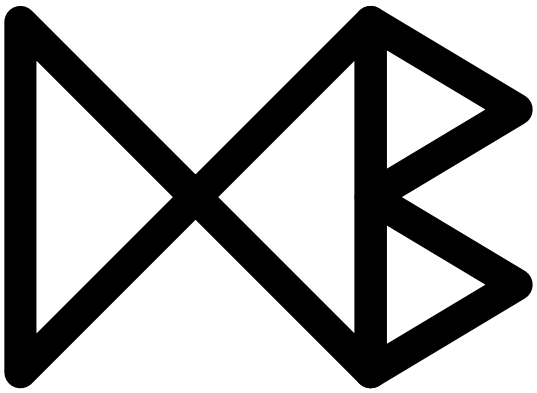


Outsourced Similarity Search on Metric Data Assets

Man Lung Yiu, Ira Assent, Christian S. Jensen, and Panos Kalnis

July 27, 2010

TR-28

A DB Technical Report

Title	Outsourced Similarity Search on Metric Data Assets
	Copyright © 2010 Man Lung Yiu, Ira Assent, Christian S. Jensen, and Panos Kalnis. All rights reserved.
Author(s)	Man Lung Yiu, Ira Assent, Christian S. Jensen, and Panos Kalnis
Publication History	Extended version of: M. L. Yiu, I. Assent, C. S. Jensen, and P. Kalnis, "Outsourced Similarity Search on Metric Data Assets," to appear in *IEEE Transactions on Knowledge and Data Engineering*. July 2010. A DB Technical Report.

For additional information, see the DB TECH REPORTS homepage: ⟨www.cs.aau.dk/DBTR⟩.

Any software made available via DB TECH REPORTS *is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranty of merchantability and fitness for a particular purpose.*

The DB TECH REPORTS icon is made from two letters in an early version of the Rune alphabet, which was used by the Vikings, among others. Runes have angular shapes and lack horizontal lines because the primary storage medium was wood, although they may also be found on jewelry, tools, and weapons. Runes were perceived as having magic, hidden powers. The first letter in the logo is "Dagaz," the rune for day or daylight and the phonetic equivalent of "d." Its meanings include happiness, activity, and satisfaction. The second letter is "Berkano," which is associated with the birch tree. Its divinatory meanings include health, new beginnings, growth, plenty, and clearance. It is associated with Idun, goddess of Spring, and with fertility. It is the phonetic equivalent of "b."

Abstract

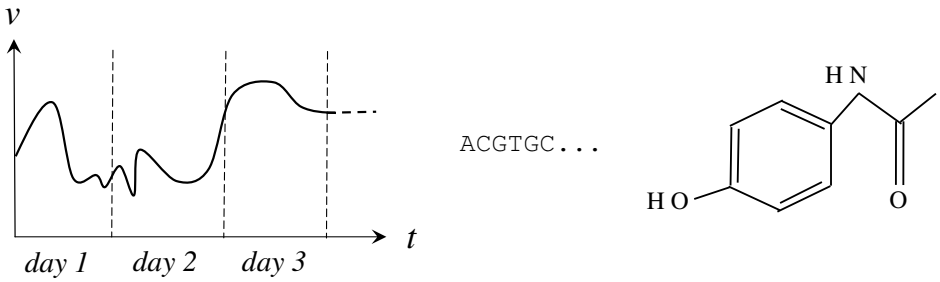
This paper considers a cloud computing setting in which similarity querying of metric data is outsourced to a service provider. The data is to be revealed only to trusted users, not to the service provider or anyone else. Users query the server for the most similar data objects to a query example. Outsourcing offers the data owner scalability and a low initial investment. The need for privacy may be due to the data being sensitive (e.g., in medicine), valuable (e.g., in astronomy), or otherwise confidential. Given this setting, the paper presents techniques that transform the data prior to supplying it to the service provider for similarity queries on the transformed data. Our techniques provide interesting trade-offs between query cost and accuracy. They are then further extended to offer two intuitive privacy guarantees. Empirical studies with real data demonstrate that the techniques are capable of offering privacy while enabling efficient and accurate processing of similarity queries.

1 Introduction

Advances in digital measurement and engineering technologies enable the capture of massive amounts of data in fields such as astronomy, medicine, and seismology. The effort for data collection and processing, as well as its potential utility for research or business, create value for the *data owner*. He wishes to store them and allow access by himself, colleagues, and other (trusted) scientists or customers. This can be supported by outsourced servers that offer low storage costs for large databases. For instance, outsourcing based on cloud computing is becoming increasingly attractive, as it promises pay-as-you-go, low storage costs as well as easy data access. However, care needs to be taken to safeguard data that is valuable or sensitive against unauthorized access. In this context, we call any item in a data collection an *object*, individuals with authorized access *query users*, and the entity offering the storage service the *service provider*.

We illustrate the sensitivity issues with several scenarios. First, consider space programs such as the NASA Apollo program on the Earth's Moon[1] or the ESA Mars Express[2] that collect scientifically valuable and rare data. The NASA data is known to be private before it is released to the public. For example, time series data is collected from sensors to study the atmosphere's density. Such data is usually analyzed by the scientists involved in setting up the instruments, prior to being made available to the general community. At the early stage, access is restricted to authorized scientists for first analysis, because of the substantial efforts invested in building, testing, and deploying instruments, and in refining the data prior to use. Such valuable data needs protection when outsourced, to ensure that the investments by scientific groups are decently rewarded.

To analyze the data, authorized scientists may search for similar patterns in collected time series, such as certain daily or hourly sub-sequences that indicate interesting phenomena. In this scenario, time series can be represented as vectors of values in chronological order (see Fig. 1a). At query time, a user specifies an example time series q and wishes to obtain those time series most similar to q; the system then retrieves the time series p in the database with the minimum distance to q.



(a) sensor time-series (b) DNA string (c) molecule (graph)

Figure 1: Application Examples of Valuable Data

As a second scenario, consider biologists analyzing DNA microarray data to understand the functioning of genes or gene groups, for instance from the Stanford Microarray Database[3]. A DNA microarray is a matrix

[1] http://ti.arc.nasa.gov/project/planetary/moon/
[2] http://sci.esa.int/marsexpress/
[3] http://genome-www5.stanford.edu

obtained by subjecting gene samples (rows in the matrix) to different experimental conditions (columns in the matrix). Genes that follow the same expression pattern on all or a subset of the experiments might be part of a common control mechanism. For a given gene, its expression values form a query vector. Biologists query the database of experiments to identify those genes that are most similar to this specific expression pattern and that are therefore most likely to be linked to this gene. Generating DNA data is very costly due to the material and time invested. Hence, as in the first scenario, the data are costly and rare, making them very valuable to their owners.

Many applications in science and business rely on similarity search of metric data other than time series and vector data. Computer-aided gene sequencing uses the similarity between an unknown sequence from one species and a known sequence from a closely related species to predict the former's function[4] (see Fig. 1b). In drug design, pharmacists search for the most similar graph structures to their quest for a suitable molecule, which can be represented as a labeled graph[5], as shown in Fig. 1c.

In general, the above diverse scenarios have the following common characteristics: valuable data in a metric space are searched based on a similarity measure. When these data are outsourced, they must be secured against leaks or attacks.

Shortcomings of existing methods.
In the literature, a number of concepts for securing databases have been studied. *Private information retrieval* techniques [15] hide the user's query, e.g., the data item searched for, but not the data being queried. To outsource valuable data to an insecure server, such techniques are clearly not appropriate. *Digital watermarking* [2] establishes the data owner's identity on the data. Additional information stored in the data helps prove ownership, but it cannot prevent an attacker from illegally copying the dataset. *Anonymization techniques* [25] secure data by releasing only a generalized version. Aggregate statistical analysis is still possible on the generalized data, but the result of a specific query is not guaranteed to be accurate.

Traditional *encryption methods* are capable of protecting the confidentiality of the data. However, this also prevents users from querying the data on the untrusted server. Obviously, transferring all the encrypted data to the query user for searching takes outsourcing ad absurdum. Moreover, when services are made available to users on a pay-as-you-go basis, the service providers are not interested in such a brute force data transfer.

Note that certain applications may involve large numbers of query users from scientific institutions, hospitals, or branch offices globally. For example, consider a query load of 10,000 queries/s. If a brute force solution transfers 10 MBytes for a single query, the server needs a network bandwidth of 100,000 MBytes/s, which is far beyond the available bandwidth of Fast Ethernet (100 Mbits/s). Also, from the user point of view, it is better to have most of the processing done in the cloud. For example, a user with a smart phone may not have enough resources (bandwidth, CPU, battery power) to download and query a large dataset locally. Therefore, the design of communication-efficient solutions is of critical importance to the success of cloud computing applications, allowing these to be operated at low cost.

Typically, cloud computing providers (e.g., Amazon, HP, and Microsoft) attempt to solve the problem by offering contractual agreement that promise not to release outsourced data to third parties. Nevertheless, even if the provider respects the contractual agreement, the data are not guaranteed to be safe. Unintended leaks of data are reported regularly, and hackers may still exploit vulnerabilities to gain access to data. Therefore, we believe that data owners will find it attractive to outsource encrypted rather than plain data.

Closest to our work are the recent outsourcing proposals [30, 28] on searching problems in the spatial domain and multidimensional space, respectively. Unfortunately, their techniques rely on specific properties of those spaces and they cannot be extended to solve our problem, which considers arbitrary metric data spaces (e.g., strings, graphs, time-series).

Objective.
The goal of this research is to develop a transformation method $t()$ for converting an original object p in a metric space into another metric space object $p' = t(p)$. First, the data owner specifies a key value $\mathcal{CK}$ in order to define the instance of $t()$ to be used. In a pre-processing phase, the data owner computes p' for each object p and uploads it to the server (i.e., service provider). At query time, the query user specifies his query object q and then submits the transformed query object q' to the server for similarity search. The transformation method must satisfy these requirements:

[4] E.g., BLAST. http://blast.ncbi.nlm.nih.gov/Blast.cgi
[5] E.g., molecule data available at http://dtp.nci.nih.gov

- Even in the worst case that the attacker knows the inverse of $t()$, he can only estimate the original object p from the transformed object $t(p)$ with bounded precision.
- It enables high query accuracy.
- It enables efficient query processing in terms of communication cost.
- It supports insertion and deletion of objects.

Our contributions are as follows. We present three transformation techniques that satisfy the above requirements. They represent various trade-offs among data privacy and query cost and accuracy.

- In our first solution, we propose an encrypted index-based technique with perfect privacy, but multiple communication rounds. This technique flexibly reduces round trip latency at the expense of data transfer.
- For our second solution, our private anchor-based indexing guarantees the correct answer within only 2 rounds of communication. Retrieval is accelerated by bounding the range of potential nearest neighbors in the first phase.
- Our third solution limits communication to a single round, and also returns a constant-sized candidate set by computing a close approximation of the query result.
- We extend our solutions in order to achieve two intuitive privacy guarantees.

The rest of the paper is organized as follows. Section 2 reviews related work. We define the problem and introduce two intuitive privacy guarantees in Section 3, before presenting our solutions in Section 4. Thorough experimental evaluation is discussed in Section 5, before we conclude in Section 6.

2 Related Work

We first introduce existing work on indexing and nearest neighbor search techniques for metric data. Then we cover work on privacy and security of outsourced data.

2.1 Indexing and NN Search in Metric Space

We review metric indexing because our proposed methods provide metric indexing on the server for efficient processing.

The R*-tree [7] and the X-tree [8] are well-known disk-based indexes for multi-dimensional objects, where each object is modeled as a vector of coordinate values. Complex data objects (e.g., DNA sequence, time series) cannot be effectively represented by coordinate values. Instead, we model them in metric space, where a (black-box) distance function $dist(p_i, p_j)$ is used to compute the dissimilarity between objects p_i and p_j. The distance function $dist(\cdot)$ is said to be a *metric* if it satisfies symmetry, non-negativity, and the triangle inequality. Interested readers are referred to two excellent surveys [10, 20] on metric space indexing. In this section, we only describe three representative indexing methods for a set P of metric space objects. They are the vantage-point tree (VP-tree) [29], the multi-vantage-point (MVP-tree) [9], and the M-tree [11].

The VP-tree is a binary tree built on P by utilizing the mutual distances among the objects in P [29]. First, we choose an arbitrary object $a \in P$ as the *root object*, and then we determine the median distance r among the distances $dist(a, p)$ from a to the objects $p \in P$. Each object $p \in P$ satisfying $dist(a, p) \leq r$ is inserted into the left subtree of a, whereas the others are inserted into the right subtree of a. The tree is built in a top-down manner by applying the above construction procedure recursively to the subtrees of a. The MVP-tree [9] is an extension of the VP-tree, so that each index node stores two anchor objects and has m^2 subtrees (m being a parameter). The VP-tree (and MVP-tree) supports insertion/deletion of objects at the risk of an unbalanced tree.

The most popular metric space index is the M-tree [11] (and its variant [22]) due to its efficient support of object insertion/deletion. Each index entry e stores a *minimum bounding sphere* consisting of (i) an anchor object $e.a$ as the sphere center, (ii) a covering radius $e.r = \max\{dist(e.a, p) \mid p \in sub(e)\}$ as the maximum distance from $e.a$ to any object in the subtree of e. In addition, the entry stores (iii) a pointer to its child node, and (iv) a

pre-computed distance from $e.a$ to the parent entry of e. In contrast, a leaf entry only stores the actual object p, and its pre-computed distance to its parent entry.

Given a query object q and a set P of objects, the nearest neighbor (NN) query retrieves the object $p \in P$ such that $dist(q, p)$ is minimized. The *best-first* paradigm [26, 20] is the state-of-the art method for performing NN search on a hierarchical metric space index (e.g., the M-tree). Given a query object q and an index entry e, the function $mindist(q, e)$ is used to compute the (conservative) minimum distance between q and any object indexed by the subtree of e. The best-first search employs a min-heap H for organizing its encountered entries in ascending order of $mindist(q, e)$. Initially, the entries in the root node of the tree are inserted into H. When an index entry e is deheaped from H, we access its child node and insert all entries of the node into H. The first object p that is deheaped from H, is guaranteed to be the NN of q.

Hashing techniques [16, 6] have also been proposed to answer the NN query efficiently. These techniques do not guarantee exact NN retrieval, but they return objects close enough to the NN in practice. The locality-sensitive hashing technique (LSH) [16] is specifically designed for the L_x norm in the multidimensional space $\mathbb{R}^d$; it is inapplicable to arbitrary metric spaces (e.g., edit distance over the domain of strings).

The distance-based hashing technique (DBH) [6] is an extension of LSH for metric spaces. It takes as input two parameters: (i) the number A of bits, and (ii) the number C of hash tables. Let $\mathcal{HT}_j$ be the j-th hash table, for indexing objects $p \in P$ based on their A-length bitmaps $\mathcal{BM}_j(p)$. To compute the i-th bit of the bitmap $\mathcal{BM}_j(p)$, we pick two anchor objects $a_i, b_i \in P$, and define the projection function [14] as:

$$\mathcal{PJF}_{a_i,b_i}(p) = \frac{dist^2(p, a_i) + dist^2(a_i, b_i) - dist^2(p, b_i)}{2 \cdot dist(a_i, b_i)}$$

Then, we determine the value r_i as the median value of $\mathcal{PJF}_{a_i,b_i}(p)$. The i-th bit of $\mathcal{BM}_j(p)$ is set to 0 if $\mathcal{PJF}_{a_i,b_i}(p) \leq r_i$; otherwise, the bit is set to 1.

During the construction phase, we insert each object $p \in P$ into the hash table $\mathcal{HT}_j$ according to the bitmap $\mathcal{BM}_j(p)$. This step is repeated for all C hash tables. At query time, the user requests the hash table $\mathcal{HT}_j$ to return all objects having the same bitmap as the bitmap $\mathcal{BM}_j(q)$ of the query object q. Again, this step is repeated for all hash tables and eventually the closest of them (to q) is reported as the result.

Nevertheless, DBH has two limitations. First, it is possible that no hash table $\mathcal{HT}_j$ contains any object with the same bitmap as the query bitmap $\mathcal{BM}_j(q)$, leading to an empty result. Secondly, once the DBH structure is built (i.e., values of A and C are fixed), its query accuracy cannot be dynamically optimized by the user. We will address the above problems by developing a flexible hashing technique in Section 4.3 that (i) prevents empty results, and (ii) allows the user to boost the query accuracy online by trading off communication cost.

2.2 Privacy and Security

The idea of outsourcing database services to a service provider was introduced by Hacigümüs et al. [18]. Since then, various techniques have been developed to maintain the confidentiality of outsourced data. Given a relational table, Hacigümüs et al. [17] map the tuples of the table into buckets and then store the encrypted tuples of those buckets at the server. At query time, the user compares the query object against the description of those buckets, and then determines the necessary buckets that need to be retrieved from the server. In another proposal [12], the data owner applies the encryption function on each node separately and then stores all encrypted tuples at the server. The method of Agrawal et al. [3] employs an order-preserving function on 1D data values such that the distribution of output values is different from that of input values.

The two works most related to ours are proposed by Yiu et al. [30] and Wong et al. [28]. Yiu et al. [30] present several transformation-based techniques for outsourcing spatial data to the (untrusted) server, such that the server is able to perform spatial range search correctly for trusted users on those transformed points, without knowing their actual coordinates. They propose spatial transformations in 2D space based on scaling, shifting, and noise injection. Also, they develop a solution using an encrypted R-tree. Those solutions operate on explicit 2D coordinates, rendering them inapplicable in our setting, where the distance function is a generic distance metric.

Wong et al. [28] propose to outsource multi-dimensional points to the (untrusted) server, by using a secure scalar product encryption technique. Methods are then provided for kNN search at the server, without the server learning the distances among the points. However, the secure scalar product relies on specific properties of the Euclidean distance in the multi-dimensional space. It is not applicable to other L_p norms, e.g., the L_1 norm (the Manhattan distance). Obviously, it also cannot be applied to our problem setting which considers arbitrary metric

space objects (e.g., strings, graphs, time-series). Another drawback of this proposal is that no indexing scheme can be built on the encrypted tuples, forcing the server to perform a linear scan over the dataset. This affects severely the scalability of the system.

In the field of privacy-preserving data mining, perturbation techniques [4, 23] have been developed for introducing *noise* into the data, before sending them to the service provider. However, such an approach does not guarantee the exact retrieval of results.

The k-anonymity model [27] has been applied extensively for the privacy-preserving publication of datasets. The idea is to generalize the tuples in a table such that each generalized representation is shared by at least k tuples. This way, each object cannot be distinguished from at least $k-1$ other objects. It is often used to generalize the medical records of patients so that the adversary cannot link a specific patient to a medical record. We contend that k-anonymity is of little relevance to data collected from nature (e.g., astronomy data, time series), which is a characteristic of most of the data that motivates this paper's study. However, as some data, e.g., DNA data, may be person related, we do afford k-anonymity some coverage in the paper.

3 Problem Setting

We start by introducing our scenario and our problem setting in Section 3.1. Then, we propose two intuitive privacy guarantees for metric data in Section 3.2. Next, we describe straightforward solutions to our problem and discuss their drawbacks in Section 3.3.

3.1 Problem Definition

We first discuss our scenario and then define our problem.

Scenario.
Figure 2 depicts our scenario for outsourcing data. It consists of three entities: a data owner, a trusted query user, and an untrusted server. On the one hand, the data owner wishes to upload his data to the server so that users are

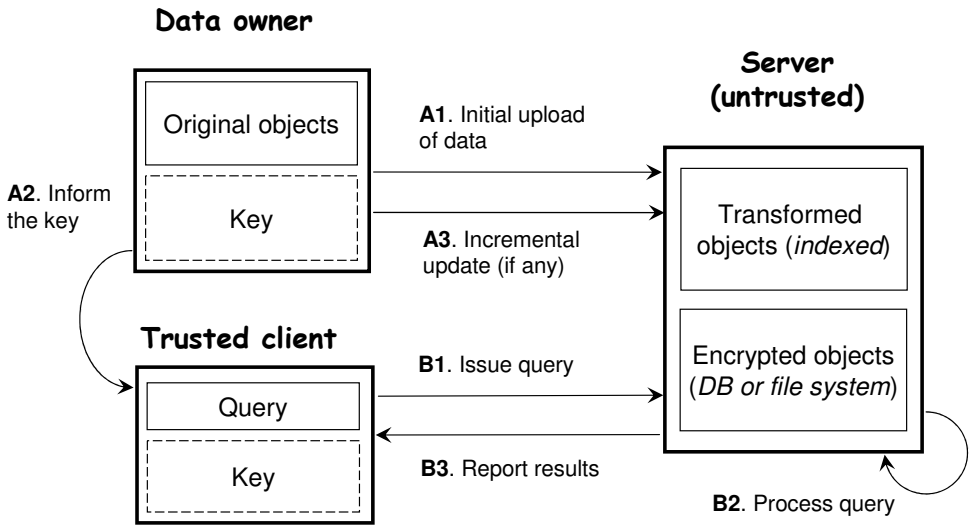


Figure 2: Scenario Overview

able to execute queries on those data. On the other hand, the data owner trusts only the users, and nobody else (including the server).

The data owner has a set P of (original) objects (e.g., actual time series, graphs, strings), and a key to be used for transformation. First, the data owner applies a transformation function (with a key) to convert P into a set P' of transformed objects, and uploads the set P' to the server (see step A1 in the figure). The server builds an index structure on the set P' in order to facilitate efficient search. In addition, the data owner applies a standard

encryption method (e.g., AES) on the set of original objects; the resulting encrypted objects (with their IDs) are uploaded to the server and stored in a relational table (or in the file system).

Next, the data owner informs every user of the transformation key (see step A2). In the future, the data owner is allowed to perform incremental insertion/deletion of objects (see step A3).

At query time, a trusted user applies the transformation function (with a key) to the query and then sends the transformed query to the server (see step B1). Then, the server processes the query (see step B2), and reports the results back to the user (see step B3). Eventually, the user decodes the retrieved results back into the actual results. Observe that these results contain only the IDs of the actual objects. The user may optionally request the server to return the actual objects that correspond to the above result set.

Table 1 summarizes the notation used throughout the paper.

Table 1: List of Notation

Notation	Meaning
P	the set of original objects
P'	the set of transformed objects
$p.id$	the ID of the object p
$dist(p_i, p_j)$	the distance between objects p_i and p_j
$\mathcal{CK}$	encryption key
$\mathcal{ECR}(X, \mathcal{CK})$	encrypt the object X using the key $\mathcal{CK}$
$\mathcal{DCR}(X, \mathcal{CK})$	decrypt the object X using the key $\mathcal{CK}$
$\mathcal{OPE}(v)$	order-preserving encryption on a data value v
$hamming(\mathcal{B}, \mathcal{B}')$	Hamming distance between bitmaps $\mathcal{B}$ and $\mathcal{B}'$
δ	the value used in the δ-gap guarantee
k	the value used in the k-sized guarantee

Problem Definition.
We will use the term *object* for the metric data of interest to the data owner. A transformed object then refers to an object obtained from a transformation.

Let $dist(p_i, p_j)$ denote the distance between two objects p_i and p_j. We focus on nearest neighbor queries, for simplicity. The extension to the case of k nearest neighbors is straightforward. We first give the definition of the nearest neighbor (NN) query as follows.

Definition 1 Nearest Neighbor Query.
Given a query object q and a set P of objects, the nearest neighbor query retrieves the object $p_{nn} \in P$ such that $dist(q, p_{nn}) \leq dist(q, p)$ for all $p \in P$.

Recall from Fig. 2 that both steps A1 and B1 require the data owner and the user to apply a transformation function. Our research objective is to design a transformation method that meets the following requirements:

- Even in the worst case where the attacker knows the inverse of the transformation function, the attacker can only estimate the original object p from the transformed object p' with bounded precision.
- It enables high query accuracy.
- It enables efficient query processing in terms of communication cost.
- It supports the insertion and deletion of objects.

3.2 Privacy Guarantees

In this section, we define two intuitive privacy guarantees that can be adapted for metric data.

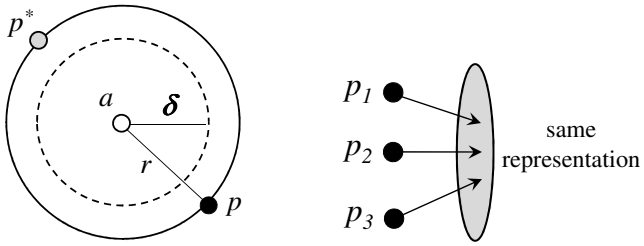
Distance Guarantee.
In two-dimensional space, obfuscation [5] has been used to represent an object's location by a superset region called the *obfuscated region*. An adversary without apriori knowledge is unable to distinguish the object's actual location from other locations in the obfuscated region. The privacy value is typically expressed as the area of the

obfuscated region in the two-dimensional space. However, for generic metric space, there is only the concept of distance but not area. Privacy thus means avoiding small distance between an object and its obfuscated representation. We propose to obfuscate an object p by using a ring $(a, dist(a,p))$ whose center is a reference object a and radius is $dist(a,p)$. This way, the object cannot be distinguished from any other possible object (not necessarily from the dataset) that have the same obfuscated ring representation. We formally define this privacy guarantee as:

Definition 2 **δ-gap guarantee.**
Let p be an object of the dataset P. The ring $(a, dist(a,p))$ satisfies the δ-gap guarantee *if $dist(a,p) \geq \delta$.*

The data owner is able to tune the value of δ such that it describes exactly the required degree of obfuscation. In the example of Fig. 3a, the object p is represented by the ring (a, r) where a is a reference object and $r = dist(a,p)$. Since $r \geq \delta$, the ring satisfies the δ-gap guarantee. Observe that any possible object (e.g., object p^*) having distance r from object a also has the same representation as (a, r).

The reference object a could be either an object picked from the dataset P or a randomly generated object that satisfies $dist(a,p) = \delta$. Regarding the choice of δ, we suggest to set δ to the average c-th nearest neighbor distance, where c is a small constant. This value causes the objects to be displaced in the vicinity of their neighbors, without significantly affecting most of the distances between pairs of objects.



(a) distance guarantee: δ-gap (b) size guarantee: k-sized

Figure 3: Privacy Guarantees

Size Guarantee.
For certain data related to individuals (e.g., DNA data), a more appropriate privacy guarantee may be k-sized guarantee (or k-anonymity) [27]. It is defined formally as:

Definition 3 **k-sized guarantee (or k-anonymity).** *[27]*
A collection $\mathcal{D}$ of objects satisfies the k-sized guarantee*, if for each object $o \in \mathcal{D}$, there exists at least $k-1$ other objects in $\mathcal{D}$ that are mapped to the same representation as o.*

Figure 3b illustrates the abstract concept of the k-sized guarantee, in which three objects p_1, p_2, p_3 are mapped to the same representation (i.e., $k = 3$). There are multiple ways of implementing such representations. For instance, an object could be represented by a hyper-rectangle (as in Section 3.3), or it could be represented by a distance range from a center reference object (as in Section 4.5).

In the context of similarity search, the k-sized guarantee confuses the ranking of transformed objects because $k-1$ of them have the same rank as the transformed object of the actual nearest neighbor.

Summary.
Table 2 summarizes the characteristics of the δ-gap guarantee and the k-sized guarantee. Referring to Section 3.1,

Table 2: Applicability of Privacy Guarantees

Guarantee	δ-gap (distance-based)	k-sized (size-based)
Fixed representation	yes	no
Original space (of P)	yes	yes
Transformed space (of P')	no	yes

suppose that the original space is the domain space of the original dataset P, whereas the transformed space is the domain space of the transformed dataset P'.

The δ-gap guarantee has a fixed representation $(a, dist(a, p))$. Its distance threshold δ is only meaningful in the original space, not in the transformed space. Regarding the k-sized guarantee, its representation is flexible and can be implemented in either the original space or the transformed space.

3.3 Straightforward Solutions

We describe two straightforward solutions to our problem and discuss their shortcomings.

Brute-Force Secure Solution (BRUTE).
The brute-force solution is the one mentioned in the introduction. In the offline construction phase, the data owner applies conventional encryption (e.g., AES) to each object and then uploads the encrypted objects to the server. At query time, the user needs to download all encrypted objects from the server, decrypt them, and then compute the actual result. As mentioned, this solution is perfectly secure, but its query and communication cost are both prohibitively high, and pay-as-you-go is not supported.

Anonymization-Based Solution (ANONY).
The anonymization-based solution achieves data privacy by means of k-anonymity—the objects are generalized in such a way that each generalized object cannot be distinguished from $k-1$ other generalized objects. In the context of similarity search, this solution disturbs the ranking of transformed objects because $k-1$ objects have the same rank as the transformed object of the actual nearest neighbor. While, we consider this solution as a competitor, k-anonymity is not a suitable privacy guarantee for many or most of the applications that motivate the paper's study (see Section 2.2).

In the offline construction phase, the data owner applies a K-D tree partitioning technique [25] on the dataset to obtain disjoint buckets such that each bucket contains at least k objects. For each bucket e, the data owner uploads to the server: (i) $e.MBR$, the *minimum bounding rectangle* (MBR) of all objects inside the bucket and (ii) encrypted strings of the tuples assigned to that bucket.

Let $mindist(q, e.MBR)$ and $maxdist(q, e.MBR)$ represent the minimum and maximum distance from the query object q to $e.MBR$ (the MBR of the bucket e).

At query time, the query user first obtains the MBRs of all buckets from the server, then computes the maximum distance from q to each bucket, and determines the smallest maximum distance (say, $\gamma = \min_e maxdist(q, e)$). Next, the query user requests from the server all encrypted tuples from buckets e that satisfy $mindist(q, e) \leq \gamma$. Eventually, the query user decrypts those tuples in order to obtain the actual result.

Observe that the anonymization technique of LeFevre et al. [25] is applicable only to multi-dimensional data. For arbitrary metric space data, the clustering-based anonymization technique of Aggarwal et al. [1] can be applied. It represents each bucket as a minimum bounding sphere (MBS) consisting of an anchor object and a covering radius, similar to M-tree index entry as described in Section 2.1.

The anonymization-based solution has two shortcomings. First, the MBRs/MBSs of the buckets contain substantial empty space, due to the curse of dimensionality. Thus, the derived upper NN distance bound γ can be loose, triggering the retrieval of a large number of buckets. Second, the solution still allows the server to know the MBRs/MBSs of the buckets, which are located in the same space as the original objects. In contrast, our proposed methods in Sections 4.2 and 4.3 do not permit the server to know any information in the original space; the anchors are converted to IDs, and distance information is transformed to numbers or bitmaps.

4 Proposed Solutions

In this section, we present the transformation methods, and the corresponding query processing techniques. We propose three transformation methods, EHI, MPT, and FDH, in Section 4.1, 4.2, and 4.3, respectively. They capture various trade-offs among data privacy and query cost and accuracy. We present extensions of MPT and FDH in Sections 4.4 and 4.5, in order to comply with the δ-gap guarantee and the k-sized guarantee. We emphasize that all the above methods view the distance function $dist()$ as a black-box; they only require the distance function to be a metric (i.e., satisfying the triangle inequality).

4.1 Encrypted Hierarchical Index Search (EHI)

This section presents a client algorithm, called *encrypted hierarchical index* (EHI), for performing NN search on an encrypted hierarchical index stored at the server. This method offers perfect data privacy for the data owner, but it incurs multiple communication round trips during query processing.

In the literature, algorithms have been developed for processing range queries on encrypted B^+-tree [12] and encrypted R-tree [30]; however, no solutions were proposed for the NN query on those encrypted indexes.

Transformation Key.
The transformation key of EHI is simply an encryption key $\mathcal{CK}$ for standard encryption algorithm (e.g., AES).

Data Transformation.
An index entry e consists of its anchor object $e.a$ and its covering radius $e.r$. Given a query object q and an index entry e, their minimum distance and maximum distance are defined as $mindist(q, e) = \max\{0, dist(q, e.a) - e.r\}$ and $maxdist(q, e) = dist(q, e.a) + e.r$, respectively [11]. In Fig. 4, the anchor object of each entry is shown as a gray dot. For the index entry e_9, we can compute $mindist(q, e_9) = 6 - 1 = 5$ and $maxdist(q, e_9) = 6 + 1 = 7$.

Our EHI method supports any disk-based hierarchical index (e.g., the M-tree [11]), provided that they permit the computation of $mindist(q, e)$ and $maxdist(q, e)$. Please refer to Section 2 for the discussion on M-tree. To construct the EHI structure, we first build a disk-based tree index TR_P on the dataset P (e.g., the example M-tree in Fig. 4). Then, for each tree node, we encrypt its content, and send the encrypted node with its disk block ID to the server. At the end, we send the disk block ID of the root node to the server.

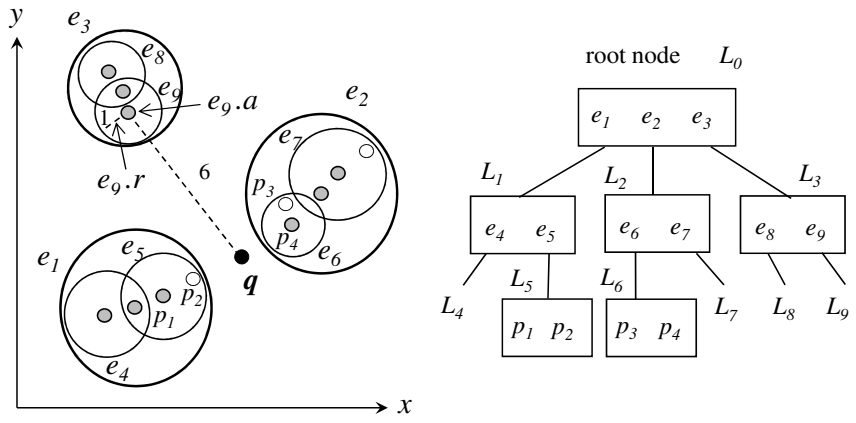


Figure 4: Example of Query Processing in EHI

Query Processing.
Since the tree index stored at the server is encrypted, the server cannot process the NN query by itself. An algorithm for communication between the client and the server needs to be developed in order to answer the NN query correctly.

The total response time of the algorithm consists of the *round trip latency* and the *data transfer time*. These two measures are analogous to the seek time and transfer time in hard disks. Traditional best-first NN search algorithms [26, 20] guarantee that the data transfer time is minimized. However, in the above context, they need to send a message to the server each time a node is requested. This would incur very high round trip latency.

This motivates us to enhance the best-first NN search algorithm in the context of a client-server architecture. Algorithm 1 is the pseudo-code for the user to search the encrypted index stored at the server side. It has two new features: (i) a parameter λ is used to reduce the number of communication messages and round trip latency (see Lines 7–8), and (ii) non-leaf entries are exploited to derive an upper bound of NN distance for pruning unqualified entries (see Lines 3 and 13).

Table 3 illustrates our algorithm on the example in Fig. 4, for different values of λ. Interestingly, our algorithm represents a generalization of different search paradigms on a disk-based tree index. When λ equals 1, our algorithm degenerates to the traditional best-first search. It has the optimal data transfer cost (5 nodes), but incurs a large number of round trips (5 rounds). When λ is set to ∞, our algorithm behaves as breadth-first search. It has

Algorithm 1 EHI Searching Algorithm for Client

Algorithm EHI-Search (Query object q, Encryption Key $\mathcal{CK}$, Integer λ)
request the server for the (encrypted) root node L_{root};
H:=new min-heap; p_{nn}:=NULL;
γ:=$\min_{e \in L_{root}} maxdist(q, e)$; ▷ derive NN distance bound
for each entry $e \in L_{root}$ such that $mindist(q, e) \leq \gamma$ **do**
    insert the entry $\langle e, mindist(q, e) \rangle$ into H;
while H is not empty and its top entry's key $\leq \gamma$ **do**
    pop next λ entries from H and insert them into a set S;
    request the server for each (encrypted) child node of S;
    for each retrieved node L_{cur} **do**
        if L_{cur} is a leaf node **then** ▷ check for closer objects
            update γ and p_{nn} by using objects in L_{cur};
        else ▷ expand the entries of L_{cur}
            γ:=$\min\{\gamma, \min_{e \in L_{cur}} maxdist(q, e)\}$;
            for each $e \in L_{cur}$ such that $mindist(q, e) \leq \gamma$ **do**
                insert the entry $\langle e, mindist(q, e) \rangle$ into H;
return p_{nn} as the result;

the optimal number of round trips (i.e., the tree height, 3), but it may lead to a high data transfer cost (7 nodes). Observe that after the first round, the distance $maxdist(q, e_2)$ is used as the upper bound of the NN distance so the entry e_3 is pruned before the second round.

When λ is set to 2, the algorithm accesses only 5 nodes in a total of 3 rounds. Thus, it has the lowest response time when compared to other values of λ. In general, when the value of λ is large, the number of round trips is small, but the data transfer cost may be high.

Table 3: Requested Nodes in EHI for Communication Rounds

Round	**Requested Nodes**		
	$\lambda = 1$ (best-first)	$\lambda = 2$	$\lambda = \infty$ (breadth-first)
1	root node	root node	root node
2	e_2	e_2, e_1	e_2, e_1
3	e_6	e_6, e_5	e_6, e_5, e_7, e_4
4	e_1	—	—
5	e_5	—	—

As a guideline, we recommend to set the value of $\lambda = \lceil \frac{\tau_{rtl}}{\tau_{node}} \rceil$, where τ_{node} is the transfer time of a node, and τ_{rtl} is the round trip latency. This scheme ensures that the data transfer time of λ nodes equals the round trip latency.

Incremental Data Update.
The data owner is able to insert and delete objects from the encrypted tree, similar to the way of using the underlying index (e.g., M-tree).

4.2 Metric Preserving Transformation (MPT)

In this section, we develop a method, called *metric preserving transformation* (MPT), for evaluating the NN query. Unlike the EHI method, MPT incurs only 2 rounds of communication during the query phase.

The basic idea behind MPT is to pick a small subset of the dataset P as the set of anchor objects [11] and then assign each object of P to its nearest anchor. For each object p, we compute its distance $dist(a_i, p)$ from its anchor a_i and then apply an order-preserving encryption function $\mathcal{OPE}$ [3] on the distance value. These order-preserving encrypted distances will be stored in the server and utilized for processing NN queries.[6]

[6]In fact, the transformation of MPT is different from iDistance [21] because iDistance is specifically designed for indexing points in a multi-dimensional space rather than in a generic metric space. Our MPT solution is applicable to arbitrary metric data (e.g., graphs, strings),

A function $\mathcal{OPE} : \mathbb{R} \rightarrow \mathbb{R}$ is said to be *order preserving* if it guarantees that $\mathcal{OPE}(x) > \mathcal{OPE}(x')$ for any values x, x' that satisfy $x > x'$.

Transformation Key.
The transformation key consists of an encryption Key $\mathcal{CK}$, an integer A, and A pairs of the form (a_i, r_i), where a_i is an (anchor) object and r_i is a distance value. We will soon elaborate on how to generate the transformation key.

Data Transformation.
Algorithm 2 is the pseudo-code for constructing MPT from the input dataset P. Integer A denotes the number of anchor objects. We apply a heuristic from the M-tree [11] to select a set of A anchor objects from P. Such a heuristic aims at optimizing the "indexing quality" of the anchors, but we do not cover the details here. In Line 2, we compute the value B, i.e., the maximum number of objects that can be assigned to the same anchor. Next, we apply a heuristic [11] to assign each object to its anchor object. For example, each object is assigned to its nearest anchor object (having fewer than B assigned objects).

Next, we examine each anchor object. Let a_i denote the i-th anchor object and the set $a_i.S$ represent its assigned set of objects. We compute the anchor's *covering radius* r_i, which denotes the maximum distance from a_i to any object in its set $a_i.S$ [11]. The anchor distance plays an important role in query processing, as we will discuss shortly. For each object p in the set $a_i.S$, we compute its distance $dist(a_i, p)$ from its anchor, and we then apply an order-preserving encryption function $\mathcal{OPE}$ [3] on $dist(a_i, p)$. A tuple consisting of the object ID $p.id$, the order-preserving encrypted distance $\mathcal{OPE}(dist(a_i, p))$, and the encrypted object $\mathcal{ECR}(p, \mathcal{CK})$ will be sent to the server. The benefit of using $\mathcal{OPE}$ is that it hides the original distance values and yet allows comparisons to be correctly evaluated at the server side.

Algorithm 2 MPT Building Algorithm for Data Owner

Algorithm MPT-Build (Dataset P, Encryption Key $\mathcal{CK}$, Integer A)
1: use a heuristic of Ref. [11] to select a set of A anchor objects from P;
2: Integer B:=$\lceil |P|/A \rceil$;
3: use a heuristic of Ref. [11] to assign each data object of P to an anchor object, subject to the capacity constraint B;
4: **for** i:=1 to A **do**
5: let a_i be the i-th anchor object;
6: let $a_i.S$ be the set of objects assigned to the anchor a_i;
7: r_i:=$\max_{p \in a_i.S} dist(a_i, p)$; ▷ compute covering radius
8: **for each** object $p \in a_i.S$ **do**
9: send the tuple $\langle p.id, \mathcal{OPE}(dist(a_i, p)), \mathcal{ECR}(p, \mathcal{CK}) \rangle$ to the server;

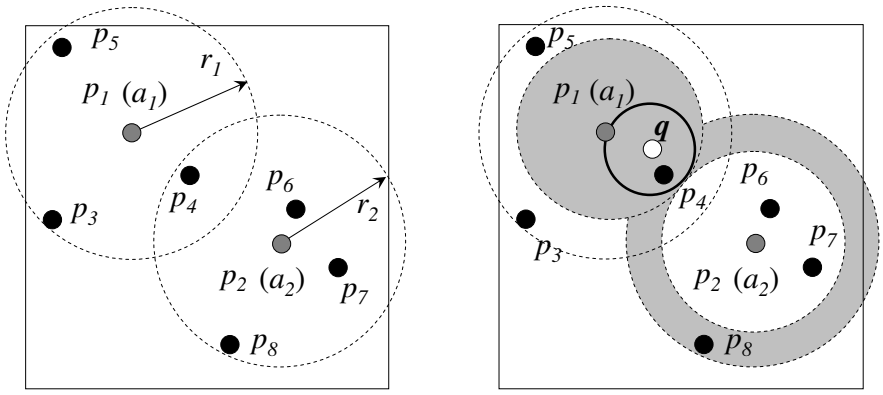
Figure 5a shows an example with eight objects. Suppose that $A = 2$ and that the objects p_1 and p_2 are selected as the anchors a_1 and a_2, respectively. Fig. 5c illustrates the transformed dataset stored in the server. Observe that objects p_1, p_3, p_4, and p_5 are assigned to anchor a_1, whereas the others are assigned to anchor a_2. The transformed dataset also stores the order-preserving encrypted anchor distance and encrypted object for each tuple. For instance, p_5 has distance $dist(a_1, p_5) = 0.32$ from the anchor a_1. To prevent the server from knowing such an actual distance value, the order-preserving function $\mathcal{OPE}$ is applied on the distance and the distance value 0.8 is stored instead.

As a remark, the transformed dataset (at the server side) can be indexed by an R^*-tree using a composite key on the anchor ID and the anchor distance.

Query Processing.
The client has the transformation key, which contains an encryption key $\mathcal{CK}$, the value of A, the set of anchor objects $\{a_i\}$, and the corresponding distance values $\{r_i\}$.

We first establish the foundation of NN search on MPT. Let q be the query object and γ be an upper bound of the NN distance (derived by the client). The client is able to compute the values $dist(q, a_i) - \gamma$ and $dist(q, a_i) + \gamma$ and their order preserving encrypted values using $\mathcal{OPE}()$. With these encrypted distances, the following lemma states when an object p cannot become the NN of q. In order to guarantee exact NN retrieval, the client needs to issue a distance range request to the server so that any object p satisfying $\mathcal{OPE}(dist(a_i, p)) \in [\mathcal{OPE}(dist(q, a_i) - \gamma), \mathcal{OPE}(dist(q, a_i) + \gamma)]$ is retrieved.

but iDistance is not.



(a) construction (b) query processing

ID	Anchor ID	Anchor distance	Encrypted object
1	1	$\mathcal{OPE}(0.00) = 0.00$	$\mathcal{ENC}(p_1) =$ E249C5F3
2	2	$\mathcal{OPE}(0.00) = 0.00$	$\mathcal{ENC}(p_2) =$ 27E90563
3	1	$\mathcal{OPE}(0.35) = 1.00$	$\mathcal{ENC}(p_3) =$ A6231D25
4	1	$\mathcal{OPE}(0.20) = 0.60$	$\mathcal{ENC}(p_4) =$ 90B23F52
5	1	$\mathcal{OPE}(0.32) = 0.80$	$\mathcal{ENC}(p_5) =$ A796437C
6	2	$\mathcal{OPE}(0.10) = 0.20$	$\mathcal{ENC}(p_6) =$ 627D3935
7	2	$\mathcal{OPE}(0.18) = 0.40$	$\mathcal{ENC}(p_7) =$ 35B690D3
8	2	$\mathcal{OPE}(0.34) = 0.90$	$\mathcal{ENC}(p_8) =$ 2F076723

(c) transformed dataset stored at the server

Figure 5: Example of MPT

Lemma 1 *Let γ be an upper bound of the NN distance. Let q be the query object and a_i be an anchor object, where $v_q = dist(q, a_i)$. An object p cannot be the NN of q if $\mathcal{OPE}(dist(a_i, p))$ does not fall in the range $[\mathcal{OPE}(v_q - \gamma), \mathcal{OPE}(v_q + \gamma)]$.*

Proof Since function $\mathcal{OPE}$ is order-preserving, we derive: $dist(a_i, p) \notin [v_q - \gamma, v_q + \gamma]$. This is equivalent to the condition: $dist(a_i, p) < v_q - \gamma$ or $dist(a_i, p) > v_q + \gamma$. By rearranging the terms, we obtain: $\gamma < v_q - dist(a_i, p)$ or $dist(a_i, p) - v_q > \gamma$. Thus, we have $|dist(q, a_i) - dist(a_i, p)| > \gamma$. According to the triangle inequality, we have: $dist(q, p) \geq |dist(q, a_i) - dist(a_i, p)|$. By combining the last two inequalities, we derive $dist(q, p) > \gamma$; thus p cannot be the NN of q. □

Algorithm 3 is the pseudo-code for answering the NN query on MPT. In addition to the parameters used in the transformation key, the client specifies an additional parameter θ used for accelerating the NN retrieval process. The algorithm consists of two phases: the distance bounding phase, and the candidate retrieval phase. Each phase incurs exactly one round trip of communication.

The objective of the distance bounding phase (Lines 1–6) is to derive a tight upper bound γ of the NN distance, in order to reduce the cost of the candidate retrieval phase substantially. In Line 1, the anchor objects are used to derive an upper bound of the NN distance γ. In Lines 2–5, the closest anchor a_{near} to q is selected and the client requests the server to return θ random objects associated with that anchor. These objects are then used to further tighten the NN distance bound γ.

The goal of the candidate retrieval phase (Lines 7–11) is to retrieve the set of potential candidates that can become the NN of q. The minimum distance between q and any object indexed by anchor a_i (with covering radius r_i) is given by [11]:

$$mindist(q, (a_i, r_i)) = \max\{dist(q, a_i) - r_i, 0\}$$

Obviously, we only need to consider the anchors satisfying $mindist(q, (a_i, r_i)) \leq \gamma$. In Line 9, we apply Lemma 1 and issue the corresponding distance range query to retrieve the potential candidates indexed by anchor a_i.

Eventually, we obtain the candidate set S_{cand} and return the object $p \in S_{cand}$ with the minimum $dist(q, p)$ value as the result.

Algorithm 3 MPT Searching Algorithm for Client

Algorithm MPT-Search (Query object q, Encryption Key $\mathcal{CK}$, Integer θ, Integer A, Pairs $\{(a_i, r_i)\}$)
/ Distance bounding phase */*
1: γ:=$\min_{i \in [1,A]} dist(q, a_i)$; ▷ initial bound of NN distance
2: let a_{near} be the anchor leading to the γ value;
3: request the server for θ random tuples whose anchor ID equals to that of a_{near};
4: let S_{samp} be the set of decrypted objects from the received tuples;
5: **for each** $p \in S_{samp}$ **do**
6: γ:=$\min\{\gamma, dist(q, p)\}$; ▷ refined bound of NN distance
/ Candidate retrieval phase */*
7: **for** i:=1 to A **do**
8: **if** $mindist(q, (a_i, r_i)) \leq \gamma$ **then**
9: request the server for all tuples (with anchor ID as a_i) whose $\mathcal{OPE}(dist(a_i, p))$ falls into the range $[\mathcal{OPE}(dist(q, a_i) - \gamma), \mathcal{OPE}(dist(q, a_i) + \gamma)]$;
10: let S_{cand} be the set of decrypted objects from the received tuples (of the above request);
11: **return** the object $p \in S_{cand}$ with the minimum $dist(q, p)$ value as the final result;

Figure 5b illustrates NN query processing on MPT. The query object q is shown as a white dot. Suppose that $dist(q, a_1) = 0.13$ and $dist(q, a_2) = 0.34$. First, the client uses the anchor objects to derive the upper distance bound $\gamma = dist(q, a_1) = 0.13$. For simplicity, we skip the steps of using θ additional objects to tighten γ (see Lines 2–6 of the algorithm). We then proceed to the candidate retrieval phase. We derive the distance range query $[0.13 - 0.13, 0.13 + 0.13] = [0, 0.26]$ for anchor a_1, and the distance range query $[0.34 - 0.13, 0.34 + 0.13] = [0.21, 0.47]$ for anchor a_2. We then apply the $\mathcal{OPE}$ function on those distance ranges and send the transformed queries to the server. The search space is depicted by the gray region in Fig. 5b. Thus, the server searches the transformed dataset of Fig. 5c and returns the encrypted tuples corresponding to objects p_1, p_4, p_8. Eventually, the client decrypts those objects and obtains object p_4 as its NN.

Incremental Data Update.
The data owner inserts or deletes an object p, for the corresponding bucket having the anchor object of p.

If the data distribution in the dataset changes gradually over time so that most of the objects in a partition become far away from their anchor. To recover the indexing quality of MPT, the data owner can perform the following periodic (say, weekly) maintenance procedure: (i) split the largest partition into two equal-sized partitions (with new anchors), and (ii) merge the smallest partition with its nearest partition. The maintenance overhead is small because it involves only three partitions.

4.3 Flexible Distance-Based Hashing (FDH)

In this section, we propose a hashing-based technique, called *flexible distance-based hashing* (FDH), for processing the NN query. The main advantage of this technique is that the server always returns a constant-sized candidate set (in one communication round). The client then refines the candidate set to obtain the final result. Even though FDH is not guaranteed to return the exact result, the final result is very close to the actual NN in practice.

During query processing, FDH allows the client to specify an integer parameter θ for increasing the accuracy of a query result, without rebuilding the transformed data stored at the server. This is a significant enhancement over earlier work [6], in which query accuracy cannot be adjusted once the index structure is built.

In addition, our FDH method employs a novel technique for conceptually linking similar hash buckets, in order to maximize the utility of the transformed data for answering queries.

In the literature, the Hilbert curve transformation has been employed for approximate NN search in 2D space [24]. However, its performance degrades rapidly for high dimensional space and it is inapplicable to the metric space. In contrast, our FDH method can be applied for any black-box distance function $dist()$.

Transformation Key.
The transformation key consists of an encryption Key $\mathcal{CK}$, an integer A, and A pairs of the form (a_i, r_i), where a_i

is an object and r_i is a distance value. The generation of the transformation key will be discussed shortly.

Data Transformation.
The data owner specifies a parameter A for the data transformation step. A typical value of A is in the range of tens to hundreds. This parameter provides a trade-off between the cost of transformation and the query accuracy, e.g., a higher value of A leads to better query accuracy, but also high transformation cost.

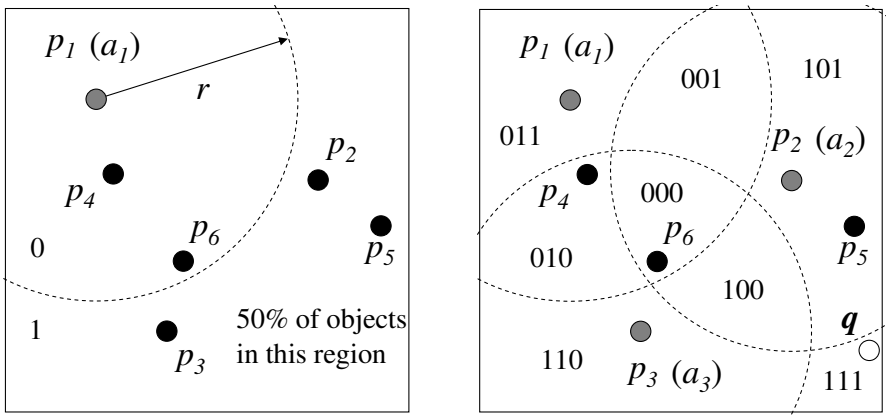
Let P be the original dataset of objects. In the data transformation phase, we choose A random objects from the set P as *anchor* objects. Let these anchor objects be $a_1, a_2, \cdots, a_i, \cdots, a_A$. For each anchor object a_i, we need to pick a distance value r_i (which will be determined shortly).

Given an object $p \in P$, we convert it into an A-length *bitmap* where the i-th bit of the bitmap is defined as:

$$\mathcal{BM}(p)[i] \quad = \quad \begin{cases} 0 & \text{if } dist(a_i, p) \leq r_i \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

Our transformation is cheap to compute because the $dist(\cdot)$ function is called exactly once for computing a bit value. In contrast, the transformation function of Athitsos et al. [6] is more expensive to evaluate as it calls the $dist()$ function three times (for computing a bit value).

Figure 6a shows a set of objects $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. Suppose that p_1 is chosen as the anchor object a_1. We illustrate how to derive the distance value r_1 for the anchor a_1. Observe that each object within the circle (i.e., p_1, p_4, p_6) has a bit value 0, whereas any other object (i.e., p_2, p_3, p_5) has a bit value 1. If the distance value r_i is too large, then most of the objects will be assigned the bit value 0, implying that we cannot distinguish pairs of objects (e.g., the pair (p_4, p_5)) that are far apart. A similar problem arises when r_i is set to a small value. To balance the number of objects on both sides, we choose the distance value r_i as the median value among the distances $dist(a_i, p)$ (of objects from the anchor a_i).



(a) finding the value of r (b) data transformation

ID	Bitmap	Encrypted object
1	011	$\mathcal{ENC}(p_1)$ = A23692F0
2	101	$\mathcal{ENC}(p_2)$ = 967990B2
3	110	$\mathcal{ENC}(p_3)$ = 6723A623
4	010	$\mathcal{ENC}(p_4)$ = 835223B6
5	101	$\mathcal{ENC}(p_5)$ = D257623E
6	000	$\mathcal{ENC}(p_6)$ = 3023C790

(c) transformed dataset stored at the server

Figure 6: Example of FDH

Figure 6b shows an example with $A = 3$ anchors a_1, a_2, a_3 (which are p_1, p_2, p_3, respectively). The domain space is conceptually partitioned into 8 regions, each of which can be represented by a bitmap as defined in Equation 1. For example, the region '010' contains objects that are close to a_1 and a_3, but far from a_2.

It is worth noticing that the above regions may not be continuous for arbitrary anchor objects in arbitrary space. This subtle issue does not lead to any problem since we do not need to describe the extents of the regions explicitly. We only define those regions implicitly by using Equation 1.

Algorithm 4 is the pseudo-code of the construction algorithm for FDH. We first choose the anchors a_i and

Algorithm 4 FDH Building Algorithm for Data Owner

Algorithm FDH-Build (Dataset P, Encryption Key $\mathcal{CK}$, Integer A)
1: **for** i:=1 to A **do** ▷ key generation
2: choose an object randomly from P as an anchor object a_i;
3: find the distance value r_i such that half of objects $p \in P$ satisfy $dist(a_i, p) \leq r_i$;
4: **for each** object $p \in P$ **do**
5: compute the encryption $\mathcal{ECR}(p, \mathcal{CK})$;
6: compute $\mathcal{BM}(p)$;
7: send the tuple $\langle p.id, \mathcal{BM}(p), \mathcal{ECR}(p, \mathcal{CK})\rangle$ to the server;

distance values r_i as described before. Then, for each object $p \in P$, we apply Equation 1 to obtain an A-length bitmap $\mathcal{BM}(p)$, and we apply a standard encryption (e.g., AES) on p to obtain an encrypted object $\mathcal{ECR}(p, \mathcal{CK})$. A tuple consisting of the object ID, transformed bitmap $\mathcal{BM}(p)$, and the encrypted object $\mathcal{ECR}(p, \mathcal{CK})$ is uploaded to the server.

Figure 6c shows the content of the transformed dataset (stored at the server), obtained from the configuration of Fig. 6b. We introduce an appropriate indexing method for the above bitmaps shortly.

Query Processing.
Note that the client has the transformation key, which contains the value of A, the set of anchor objects $\{a_i\}$, and the corresponding distance values $\{r_i\}$. At query time, the user specifies a query object q and then applies Equation 1 to obtain the bitmap $\mathcal{BM}(q)$. The user then requests the server to return an encrypted object $\mathcal{ENC}(p, \mathcal{CK})$ such that its bitmap $\mathcal{BM}(p)$ is identical to $\mathcal{BM}(q)$.

A closer look reveals that the above query method has two potential problems. In Fig. 6b, different regions contain different numbers of objects. In case the query object q falls into the region '111', none of the data objects can be retrieved. On the other hand, if the query bitmap is '101', the corresponding region contains multiple objects, and the server cannot decide which of them is closer to q solely based on the bitmap.

These two important problems have not been considered in earlier work [6]. Intuitively, if the query bitmap falls in a region (say, '111') without objects, we need to expand the search space to neighborhood regions (e.g., regions '110', '100', '101'). In case the query object q falls into a region (say, '101') with multiple objects, we need to gather extra information to decide which of them is closer to q.

Is there any elegant solution for realizing the above ideas? It turns out that the answer is positive. First, we define the *Hamming distance* between two bitmaps $\mathcal{B}$ and $\mathcal{B}'$ as follows:

$$hamming(\mathcal{B}, \mathcal{B}') = \mid \{\mathcal{B}[i] \neq \mathcal{B}'[i] \mid i \in [1, A]\} \mid \quad (2)$$

It is well known that the Hamming distance is an intuitive function for capturing the distances among bitmaps. In our context, two regions close together are expected to have a low Hamming distance between their corresponding bitmaps.

At the server side, we employ a metric space index (e.g., the M-tree [11]) for indexing the bitmaps based on the Hamming distance function $hamming()$. In other words, hash buckets (for the regions) are no longer explicitly maintained in the system. This eliminates the problems raised by empty hash buckets or buckets with many objects. As we will see later, the query processing strategy is to apply a similarity search on the above metric space index.

Algorithm 5 is the pseudo-code of the searching algorithm for FDH. The client specifies an additional integer parameter θ, and requests the server to retrieve the θ tuples whose bitmaps are the closest to the query bitmap $\mathcal{BM}(q)$. After receiving the result tuples from the server, the client decodes them into original objects and computes their distances from q. Among those objects, the one closest to q is reported as the result.

The advantage of the above algorithm is that it provides the clients the flexibility of choosing different values of θ at query time. In our experimental study, we will show that a small value of θ is sufficient to achieve a reasonable accuracy for the reported result.

Algorithm 5 FDH Searching Algorithm for Client

Algorithm FDH-Search (Query object q, Encryption Key $\mathcal{CK}$, Integer θ, Integer A, Pairs $\{(a_i, r_i)\}$)

1: compute the query bitmap $\mathcal{BM}(q)$;
2: request the θ tuples $\langle p.id, \mathcal{BM}(p), \mathcal{ECR}(p, \mathcal{CK})\rangle$ with the lowest Hamming distance from q from the server (ties are resolved arbitrarily);
3: let S be the set of decrypted objects from the received tuples;
4: **return** the object $p \in S$ with the minimum $dist(q, p)$ value as the final result;

Incremental Data Update.
The data owner efficiently computes the bitmap of an object and then requests the server to insert or delete that object (with a particular object ID).

We need a mechanism to cope with the scenario that the data distribution in the dataset changes gradually over time. We suggest that the data owner pre-computes a bitmap (for each object) such that its length equals, e.g., $10|A|$. The data owner then performs a periodic (say, weekly) procedure as follows: (i) asks the server to find $|A|$ bit positions (out of all $10|A|$ positions) such that the ratio of 0/1 bits is the most balanced, and (ii) publishes to the users a new key with those A bit positions only. This maintenance task is very efficient as the server needs not ship any bitmaps or encrypted data objects to the data owner.

4.4 Enforcing δ-Gap Guarantee in Original Space

We now discuss how the δ-gap guarantee (in Definition 2) can be achieved by adapting our transformation methods MPT and FDH. Recall that this guarantee is used to define privacy in the original space, and it requires that: each original object $p \in P$ must be represented by a tuple $(p, dist(a, p))$ such that a is a reference object and $dist(a, p) \geq \delta$. This requirement has to be fulfilled in the original space before applying our transformation method. We call these extended solutions δ-MPT and δ-FDH, respectively. Only their construction algorithms are modified; the query algorithms of MPT and FDH can be directly reused for processing queries as the schema used for the transformed data remains unchanged.

δ-Gap Variant of MPT.
In order to provide the δ-gap guarantee in MPT, we modify Line 3 of Algorithm 2 as follows, by restricting how a data object p can be assigned to an anchor object of the set A. Specifically, we define the set $A_\delta(p) = \{a_i \in A \mid dist(a_i, p) \geq \delta\}$ for object p. Every anchor in $A_\delta(p)$ always satisfies the δ-gap guarantee with p. Then, we assign p to an anchor from $A_\delta(p)$, by using a heuristic [11] (e.g., finding nearest anchor).

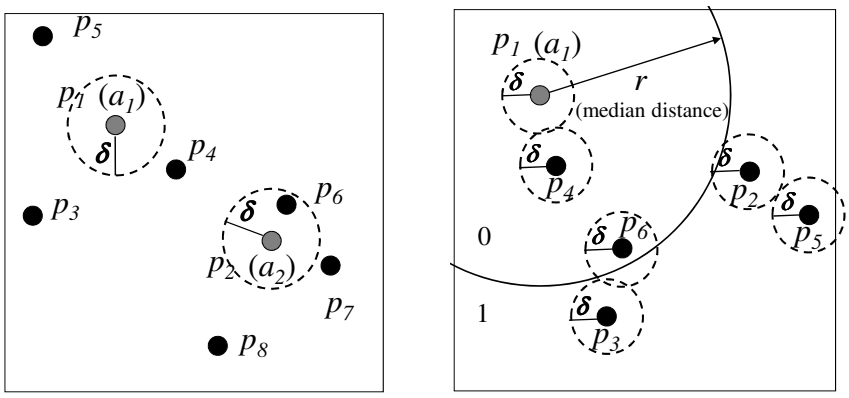
Suppose we now use the above variant to assign objects to anchors in the example of Fig. 7a, where objects p_1 and p_2 are anchors a_1 and a_2 respectively. Each dotted circle has its center at anchor a_i and its radius as δ. To meet the δ-gap requirement, an object located within a dotted circle cannot be assigned to the anchor of that circle. For instance, object p_1 can only be assigned to anchor a_2. Similarly, objects p_2 and p_6 are assigned to anchor a_1. Each remaining object can be simply assigned to its nearest anchor.

Observe that a small δ value only forces a small number of objects to be assigned to farther anchors.

δ-Gap Variant of FDH.
To offer the δ-gap guarantee in FDH, we displace each original object $p \in P$ by the distance δ. This is performed before Equation 1 is used to convert p into a bitmap in the transformed space.

Figure 7b shows how this variant works. In this example, locations within the solid circle are mapped to the bit value 0, whereas other locations are mapped to the bit value 1. Each dotted circle represents the possible location of an object p_i after the displacement. Observe that objects p_1 and p_4 are mapped to the bit value 0, regardless of how they are displaced. Similarly, p_3 and p_5 are always mapped to the bit value 1. Each remaining object (e.g., p_2, p_6) can be mapped to the bit value 0 or 1, depending on its displaced location.

Observe that a small δ value only causes a small number of bit values to be flipped.



(a) δ-MPT (b) δ-FDH

Figure 7: δ-Gap Variants of Transformation Methods

4.5 Enforcing k-Sized Guarantee in Transformed Space

We proceed to discuss how the k-sized guarantee (in Definition 3) can be achieved by adapting our transformation methods MPT and FDH.

The method ANONY (see Section 3.3) achieves the k-sized guarantee by applying k-anonymization techniques to the dataset in the original space. However, this has high query costs due to large empty space in generalized rectangles. Also, the server may still be able to learn approximate information on the original object.

We thus extend the construction phase of our transformation methods (MPT and FDH) such that (i) they satisfy the requirements of the k-sized model and (ii) at the same time achieve much lower query cost than ANONY. The idea is to perform generalization in the transformed space rather than in the original space. We call these extended counterparts k-MPT and k-FDH.

k-Sized Variant of MPT.
The goal of k-MPT is to generalize the tuples of the transformed table (e.g., Fig. 5c) such that each tuple cannot be distinguished from (at least) $k-1$ other tuples.

In order to reuse the existing query algorithm, we decide to not generalize the anchor ID. As a consequence, the only choice is to generalize the (order-preserving encrypted) anchor distances into intervals. Specifically, we sort tuples with the same anchor ID in ascending order of their anchor distances. Then we partition the distances into disjoint distance intervals such that each interval covers at least the anchor distances of k objects. To limit the number of objects sharing the same distance interval, we impose an additional capacity constraint that each distance interval cannot cover the anchor distances of more than $2k$ objects.

As an example, assume that $k = 2$ and that we wish to generalize the anchor distances stored in Fig. 5c. Observe that the tuples p_1, p_3, p_4, p_5 all have anchor ID 1. After sorting them by their anchor distances (0, 0.6, 0.8, 1), we generalize the anchor distances of p_1 and p_4 into the interval $[0, 0.6]$, and we generalize those of p_5 and p_3 into the interval $[0.8, 1]$.

At query time, an (encrypted) object is retrieved if its anchor distance interval overlaps the query interval distance range (with respect to the same anchor ID). This procedure guarantees that the candidate set returned by k-MPT always contains the actual NN query.

The introduction of the parameter k in k-MPT results in a higher communication cost in the query phase when comparing to the original MPT. Fortunately, the following lemma shows that the extra communication cost is bounded by $4kA$, where A is the number of anchor objects.

Lemma 2 *When compared to MPT, the query method of k-MPT retrieves at most $4kA$ additional objects from the server.*

Proof We first focus on the distance range query (say, $[v_l, v_h]$) for a single anchor. The transformed objects having distance intervals covered by the range $[v_l, v_h]$ are guaranteed to be retrieved by MPT, so they incur no communication overhead in k-MPT.

Since distinct distance intervals in the transformed dataset are disjoint, the lower bound value v_l intersects at most one (distinct) distance interval. Such an interval covers at most $2k$ objects, due to the capacity constraint. Similarly, the upper bound value v_h intersects at most one distance interval, corresponding to at most $2k$ objects.

After considering the distance range queries for all anchors, we conclude that k-MPT retrieves at most $2(2k)(A) = 4kA$ additional objects from the server, when compared to the original MPT. □

k-Sized Variant of FDH.
The objective of k-FDH is to generalize the tuples of the transformed table (e.g., Fig. 6c) such that each tuple cannot be distinguished from $k-1$ other tuples.

An intuitive solution works as follows: An object p is mapped to a bitmap $\mathcal{BM}(p)$ that is shared by fewer than $k-1$ other objects. We then compute the nearest bitmap of $\mathcal{BM}(p)$ (say, $\mathcal{BM}(p')$) and replace all occurrences of $\mathcal{BM}(p)$ in the transformed table by the bitmap $\mathcal{BM}(p')$. By applying this above procedure iteratively, we eventually obtain a transformed table such that each distinct bitmap is shared by at least k different objects. This nearest bitmap replacement policy aims at reducing the amount of information loss.

For instance, we want to modify the bitmaps of the transformed table of Fig. 6c such that each bitmap is shared by at least $k = 2$ objects. The bitmap of p_1 is "011" and occurs only once in the table. Thus, we find its nearest bitmap (i.e., "010") and replace the bitmap of p_1 by "010." The bitmap of p_2 appears twice in the table, so its bitmap needs not be changed.

As a remark, the k-FDH method has the same communication cost as the original FDH in the query phase. The only added disadvantage of k-FDH is that the introduction of k leads to less accurate query results (the larger k is, the less accurate), due to the relocation of transformed objects' bitmaps.

5 Experimental Evaluation

We first describe our experimental setting, including real datasets, methods, parameters, and measurements. Then we study the effect of various factors on the performance and quality of different methods.

5.1 Experimental Setting

In this section we evaluate our proposed techniques on four real-world datasets, whose properties are summarized in Table 4. For each tuple in a dataset, its ID takes 4 bytes, and each attribute takes 4 bytes to store. YEAST[7] is

Table 4: Summary of Real Datasets

Name	Attributes	Number of tuples	Distance
YEAST	17, numeric	2,882	L_1 norm
MUSH	22, categorical	8,124	Jaccard dist.
SHUTL	9, numeric	43,500	L_1 norm
GFC	16, numeric	86,984	L_1 norm

a gene expression data matrix obtained from a Microarray experiment on yeast. Each entry indicates the expression level of a specific gene (row/tuple) at a specific condition (column/attribute). Both MUSH and SHUTL are obtained from the UCI Machine Learning Repository[8]. MUSH contains a wide variety of categorical attributes for describing the characteristics of North American mushrooms. SHUTL contains the log records describing the states of shuttles from NASA. The dataset GFC, obtained from the Georgia Forestry Commission Fire Weather System[9], consists of temperature time series measured by weather stations gathering agrarian meteorological data. It is worth noticing that, all these real datasets have been valuable and/or protected data once in the past. For instance, gene expression matrices (YEAST) and data from space programs (SHUTL) are both expensive and time-consuming to collect, as elaborated upon in the introduction.

[7] http://arep.med.harvard.edu/biclustering/
[8] http://archive.ics.uci.edu/ml/
[9] http://weather.gfc.state.ga.us

It is known that the L_1 norm (i.e., the Manhattan distance) is better than other L_p norms at retrieving *meaningful* nearest neighbors in high dimensional spaces [19]. Thus, we use the L_1 norm as the distance metric for numeric datasets. The Jaccard distance [13] is used as the distance metric for the categorical dataset (MUSH).

Table 5 provides an overview over the compared algorithms. BRUTE and ANONY are the straightforward solutions described in Section 3.3. EHI, MPT, and FDH are our solutions as presented in Section 4. Finally, δ-MPT and δ-FDH denote the δ-gap variants from Section 4.4. The distance-based hashing method (DBH) [6] was reviewed in Section 2.

Table 5: Summary of Algorithms

Algorithm	Name	Presented in
BRUTE	Brute-force Secure Solution	Sec. 3.3
ANONY	Anonymization-based Solution	Sec. 3.3
EHI	Encrypted Hierarchical Index Search	Sec. 4.1
MPT	Metric Preserving Transformation	Sec. 4.2
FDH	Flexible Distance-based Hashing	Sec. 4.3
DBH	Distance-based Hashing	Ref. [6]
δ-MPT / δ-FDH	δ-gap variants of MPT and FDH	Sec. 4.4
k-MPT / k-FDH	k-sized variants of MPT and FDH	Sec. 4.5

For the construction phase and the query phase, we use the following default parameter values: $A = 100$, $\theta = 500$, $\lambda = 10$, and $k = 32$. Experiments on queries are run with a query workload of 100 query objects. We measure the communication cost (in KBytes), server CPU cost, the distance ratio (of the result NN distance to the actual NN distance), and the rank (of the result NN on the dataset); and we report their average values for the query workload. We also measure the time taken in the construction phase.

5.2 Experimental Results

Experiment on Default Parameter Values.
Before evaluating the effects of the parameters, we give an overview of the general performance of the studied algorithms.

Table 6 shows the query communication cost of the solutions on all the datasets. The general tendency is the

Table 6: Communication Cost on Real Datasets, at Default Setting

	Communication cost (KBytes)				
Dataset	BRUTE	ANONY	EHI	MPT	FDH
YEAST	202.64	144.60	148.95	66.15	35.15
MUSH	729.89	648.38	655.90	109.34	44.92
SHUTL	1699.21	1235.30	815.40	170.11	19.53
GFC	5776.28	1309.52	533.96	315.94	33.20

same for all datasets. The straightforward approaches BRUTE and ANONY have very high communication cost when compared to the others. EHI has a moderate communication cost because the client performs the actual search procedure. Observe that MPT and FDH greatly reduce the communication cost by outsourcing search functionality to the server. MPT obtains its reduction in the communication cost by providing the server with information on the relative distances of objects (i.e., order preserving encryption). The best performance in terms of communication cost is achieved by the FDH method. Compact bitmap representatives allow efficient search with very low communication cost. As shown in Table 7, both the construction time and server CPU time (per query) of all our proposed solutions are reasonable.

Due to the high cost of ANONY, it is omitted from the subsequent experiments.

Comparison Between DBH and FDH.
To facilitate a fair comparison between DBH and FDH, we allocate the same amount of disk storage space to them.

Table 7: Construction Time and Server Time on Real Datasets, at Default Setting

Dataset	Construction time (s)			Server CPU time (s)		
	EHI	MPT	FDH	EHI	MPT	FDH
YEAST	0.016	0.094	0.313	0.001	0.001	0.049
MUSH	0.234	0.531	1.344	0.006	0.002	0.083
SHUTL	2.438	1.187	4.672	0.010	0.006	0.097
GFC	12.141	3.063	10.078	0.007	0.005	0.141

Both DBH and FDH use the same parameter value of A. However, for DBH, its number C of hash tables is fixed to 1, so that its storage space is comparable to that of FDH.

Table 8 shows the percentage of empty results, average NN distance, and average communication cost on the GFC dataset when varying the parameter A. It is worth noticing that DBH may often return empty results when A is above 100 (see the highlighted cells). This is because each hash bucket contains very few objects in those cases. On the other hand, FDH does not return empty result in any case, thanks to its search strategy based on Hamming distances among the bitmaps. The NN distances returned by DBH and FDH are similar, but the communication cost of FDH is much lower than that of DBH.

Table 8: Quality of DBH and FDH, on GFC, as a Function of A

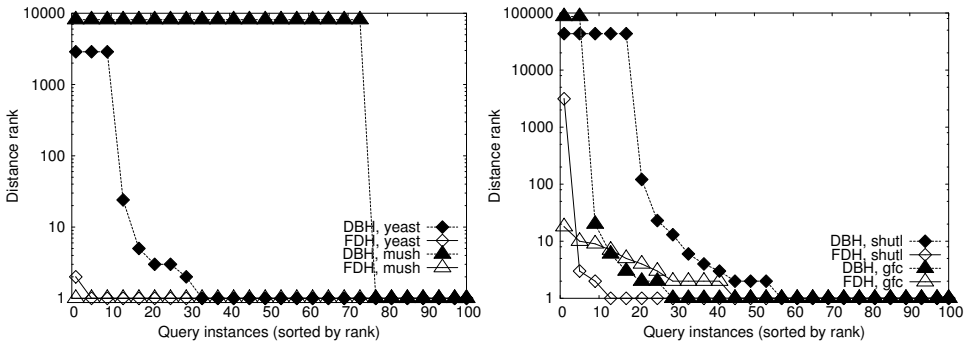
A	Empty result		Avg. NN distance		Comm. cost (KBytes)	
	DBH	FDH	DBH	FDH	DBH	FDH
20	0%	0%	0.0855	0.1059	1051.27	33.20
50	3%	0%	0.0864	0.0935	736.94	33.20
100	7%	0%	0.0879	0.0906	382.68	33.20
200	12%	0%	0.0950	0.0870	198.25	33.20
500	21%	0%	0.0874	0.0854	122.06	33.20

Next, we study the rank of result NN for individual queries (from default query workload) on all real datastes, whereas other parameters are set to their default values. For an empty result (e.g., obtained in DBH), its rank is set to the cardinality of the dataset. Figure 8a shows the rank of result NN on the datasets YEAST and MUSH, and Fig. 8b shows the rank of the result NN on the datasets SHUTL and GFC. For ease of visualization, the ranks are plotted in descending order. Observe that the result rank of FDH outperforms that of DBH on the datasets YEAST, MUSH, and SHUTL. For the dataset GFC, the 10% of the query instances with that largest ranks favor FDH much more than DBH, but the next 30% of the query instances favor DBH slightly more than FDH. Interestingly, a certain query object has a large rank (3135) on the dataset SHUTL. According to our log for FDH, the result NN distance of that query object is 0.014379, which is much higher than the average NN distance of the workload (0.000498). This query object is an outlier whose NN resides in a dense "cluster" located far away. Even with a small error in the result NN distance, the rank of the result NN can become large. A similar behavior is observed in DBH, and it performs worse than FDH.

In the remaining experiments, we report the average distance ratio (of the result NN distance to the actual NN distance) and the average rank (of the result NN) for the query workload, rather than result measures of individual queries. Also, we omit DBH from further studies due to its inferior performance and result quality.

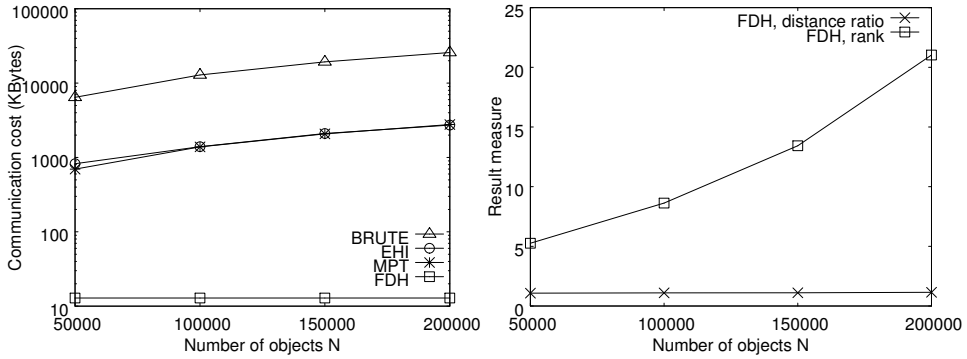
Scalability Experiment.

To analyze the scalability, we generate large synthetic datasets of cardinality N between 50,000 and 200,000. SYN consists of 32 dimensional feature vectors, distributed into 10 different clusters. As we can see in Fig. 9, the relative performance of all methods is stable across different data sizes. The communication cost of the BRUTE method corresponds to the size of the dataset. EHI and MPT achieve the same reduction in communication cost. Unlike MPT, however, EHI cannot be further improved by tuning parameter values (A, θ). The best method in terms of communication cost is once again the FDH approach which scales extremely well. The distance ratio of FDH is insensitive to N. Due to the fixed number of clusters, the number of objects within a cluster increases as N proportionally, and thus the result rank of FDH increases.



(a) YEAST and MUSH data (b) SHUTL and GFC data

Figure 8: Rank of Result NN for Individual Queries on Real Datasets



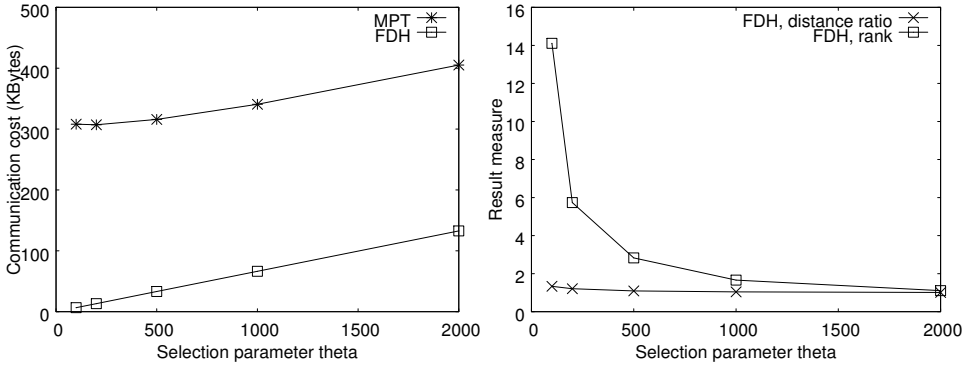
(a) communication cost (b) result measure

Figure 9: Effect of Data Size N, on Synthetic Data SYN

Effect of the Selection Parameter θ.
We now turn to the parameter θ in the MPT and FDH techniques to determine a good value that optimizes communication cost. Figure 10 depicts the results: increasing θ means that the client requests more tuples/bitmaps from the server. In MPT, θ is used in the initial communication round. Hence, it influences only the quality of the initial approximation. The better this initial approximation, the lower the communication cost is in subsequent rounds. This is reflected in the experimental results where increasing θ lowers the communication cost of MPT. Consequently, in FDH with only one round of communication, we observe the opposite effect. In this single round, the communication cost increases proportionally to the number of objects requested, and the approximation quality improves.

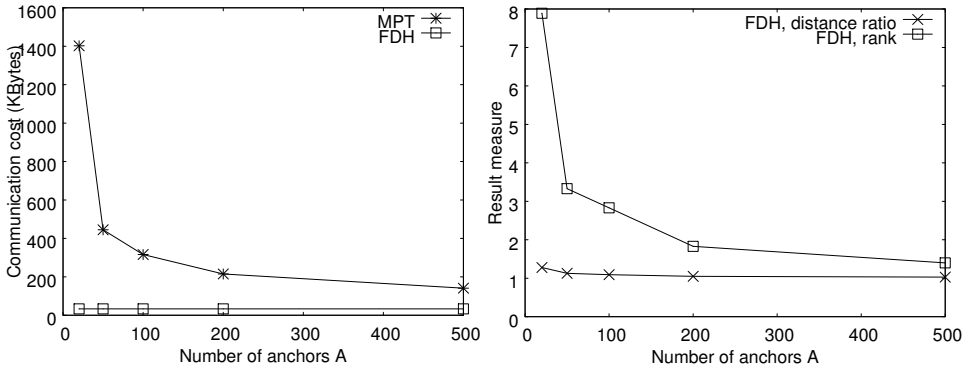
Effect of the Number of Anchors A.
We then study the effect of varying the number of anchors A that are used in the MPT and FDH techniques. Figure 11 exhibits the effects on both the communication cost and the nearest neighbor result. We observe a gradual improvement in communication cost with increasing numbers of anchors. This is due to the fact that more anchors provide more detailed information on the locations of objects. This is also reflected in the approximation quality of FDH, which improves with increasing number of anchors. With only 100 anchors, we achieve a low communication cost and a good approximation of the nearest neighbor.

Effect of the Order c on the δ-Gap Solutions.
We now examine the performance and accuracy of the extended solutions that satisfy the δ-gap guarantee: δ-MPT and δ-FDH. Recall from Section 3.2 that we recommend setting δ to the average c-th nearest neighbor distance in the dataset. Figure 12 shows the communication cost and result measure with respect to c. The cost of EHI is independent of c and it is shown as a reference for comparison. The communication cost of δ-MPT rises with c,



(a) communication cost (b) result measure

Figure 10: Effect of θ, on GFC Data



(a) communication cost (b) result measure

Figure 11: Effect of Anchors A, on GFC Data

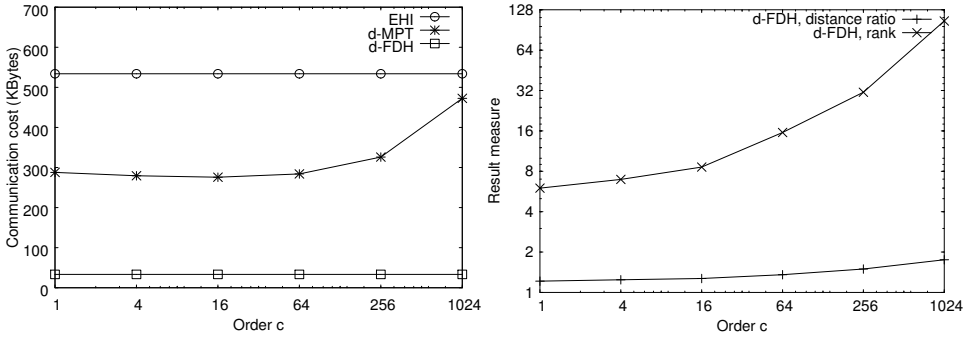
and it can be higher than that of EHI at large values of c. Regarding δ-FDH, its communication cost is insensitive to c but its result distance ratio and rank increase with c. Note that the performance and accuracy of MPT and FDH are close to δ-MPT and δ-FDH at a small c (e.g., 1).

Effect of k on the k-Sized Solutions.
In the final experiment, we investigate the effect of the size k on the performance and accuracy of the extended solutions: k-MPT, and k-FDH. Figure 13 plots the communication cost and result measure as a function of k. The communication cost of k-MPT increases with k. The communication cost of k-FDH is independent of k; however, both its result distance ratio and rank rise with k. We find that the performance and accuracy of MPT and FDH are close to k-MPT and k-FDH at a small k (e.g., 1).

Summary.
In summary, δ-FDH/k-FDH is preferred when the user has very limited communication bandwidth and δ-MPT/k-MPT is preferred when the user requires exact answers. EHI is an appropriate solution when the data owner needs a high data privacy guarantee (i.e., large δ or k) and the user requires exact answers.

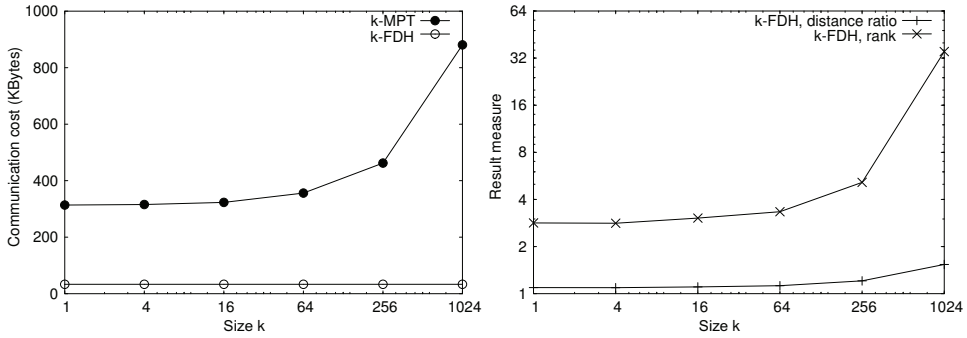
6 Conclusions

We proposed similarity search techniques for sensitive metric data, e.g., bioinformatics data, that enable outsourcing of such search. Existing solutions either offer query efficiency at no privacy, or they offer complete data privacy while sacrificing query efficiency. We introduce approaches that shift search functionality to the server. The proposed Metric Preserving Transformation (MPT) stores relative distance information at the server with respect to a private set of anchor objects. This method guarantees correctness of the final search result, but at the cost of two



(a) communication cost (b) result measure

Figure 12: Effect of Order c, on GFC Data



(a) communication cost (b) result measure

Figure 13: Effect of Size k, on GFC Data

rounds of communication. The proposed Flexible Distance-based Hashing (FDH) methods finishes in just a single round of communication, but does not guarantee retrieval of the exact result. The experimental evaluation shows that the approximation is very close to the exact result. Both MPT and FDH are extended to satisfy the δ-gap and the k-sized privacy guarantees. We demonstrate their efficiency and privacy on synthetic and real-world data.

Acknowledgements

The research was conducted when I. Assent and C. S. Jensen were full-time employees at Aalborg University, Denmark. C. S. Jensen is an Adjunct Professor at University of Agder, Norway.

References

[1] G. Aggarwal, T. Feder, K. Kenthapadi, S. Khuller, R. Panigrahy, D. Thomas, and A. Zhu. Achieving Anonymity via Clustering. In *PODS*, pages 153–162, 2006.

[2] R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking Relational Data: Framework, Algorithms and Analysis. *VLDB J.*, 12(2):157–169, 2003.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-Preserving Encryption for Numeric Data. In *SIGMOD*, pages 563–574, 2004.

[4] R. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In *SIGMOD*, pages 439–450, 2000.

[5] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati. Location Privacy Protection Through Obfuscation-Based Techniques. In *DBSec*, pages 47–60, 2007.

[6] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest Neighbor Retrieval Using Distance-Based Hashing. In *ICDE*, pages 327–336, 2008.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, pages 322–331, 1990.

[8] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An Index Structure for High-Dimensional Data. In *VLDB*, pages 28–39, 1996.

[9] T. Bozkaya and Z. M. Özsoyoglu. Indexing Large Metric Spaces for Similarity Search Queries. *TODS*, 24(3):361–404, 1999.

[10] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in Metric Spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[11] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*, pages 426–435, 1997.

[12] E. Damiani, S. D. C. Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In *CCS*, pages 93–102, 2003.

[13] M. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall, 2002.

[14] C. Faloutsos and K.-I. Lin. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *SIGMOD*, pages 163–174, 1995.

[15] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K. L. Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *SIGMOD*, pages 121–132, 2008.

[16] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *VLDB*, pages 518–529, 1999.

[17] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database-Service-Provider Model. In *SIGMOD*, pages 216–227, 2002.

[18] H. Hacigümüs, S. Mehrotra, and B. R. Iyer. Providing Database as a Service. In *ICDE*, pages 29–40, 2002.

[19] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What Is the Nearest Neighbor in High Dimensional Spaces? In *VLDB*, pages 506–515, 2000.

[20] G. R. Hjaltason and H. Samet. Index-Driven Similarity Search in Metric Spaces. *TODS*, 28(4):517–580, 2003.

[21] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Z. 0003. iDistance: An Adaptive B^+-Tree Based Indexing Method for Nearest Neighbor Search. *TODS*, 30(2):364–397, 2005.

[22] C. T. Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes. In *EDBT*, pages 51–65, 2000.

[23] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the Privacy Preserving Properties of Random Data Perturbation Techniques. In *ICDM*, pages 99–106, 2003.

[24] A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*, pages 239–257, 2007.

[25] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *ICDE*, page 25, 2006.

[26] T. Seidl and H. P. Kriegel. Optimal Multi-step k-Nearest Neighbor Search. In *SIGMOD*, pages 154–165, 1998.

[27] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *IJUFKS*, 10(5):557–570, 2002.

[28] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis. Secure k-NN Computation on Encrypted Databases. In *SIGMOD*, pages 139–152, 2009.

[29] P. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *SODA*, pages 311–321, 1993.

[30] M. L. Yiu, G. Ghinita, C. S. Jensen, and P. Kalnis. Outsourcing Search Services on Private Spatial Data. In *ICDE*, pages 1140–1143, 2009.