The VLDB Journal (2012) 21:333–358
DOI 10.1007/s00778-011-0245-7

REGULAR PAPER

Secure multidimensional range queries over outsourced data

Bijit Hore · Sharad Mehrotra · Mustafa Canim · Murat Kantarcioglu

Received: 21 September 2010 / Revised: 4 June 2011 / Accepted: 1 July 2011 / Published online: 9 August 2011

Abstract In this paper, we study the problem of supporting multidimensional range queries on encrypted data. The problem is motivated by secure data outsourcing applications where a client may store his/her data on a remote server in encrypted form and want to execute queries using server's computational capabilities. The solution approach is to compute a *secure indexing tag* of the data by applying *bucketization* (a generic form of data partitioning) which prevents the server from learning exact values but still allows it to check if a record satisfies the query predicate. Queries are evaluated in an *approximate* manner where the returned set of records may contain some false positives. These records then need to be weeded out by the client which comprises the computational overhead of our scheme. We develop a bucketization procedure for answering *multidimensional range queries* on multidimensional data. For a given bucketization scheme, we derive cost and disclosure-risk metrics that estimate client's computational overhead and disclosure risk respectively. Given a multidimensional dataset, its bucketization is posed as an optimization problem where the goal is to minimize the risk of disclosure while keeping query cost (client's computational overhead) below a certain user-specified threshold value. We provide a tunable data bucketization algorithm that allows the data owner to control the trade-off between disclosure risk and cost. We also study the trade-off characteristics through an extensive set of experiments on real and synthetic data.

Keywords Privacy · Disclosure · Confidentiality · Outsourcing · Security · Query execution · Relational

B. Hore (✉) · S. Mehrotra
Donald Bren School of Computer Science,
University of California, Irvine, CA, USA
e-mail: bhore@ics.uci.edu

S. Mehrotra
e-mail: sharad@ics.uci.edu

M. Canim
IBM T. J. Watson, New York, NY, USA
e-mail: mustafa@us.ibm.com

M. Kantarcioglu
University of Texas at Dallas, Richardson, TX, USA
e-mail: muratk@utdallas.edu

1 Introduction

The problem of querying encrypted data is primarily motivated by data outsourcing applications. In a typical setup, the service provider hosts the data of its clients and provides a variety of data management functionalities, like queries, updates, modifications etc. While the server is expected to implement the various operations correctly, the client may not trust the server side with complete access to its sensitive data. The solution is to store the sensitive data in encrypted form and keep the keys secret from the server at all times. To query this encrypted data, however, the server should be able to evaluate various query predicates against the data without having to decrypt it.

In this paper, we study the problem of supporting the important class of *multidimensional range* queries over relational data in a privacy-preserving manner. There are many practical applications where such range queries need to be supported—for example, an internet service provider may outsource the management of the web logs of his clients. He may want to query this data to analyze traffic patterns by date ranges and IP address ranges specified as 128.54.$*$ etc. Consider another scenario where a large company keeps an inventory of its stock across its warehouses using RFID tags and readers. This data may be outsourced to a service provider for cost-saving purposes, but needs to be frequently queried by analysts managing the supply chain. The queries pertaining to location of goods may be specified using geographic regions and/or time ranges. Yet, another example could be of a credit-rating agency that outsources its database to a remote service provider where data are queried by predicates specified on various financial attributes of an individual. Range queries in such applications comprise an important class of queries. All this data contain sensitive information which the owner may want to keep hidden from the server. As a concrete example, consider the following relational table which may be stored remotely by the credit-rating agency:

```
CUSTOMER(ssn,name,sex,salary,age,credit-rating)
```

The goal is to support multidimensional range queries like the ones shown below without revealing the content of the relation CUSTOMER to the database server.

```
SELECT * FROM CUSTOMER as C,
WHERE C.salary > 50K AND C.salary < 75K

SELECT * FROM CUSTOMER as C,
WHERE C.salary > 100K AND C.age < 35
```

The general approach is to encrypt the actual data using some semantically secure encryption algorithm like AES [1]. The query predicate is evaluated by the server against some *indexing* information created from the plaintext and stored along with the original data as a "tag". In cryptographic approaches, the tag is the ciphertext generated by applying the "searchable encryption" algorithm that as input a secret key and the plaintext data generate the ciphertext. The server can then evaluate the query predicate against this tag once an appropriate "trapdoor" has been released by the client. An alternative approach, one that we use in this paper, is based on data partitioning (also known as *bucketization* in the literature). Here, the data are first partitioned into buckets and the bucket-id is set as the tag for each data item in the bucket. A query posed by the client is then translated suitably before issuing it to the server who can evaluate it using only the information in the index tags corresponding to the data items. We provide a little background on bucketization next.

1.1 Bucketization background

Hacigumus et al. [23] were the first ones to propose the bucketization-based data representation for query processing in an *untrusted* environment (described in detail in Sect. 3.2). Their bucketization was simply a data partitioning step similar to those used for histogram construction, for instance, equi-depth, equi-width partitioning, etc. followed by assignment of a random (index) tag to each bucket effectively making every element within a bucket indistinguishable from another. When a query is issued by the client (data owner), it is first determined which buckets intersect the query using the index tag stored on the client (this is typically a small amount of information) and all contents of the intersecting buckets are retrieved from the server. The disadvantage is that the query result almost certainly consists of false positives which have to be eliminated by the client in the post-processing phase. Unfortunately, the bucketization schemes proposed in [23] are neither optimal for minimizing false positives, nor are they especially suited for preventing disclosure of data values against the simplest of adversarial attacks. A more principled approach for one-dimensional range queries was developed by the authors of the current paper in context of single-dimensional range queries over numeric attributes [29].

Range queries over data with numeric attributes are an important class of queries that has received a lot of attention in the literature [4,6,9,29,34,43]. There are essentially three categories of solutions that have been developed for range queries—(i) those that use some specialized data structure for range query evaluation while trying to preserve notions of semantic security of the encrypted representation [9,34,43]. (ii) Order-preserving encryption-based techniques [4,6] that ensures that order amongst plaintext data is preserved in the ciphertext domain. This allows direct translation of range predicates from the original domain to the domain of the ciphertext. (iii) Bucketization-based techniques like [29] and this paper, that use distributional properties of the dataset to partition and index them for efficient querying while trying to keep the information disclosure to a minimum. The information disclosure analysis is akin to statistical disclosure control and privacy-preserving data publishing approaches [35,47]. We will describe some of the techniques in the first two classes in more in depth in Sect. 7 on related works. Let us now delve into the bucketization-based approach for multidimensional range queries.

1.2 Bucketization-based techniques for range search

As mentioned above, bucketization can be seen as a generalized partitioning algorithm that induces indistinguishability amongst data objets in a controlled manner. The nature of disclosure in bucketization-based schemes is different from that in cryptographic schemes. In the latter, the disclosure risk is inversely proportional to the difficulty of breaking the encryption scheme and if broken, there is complete disclosure of the plaintext values. In contrast, the information disclosure in bucketization approaches could be partial or probabilistic in nature. That is, there could be a non-negligible probability of partial disclosure of a sensitive value given the transformed data, e.g., the bucket identity might identify some intervals to which the true value of the sensitive attribute belongs to. Also, unlike cryptographic schemes that aim for exact predicate evaluation, bucketization admits false positives while ensuring all matching data are retrieved. A post-processing step is required at the client to weed out the false positives. Data bucketization is set up as a cost-based optimization problem where buckets are created so as to minimize the average number of false positives per query. One of the biggest advantages of bucketization framework is that expressive and complex queries can be evaluated relatively efficiently. Further, the implementation of bucket-based query evaluation is often much simpler than cryptographic protocols and do not require any specialized algorithms to be executed on the server. Finally, bucketization can actually be composed with many of the cryptographic techniques to enhance the confidentiality properties of searchable encryption schemes, but we will not go into them in detail here.

1.3 Issues with extending single-dimensional schemes (like [29]) to multiple dimensions

Out of all the works mentioned above, only [43] explicitly addresses the multidimensional range search technique over encrypted data. We note (as has been observed in [43]) that one can adapt 1-dimensional search techniques to the multidimensional case, but it leads to unwanted leakage of information. As illustrated in the following example, if one were to retrieve the data satisfying the query predicate along each dimension independently and then compute the final set by taking the intersection of these sets, it can lead to information leakage.

1.4 Example

Consider two 2-d range queries $R_1(r_1^1, r_1^2)$ and $R_2(r_2^1, r_2^2)$ where r_i^j represents the predicate of the ith query along jth dimension. If we were to implement the range queries naively by retrieving the set of records corresponding to each r_i^j separately and then taking their intersection, after answering the two queries above, the server would also be able to determine all the records that satisfy the two range predicates (r_1^1, r_2^2) and (r_2^1, r_1^2) by simple intersection of the lists retrieved for R_1 and R_2. This is clearly more information than what the server needs to know. While the technique proposed in this paper (and in [29]) only return "noisy" sets (i.e., that may contain false positives), the simple list intersection attack reveals more information than necessary and can help the adversary guess the true value of a sensitive attribute.

Hence, it is clear that there is a need for developing efficient algorithms for supporting multidimensional range queries on encrypted data.

1.5 Contributions of this paper

In this paper, we generalize the bucketization-based approach for 1-dimensional range queries proposed in our previous work [29] to the multidimensional case. Specifically, we describe how the bucketization technique can be applied to multidimensional data. We will derive appropriate cost function for measuring the overhead of evaluating multidimensional range queries in this model. This, as we will see, turns out to be substantially more complicated than the single-dimensional case. We will develop a heuristics-based polynomial time algorithm that works very well in practice. To summarize, the additional contributions of this paper over and above [29] are:

- We derive the disclosure metrics for bucketization from first principles and confirm the correctness of the intuition that was given in [29].
- We derive a new cost metric that measures the expected performance overhead for multidimensional range queries over bucketized data.
- We propose an efficient multidimensional bucketization algorithm that uses novel heuristics to minimize the cost.
- We provide insightful discussions to the nature of privacy-performance trade-off that bucketization facilitates for multidimensional data.
- Finally, we carry out extensive experimentation using real and synthetic data to measure the performance and quality of our algorithms and illustrate how they are superior to off-the-shelf data partitioning approaches.

1.6 Outline of paper

In the next section (Sect. 2), we describe the typical data outsourcing architecture and give an overview of the data representation and query execution method proposed in this work. In Sect. 3, we derive the metrics that reflect the cost of executing single and multidimensional range queries over bucketized data. In the same section, we also propose two measures of disclosure risk for the bucketized data. Section 4 develops the 2-stage bucketization algorithm. The first stage consists of an optimal partitioning algorithm that creates buckets that minimize query execution overhead. The second stage is the controlled diffusion algorithm that mixes the contents of the buckets generated in the first stage to create "safer" buckets while trading off a specified amount of performance (i.e., while keeping the performance overhead below an user-specified threshold). In Sect. 5, we discuss the results from our extensive set of experiments for both single and multidimensional data and queries. We provide some discussion on practical aspects of our approach in Sect. 6. We discuss some related work in Sect. 7 and conclude in Sect. 8.

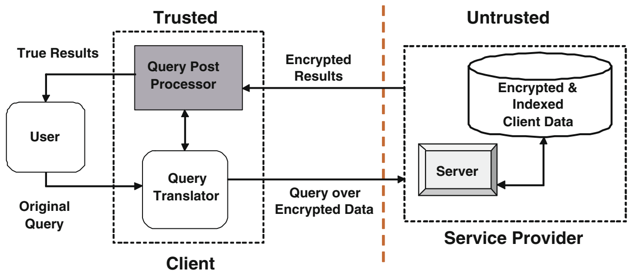

Fig. 1 Data outsourcing model

2 Preliminaries

2.1 The outsourcing model

Figure 1 shows a simplified architecture for data outsourcing application. Here, the server-side environment is assumed to be untrusted and therefore the sensitive data must always remain encrypted on the server. We describe the untrusted server model and corresponding adversarial attacks in Sect. 3.2. To evaluate SQL queries, the client can download the whole table onto the server side and then decrypt the tables and execute the query. A trivial approach such as this, however, would defeat the purpose of database outsourcing, reducing it to essentially a remote secure storage. Instead, the goal is to enable query processing at the server without decrypting the data.

The client consists of two main components: the *query translator* and *query post-processor* which are always assumed to operate within the secure environment. To enable query processing, auxiliary data are added to the each encrypted record and referred to as the *index tag*. The plaintext queries of the user needs to be translated appropriately so that the server can evaluate it against the secure indexes. This is carried out by the query translator using the *mapping functions*. The server evaluates the translated query predicate against the indexes and returns a set of encrypted records. The returned set contains all true answers possibly along with some false-positive records. The post-processor decrypts the data and eliminates the false positives. We will illustrate the query translation and response process shortly. The client also uses the mapping functions for generating the index information on updates and new insertions.[1] The re-computation of these secure indexes is expected to be infrequent[2] and therefore the amortized cost is not too great on the client. Additionally, key management and encryption/decryption of data are client's responsibility and are all carried out exclusively within its secure perimeter. The key management issues are outside the scope of the current paper and will not be discussed here any further.

2.2 Bucketization-based indexes for query processing

2.2.1 *Encrypted representation of relational data*

Consider the CUSTOMER table introduced earlier:

```
CUSTOMER(ssn,name,sex,salary,age,credit-rating)
```

The client may want to store values of "salary" and "age" encrypted so that they are not disclosed but still support queries with predicates defined on these attributes. We will assume that data are encrypted at the row level, i.e., each row of the table is encrypted as a single unit which we will refer to as an *e-tuple* [23,29]. In a d-dimensional model for the relational data, the d dimensions typically correspond to the attributes which are used in query predicates. To enable queries, the server-side indexing data are generated as follows. The set of data points are partitioned into M d-dimensional buckets on the client (partitioning algorithm will be described in Sect. 4) and the bucket label of each record is stored alongside the corresponding e-tuple on the server. The client stores the extents of each of the M rectangular buckets which are then used to determine whether an user query intersects with a bucket. This requires an additional O(dM) space on the client. We illustrate the bucketization and query translation process for multidimensional range queries below.

2.2.2 *Multidimensional data and range queries*

A multidimensional dataset R of dimension d is any set of points where each point is specified by an ordered set of d co-ordinates, where each co-ordinate takes values from a specified domain $\mathcal{D}_i, i = 1, \ldots, d$. A multidimensional range query in a d dimensional space is specified by two d-dimensional points p_1 and p_2 which determine the end points of the longest diagonal where all co-ordinates of p_1 have values smaller than or equal to the corresponding co-ordinates of p_2. Answering a multidimensional range query q consists of selecting the set of points from R that satisfy the query predicate, i.e., lie within or on the query rectangle. In the outsourcing model, the client specifies the query and the server retrieves the set of points from R that satisfy the predicate and returns them to the client over the network.

In Fig. 2, the original CUSTOMER table is shown in the bottom left-hand corner. Let the queried attributes be *salary* and *age*. The partitioning algorithm maps the projected two-tuple (salary, age) of each row in R to a two-dimensional

[1] For generating the secure index information when new data are added, the client makes use of the mapping functions which were constructed during the creation of the secure indexes the first time. We assume that a table is indexed only when it is of substantial size and significant efficiencies can be expected by selective access during query evaluation on the server side (as opposed to simply downloading the whole dataset onto the client).

[2] A re-computation of indexes is required only when the data distribution changes substantially and this we assume is a rare event.

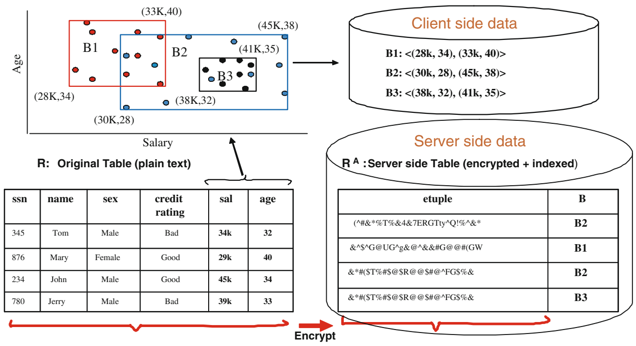

Fig. 2 Queries on bucketized data

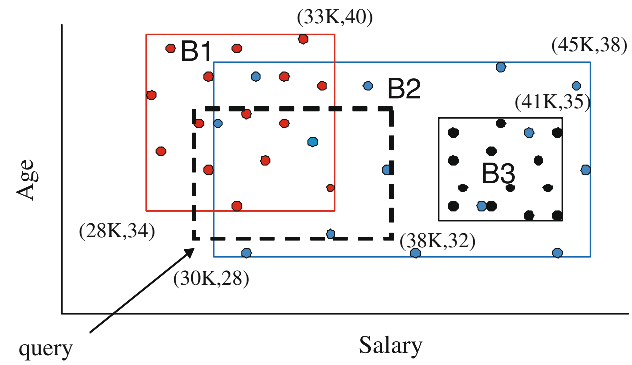

Fig. 3 Query cost estimation

space and clusters them into 3 groups represented by the rectangles B_1, B_2 and B_3. A new column is added to the table, for the bucket label and denoted B. The B-value of each tuple denotes the bucket to which that tuple is assigned. The bucket labels by themselves give out no information regarding the nature of values within the buckets. Each original tuple is then encrypted and stored as a single encrypted string (*e-tuple*) along with its assigned bucket label on the server side in the table R^A. The bucket label is stored in plaintext to allow queries. Now consider a range query q: "*Select* $*$ *From R Where salary* $\in [25k, 35k] \wedge$ *age* $\in [30, 36]$." To fetch the tuples meeting the selection criteria of q, the client utilizes the translation information (client-side data) to map q to a query over the buckets. As a result, q is translated to q': "*Select* $*$ *From* R^A *Where bucket* $= B_1 \vee B_2$". This not only retrieves the true answer set, but also some false positives. Since the false positives comprise the overhead of the scheme, we set the *cost* of a bucket as an estimate of the total number of false positives that it contributes over the set of all possible range queries. We will see in Sect. 3 how the cost of buckets can be estimated.

2.2.3 Encryption of bucket labels

One may think that using non-deterministic encryption of the bucket labels for each e-tuple on the server side will raise the level of security, but that is not so. The reason being, in our query model, all elements in a bucket need to be retrieved together. Therefore, irrespective of whether bucket labels have been encrypted or not, over sufficient number of distinct queries, an adversary will be able to determine which elements belong to the same bucket with a reasonably high probability. As a result, in the asymptotic analysis, encryption of bucket labels do not make a difference.

3 Performance cost and disclosure risk

With respect to a query, a bucketization (partitioning) of the dataset is considered good if the number of false positives retrieved is not too large. Intuitively, one can see that greater the number of buckets, smaller is the overhead per query. In contrast, from security point of view, the best scheme is to assign all data points to one bucket. Such a scheme does not leak any information whatsoever, to a server-side adversary, but it is obviously unacceptable for large datasets since every query will lead to retrieval of the whole dataset. In this section, we will assume that the owner specifies the number of buckets M that are to be used. Later in Sect. 6, we will describe how a good value of M may be selected in practice. For a given set (or class) of queries, the optimal partitioning scheme (from performance perspective) is the one that minimizes the total number of false-positive retrievals over all queries in this class. We analytically derive expressions for the total number of false positives retrieved for the class of multidimensional *range queries*. We will refer to the expression of number of false positives as the *cost measure* or *objective function*. The objective is then to minimize the cost of the bucketization scheme. In [29], we derived the cost measures for single-dimensional case where the attribute to be queried is also the sensitive attribute. Here, we will derive the cost measure for the generic multidimensional case. One important distinction between the solution for single and multidimensional cases is that while the former has an optimum polynomial time solution, the multidimensional partitioning problem is most likely NP-hard.

3.1 Cost metrics for multidimensional range queries

Consider the two-dimensional Euclidean space where the buckets are 2-dimensional rectangles as shown in Fig. 3. Assume the attributes are integer valued. Then, we can specify any query Q by an ordered pair of two reals. For example, $Q_{(k_1,k_2)}$ denotes the set of rectangular queries with extents k_1 and k_2 along the first and second dimensions respectively. Consider a 2-dimensional bucket B with edges (b_1, b_2). The number of distinct queries from $Q_{(k_1,k_2)}$ that overlap with B are $(b_1 + k_1)(b_2 + k_2)$. Each distinct data value p

(multidimensional point)[3] in B overlaps with exactly $k_1 * k_2$ of the queries from this set. For the remaining queries, p contributes only false positives. Therefore, each point within a bucket contributes f_p false positives for $[b_1b_2+b_1k_2+b_2k_1]$ of the queries in $Q_{(k_1,k_2)}$ where f_p is the frequency of point p in the dataset. For instance, the query shown in Fig. 3 will end up retrieving all the points assigned to the buckets $B1$ and $B2$ (all the red and blue points), thereby retrieving a number of false positives along with the true solution set.

Over all queries that overlap a bucket B, the total number of false positives that each point p with frequency f_p contributes, is $False(p) = f_p \sum_{k_1} \sum_{k_2} [b_1b_2 + b_1k_2 + b_2k_1]$. The total number of false positives gives a good estimate of the query overhead under the assumption that all possible range queries are equi-probable. Assume that k_i takes values from $1, \ldots, N$. Breaking the summation we get the following:

$$\begin{aligned}
False(p) &= f_p \sum_{k_1=1}^{N} [Nb_1b_2 + Nb_2k_1] + f_p N b_1 \sum_{k_2=1}^{N} k_2 \\
&= f_p N^2 b_1 b_2 + f_p N b_2 \frac{N(N+1)}{2} \\
&\quad + f_p N b_1 \frac{N(N+1)}{2} \\
&= f_p N^2 \left[b_1b_2 + \frac{(N+1)b_2}{2} + \frac{(N+1)b_1}{2} \right] \\
False(p) &= f_p N^2 \left[b_1b_2 + \frac{(N+1)(b_1+b_2)}{2} \right]
\end{aligned}$$

$False(p)$ denotes the number of false positives for a point $p \in B$. Therefore, the total number of false positives for B is given by

$$TFP(B) = \sum_{p \in B} N^2 f_p \left[b_1b_2 + \frac{(N+1)}{2}(b_1+b_2) \right]$$

where F_B denotes the total number of data points (including multiple copies) in B. Since N is a constant for the domain, we observe that $TFP(B)$ is proportional to $F_B[b_1b_2 + C(b_1 + b_2)]$ where $C = \frac{N+1}{2}$, a constant.

Now let us see the case of 3-dimensional data: assume again that we have a fixed bucket $B = (b_1, b_2, b_3)$ and the class of queries with edge fixed edge lengths (k_1, k_2, k_3) be denoted $Q_{k_1k_2k_3}$. From this class, $k_1 * k_2 * k_3$ queries overlap any single point "p". For the other queries from this group that overlap the bucket B, p is a false positive. Similar to the 2-dimensional case, we compute the number of false positives $False(p)$, the point p contributes over all possible range queries where k_i's can vary over the complete range $[1, N]$.

[3] We use the term *value* and *point* to mean the same thing when referring to a multidimensional dataset.

$$\begin{aligned}
False(p) \propto f_p[&b_1b_2b_3 + b_1b_2k_3 + b_1k_2b_3 \\
&+ k_1b_2b_3 + b_1k_2k_3 + k_1b_2k_3 + k_1k_2b_3]
\end{aligned}$$

Computing exactly

$$\begin{aligned}
False(p) &= f_p \sum_{k_1} \sum_{k_2} \sum_{k_3} [b_1b_2b_3 + b_1b_2k_3 + b_1k_2b_3 \\
&\qquad + k_1b_2b_3 + b_1k_2k_3 + k_1b_2k_3 + k_1k_2b_3] \\
False(p) &= f_p \Bigg[N^3 b_1b_2b_3 + N^2 b_1 b_2 \frac{N(N+1)}{2} \\
&\qquad + N^2 b_1 b_3 \frac{N(N+1)}{2} + N^2 b_2 b_3 \frac{N(N+1)}{2} \\
&\qquad + N b_1 \frac{N^2(N+1)^2}{2^2} + N b_2 \frac{N^2(N+1)^2}{2^2} \\
&\qquad + N b_3 \frac{N^2(N+1)^2}{2^2} \Bigg] \\
False(p) &= N^3 f_p \Bigg[b_1b_2b_3 + \frac{N+1}{2} \sum_{i<j} b_i b_j \\
&\qquad + \left(\frac{N+1}{2}\right)^2 \sum_i b_i \Bigg]
\end{aligned}$$

For the whole bucket (with F_B total points), the total number of false positives over all queries that intersect with B, we have:

$$\begin{aligned}
TFP(B) &= \sum_{p \in B} False(p) \\
&= N^3 F_B \Bigg[b_1b_2b_3 + \frac{N+1}{2} \sum_{i<j} b_i b_j \\
&\qquad + \left(\frac{N+1}{2}\right)^2 \sum_i b_i \Bigg]
\end{aligned}$$

It can be shown easily by induction on the number of dimensions d, that the total false positive for any d-dimensional bucket B, is given by:

$$\begin{aligned}
TFP(B) = N^d F_B \Bigg[\sum_{k=1}^{d} \Bigg\{ &\left(\frac{N+1}{2}\right)^{k-1} \\
&\times \sum_{i_1 < \cdots < i_t} (b_{i_1} b_{i_2} \cdots b_{i_t}) \Bigg\} \Bigg] \qquad (1)
\end{aligned}$$

where $t = d - k + 1$. Therefore, our objective is to minimize the value of $\sum_{B_i, i=1}^{M} TFP(B)$. In most cases since $N \gg b_i$, we see from the generic expressions of $TFP(B)$ (Eq. 1) that the dominant term is due to the last term when the expression inside the bracket is expanded. For example, for the 3-dimensional case, the dominant term within the bracket is $(\frac{N+1}{2})^2 \sum b_i$. Hence for all practical purposes, it is desirable that the bucketization minimize the term $\mathbf{F_B} \sum \mathbf{b_i^B}$ for a bucket B (where $b_i^B, i = 1 \dots d$ denote the edge lengths). Hence we use the following expression as the cost measure that needs to be **minimized**.

$$Cost = \sum_{\mathbf{t=1}}^{\mathbf{M}} \left(\mathbf{F_{B_t}} \sum_{\mathbf{i=1}}^{\mathbf{d}} \mathbf{b_i^{B_t}} \right) \tag{2}$$

where $b_i^B, i = 1 \cdots d$ denote the edge lengths of the bucket B. Note, the cost is proportional to the sum of edge lengths instead of the area of the buckets. This is intuitively correct since a bucket may have a very small volume (for instance, if one of its edges is very small) but still overlap with large number of data points.

In spite of the fact that buckets are allowed to overlap in arbitrary manner, there is no ambiguity in the querying process since we retrieve a bucket if and only if it overlaps the query rectangle. This class of clustering obviously covers the set of all non-overlapping partitioning schemes as well. Now, let us turn our attention to disclosure analysis of the bucketization process.

3.2 Disclosure model

In this paper, we analyze the security of our scheme in the *passive adversary* model. It is the most popular adversarial model considered in context of the secure data outsourcing problem by far. It is alternatively referred to as the *curious intruder* model in literature [22]. In this model the server is considered truthful in general, i.e., the server implements various data storage and query processing functionalities correctly. The adversary is one or more inquisitive individual(s) on the server side who has access to the data, e.g., a database administrator. Such an adversary only tries to learn sensitive information about the data without actively modifying it or disrupting any other kind of services.[4]

Since the actual data always remains encrypted on the server, there is no direct way in which the adversary can access this data unless he is able to break the encryption. We will assume that the user employs computationally strong encryption algorithms that makes the possibility of such an exposure negligible even if a hacker gets access to the encrypted records. To prevent any kind of information leakage, we will assume that the whole record (all attributes) is encrypted. Therefore, the only information that is visible to the adversary are the bucket labels corresponding to each row of the table.

3.2.1 Attack model

Given that all attributes of a record are encrypted on the server, the attack model we consider here is one where the adversary is able to determine one or more attribute values of some records in a bucket through an alternate channel. A real scenario where such an attack is plausible is when the adversary is able to insert a few known tuples into the dataset and see the corresponding bucket labels that are generated.[5] Such an adversary may comprise two or more individuals, one on the server side and the other on the client side who collude to insert "test records" and gather information about the bucket labels.

Next, we derive metrics for measuring how much information buckets leak about sensitive attribute values.

3.3 Measures of disclosure risk

The bucketization process can be seen as a process of "*Creating a new variable* **B** *(bucket label) which directly or indirectly captures some information about the sensitive variable* **S**". In our case, since all attributes are considered sensitive, **S** denotes the set of all data attributes. A subset of the attributes are labeled as the set of *queriable* attributes if the corresponding dimensions are used in the bucketization algorithm. For example, if *salary* and *age* are both queriable attributes, bucketization will be carried out on the 2-dimensional projection of the dataset along these two dimensions. The buckets may look something like the ones shown in Fig. 2. In this case, the bucket extents along the *salary* dimension directly captures some information about the salary values of the records in the bucket. If the adversary can learn the salary values of a few encrypted records, he can use this knowledge to predict the salary values of other records in the same bucket (by using some localization property). As an example of indirect disclosure, consider the *credit rating* attribute which is not designated as a queriable attribute. The two-dimensional buckets created on *salary* and *age* do not reveal information about the *credit rating* directly, but may do so indirectly since there may be a correlation between the *credit rating* of an individual and his/her *age* and *salary* attributes. For instance, if younger people with lower salaries tend to have bad credit, then knowing that a certain individual belongs to a bucket that spans lower salaries and ages indicates a higher probability that the person's credit ratings is bad. In such cases, we say that the bucket leads to indirect exposure about the particular attribute.

Therefore, we need an approach to measure and more importantly control the association between B and S. In other words, we want to compute the predictive power of B, i.e., "how well is the adversary able to predict the value of an attribute of a record by simply looking at its bucket label?".

[4] Almost all the approaches we describe in this paper address data confidentiality and privacy issues for the passive adversary scenario. We will refer to this model interchangeably as the *semi-trusted* model.

[5] In this work, we do not consider the cases of partial disclosure of encrypted records on the server. This is another interesting attack scenario which we intend to look into in future.

The information conveyed by variable X about Y can be quantified by the *mutual information* expression [15]:

$$I(X;Y) = H(X) - H(X|Y) = I(Y;X) \quad (3)$$

where $H(X)$ denotes the entropy of the variable X and $H(X|Y)$ is the conditional entropy of X given Y. When X is a discrete variable, the entropy is denoted by the following formula (here log denotes the logarithm to base 2 and $x \in domain(X)$ is denoted as $x \in X$ for short) [15].

$$H(X) = -\sum_{x \in X} p(x) \log(p(x)) \quad (4)$$

and the conditional entropy $H(X|Y)$ is given by the formula:

$$H(X|Y) = -\sum_{x \in X}\sum_{y \in Y} p(x,y) \log p(x|y) \quad (5)$$

Therefore, a measure of the information that B captures about the sensitive attribute S is given by

$$I(S;B) = H(S) - H(S|B)$$

Since $H(S)$ is constant, to minimize the risk, one has to minimize $-H(S|B)$ or in other words, maximize the **entropy** of the distribution of S within each bucket. This is same as saying that the distribution of the values that attribute S takes within each bucket is as uniform and spread out (over a large domain) as possible. This is analogous to the notion of l-diversity proposed for privacy-preserving data publishing [35].

If the sensitive attribute takes values from a discrete numeric domain,[6] then entropy by itself might not be a adequate measure of security. This is because, entropy measure for a discrete random variable does not take into consideration the order inherent in a domain. For example, let the *salary* domain be [50k, 150k] and let it be discretized into 100 "classes" [50k, 51k], [51k, 52k], …, [149k, 150k]. Let bucket B1 contain 9 tuples, 3 each with salary values in ranges [50k, 51k], [53k, 54k] and [55k, 56k]. Consider a second bucket B2 which also has 9 tuples with 3 each having values in [60k, 61k], [100k, 101k] and [145k, 146k]. The entropy of the distribution of values within buckets B1 and B2 is the same whereas, the spread of the first bucket is merely 6k and that of the latter is approximately 85k. As a result, if one salary value is exposed from B1, say accidentally then the adversary is able to estimate the remaining values in B1 within a relatively small error bound. Therefore, the second distribution is clearly more preferable from the point of view of disclosure risk. To capture this notion of security, we propose the **variance** of the bucket-level distribution (of the sensitive values) as the second measure of disclosure risk. That is, higher the variance of the value distribution within each bucket, more secure is the bucketization scheme. The following theorem shows how variance of the value distribution within a bucket captures the adversarial uncertainty in estimating the correct value of the attribute for any record. We first define the term *Average Squared Error of Estimation* (*ASEE*) (This is reproduced here from [29] for completeness):

3.3.1 ASEE

Let a random variable X_B follow the same distribution as the sensitive values in bucket B, denoted as P_B. For the case of a discrete (continuous) random variable, let the corresponding probability mass (density) function be denoted by p_B. Then, a measure of disclosure risk will be denoted by the adversary's ability to **estimate** the true value of a random element chosen from this bucket. We assume that A employs a statistical estimator for this purpose denoted X'_B which follows the probability distribution P'_B (the statistical estimator is itself a random variable). If A assigns the value x_i to X'_B with probability $p'_B(x_i)$ and there are N distinct values in the domain of B. Now, since there is no information to distinguish between two elements of the same bucket, the variables X_B and X'_B are independent for every bucket B. Now, we define **Average Squared Error of Estimation (ASEE)** as:

Definition 1

$$\begin{aligned}\mathrm{ASEE}(X_B, X'_B) &= \sum_{j=1}^{N}\sum_{i=1}^{N} p'_B(x_i|x_j)p_B(x_j)(x_i - x_j)^2\\ &= \sum_{j=1}^{N}\sum_{i=1}^{N} p'_B(x_i)p_B(x_j)(x_i - x_j)^2\end{aligned}$$

Proposition 1 **ASEE(X, X′) = Var(X)+Var(X′)+(E(X) − E(X′))²** *where X and X' are random variables with probability mass (density) functions p and p', respectively. Also* **Var(X)** *and* **E(X)** *denote variance and expectation of X respectively.*

Proof We have from definition of $ASEE(X, X')$

$$\begin{aligned}&= \sum_{i=1}^{N}\sum_{j=1}^{N} p'(x_i)p(x_j)(x_i - x_j)^2\\ &= \sum_{i=1}^{N} p'(x_i)\sum_{j=1}^{N} p(x_j)(x_i - x_j)^2\\ &= \sum_{i=1}^{N} p'(x_i)\sum_{j=1}^{N} p(x_j)\left(x_i{}^2 + x_j{}^2 - 2x_ix_j\right)\\ &= \sum_{i=1}^{N} p'(x_i)\left[\sum_{j=1}^{N} p(x_j)x_i{}^2 + \sum_{j=1}^{N} p(x_j)x_j{}^2\right.\\ &\qquad \left. - 2\sum_{j=1}^{N} p(x_j)x_ix_j\right]\end{aligned}$$

[6] Even in the case of continuous domain, the variable is often discretized for entropy computation.

Using $Var(X) = \sigma^2 = E(X^2) - \mu^2$ (where $\mu = E(X)$) we get

$$= \sum_{i=1}^{N} p'(x_i) \left[1.x_i{}^2 + \left(\sigma^2 + \mu^2\right) - 2\mu x_i\right]$$
$$= \sum_{i=1}^{N} p'(x_i)x_i{}^2 + \left(\sigma^2 + \mu^2\right) \sum_{i=1}^{N} p'(x_i) - 2\mu \sum_{i=1}^{N} p'(x_i)x_i$$
$$= \left(\sigma'^2 + \mu'^2\right) + \left(\sigma^2 + \mu^2\right) - 2\mu\mu'$$
$$= \sigma^2 + \sigma'^2 + (\mu - \mu')^2$$
$$= Var(X) + Var(X') + (E(X) - E(X'))^2$$

□

It is easy to see that adversary can minimize $ASEE(X_B, X'_B)$ for a bucket B in two ways: (1) by reducing $Var(X'_B)$ and (2) by reducing the absolute value of the difference $E(X_B) - E(X'_B)$. Therefore, the best estimator of the value of an element from bucket B that the adversary can get is the constant estimator equal to the mean of the distribution of the elements in B (i.e., $E(X_B)$). For the *constant estimator* X'_B, $Var(X'_B) = 0$. Also, as follows from basic sampling theory, the "mean value of the sample-means is a good estimator of the population (true) mean" [13]. Thus, the adversary can minimize the last term in the above expression by obtaining a large number of samples,[7] i.e., *plaintext* values from B. However, note that the one factor that the adversary cannot control (irrespective of the statistical estimator he uses) is the true variance of the bucket values, $Var(X_B)$. Therefore, even in the best case scenario (i.e., $E(X'_B) = E(X_B)$ and $Var(X'_B) = 0$), A still cannot reduce the ASEE below $Var(X_B)$, which, therefore, forms the lower bound of the accuracy achievable by A. Hence, we conclude that the data owner (client) should try to bucketize data in order to **maximize the variance** of the distribution of values within each bucket.

3.3.2 Information leakage through query patterns

In the outsourced data security, we need to deal with multiple security issues: (a) secure storage and retrieval of encrypted data (b) prevent information disclosure based on query access patterns. Using encryption and bucketing ideas, we address the first issue. On the other hand, our work and other work on encrypted key-word search [29] do not hide access patterns. In fact, most cryptographic solutions [6,8,9,14,20,43,44] are even more vulnerable since they attempt to return the exact set of answers. Maintaining access statistics accurately for such techniques is even easier for such techniques. To our knowledge, there are two types of techniques proposed in the literature to address the issue: the first is inspired by the Oblivious RAM [38]. The basic idea is to replace any single data request by a set of requests. While the original Oblivious RAM model required $O(log^4(n))$ requests (another version of the solution requires $O(log^3(n)$ requests with a large constant), authors in [38] came up with a technique that only requires $O(log^2(n))$ requests, where n is the number of records in the database. An alternate strategy is to periodically re-encrypt the data (thus changing the representation of the data at the server) at a frequency greater than what would be needed to launch an access pattern-based attack.

Query access pattern attacks are also possible in the context of bucketization-based approach, albeit they are somewhat less severe since bucketization-based approaches retrieve all records in the same bucket together there by making accurate determination of access frequency difficult. Furthermore, even if the adversary, through frequency-based attacks can determine the exact set of values (along with their frequencies) that are present in the bucket, there is a degree of protection since high entropy and high variance of sensitive values within each bucket guarantees a certain minimum level of uncertainty (security) against such attacks.

Nonetheless, our technique is susceptible, by itself to access pattern attacks. Techniques such as oblivious RAM and/or re-bucketization could be used to prevent against adversaries being able to make inferences. Some of the recent works on PIR [48] and access pattern hiding in untrusted stores [49] have proposed some interesting approaches keeping performance considerations in mind. We, however, believe that an exploration in any depth of how some of these techniques may apply to answer multidimensional range queries is outside the scope of the current paper.

Now, that we have given our cost and disclosure metrics, we turn our attention to algorithms for bucketizing the data.

4 Bucketization algorithms

The central goal of bucketization is to minimize the disclosure risk while keeping the query performance (number of false-positive retrievals) bounded within specified limits. Our bucketization consists of two phases—first create buckets that are optimized only from the performance perspective; then in a controlled manner re-distribute the contents of each optimal bucket into multiple *composite buckets*. The second phase is called *controlled diffusion*, where in one decreases the disclosure risk of bucket by making the contents more "diverse" by increasing the entropy and variance of the value distribution within each bucket. The controlled diffusion mechanism allows us to limit the worst-case degradation in query performance for the resulting composite

[7] While it is unlikely, but nonetheless quite possible that the adversary can monitor the plaintext values of a large fraction (or even all but one) of the records in a bucket.

buckets (from the performance achieved by the optimal buckets). One important point to note is that if we assign only a small number of data items to a bucket, then, one might not be able to achieve a desired level of entropy or variance of the value distribution. However, we do not propose a more principled way to derive the optimal size of buckets in this paper. This remains an important problem for future work.

Now, we present our multidimensional partitioning algorithm that tries to minimize the number of false positives retrieved for all range queries.

4.1 Partitioning multidimensional data

As mentioned earlier, in the first phase of the two-phase bucketization algorithm the objective is to determine the least cost partitioning of a multidimensional dataset into M of buckets (where M is smaller than the number of points in the dataset). The quality of a bucket is measured by the cost function derived in the previous section (Eq. 2). Intuitively, the cost measure captures the compactness of a bucket. A good bucketization minimizes the sum (over all buckets) of the product of a bucket's weight (number of data points) and its perimeter (or sum of its extents along each dimension). Since we are only interested in rectangular buckets, the following observation holds: "the longest diagonal of the rectangle is proportional to its perimeter". Now, the algorithm can utilize this fact to determine good new candidate buckets to form as follows—start with all points in a single bucket (note that this has maximum possible cost); then consider all pairs of data points such that a new rectangle defined with these two as the end points of its longest diagonal reduces the cost measure by the maximum amount, i.e., the reduction in cost when the data are partitioned into two rectangles, one set covered by the new rectangle, and remainder covered by the (modified) first rectangle. Since the total cost monotonically decreases in each iteration, the algorithm iterates till M distinct buckets are formed. We will illustrate the algorithm with an example shortly.

Most problems of optimal partitioning of multidimensional datasets are NP-hard [32,36,37]. While we do not explicitly show our problem to be NP-hard, we found that the following, very related partitioning problem [11] has been long known to be NP-hard.: "Given a set of objects X with pairwise distance d defined between each pair of elements in X, find a partitioning P of X into fixed number $q (q \geq 3)$ of buckets that minimizes the objective function $F(P) = \sum_{C \in P} diam(C)$ where $diam(C)$ is defined as $diam(C) = max_{x,y \in C} d(x, y)$". Our problem can be seen as a weighted version of this problem and is likely to be NP-hard as well.

Below, we present the "Query Optimal Bucketization (QOB)" algorithm below (Algorithm 1). It is a polynomial time algorithm which produces good buckets in practice. The

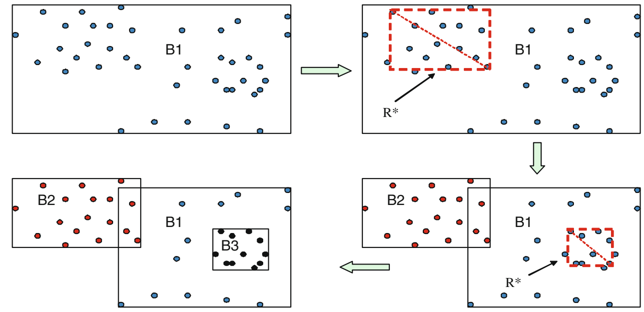

Fig. 4 Example: greedy optimal clustering

algorithm starts with a single bucket and greedily generates a new one in each iteration till the specified number of buckets is formed. We refer to the resulting buckets as "optimal" buckets.

Algorithm 1 Query Optimal Bucketization (QOB)

1: Input: Dataset of multidimensional points D, # buckets M;
2: Output: M buckets, Total Cost;
3:
4: $Cost(D, R_1) = |D| * \sum_{i=1}^{d} r_1^i$;
5: /* r_j^i is the length of the i^{th} edge of rectangle R_j */
6:
7: **for** $j = 2$ to M **do**
8: Over all pairs of points in D, choose the pair (p_1^*, p_2^*) and the corresponding rectangle R^* s.t. the cost reduction $Cost(D, \cup_{t=1}^{j-1} R_t) - [Cost(D \setminus D_{R^*}, \cup_{t=1}^{j-1} R_t) + Cost(D_{R^*}, R^*)]$ is maximized;
9: Assign points within R^* to a new cluster R_j & recompute the minimum bounding rectangles (MBRs) of all the affected clusters;
10: Make one pass on D & reassign all points to these j clusters (readjusting MBRs if necessary) to further reduce total cost;
11: **end for**
12: Output the M clusters and the total cost of scheme;

Note that the total cost decreases monotonically with increasing number of bins. The term $Cost(D, \cup_{t=1}^{j-1} R_t)$ is the sum over all rectangles R_t $(t = 1, \ldots, j-1)$ of the product of the *number of data points within* R_t and the *sum of the d edge lengths of* R_t (recall, R_t is a d-dimensional rectangle). For example, in the Fig. 4 above, consider the final bucketization. The cost of this scheme is given by $\sum_{i=1}^{i=3}$(#points in B_i)*(length(B_i)+width(B_i)). For each of the $M-1$ iterations, in step 8 of Algorithm 1, $O(|D|^2)$ candidates for a new cluster are considered. A candidate subset of points is specified by selecting all points within a bounding rectangle R whose diagonal endpoints correspond to a pair of points in D. The rectangle R^* that maximally reduces the cost is the chosen candidate and a new bucket is initialized using the points within R^*. This requires re-computing the MBRs (*Minimum Bounding Rectangles*) of the clusters that lost data elements to R^*. Since the candidate selection was restricted only to the rectangles which have one pair of their diagonal endpoints in the dataset, we can potentially reduce the cost further by readjusting the extents of the MBRs. Step 10 makes one more pass to carry out such a reassignment of points if it leads to cost reduction. An illustrative example with $M = 3$ is shown in Fig. 4. Beginning at the top left corner, initially all data points belong to the single bucket denoted as $B1$. The algorithm then chooses two data points in this set (corresponding to the two end points of the dotted line in the top right figure) that determine the "best rectangle" R^* to create a new bucket. After local re-arrangements, the new bucket is shown as $B2$. After the second iteration, the third bucket that maybe created is shown by $B3$ and so on.

4.1.1 Implementation of the greedy partitioning algorithm

In step 8 of the algorithm, for all pairs of points in $D\,(p_1, p_2)$, a temporary rectangle is created whose diagonal endpoints correspond to (p_1, p_2). Next, all of the points within the boundaries of this rectangle are moved to a new rectangle. After generating these temporary rectangles, the one which maximizes the cost reduction is selected and created which is named as R^*. In our implementation, we do not leverage any special data structure to find the set of points between p_1, p_2. Therefore, the cost of finding these points is $O(|D|)$. However, an R-tree implementation could be used to eliminate the linear scan cost. Once a multi-dimensional index is built on the dimensions, the points in the bounding rectangle could be retrieved efficiently. For a detailed discussion on R-tree and its variants, please refer to [42].

We found that step 9 does not necessarily lead to substantially increased efficiency in all cases and hence can be implemented as an optional optimization step. We implemented step 10 of the algorithm as follows. For every point p_i in dataset D, the following operations are performed: First, we iterate over each of the j-1 other rectangles (that do not contain the point) as follows: we move the point p_i to the rectangle and update its MBR and compute the cost of the rectangle with this new point. We determine the rectangle (R_{min}) which has the least cost after the addition of the point. Next, we remove the point from the owner rectangle (R_{owner}) and update its MBR. We compute the cost without point p_i with the updated MBR. If the former is greater than the latter, we move the point from R_{owner} to R_{min}. Otherwise, we conclude that moving p_i to any of the j-1 rectangles does not provide any improvement.

4.1.2 Complexity of greedy algorithm

A worst-case analysis for the straightforward implementation of the algorithm has a time complexity of $O(M|D|^3)$, where M is the number of buckets to be created and $|D|$ is the size of the dataset. In the first step within the for-loop, each pair of data points is considered which contributes $O(|D|^2)$ units and for each of these pairs, computing the magnitude of cost reduction may take another $O(|D|)$ units.

4.2 Security-performance trade-off: controlled diffusion

In Sects. 3.2 and 4.1, we analytically derived security and cost measures for data partitioning schemes to tackle multi-dimensional data. We also proposed algorithms to partition a dataset into a given number of buckets that minimizes the overhead of answering queries. We will refer to the resulting buckets as *optimal buckets*.[8] Since the algorithms proposed in Sect. 4.1 do not take security into consideration, indexing the user's data on the server using the optimal buckets might lead to a higher than desired disclosure risk. Intuitively, we can see that there is a direct trade-off between the level of security and performance. On one hand, security is best served when all data belongs to just a single bucket and each tuple is indistinguishable from any other on the server side. This, of course, leads to poor performance since all data needs to be retrieved on each query. The other extreme is actually poor from both the security and performance perspective. For instance, if we assign a bucket to every distinct value appearing in the dataset, there are no false positives, and as a result, the server always returns the exact solution set (i.e., no false positives) for each query. But, in this case, the size of the translated query (i.e., one that is posed to the server) could be quite large, therefore, offsetting the benefits of the accurate solution set. Additionally, the bucket contents are vulnerable to potential disclosure by statistical attacks due to low variance and entropy measures of the buckets.

From the above discussion it is clear that there is a trade-off between security and performance. An ideal partitioning approach should allow the user to explore this trade-off space and choose a partitioning scheme that best suits his requirements. Toward this goal, we propose a 2-phase parameterized partitioning algorithm that lets the user choose a desired level of trade-off between security and performance. Below, we present a simple trade-off algorithm that is "distribution agnostic" and works very well in practice.

4.2.1 Distribution-agnostic trade-off algorithm

The optimal buckets generated in the first phase are compact and tend to have a value distributions with small variance and/or entropy. Therefore, in the second phase we re-distributes the data in these buckets into a new set of buckets such that the average entropy and variance of the sensitive attribute's value distribution is substantially increased. We call this the *controlled diffusion algorithm* (CDf-algorithm)

[8] These buckets are not optimal and in fact, computing an optimal partitioning is NP-Hard for most cost functions. Nonetheless, we use the qualifier *optimal* to distinguish the buckets resulting from the first phase from the final ones which are called *composite* buckets.

(first proposed in [29]). The diffusion algorithm admits a control parameter *max-degradation factor* that lets the data owner control the trade-off between cost (performance overhead) and security. The problem is formally stated below:

Problem 1 (Trade-off Problem) *Given a dataset $D = (V, F)$ and an optimal set of M buckets on the data $\{B_1, B_2, \ldots, B_M\}$, re-bucketize the data into M new buckets, $\{CB_1, CB_2, \ldots, CB_M\}$ such that no more than a factor K of performance degradation is introduced and the* **minimum variance** *and* **minimum entropy** *amongst the M random variables $X_1, \ldots, X_M$ are simultaneously* **maximized**, *where the random variable X_i follows the distribution of values within the ith bucket.*

4.2.2 Controlled diffusion

Let optimal buckets be denoted by B_i's for $i = 1, \ldots, M$. The controlled diffusion process creates a new set of M approximately equi-depth buckets which are called the *composite buckets* (denoted by CB_j, $j = 1, \ldots, M$) by *diffusing* (i.e., re-distributing) elements from the B_i's into the CB_j's. The diffusion process is carried out in a controlled manner by restricting the number of CB's that the elements from a particular B_i get diffused into. The resulting set of composite buckets, the $\{CB_1, \ldots, CB_M\}$ form the final bucketized representation of the client data on the server, i.e., the e-tuples are tagged with these bucket labels. To retrieve the data elements in response to a range query q, the client first computes the query overlap with the optimal buckets (denoted as $q(B)$) and then determine the composite buckets that contain at least one element from any bucket in $q(B)$ which are then requested from the server.

A requirement for guaranteeing the bounds on false positives is that the M composite buckets should be approximately equal in size. This equi-depth constraint sets the target size of each CB to be a constant[9] $= f_{CB} = |D|/M$ where $|D|$ is size of the dataset (i.e., rows in the table). The performance constraint is enforced as follows: If the maximum allowed performance degradation $= K$, then for an optimal bucket B_i of size $|B_i|$ its elements are diffused into no more than $d_i = \frac{K*|B_i|}{f_{CB}}$ composite buckets. The diffusion factor d_i is rounded-off to the closest integer. In response to a range query q, the server will retrieve at most K times the number of records it would have retrieved using the optimal buckets. For example, if the optimal buckets retrieved in response to a query q were B_1 and B_2, then using CB_j's the server would not retrieve any more than $K * |B_1| + K * |B_2|$ elements. This ensures that precision of the retrieved set does not reduce by a factor greater than K.

[9] The actual size of the composite buckets needs to be approximately close to the target and need not be exactly $= f_{CB}$.

An added advantage of this approach is that, it guarantees the performance lower bound not just for the average precision of queries but for every query in the class individually. The important point to note is that the domains of the composite buckets overlap where elements with the same value can end up going to multiple CB's. This extra degree of freedom for the composite buckets allows the user to substantially increase the security level by trading off relatively small degree of performance (i.e., allowing for small degradation). In the extended paper [28], we derive an upper bound on the expected degradation in query precision resulting from the diffusion algorithm when the attribute value distribution is uniform. Interestingly, this factor turns out to be independent of the value of the parameter "Max-degradation-factor" K. The CDf algorithm allows the user to explore the "privacy-performance trade-off space" by choosing the value of K. We illustrate the diffusion process with an example below.

Algorithm 2 Controlled Diffusion (D, M, K)

Input: Dataset $D = (V, F)$, M = # of CB's (usually same as # opt buckets) K = maximum performance degradation factor;
Output: An M partition of the dataset (i.e., M buckets)

Compute optimal buckets $\{B_i, \ldots, B_M\}$ using QOB algorithm;
Initialize M empty composite buckets $CB_1 \ldots, CB_M$;
for each B_i **do**
  Select $d_i = \frac{K*|B_i|}{f_{CB}}$ distinct CB's randomly, $f_{CB} = \frac{|D|}{M}$;
  Assign elements of B_i **equiprobably** to the $d_i CB$'s;
  /* (roughly $|B_i|/d_i$ elements of B_i go into each CB) */
end for
Return the set buckets $\{CB_j | j = 1, \ldots, M\}$;

4.2.3 Example

Figure 5 illustrates the diffusion process using a single-dimensional dataset. Another, multidimensional version of

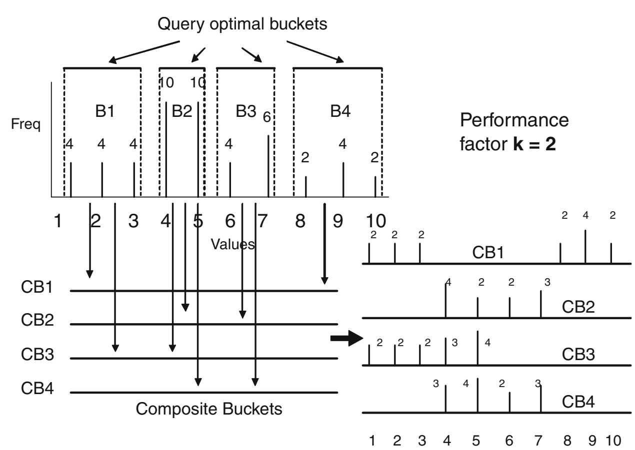

Fig. 5 Controlled diffusion

this example is shown in the extended version of the paper [28]. Consider the optimal buckets shown in the figure and say a performance degradation of up to 2 times the optimal ($K = 2$) is allowed. In the figure, the vertical arrows show which of the composite buckets, the elements of an optimal bucket gets assigned to (i.e., diffused to). The final resulting buckets are shown in the bottom right-hand side of the figure, and we can see that all the 4 CB's roughly have the same size (between 11 and 14). The average entropy of a bucket increases from 1.264 to 2.052 and standard deviation increases from 0.628 to 1.875 as one goes from the B's to CB's. In this example, the entropy increases since the number of distinct elements in the CB's are more than those in the B's. The variance of the CB's is also higher (on an average) than that of the B's since the domain (or spread) of each bucket has increased. Note that average precision of the queries (using the composite buckets) remains within a factor of 2 of the optimal. For instance, take the range query $q = [2, 4]$, it would have retrieved the buckets B_1 and B_2 had we used the optimal buckets resulting in a precision of $18/32 = 0.5625$. Now evaluating the same query using the composite buckets, we would end up retrieving all the buckets CB_1 through CB_4 with the reduced precision as $18/50 \approx 0.36 > \frac{1}{2} * 0.5625$. (Note: Due to the small error margin allowed in the size of the composite buckets (i.e., they need not be exactly equal in size), the precision of few of the queries might reduce by a factor slightly greater than K).

We will now see the performance of the bucketization algorithms in the next section.

5 Experiments

We primarily carried out experiments to (i) clearly illustrate the benefit of bucketization over a naive strategy that retrieves all records from the server for every query; (ii) measure the (performance) cost of query evaluation as a function of number of buckets used; (iii) measure the disclosure risk due to bucketization as a function of number of buckets used; (iv) gain insight into the trade-off between performance cost and disclosure risk. We also compared the trade-off characteristics of our partitioning algorithm with other off-the-shelf multidimensional data partitioning/indexing techniques that could have been used in the first phase of bucketization (like R-trees, KD-trees etc.) and show that ours is better in most cases.

Now, we describe each of these in detail.

5.1 Performance: Bucketization versus naive strategy

Here, we compare the performance of the proposed multi-dimensional bucketization approach with the performance of a simple client-server architecture where bucketization has not been used. Suppose that a client wants to store his data on a server and query this data periodically. To be able to do that the client first encrypts the data records with an encryption algorithm such as AES [1] and stores them on a server in encrypted format. Whenever there is a query, the server simply streams all the encrypted records to the client over the network. The client then decrypts all the records and evaluates the query predicate locally against each record. Below, we describe the details of the experiments we have conducted to analyze the benefits of using bucketization approach compared to this trivial solution.

Using two different computers, we simulated the scenario described above to conduct the performance experiments. As for the server, we used an IBM x3500 which has 16GB of main memory and a 8 core Intel(R) Xeon(R) CPU E5420 @ 2.50 GHz processor running on a 32 bit Suse Linux operating system. For the client side, we used a 2.79 GHz Intel Pentium D Dell Precision with 2 GB memory running on Windows XP platform.

As for the dataset, we used the *Lineitem* table of TPC-H dataset generated with the default scale (scale 1). TPC-H is a decision support benchmark widely used in the database community to assess the performance of database management systems [83]. In the experiments, we constructed the buckets using 10K records of Lineitem table and used four different attributes for the range queries. These are *PartKey*, *ExtendedPrice*, *ShipDate*, *CommitDate*. We prepared a workload of 10K range queries where a single range predicate is defined for each attribute. The range of the predicates was selected randomly from a Gaussian distribution. As for the mean of the Gaussian distribution, we used the median value of the attributes. Standard deviation of the distribution is selected as 10 % of the range of the attribute domains.

5.1.1 Client-side processing cost

We measured the workload execution time on both the server and client side for different number of buckets. The experimental results are shown in Fig. 6. Each point in the figure corresponds to a different run of the workload. Each run was repeated 20 times and the average duration of these runs were recorded. The x axis shows the number of buckets applied to the dataset on the server side and y axis shows the total execution time of the workload on the client side which is dominated by the cost of decrypting the records. The straight line at the bottom of the plot shows the execution time of the workload when there are no false positives (less than one second). We use this as the base line for evaluation. The second line which is named as "With false positives" shows the execution time using the optimal buckets (i.e., without applying diffusion). The remaining three lines show the execution time when we apply diffusion after bucketization. The topmost

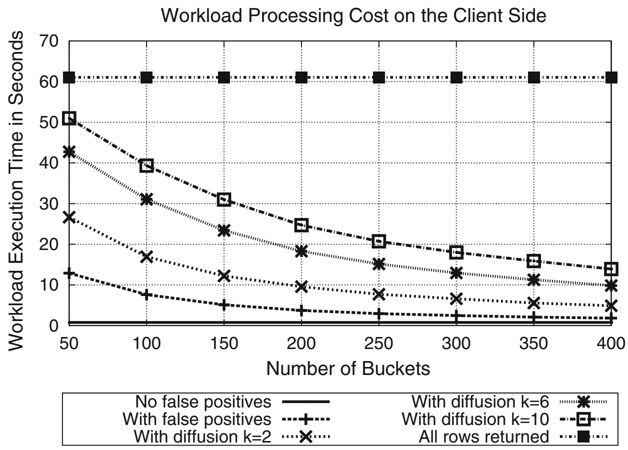

Fig. 6 Workload processing cost on the client side

straight line shows the execution time when we use the trivial idea described above (it takes 61035 milliseconds).

The experiment results show that the bucketization approach is quite effective in reducing the client-side processing cost. This cost gets smaller as larger number of buckets are used. For example, if the records are clustered into 400 buckets, the execution of the workload takes 1,818 ms which is about 33 times less than the cost of the trivial implementation. Even if diffusion is applied, the execution time is significantly less than the cost of the trivial implementation. For the diffusion factor of $K = 6$, the bucketization approach provides about 6 times improvement over the naive strategy. We decided to limit the number of buckets used in all our experiments to 400 since we found that for these many buckets, the query execution overhead was almost similar to the case where there are no false positives in most cases.

5.1.2 *Server-side processing cost*

We also measured the workload execution time on the server side and observed that the execution of the workload takes about 2,665 ms. For many different values of the diffusion factor k, the client-side processing cost is greater than the server-side processing cost. Based on this observation, we can claim that the decryption cost on the client side constitutes the bottleneck during query processing. Hence, bucketization provides significant improvement over the trivial approach.

Next, we describe the results of our experiments using multidimensional datasets.

5.2 Performance, disclosure and trade-off characteristics of bucketization

We used three types of datasets to perform the multidimensional experiments. We used the "Co-occurrence Texture" table of the "Corel Image" dataset in UCI-KDD archive [46] as a sample real-world dataset. The dataset includes 16 attributes where the attribute values have a normal distribution. Since many real-world data follow the normal distribution, we believe that this dataset would be a good choice for real dataset experiments. We illustrate the trade-off characteristics using one of these six attributes.

We have also conducted some experiments with the TPC-H dataset. We used 10K records of the *Lineitem* table of TPC-H dataset generated with the default scale factor (scale 1). For the multidimensional columns, we used *PartKey*, *ExtendedPrice*, *ShipDate*, *CommitDate* attributes.

As for the synthetic dataset, 10^4 integer values are generated by sampling uniformly at random from the domain [0,999].

We generated four different workloads to conduct the precision experiments. The first two workloads, Q_{syn} and Q_{real}, correspond to synthetic and real datasets. Each of these sets has 10K queries. The range of each query in Q_{syn} and Q_{real} was selected randomly from a uniform distribution which has the same range as the datasets. The third and fourth workloads, each consisting of 10K queries, were prepared for the TPC-H experiments. The range of the queries in these workloads was selected randomly from a Gaussian and uniform distribution respectively. As for the mean of the Gaussian distribution, we used the median value of the attributes. Standard deviation of the distribution is selected as 10% of the range of the attribute domains.

Four types of experiments were carried out to evaluate the performance of the proposed algorithms. Below we present the results of experiments conducted for a 4-dimensional dataset only. Experiment results for data with other dimensionality are given in the extended version [28].

1. **Precision:** In this experiment, we measure the decrease in precision of evaluating the benchmark queries using optimal buckets(QOB-buckets) and composite buckets (CB's). In Fig. 7, we plot the ratio of the *average precision* using optimal buckets to that using the composite buckets as a function of *# of buckets*. We use the benchmark query set Q_{syn}. Plots are shown for multiple values of the maximum allowed *performance degradation factor* $K = 2, 4, 6, 8, 10$. Figure 8 shows the corresponding plot for the real dataset on query set Q_{real}. Figures 9 and 10 show the same plot for TPC-H dataset on two different TPC-H query workloads.
2. **Privacy measure:** We plot the ratio of *variance* of the CB's to that of the QOB buckets as a function of *# of buckets* in Fig. 11 for the synthetic dataset and in Fig. 12 for the real dataset and Fig. 13 for the TPC-H dataset (for values of $K = 2, 4, 6, 8, 10$). Figure 14 plots the ratio of *average entropy* for the two sets of buckets, on the synthetic dataset. Figures 15 and 16 displays the corresponding ratio for the real and TPC-H datasets respectively.

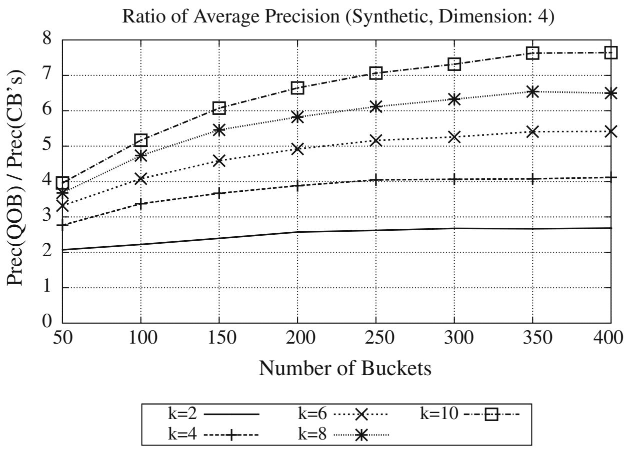

Fig. 7 Decrease in precision (precision ratio vs. number of buckets)—synthetic dataset

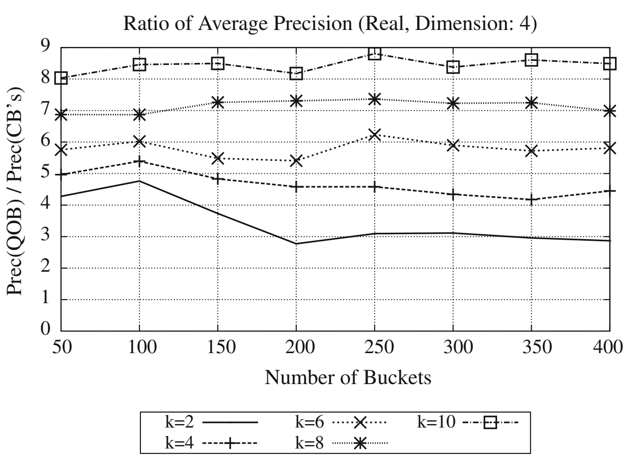

Fig. 8 Decrease in precision (precision ratio vs. number of buckets)—real dataset

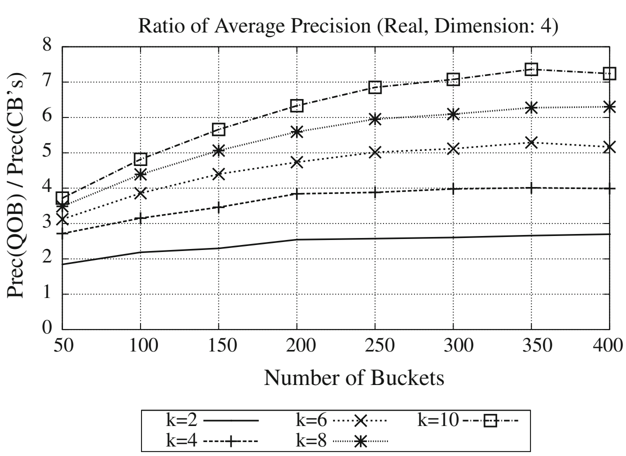

Fig. 9 Decrease in precision (precision ratio vs. number of buckets)—TPCH Dataset, Uniform Query Workload

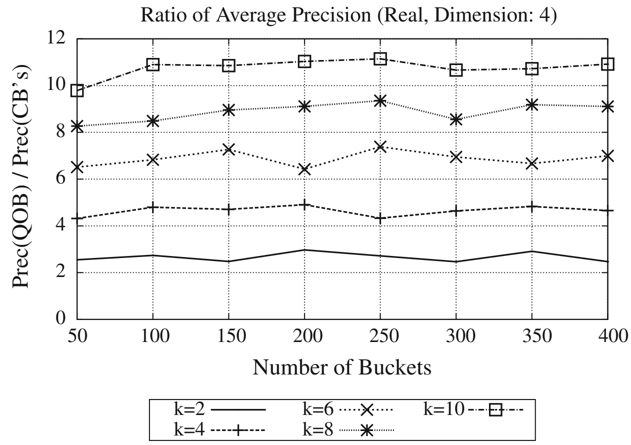

Fig. 10 Decrease in precision (precision ratio vs. number of buckets)—TPCH Dataset, Gaussian Query Workload

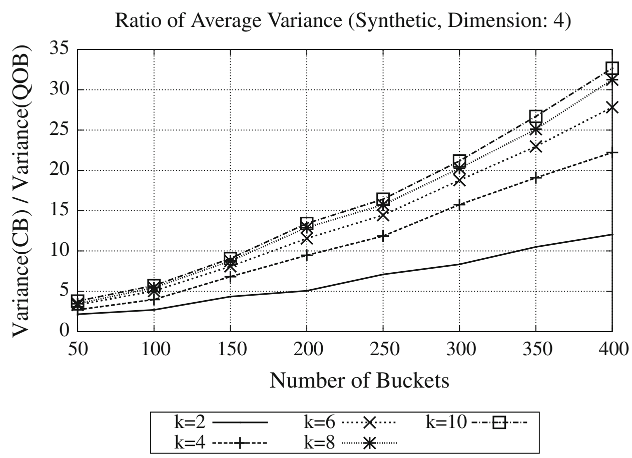

Fig. 11 Increase in variance (variance ratio vs. number of buckets)—synthetic dataset

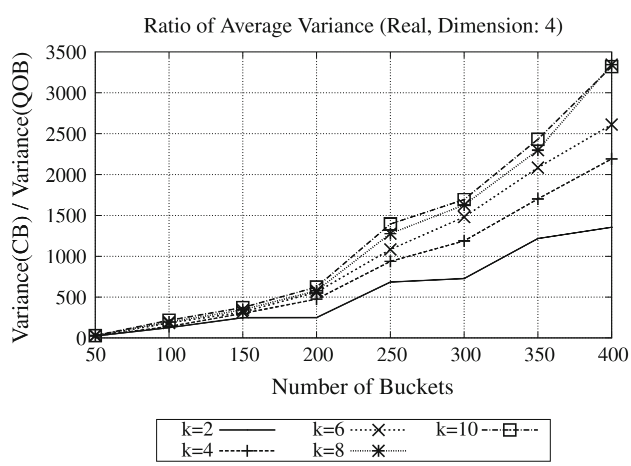

Fig. 12 Increase in variance (variance ratio vs. number of buckets)—real dataset

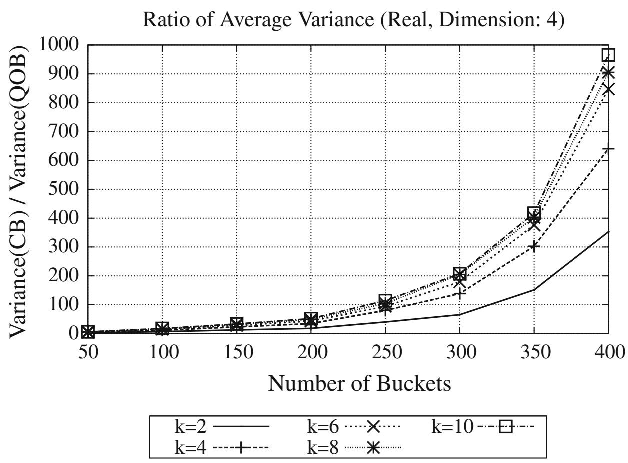

Fig. 13 Increase in variance (variance ratio vs. number of buckets)—TPCH dataset

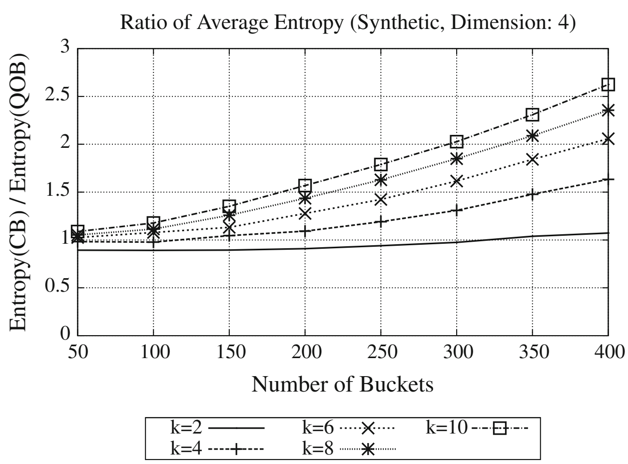

Fig. 14 Increase in entropy (entropy ratio vs. number of buckets)—synthetic dataset

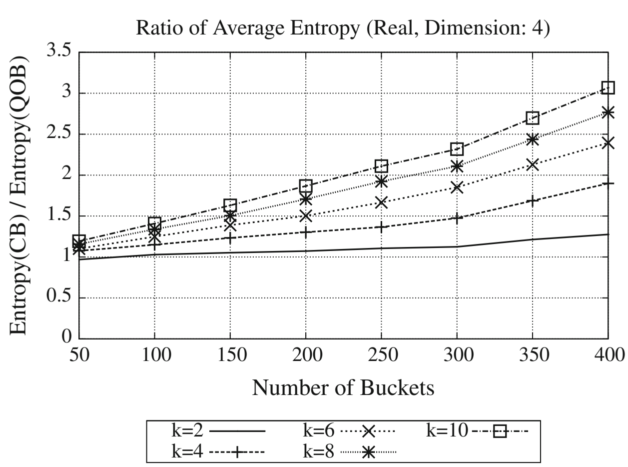

Fig. 15 Increase in entropy (entropy ratio vs. number of buckets)—real dataset

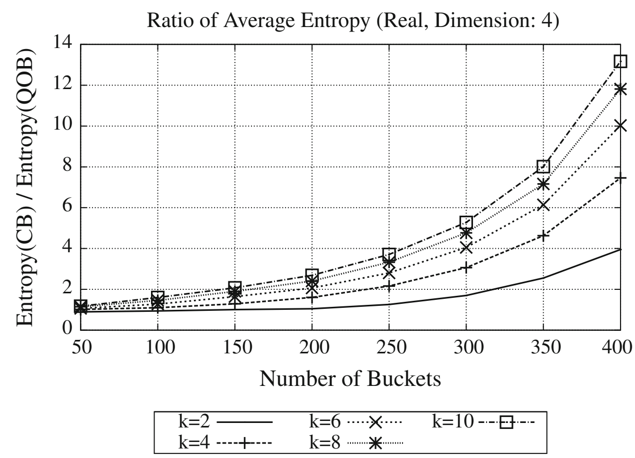

Fig. 16 Increase in entropy (entropy ratio vs. number of buckets)—TPCH dataset

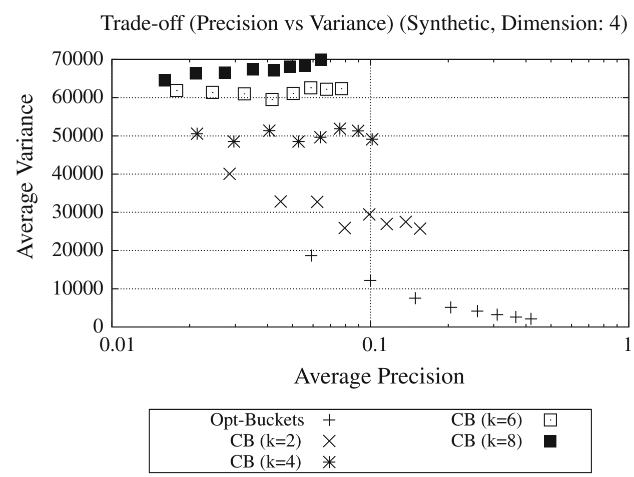

Fig. 17 Privacy-performance trade-off (variance vs. precision)—synthetic dataset

3. **Performance-privacy trade-off:** Figures 17, 18, 19, and 20 display the trade-off between *average variance* of buckets and *average precision* for the synthetic real TPC-H datasets respectively (Figs. 19 and 20 correspond to two different workloads on the TPC-H data). We run the experiment with 6 different values of M (# of buckets) $M = 100, 150, \ldots, 350$. For each M, we plot the average variance of the optimal set of buckets (QOB's) as well as the average variance for the 5 sets of composite buckets. That is, for each of 6 values of M, we apply the controlled diffusion algorithm for 5 different values of maximum degradation factor ($K = 2, 4, 6, 8, 10$). In effect, we plot 6 different points for each value of K. Similar plots of *entropy-precision trade-off* for the same sets of buckets are shown in Figs. 21, 22, 23 and 24 respectively.

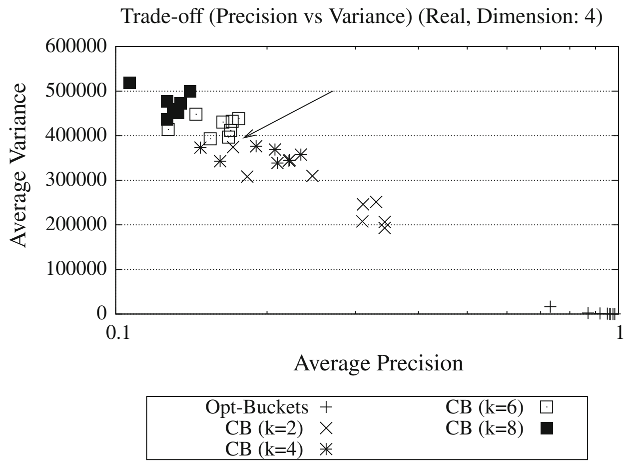

Fig. 18 Privacy-performance trade-off (variance vs. precision)—real dataset

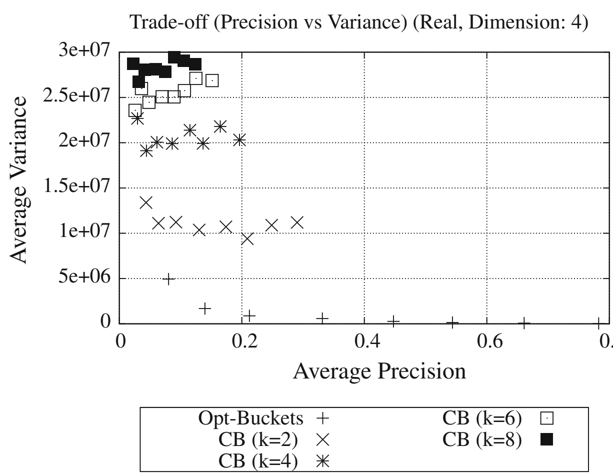

Fig. 19 Privacy-performance trade-off (variance vs. precision)—TPCH dataset, uniform query workload

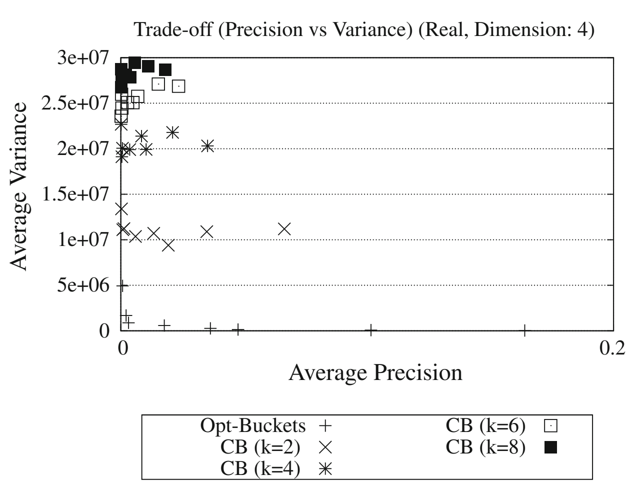

Fig. 20 Privacy-performance trade-off (variance vs. precision)—TPCH dataset, Gaussian query workload

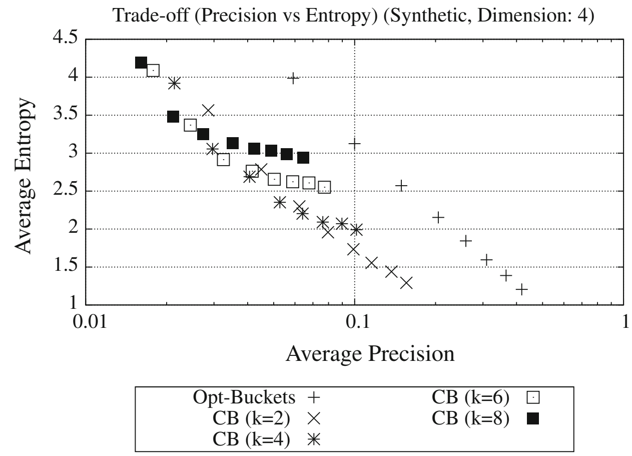

Fig. 21 Privacy-performance trade-off (entropy vs. precision)—synthetic dataset

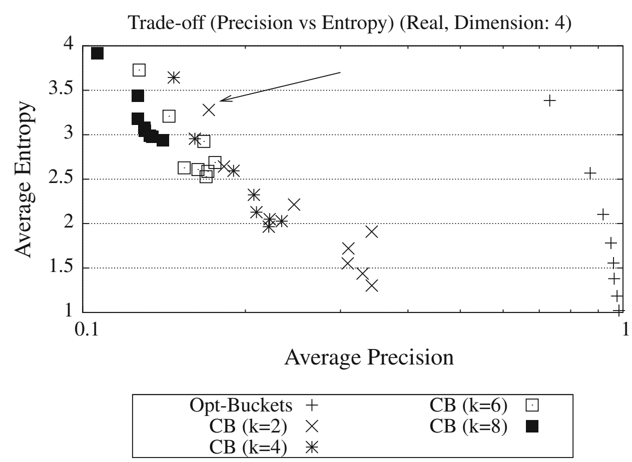

Fig. 22 Privacy-performance trade-off (entropy vs. precision)—real dataset

Figures 19 and 20 display the trade-off between *average variance* of buckets and *average precision* for the two TPC-H workloads. Similar plots of *entropy-precision trade-off* for the two TPC-H workloads are shown in Figs. 23 and 24 respectively.

5.2.1 Observations from the experiments

1. **Precision:** The relative precision measurements show that the decline in average precision is below the allowed maximum factor K particularly for higher values of K. In Fig. 7, an interesting observation is that for smaller values of K (e.g., $K = 2$) the observed degradation factor is larger than K! This is possible since the algorithm assumes that the size of the composite buckets is approximately the same. If for some reason the sizes deviate significantly from the intended (average) size, then it may lead to larger degradation than the intended

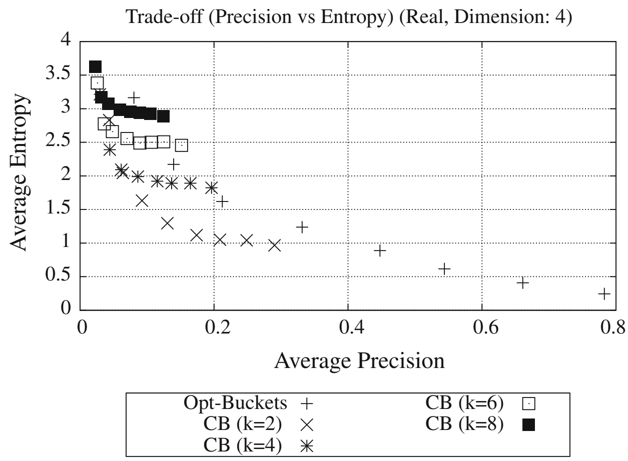

Fig. 23 Privacy-performance trade-off (entropy vs. precision)—TPCH dataset, uniform query workload

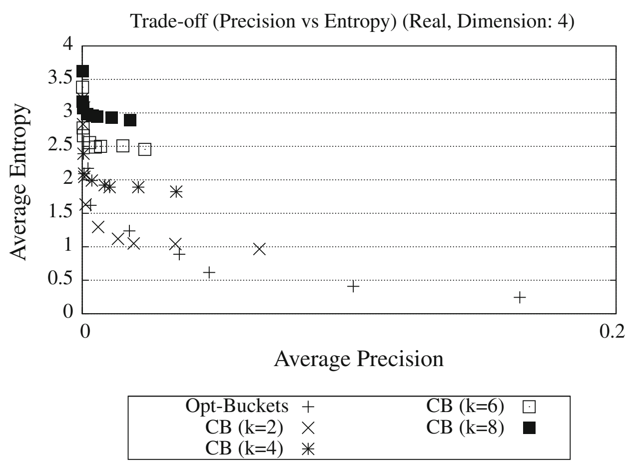

Fig. 24 Privacy-performance trade-off (entropy vs. precision)—TPCH dataset, Gaussian query workload

maximum denoted by K. This is what we suspect was the case for the smaller values of K for the synthetic dataset. Also, unlike the single-dimensional case, for the synthetic dataset we saw that relative query precision degraded as the number of buckets increased for a given value of K. This phenomenon is due to the fact that average precision was poor for the optimal case to start with (for the synthetic data) when small number of buckets were used and the relative degradation due to diffusion was not as much as that when larger number of buckets were allocated.

2. **Privacy measure:** The privacy measurement experiments show that the variance increases by a large factor in most cases even for small values of K. This is in line with the pattern we observed for single-dimension case as well. Similar is the change in values of entropy after diffusion.
3. **Performance-Privacy trade-off:** The plots in the privacy-performance trade-off space display all the **design points** available to the data owner. These plots provide a good estimate of the degree of privacy available if one is willing to sacrifice efficiency by a given amount. For example, by looking at the plots of Figs. 18 and 22, the data owner might choose a bucketizaton scheme that uses 100 buckets and sets $K = 2$ (shown by the arrows). $K = 2$ provides a high value for average entropy as well as a sufficiently high value of variance of the buckets for a acceptable loss in efficiency. The owner can also use non-integral values of K and explore more points in the trade-off space. The trade-off plots could be made more accurate by choosing an appropriate set of benchmark queries for a given application.

5.2.2 Greedy algorithm versus partitioning algorithms

In Sect. 4.1, we described a multi-dimensional data partitioning algorithm that aims to minimize the cost function derived in Sect. 3.1. We showed that minimizing the cost function in turn improves the precision of the queries and therefore improves the quality of the buckets. Instead of using the greedy algorithm if one were to use some of the well-known multidimensional data structures to partition the data, how would the performance vary? To get an answer to this question, we carried out the initial bucketization using some of the well-known data structures. We carried out some experiments where we compare these algorithms with the greedy algorithm in terms of precision achieved for multidimensional range queries. We describe the bucket formation strategies below.

1. **BSP tree (Binary Space Partitioning):** Binary Space Partitioning is a method for dividing a multidimensional space into sub-spaces using hyperplanes. The generated sub-spaces can be represented as a tree data structure known as a BSP tree (See [42] for more details). In our implementation, the sub-spaces are partitioned with axis-orthogonal hyperplanes. The axis is chosen based on depth of the tree node which is being split. Specifically, if the depth of node n is denoted Depth(n), the splitting hyperplane is orthogonal to axis Depth(n) mod d + 1 and splits the space into two equal halves. The splitting operations are performed until obtaining the desired number of leaf nodes. Once the splitting operations are completed, the buckets are created using the data points in each leaf node.
2. **Kd-tree:** Kd-tree is a data structure used for multidimensional space partitioning. The construction of kd-tree is very similar to that of BSP tree. The root node covers the entire space. At each step, a splitting dimension and

a split value from the range of the current node on that dimension are chosen heuristically to divide the space into sub-spaces. The algorithm halts when pre-defined requirements (such as tree height or number of elements covered by a node) are met (see [42] for more details). In our implementation, the sub-spaces are partitioned with axis-orthogonal hyperplanes and the axis is chosen based on the depth of the tree node as done in the BSP construction. For a given partition, the median point is chosen as the splitting point. As in the BSP construction, the splitting operations are performed until we obtain the desired number of buckets.

3. **R-tree and its variants:** R-tree is a data structure used to index multi-dimensional data. At each iteration, the data structure creates possibly overlapping MBRs containing the data points. Each non-leaf node keeps a minimum boundary box of all data points within this child node. There are numerous variants of R-trees discussed in the literature. One of these data structures is called R*-tree which aims to minimize the volume of partitions. In addition to the original version of R-trees, we have conducted experiments with R*-tree. For a detailed discussion on R-tree and its variants, please refer to [42].
4. **k-means:** In addition to the partitioning algorithms discussed above, we have conducted some experiments with k-means clustering algorithm which is a popular and practical algorithm. For this purpose, we used the implementation of k-means in Weka library with the default parameters [50].

To perform these experiments, we used the datasets described in Sect. 5.2. Further we generated a third dataset similar to the synthetic dataset except that the data points are selected from a standard normal distribution rather than a uniform distribution.

The experiment results for only dimension 4 are given in Figs. 25, 26 and 27. The experiment results for the other dimensions are given in the extended version. These results imply that the precision of the buckets generated using our greedy multidimensional bucketization algorithm is much higher than those generated with other algorithms for the real dataset and the synthetic dataset with a normal distribution. At the same time, for the special case when the data are known to be uniformly distributed, our experiments show that many existing space and data partitioning algorithms may perform comparable or better than the greedy algorithm and therefore should be used instead. However, since most real datasets are unlikely to be uniformly distributed, the greedy algorithm should be the algorithm of choice in the majority of cases. The results also reiterate the fact that using the cost measure in the optimization pays off, and in a big way in many cases, as compared to using an off-the-shelf multidimensional data structure.

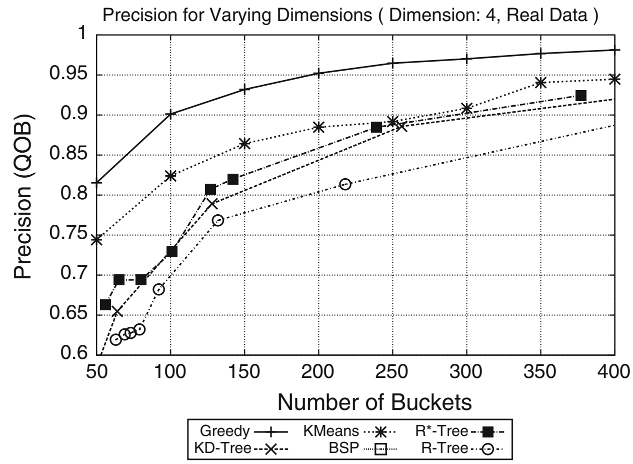

Fig. 25 Precision comparison (precision vs. number of buckets)—real dataset

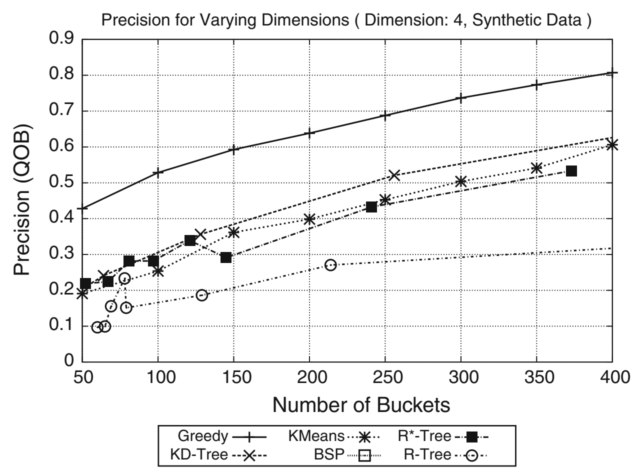

Fig. 26 Precision comparison (precision vs. number of buckets)—synthetic dataset normal distribution

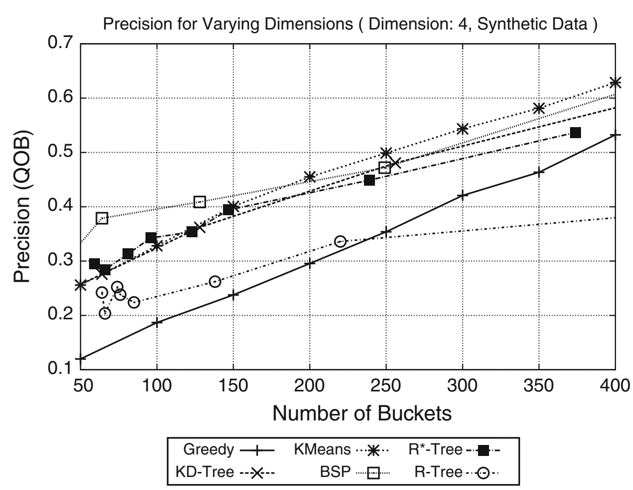

Fig. 27 Precision comparison (precision vs. number of buckets)—synthetic dataset uniform distribution

We also conducted some experiments to compare these algorithms in terms of variance and entropy. These results are shown in Figs. 28, 29, 30, 31, 32 and 33. We notice that while the precision achieved by the optimal buckets is greater than all other data structures (except for the uniformly distributed dataset), the variance and entropy lies somewhere in the middle. However, this is to be expected, since the first phase of the bucketization process only optimizes for performance and does not take into account the entropy and variance measures of buckets.

In these experiments, we used numerical attributes for each dimension. However, our methods extend easily to schemas containing categorical attributes as well. As a preprocessing step, such attributes can be converted to numerical attributes by assigning unique values to every category.

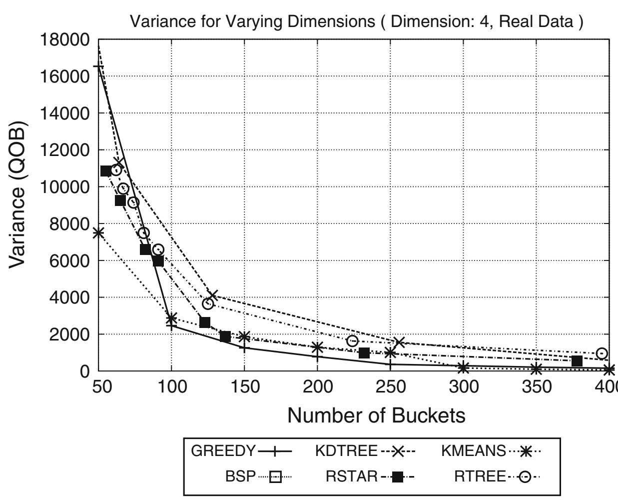

Fig. 28 Variance comparison (variance vs. number of buckets)—real dataset

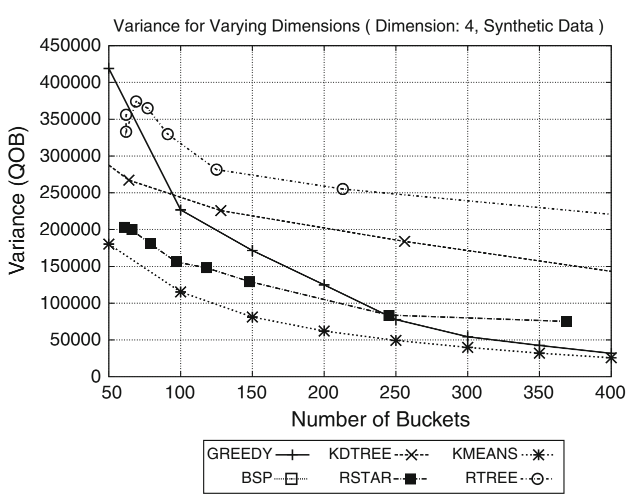

Fig. 29 Variance comparison (variance vs. number of buckets)—synthetic dataset normal distribution

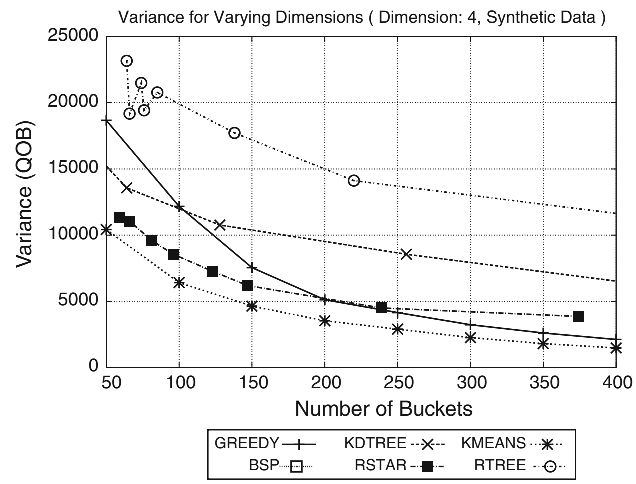

Fig. 30 Variance comparison (variance vs. number of buckets)—synthetic dataset uniform distribution

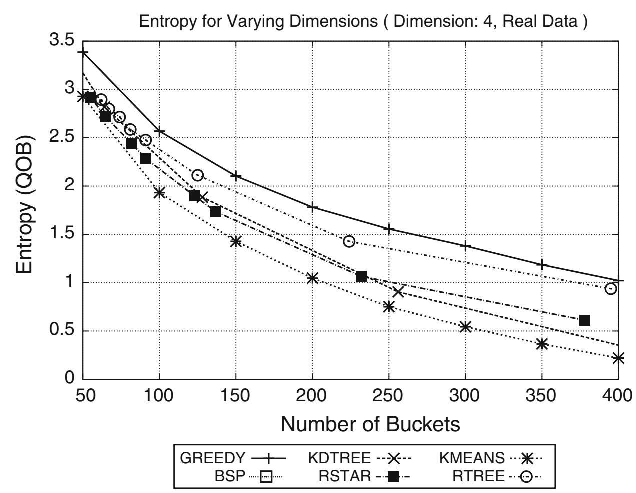

Fig. 31 Entropy comparison (Entropy vs. number of buckets)—real dataset

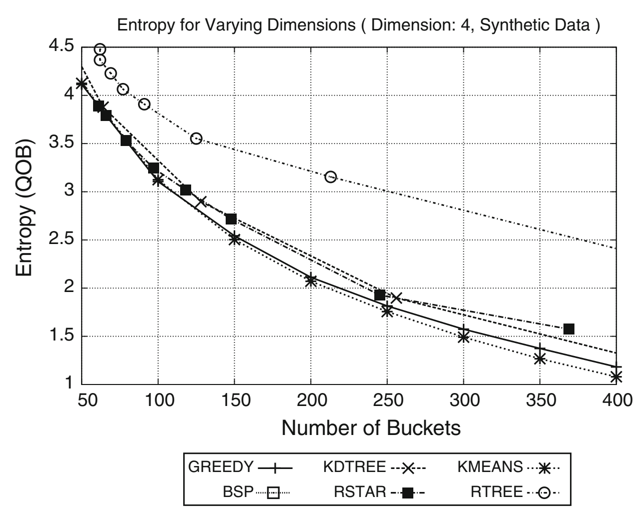

Fig. 32 Entropy comparison (entropy vs. number of buckets)—synthetic dataset normal distribution

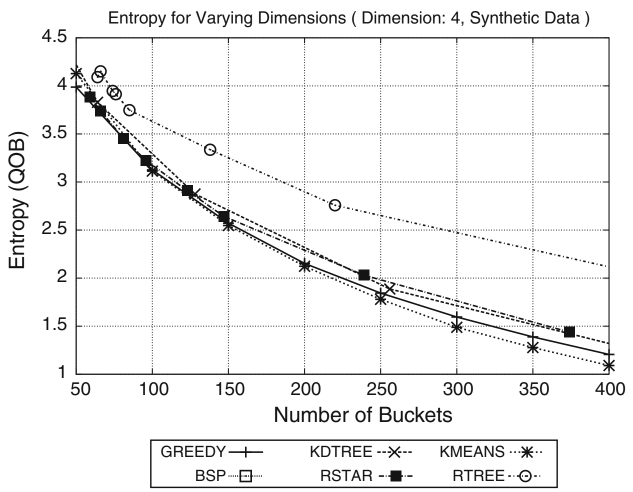

Fig. 33 Entropy comparison (entropy vs. number of Buckets)—synthetic dataset uniform distribution

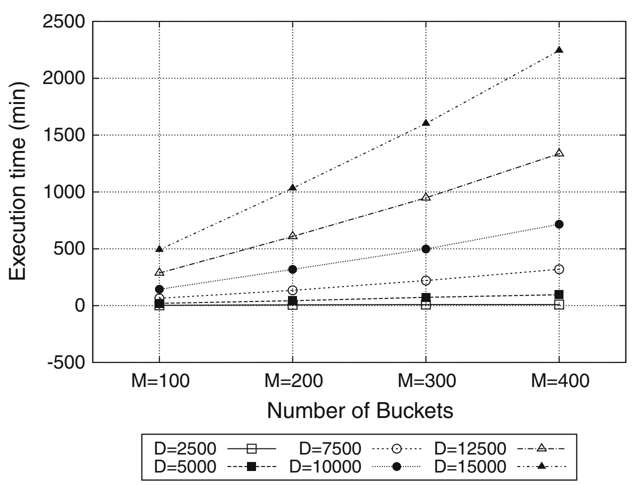

Fig. 34 Greedy algorithm execution time versus number of buckets

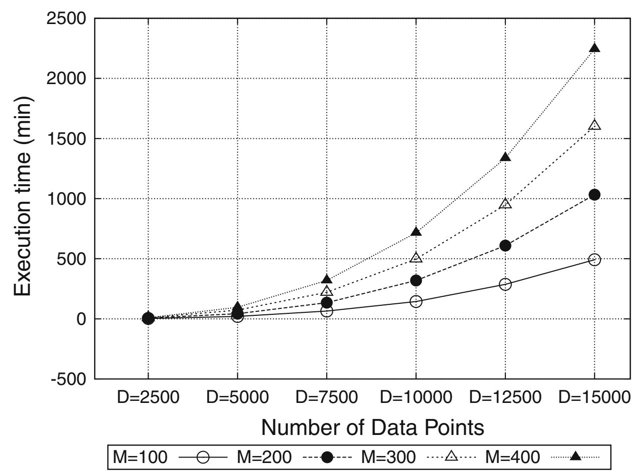

Fig. 35 Greedy algorithm execution time versus size of data

5.3 Timing experiments

In this section, we present the experiment results where we measure the time spent for the partitioning of the datasets. We have conducted some experiments with different number of buckets and data points. The results are displayed in Figs. 34 and 35. As we mentioned earlier, a worst-case analysis for the straightforward implementation of the algorithm has a time complexity of $O(M|D|^3)$, where M is the number of buckets to be created and $|D|$ is the size of the dataset. As shown in Fig. 34, the total execution time increases linearly as the number of buckets increases. Also in Fig. 35, we show the increase in the execution time as the number of data points increases in the dataset.

These experiments reinstate the fact that the greedy algorithm has a high time complexity. However, the focus in this paper has not been so much about scalability as it has been to derive a heuristic that results in high quality buckets, i.e., those resulting in relatively low overhead of false positives. In this regard, we have been able to clearly show that the proposed greedy algorithms results in better buckets than other standard multidimensional data structures. Regarding scalability of our algorithm, we outline an approach in the next section.

6 Practical considerations and future work

There are various issues to address in making the proposed scheme more practical. The first major issue is how should a data owner decide the value of M which is required as an input to the bucketization algorithm. We note that increase the value of M lowers cost (improves performance) but also lowers security. One strategy could be to let the user specify the maximum cost overhead he is willing to tolerate as well as the minimum security he requires (he needs to specify minimum values of both average entropy and variance of buckets). Since the cost function and the two privacy metrics vary monotonically with M, one can simply employ a binary search mechanism to identify the feasible values of M which meet both criteria. If privacy is more important one can use the minimum value of M in the feasible region, since that tends to offer the maximum privacy while meeting the performance constraints. Alternatively, one may combine the cost and disclosure measures into a joint (weighted) measure and pose it as a minimization problem. The goal would be to search for the value of M that minimizes this joint measure.

In general, designing appropriate mechanisms to set the correct values for maximum cost overhead and minimum security is an important problem which has implications on usability of the proposed system. As mentioned above, the data owner can start with some default values for the two parameters M and the *diffusion factor*, say those which achieve high degree of privacy (i.e., high values for variance and entropy) and then adjust the parameter(s) as required to achieve a more desirable trade-off between performance and privacy. This would require re-bucketization of the dataset of course.

Another issue of concern is measuring how secure we are. Our security model is geared toward estimating the worst-case disclosure risk under the attack model mentioned in Sect. 3.2. In that respect, it is more akin to information theoretic security than computational security. It is therefore not conducive for security analysis where specific attacks are proposed and schemes are shown to be safe against them for a computationally bounded adversary (i.e., polynomial time adversary). Here, the worst-case exposure corresponds to the situation when the adversary has learnt the complete distribution for a bucket, i.e., he knows the set of values along with their frequencies for the bucket. Now, given any random encrypted record from this bucket, he can restrict (guess) a sensitive attribute's value to within a certain interval with a certain probability of error. From the security stand point, a bucket is more robust if the error in the adversary's estimate is large on an average. We show that this is the case when both entropy and variance of the value distribution within the bucket is large. However, it is not easy to determine whether a partitioning of the data exists, such that a specified (high) value can be attained for the average entropy and average variance. Hence, we simply try to increase these measures in a relative sense, i.e., in comparison to the optimal buckets. We show via experiments that indeed, one can enhance these measures substantially via the diffusion mechanism.

Another important issue is that of rebucketization—when to rebucketize the data? The performance of a bucketization scheme is dependent upon the data distribution. Therefore, for large datasets where distribution of values is mostly stable, we expect that rebucketization will be required only infrequently. In such cases, a trivial solution is to carry out the bucketization periodically at a secure location such as the client. However, if the client is computationally constrained, such as a mobile device, it can periodically access a server with sufficient computation power and storage which is also secure. Of course this is not an ideal solution which works in all cases. In other cases, one would possibly want a dynamic scheme where bucketization follows the data distribution more faithfully and remains optimal. This problem, however, is out of the scope of the current paper and would be a very interesting direction for future extensions.

Regarding scaling our bucketization algorithm to large datasets, a sampling-based approach can potentially be employed. The basic approach would be to create the buckets using a random sample from the underlying dataset and then assign the remaining points to a bucket such that the cost increase is minimal in the process. This process may require bucket extents to be enlarged in order to cover all points. We do not explore such a sampling-based algorithm in this paper, and again, this remains an important piece of future work.

7 Related work

The problem of querying encrypted data has been deeply researched in both cryptography and database communities [2–4, 6, 8–10, 14, 16, 20, 23–25, 29, 43, 44]. The specific scheme that may be applicable depends on the data type and the type of query that needs to be supported. We first describe some of the techniques proposed specifically for evaluating range queries.

7.1 Order-preserving encryption

An ideal cryptographic scheme for supporting range queries would be one that allows range-predicate evaluation over encrypted data directly. Agrawal et al. [4] were the first ones to propose an order-preserving encryption scheme for numeric data. Such a scheme allows the server to execute range queries directly on the encrypted representation as well as maintain indexes which would enable efficient access of the encrypted data. Recently, Boldyreva et al. [6] gave a formal security analysis and a relatively efficient implementation of such an order-preserving encryption scheme. The disadvantage of order-preserving schemes is that the encryption needs to be deterministic (i.e., all encryption of a given plaintext should be identical). As a result, it reveals the frequency of each distinct value in the dataset and hence susceptible to statistical attacks. Both these techniques suffer from the problem that the adversary is able to progressively improve his estimate of the true value of a ciphertext by evaluating more predicates against the ciphertext (i.e., as number of queries increase). For example, consider two queries $q_1 = [50, 100]$ and $q_2 = [70, 120]$. If the same element x is retrieved for both the queries, the adversary is able to tell that $x \in [70, 100]$ which is better than what he learns from either one of the queries in isolation. We will refer to this as the "value-localization" problem.

7.2 Pattern hiding by oblivious data structure traversal

In a recent paper by Vimercati et al. [12], they create bucket based on the B+tree index. In order to hide the access patterns, they run fake queries. In addition, they buffer the first few levels of the tree on the client side. Finally, they shuffle from time to time. However, this technique while being related is trying to solve a different problem—outsourcing a B+tree securely and obliviously traversing this data structure to retrieve data. While our technique tries to prevent approximate leakage under certain attack scenarios (e.g., when the adversary may learn the distribution of values within a bucket), this paper does not analyze the security under any specific attacks. Also, how the technique will scale with dimensions or be applied to multidimensional data structures remains unclear at this point. However, practical oblivious data structure traversals techniques could potentially be integrated into a secure data outsourcing framework.

7.3 Cryptographic techniques for range queries

More recently [9,43] have proposed techniques for evaluating range predicates directly over encrypted data. For example, Shi et al. in [43] propose a searchable encryption scheme that supports multidimensional range queries over encrypted data (MRQED). The technique utilizes an interval tree structure to form a hierarchical representation of intervals along each dimension and stores multiple ciphertexts corresponding to a single data value on the server (one corresponding to each level of the interval tree). Consider the case when the scheme is applied to a single-dimensional dataset with values belonging to a domain of size T. The scheme not only generates a ciphertext representation that is O(log T) times the actual dataset (due to multiple encrypted representation), each search query has a complexity of O(N log T) where N is the size of the dataset and T is the size of the domain. The performance only gets worse as the dimension increases—each query requires an O(dN log T) time to execute. The scheme is proven secure under the selective identity security model [7] similar to that used for identity-based encryption techniques. Authors in [34] present a binary string-based encoding of ranges. Similar to [43], they show that they can represent each range as a set of prefixes (this is same as saying that a set of nodes in the interval tree can represent a range as done in [43]). Then, using the bit-flipping-based encryption scheme of [52] they modify the encoding function (i.e., flip the bits corresponding to a subset of branches in the interval tree). This modified encoding function is called the ciphertext tree which is then used for encoding the numeric values and also translating a range query into appropriately represented set of prefixes.

There are severe performance limitations in the techniques described above—for instance public-key encryption-based schemes like [9] are substantially slower than symmetric key encryption algorithms. Further, neither one of these techniques admit any form of indexing for faster access. This leads to inefficient access mechanism as the server has to check the query predicate against each record, i.e., make a complete scan of the whole table. This could be prohibitively expensive for large tables. Additionally, like the order-preserving encryption scheme, the techniques of [9,43] are also susceptible to the value-localization problem described above.

7.4 Related query types on encrypted multidimensional data

There are a few other encrypted query mechanisms that have been proposed for related problems to ours. The authors in [51] address the problem of nearest neighbor search over encrypted multidimensional data. They develop a specialized encryption technique to estimate distances between an encrypted query point and a set of encrypted database points. However, their technique is not directly useful for our problem since there is no easy mechanism to represent a typical range predicate as a nearest neighbor query. A more severe restriction is that they have to scan all data to evaluate a k-nearest-neighbor query which would prove to be prohibitively expensive for large datasets. A more recent work on secure outsourcing of spatial datasets was studied in [53] where the goal is to support spatial (2-dimensional) predicate-based retrieval of data. They specifically protect the database from an attacker who tries to use location-based information to attack the dataset. Their technique involves mapping a 2-dimensional point to another 2-dimensional point by applying a linear transformation along with noise addition. The original space is first partitioned into disjoint buckets and subsequently, each bin is transformed independently to mimic a target distribution. While the authors outline an interesting approach which seems to work for data in lower dimensions, it is not apparent if this can be applied to high dimension data or heterogenous datasets easily. Also, one has to be able to provide a target distribution of points for the linear transformation parameters to be computed. Also, since the goal in the paper is to support nearest neighbor queries, their approach tends to preserve more locality information than ours. This in turn implies that the adversary will be better able to localize neighbors of a point whose true value has been compromised. Our scheme tries to protect against worst-case attacks which is not specific to any particular attack algorithm. In contrast, the scheme in [53] is geared toward protection from a specific kind of attack where the adversary employs a linear least-squares fitting-based algorithm to reconstruct the original points.

A more recent work involving homomorphic encryption can potentially evaluate any function directly on the encrypted data [18]. The proposed approach is not very practical and its technical details are out of the scope of the current paper. A good overview of the approach is described in a recent CACM article [19].

7.5 Data partitioning problems

Many optimization problems in computer science and related areas can be modeled as a multidimensional partitioning problem, e.g., load balancing in parallel computing applications [31], histogram construction for selectivity estimation in query optimizers [39,40], data anonymization for privacy-preserving data publication [5,27,33] etc. to name a few. In a typical setting, the partitioning problem consists of set of data elements along with an objective function and a set of constraints. The data element is represented as points in some suitably chosen d-dimensional space and the optimization goal is to partition this set into a number of bins[10] such that the value of the objective function is minimized (maximized) and at the same time, all the constraints are met. The objective function is often an aggregate function over the resulting bins. For instance, in the problem of multidimensional histogram creation, if data elements are associated with a weight equal to their frequency in the dataset, the objective is to minimize the total sum of square of deviations from the mean value in each bin. Such a partition tends to make within-bin distribution more uniform and leads to high-quality histograms for density estimation problems. The constraints define the subset of partitioning schemes that are admissible (feasible) for the given application. For instance, space constraints might require the number of total bins to be less than some number or the weight of each bin to be less than some maximum weight bound etc.

The first phase in our bucketization algorithm is nothing but a multidimensional data partitioning algorithm like many previously proposed histogram techniques [39,40]. However, the similarity is rather deceptive, since histograms are meant for selectivity estimation of query predicates while ours are meant for reducing the overhead of false positives in the bucketized query evaluation approach. These two objectives can potentially result in very different partitions. For instance, take a uniformly distributed multidimensional dataset. For selectivity estimation, using a single bucket would be as good as using all of the M buckets. However, for our problem, using larger number of buckets would lead to greater reduction in the number of false-positive retrievals for any underlying data distribution. Therefore, the optimal answers in the two cases could deviate arbitrarily. Also, techniques to map the multidimensional data first to a one-dimensional space by imposing say an Hilbert ordering [26] will most probably not work very well either, for the same reason that they do not works very well for selectivity estimation for multidimensional predicates, because the mappings tend not to be locality preserving [39]. Nonetheless, it would be interesting to empirically compare the performance of the standard histogram techniques with ours, especially with newer approaches like [17] which combine ideas of multidimensional data structures with Hilbert ordering to linearize multidimensional distributions. However, but we do not carry out such comparison in this paper and this remains a direction for future extensions.

Whichever formulation one chooses, whether to set the cost as a constraint and the security metrics as objective functions or vice versa, we found that the resulting optimization function is very distinct from any of the existing multidimensional data/space partitioning problems we are aware of. As a result, we decided to design partitioning techniques from scratch rather than try and adapt existing histogram techniques.

8 Conclusion

In this paper, we studied the problem of securely executing range queries over outsourced data. We looked at multidimensional data and range queries. The quality metric is captured using cost metrics that estimate the expected number of false-positive matches given a bucketized representation. We show that this number is proportional to the number of points (data items) allocated to a bucket times the sum of edge lengths of the bucket. Therefore, our quality objective is to keep this measure as small as possible. Given the bucketized view of the data, the best that an adversary can do is to learn a distribution over the value range of the sensitive attribute. In other words, bucketization induces an uncertainty regarding the true value of the each record by inducing a probability distribution over the value of data. Therefore, if we can ensure that this distribution satisfies some "safety" properties we can provide better protection against adversarial estimation. We identify two distributional properties—*variance* and *entropy* of the bucket value distribution as measures of disclosure risk. Higher the value of these two metrics, lower is the worst-case risk that an adversary will learn the true value of a data item given the distribution. In contrast, for higher values of variance and entropy the number of false-positive retrievals tend to increase for queries. We propose a trade-off algorithm called "controlled diffusion (CDf)" where we take query optimal buckets and redistribute them in a controlled manner to generate new buckets with value distributions having substantially greater entropy and variance than in the optimal buckets but are still able to guarantee bounds on the query precision within a small factor of the optimal. Finally, we carried out extensive experimentation to test the quality and performance of our optimal partitioning and diffusion algorithms. Below, we outline some directions of future work and present some generic discussion.

8.1 A principled approach to trade-off exploration

Since the CDf-algorithm is distribution agnostic, it is not necessarily the optimal approach to exploring such a trade-off. The intuition is that we might be able achieve better

[10] In many problems the number of bins might not be specified directly, but instead derived from other constraints specific to the application at hand.

results (i.e., further lower disclosure risk while paying smaller cost) by designing an algorithm that takes the nature of correlations between the bucket labels, sensitive attribute and the other attributes into consideration.

8.2 Background information of adversary

We also need to take into consideration the domain knowledge that might be available to the adversary which may be combined with the exposed information to better estimate the value of the sensitive attribute(s). One way to model the adversary is like a powerful *learning machine* who tries to learn the association (approximate joint-probability distribution) between non-sensitive and sensitive attributes as accurately as possible. An exposure scenario may be as follows—Say the dataset is horizontally partitioned across a set of disks and the contents of one of the disks gets partially exposed due to a security breach e.g., hacker attack. For example, in the table shown in Fig. 2 one or more attributes from the set {age, salary, credit_rating} might get exposed for some of the records. Once the credit_rating (the sensitive attribute) has been exposed for a record it can be utilized by the adversary as a part of a training set to learn the association between age, salary, bucket label and credit_rating. Let us denote the corresponding learning model as "{*age, salary, bucket label*} $\rightarrow$ *credit_rating*". In this case, the adversary could learn associations of the following form "(age = 30, sex = M, salary < 40,000) $\rightarrow$ (credit_rating = BAD)", "(age = 25, sex = F, salary > 30,000) $\rightarrow$ (credit_rating = GOOD)" etc. with some level of confidence. In fact, the dependent attribute *bucket label* is always available to the adversary, therefore we can assume that his background knowledge comprises rules of the form $\{\mathfrak{F}, B\} \rightarrow S$, where $\mathfrak{F}$ consists of 0 or more of the non-sensitive attributes. This background knowledge allows him to predict the value of a record's sensitive attribute.

8.3 Heterogenous data

The algorithms proposed in Sect. 4 works best when all the dimensions used are numeric or there is a clear notion of a distance metric associated between two tuples in the dataset. Real datasets, more often than not, have mixed attributes: numeric as well as categorical. Categorical attributes may further have taxonomies defined on them where distance metrics based on these taxonomies might be considered more appropriate rather than imposing some arbitrary order to the categorical values. In case of such heterogenous data, the algorithm proposed above might not give the best results. In another paper [27], we develop a hierarchical data partitioning algorithm for heterogenous data. The partitioning scheme (known as a *guillotine* partitioning) yields non-overlapping buckets. The enumeration-based minimization algorithm developed in [27] was primarily developed to search for optimal partitioning of such heterogenous datasets to implement a variety of privacy constraints like k-anonymity [41,45] and l-diversity [35]. The enumeration algorithm can be easily modified for the cost functions proposed in Eq. 2 above.

8.4 Alternate techniques for bucketization

One way to view this problem is in the form of a multi-objective constrained optimization problem, where the entities *minimum entropy* and *minimum variance* amongst the set of buckets are the two objective functions and the constraint is the *maximum allowed performance degradation factor* K. Most popular solution techniques for such problems seem to revolve around the *Genetic Algorithm* (GA) framework [21,30]. GA's are iterative algorithms and cannot guarantee termination in polynomial time. Further, their efficiency degrades rapidly with the increasing size of the dataset. Nonetheless, it will be interesting to see how a GA-based approach will perform and compare it against our two-phased approach.

Acknowledgments This work was made possible by grant from NSF (award number 1045296) and gift from NEC laboratories USA.

References

1. AES: Advanced Encryption Standard. FIPS 197, Computer Security Resource Center, National Institute of Standards and Technology (2001). http://en.wikipedia.org/wiki/Advanced_EncryptionStandard
2. Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Srivastava, U., Thomas, D., Xu, Y.: Two can keep a secret: a distributed architecture for secure database services. In: CIDR (2005)
3. Agrawal, R., Asonov, D., Kantarcioglu, M., Li, Y.: Sovereign joins. In: ICDE (2006)
4. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order-preserving encryption for numeric data. In: SIGMOD (2004)
5. Bayardo, R.J., Agrawal, R.: Data privacy through optimal K-anonymization. In: ICDE (2005)
6. Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: EUROCRYPT (2009)
7. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: EUROCRYPT (2004)
8. Boneh, D., Crescenzo, G., Ostrovsky, R., Persiano, G.: Public-key encryption with keyword search. In: EUROCRYPT (2004)
9. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: TCC (2007)
10. Bouganim, L., Pucheral, P.: Chip-secured data access: confidential data on untrusted servers. In: VLDB (2002)
11. Brucker, P.: On the complexity of clustering problems. In: Optimizations and Operations Research. Springer (1978)
12. Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Pelosi, G., Samarati, P.: Efficient and private access to outsourced data. In: ICDCS (2011)

13. Casella, G., Berger, R.L.: Statistical inference. Duxbury Advanced Series (2001)
14. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: ACNS (2005)
15. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, New york (1991)
16. Damiani, E., Vimercati, S., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: CCS (2003)
17. Eavis, T., Lopez, A.: Rk-hist: an r-tree based histogram for multi-dimensional selectivity estimation. In: CIKM (2007)
18. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC (2009)
19. Gentry, C.: Computing arbitrary functions of encrypted data. CACM **53**(3) (2010)
20. Goh, E.: Secure Indexes. Unpubished manuscript (2003)
21. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1988)
22. Goldreich, O.: The Foundations of Cryptography, vol. 1. Cambridge University Press, Cambridge (2001)
23. Hacıgümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in database service provider model. In: SIGMOD (2002)
24. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Providing database as a service. In: ICDE (2002)
25. Hacıgümüş, H., Iyer, B., Mehrotra, S.: Efficient execution of aggregation queries over encrypted relational databases. In: DASFAA (2004)
26. Hilbert, D.: Ueber die stetige abbildung einer line auf ein flchenstck. In: Mathematische Annalen (1891)
27. Hore, B., Jammalamadaka, R.C., Mehrotra, S.: Flexible anonymization for privacy preserving data publishing: a systematic search based approach. In: Siam Conference on Data Mining (2007)
28. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure Multidimensional Range Queries Over Outsourced Data (extended version). ICS technical report, UCI, http://www.ics.uci.edu/~bhore/papers/range-queries-in-das(journal-extended).pdf
29. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: VLDB (2004)
30. Jones, D.R., Beltramo, M.A.: Solving partitioning problems with genetic algorithms. In: Proceedings of the 4th International Conference of Genetic Algorithms (1991)
31. Khanna, S., Muthukrishnan, S., Paterson, M.: On approximating rectangle tiling and packing. In: SODA (1998)
32. Khanna, S., Muthukrishnan, S., Skiena, S.: Efficient array partitioning. In: ICALP (1997)
33. LeFevre, K., DeWitt, D., Ramakrishnan, R.: Mondrian multidimensional K-Anonymity. In: ICDE (2006)
34. Li, J., Omiecinski, E.: Efficiency and security trade-off in supporting range queries on encrypted databases. In: DBSec (2005)
35. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-Diversity: privacy beyond K-Anonymity. In: ICDE (2006)
36. Muthukrishnan, S., Poosala, V., Suel, T.: On rectangular partitionings in two dimensions: algorithms, complexity and applications. In: ICDT (1997)
37. Muthukrishnan, S., Suel, T.: Approximation algorithms for array partitioning problems. J. Algorithms **54**, 85–104 (2005)
38. Pinkas, B., Reinman, T.: Oblivious RAM revisited. In: CRYPTO (2010)
39. Poosala, V., Ioannidis, Y.: Selectivity estimation without attribute value independence assumption. In: VLDB (1997)
40. Poosala, V., Ioannidis, Y., Haas, P.J., Shekita, E.J.: Improved histograms for selectivity estimation of range predicates. In: SIGMOD (1996)
41. Samarati, P.: Protecting respondents' identities in microdata Release. IEEE TKDE **13**(6) (2001)
42. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco (2005)
43. Shi, E., Bethencourt, J., Chan, H.T.-H., Song, D.X., Perrig, A.: Multi-dimensional range query over encrypted data. In: IEEE S&P (2007)
44. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P (2000)
45. Sweeney, L.: Achieving K-anonymity privacy protection using generalization and suppression. Int J Uncertain Fuzz Knowl Base Syst (2002)
46. UCI Machine Learning Repository. http://kdd.ics.uci.edu
47. Willenborg, L., De Waal, T.: Statistical Disclosure Control in Practice. Springer, New York (1996)
48. Williams, P., Sion, R.: Usable PIR. In: NDSS (2008)
49. Williams, P., Sion, R., Carbunar, B.: Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In: CCS (2008)
50. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
51. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N.: Secure kNN computation on encrypted databases. In: SIGMOD (2009)
52. Xu, J., Fan, J., Ammar, M.H., Moon, S.B.: Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography based scheme. In: ICNP (2002)
53. Yiu, M.L., Ghinita, G., Jensen, C.S., Kalnis, P.: Enabling search services on outsourced private spatial data. VLDB J. **19**(3), 363–384 (2010)