] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, “Numerical method for comparison on homomorphically encrypted numbers,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2019, pp. 415–445.

[34] J. H. Cheon, D. Kim, and D. Kim, “Efficient homomorphic comparison methods with optimal complexity,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020, pp. 221–256.

[35] W. Kießling, “Foundations of preferences in database systems,” in *VLDB’02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 311–322.