Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 2910 – 2928
© TÜBİTAK
doi:10.3906/elk-1405-177

Authentication of uncertain data based on k-means clustering

Levent ÜNVER*, **Taflan İ. GÜNDEM**
Computer Engineering Department, Boğaziçi University, Bebek, İstanbul, Turkey

Received: 27.05.2014 • **Accepted/Published Online:** 10.01.2015 • **Final Version:** 15.04.2016

Abstract: Probabilistic databases and database outsourcing are two recent and important applications of database systems. In this paper, we introduce authenticated query processing in outsourced probabilistic databases. We have proposed a novel authenticated data structure (ADS) called PH-tree, which is a combination of PDR-tree and MH-tree. We have also implemented k-means clustering as a preprocessor. We have compared our algorithm with an existing ADS called MR-tree and showed that PH-trees outperform MR-trees significantly.

Key words: Database outsourcing, authentication, probabilistic databases, k-means clustering, hash-tree

1. Introduction

The need for uncertain data management and probabilistic queries is rising in many areas of computer applications. The most common examples are sensor networks, moving objects' tracking, data cleaning, and marketing applications [1–3]. In order to manage uncertain data, several methods have been proposed for query processing and efficient indexing [4–7].

Outsourcing databases (proposed by Hacıgümüş et al. [8]) and cloud computing are becoming popular. In this paper, we focus on outsourcing uncertain data. Authenticated query processing is central to database outsourcing. The database service provider (DSP) is responsible for the functionality of the data management as a third party and the data owner (DO) provides the data for this service. The queries are delivered from the DO side to the DSP and the DSP responds with the result of this query. While processing the query, the DSP should also deliver a verification object (VO) to the client. The client authenticates the result set with the given public key and the incoming VO. In order to achieve this functionality, the DSP stores the data in a special structure called an authenticated data structure (ADS). Figure 1 illustrates this model. The client checks for soundness and completeness with the given VO and public key. Soundness means the result set is correct and not modified. Completeness means that all the valid results are included within the result set. These two criteria are explained with the following example.

Consider that we have the sample data given in Table 1. An example of a simple query is "The tuples with over 65K income". The sound and complete result set, *RS*, is *RS* = {*(d_1, Jim, Tech-Support), (d_2, Brian, Exec-Managerial), (d_3, Nancy, Sales)*}. The result set *RS* = {*(d_1, Jim, Tech Support), (d_6, Brian, Exec-Managerial), (d_3, Sally, Sales)*} would not be sound because it contains altered data. The result set *RS* = {*(d_1, Jim, Tech Support), (d_2, Brian, Exec-Managerial)*} is not complete because it has missing data.

*Correspondence: levent.unver@boun.edu.tr

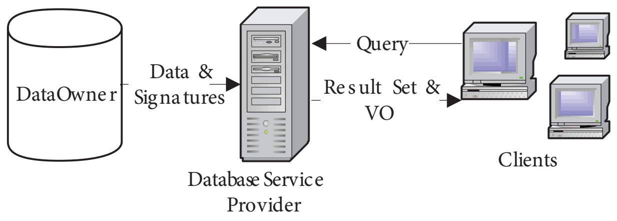

Figure 1. Authentication model in outsourced databases.

Table 1. Authenticated querying example.

Id	**Name**	**Occupation**	**Income**
d_1	Jim	Tech-support	75K
d_2	Brian	Exec-managerial	90K
d_3	Nancy	Sales	70K
d_4	Carl	Agricultural	40K
d_5	Chris	Armed forces	60K

Soundness and completeness are the basic criteria that should be provided by the DSP, but there are also some other important criteria that should be considered. These are VO size, query processing time in the DSP, and the verification time at the client side. The objective of the database outsourcing is to deliver the service to a third party, and so minimizing VO size and verification time is much more important than minimizing query processing time [9]. VO size and verification time increase the overhead at the client side.

In this paper, we propose an authentication method for probabilistic database outsourcing. For the authentication, we propose a novel authenticated data structure called a probabilistic hash-tree (PH-tree). A PH-tree is a combination of the probabilistic distribution R-tree (PDR tree) [10], designed for probabilistic databases, and the Merkle-hash tree [11]. PH-trees also benefit from a machine learning technique, clustering. K-means clustering is implemented as a preprocessor and the results are stored in the root node of the PH-tree. The K-means algorithm suits our methodology very well and greatly reduces the overhead at the client side, which is the main objective of authenticated query processing algorithms, as mentioned earlier.

The following sections are organized as follows. Section 2 gives information about the related algorithms and some preliminary work. The PH-tree is explained in detail with the pseudo codes and figures in Section 3. Section 4 contains the experiments and their corresponding results. Finally, we have the conclusions.

2. Preliminary fundamentals

2.1. Uncertain data model

There have been quite a few papers on probabilistic databases in the literature [4,12,13]. In probabilistic databases, possible worlds are generated with the given imprecise data considering the set of all outcomes. In these possible worlds, we have uncertain attributes that have probability values. The sum of all probability values of a component of a tuple associated with an attribute should be one. Table 2 presents a relational table whose *occupation* attribute is uncertain and consequently the *occupation* components of the tuples have probabilistic values.

In a probabilistic data model, a relation can have attributes that are allowed to take probabilistic values. These attributes are called *uncertain attributes* [10]. In Table 2, a sample table with an uncertain attribute in a manner similar to *Adult* dataset is illustrated [14]. Tuples (represented by rows of the table) have certain information associated with *name*, *income*, *age*, and *sex* attributes and uncertain information associated with uncertain attribute *occupation.* The values of *occupation* are estimated by prior data. A sample query is "Get the persons who highly likely work on *Sales"*.

Table 2. Uncertain data model.

Id	**Name**	**Income**	**Age**	**Sex**	**Occupation**
d_1	Jim	75K	37	male	{(Tech-Support, 0.7), (Sales, 0.3)}
d_2	Brian	90K	48	male	{(Managerial, 0.6), (Sales, 0.4)}
d_3	Nancy	70K	32	female	{(Sales, 1.0)}
d_4	Carl	40K	41	male	{(Transport-Moving, 0.3), (Agricultural, 0.7)}
d_5	Chris	60K	29	male	{(Armed-Forces, 0.8), (Tech-Support, 0.2)}

An uncertain attribute u can be defined as a probability distribution related to A, which is a discrete categorical domain representing the current values associated with the attribute u. $A = \{a_1, a_2, a_3, ..., a_M\}$, where M is the number of certain attributes in a relation scheme. The probability vector associated with attribute u is $P = <p_1, p_2, p_3, ..., p_N>$and can be defined as $u.p_i$ for a tuple in the relation, where i is the index of probability values and N represents the total number of values that u can take. Data item d_i refers to a tuple of the relation instance in this model.

Uncertainty comes from the lack of knowledge of the exact values. However, we describe the component of an uncertain attribute with exact probability values, so that the operators can be used as if they are dealing with certain attributes.

In query processing for uncertain data, we need to utilize distance functions. These functions will especially be useful when we are dealing with distributional similarities.

Manhattan distance, M, (city-block distance) between two distributions u and v with N instances can be calculated as

$$M(u,v) = \sum_{i=1}^{N} |u.p_i - v.p_i|, \tag{2.1}$$

where $u.p_i$ and $v.p_i$ are the probability values of the *ith* instance of u and v, respectively.

Euclidian distance, E, is given as follows:

$$E(u,v) = \sqrt{(\sum_{i=1}^{N}(u.p_i - -v.p_i)^2)} \tag{2.2}$$

Kullback–Leibler distance [15], *KL*, can also be used as

$$KL(u,v) = \sum_{i=1}^{N} u.p_i log(u.p_i/v.p_i) \tag{2.3}$$

These three distance functions are the ones that are used in probabilistic distribution R-trees, which are described in Section 2.2 in detail.

KL distance is based on cross-entropy measure in information theory and is useful in cluster implementation, which is described in Pereira et al.'s work [16]. We make use of KL distance in our paper. Clustering is summarized in Section 2.4.

Top-k queries are the most common type of queries in probabilistic databases. Research on the implementation of top-k query processing over uncertain data includes [7,17–19]. Note that these implementations are not identical to those of deterministic top-k query processing.

In this paper, we will be focusing on distributional similarity threshold queries (DSTQ).

Definition 1 *Distributional Similarity Threshold Query q can be defined via relation R over attribute u such that the result set of the query q returns all tuples from R that satisfies F(q, u) ≤ T, where F is a distance function and T is the given threshold value.*

In distributional similarity threshold queries, if the distributional distance between the tuple and the query is smaller than the given threshold T, then the tuple qualifies for the result of the given query. For calculating the distributional divergence, we can use any distance function that is described in this section.

2.2. PDR tree

In the literature, various indexing methods exist in the field of uncertainty. Kanagal et al. [6] proposed a structure for correlated probabilistic data based on junction trees, aka clique trees [20]. This subject has been extended to general metric spaces and multidimensions by Fabrizio et al. [21] and Zhang et al. [22]. Cheng et al. [5] and Singh et al. [10] developed other indexing methods based on traditional R-trees [23]. These trees are also called probabilistic distribution R-trees (PDR-trees) and they have query processing mechanisms that differ from those of top-k query processing. Our proposed authenticated data structure (PH-tree) is based on PDR-trees.

In a PDR-tree for an attribute, each node is stored in a memory page with closest values of the uncertain attribute in terms of distributional distance and organized as an R-tree. Additionally, new definitions and methods for minimum bounding range (MBR), the area of an MBR, and insertion criteria are defined. The method of distributional grouping is central to PDR-tree indexing. Similar to the R-tree approach, each page contains more than one uncertain attribute and items.

Figure 2 illustrates a sample PDR-tree. The symbol d_i represents the tuple ids that are contained in the leaf pages only. MBRs are formed by the probability vectors. These vectors are represented using numerical values in the figure. MBR describe the nodes of the PDR tree. For nonleaf nodes, these boundaries are determined by the vector $v = <v_1, v_2, v_3, ..., v_N>$, where v_i is the maximum probability of the items d_i in the nodes indexed in the children of the current page. For example, the leftmost internal node has MBR represented by the vector $<0.7,\ 0.3,\ 0.3>$, which is the maximum of the probabilities associated with the children nodes d_1 and d_2 (whose probability vectors are $d_1.v = <0.4,\ 0.3,\ 0.3>$, $d_2.v = <0.7,\ 0.1,\ 0.2>$, respectively). Uncertain attributes of the rest of the tree can be calculated in a similar bottom-up manner, starting from the leaf nodes going up to the root node. If the MBR does not satisfy a query for any of the internal nodes, then we can be sure that none of its children does. The area of an MBR can be calculated by using any of the distance functions described in Section 2.1.

A distributional similarity threshold query is processed, in PDR-trees, in a straightforward manner. A depth-first search is carried out via pruning the MBR boundaries. If the query is satisfied by the MBR of a node, then the subtrees of the node are also included in the search. Otherwise, the node is pruned out. When the search reaches the leaf level, the nodes that satisfy the query are included in the result.

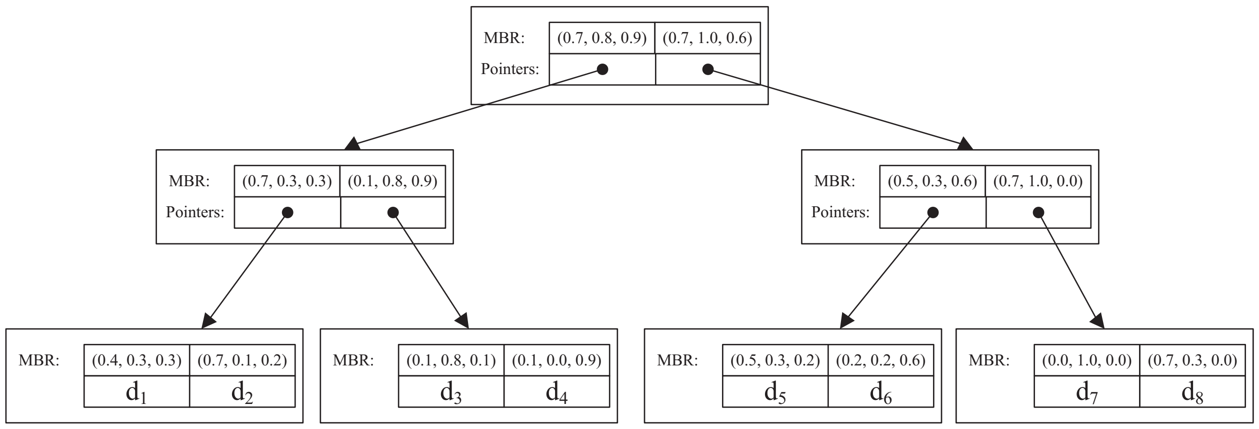

Figure 2. Sample probabilistic distribution R-tree.

2.3. Authenticated query processing

Authentication has been studied by Merkle [11] and he proposed a novel method called the Merkle-hash tree (MH-tree). The mentioned paper is considered by Yang et al. [9] as the fundamental work for a wide range of authentication methods and structures. The Merkle B-tree has been proposed by Li et al. [24]; it is a combination of MH-tree and B+tree. Authentication has been extended to the spatial data domain recently [25–27]. These papers enable the implementation of range query processing in a multiattribute environment.

Hash functions are essential to authenticated query processing. A hash function H should be easy to compute but difficult to invert. Assuming $H(x) = y$, it should be easy to compute y with the given H and x, but very difficult to compute x with given y and H [28,29]. Moreover, H should be able to compress a large input into a small and fixed output size. A hash function H should satisfy the following properties [11]:

- $H(x)$ function should be applied to any size of x.
- H function should always produce a fixed size output in order to achieve concreteness.
- Given x, it should be easy to compute $H(x)$.
- It should be computationally infeasible to find $y \neq x$ such that $H(x) = H(y)$.
- Given H and y, it should be computationally infeasible to find x for $H(x) = y$.

In order to execute authentication, hash functions are vital. It is possible to authenticate x by computing $H(x) = y$, considering the properties of hash functions described and even a large value of x should produce a small fixed output of y. This is crucial for the authentication of large amounts of data. For this reason, we make use of Tiger hashing [30]. The Tiger hash function is a strong and fast hash function that produces 192-bit fixed output regardless of the input. The design of the Tiger hash function is based on parallel lookup operations, which improve the efficiency of hardware implementation. A preimage attack has been found on Tiger hashing recently by Mendel [31], but no second preimage attacks have been discovered to date. This statement implies that the Tiger hash function is a collision-resistant hashing method [32,33].

Figure 3 illustrates a sample MH-tree. In this tree, hash values, h_i, are computed as $H(d_i)$ (i.e. $h_i = H(d_i)$. Internal node hash values are computed by concatenating the children nodes' hash values.

The concatenation operation is denoted as "|". For the given example $h_{(1-2)} = H(h_1|h_2) = H(H(d_1)|H(d_2))$ and $h_{(3-4)} = H(h_3|h_4) = H(H(d_3)|H(d_4))$. Finally the root node is constructed as $h_{(1-4)} = H(h_{(1-2)}|h_{(3-4)})$ $= H((H(h_1|h_2))|(H(h_3|h_4)))$. While constructing MH-trees, recursive functions become very beneficial.

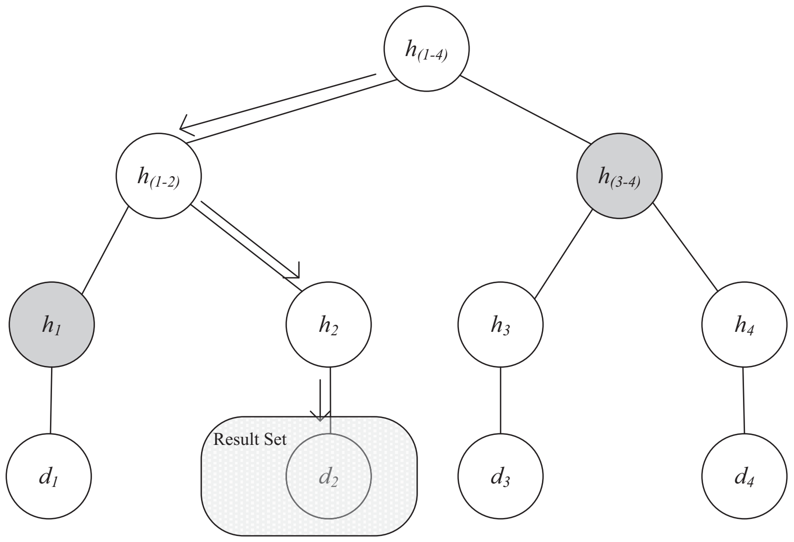

Figure 3. Merkle-hash tree.

While executing queries, the tree is pruned starting from the root. If a query is satisfied by a node, then the processing goes down one level to the children of that node. Otherwise, the hash value of the excluded node is included in the verification object. The traversing order is important in order to achieve a successful authentication, because of the concatenation used during the tree construction. The verification object should provide the hash values exactly in the same order as they are created during the tree construction. Only in this manner can we provide the same input into the hash functions, which is used in the authentication of the verification object. Consequently, the tree is traversed in a depth-first manner to satisfy the preordering.

For a given query, assume that the result set only includes d_2 in Figure 3. Arrows are included to show the path of the query from the root to the leaf. The algorithm starts from the root node and the left child node containing the hash value $h_{(1-2)}$ satisfies the query. Next, the depth-first search encounters h_1 and includes it in the *VO*, because the result set does not contain d_1. The node d_2 is included in the *VO* later, and the algorithm finishes by adding $h_{(3-4)}$ into the VO. The final state of the *VO* is $[[h_1, [d_2]], h_{(3-4)}]$. At the client side, authentication will be done by the following computation:

$$H(VO) = H(H(h_1 — H(d_2)) — h_{(3-4)}) \tag{2.4}$$

$$H(VO) = H(H(h_1 — h_2) — h_{(3-4)}) \tag{2.5}$$

$$H(VO) = H(h_{(1-2)} — h_{(3-4)}) \tag{2.6}$$

$$H(VO) = h_{(1-4)} \tag{2.7}$$

Note that $h_{(1-4)}$ is equal to the hash value of the root, which means that the authentication is done successfully at the client side.

2.4. k-Means clustering

K-means clustering [34] is an essential part of our algorithm. Basically, clustering methods are unsupervised machine learning techniques and they can be used for different types of applications [35–40]. The most common types of clustering methods are k-means clustering and expectation–maximization algorithms [41]. For the sake of simplicity, we will be concentrating on k-means clustering.

Intuitively, the k-means clustering algorithm is trained to find out the k points that represent the data the most. These points are mean values of the clusters of the underlying data. In order to get these mean values, an iterative approach is carried out to minimize the error. Reconstruction error, E, can be calculated as follows:

$$E = \sum_i \sum_N b_n^i ||d_i - m_n||^2, \tag{2.8}$$

where

$$\begin{array}{lll} \text{if} & ||d_i - m_n|| = min||d_i - m_l||, & b_n^i = 1 \\ \text{otherwise} & b_n^i = 0 & \end{array} \tag{2.9}$$

x_i are the data points, b_n^i are the cluster labels, and m_l, m_n are the mean values in the dataset. First, m_n values are randomly picked up from the initial dataset, and then at each iteration, the labels b_n^i are estimated. Once the labels are set, total reconstruction error is minimized taking its derivative with respect to m_n.

$$m_n = \sum_i b_n^i d_i / \sum_i b_n^i \tag{2.10}$$

When we calculate m_n values, we recompute b_n^i values according to the new mean values, which is the start of the second iteration. These two steps are repeated until m_n values converge.

3. Authentication of uncertain data

In this section, we explain our methodology for the authentication of uncertain data. Section 3.1 explains the PH-tree structure, Section 3.2 shows how the query processing is done in PH-trees, and finally Section 3.3 describes the authentication at the client side.

3.1. PH-tree structure

For our proposed model, we present a novel structure called the probabilistic hash-tree. Basically the PH-tree is a combination of the MH-tree and PDR-tree with a preprocessing mechanism using clustering. The clustering algorithm plays an important role in tree creation. The mean values gathered from clustering are stored in the root node of the PH-tree. Figure 4 shows a sample PH-tree and its internal node structure.

In Figure 4, the clouds represent the clusters and their corresponding sub-PH-trees. These clusters are represented by C_1, C_2, C_3, and C_4. Only C_1 and C_4 are shown in detail in Figure 4. Each cluster is created as a sub-PH-tree. Note that the roots of these subtrees are not the same as the root of the whole tree. The mean value of each cluster C_i is notated as m_i and stored in the root node. The mean values are the output of the k-means clustering algorithm. After all the clusters are formed as a sub-PH-tree and the mean values are included in the root, all the subtrees (clusters) are connected to the root node.

In the PH-tree, only leaf nodes hold the data, just like in the PDR-tree and MH-tree. Data values, d_i, and hash values, h_i, are stored inside the leaves. These probability vectors, $p_{i,j}$, determine the MBR of the node.

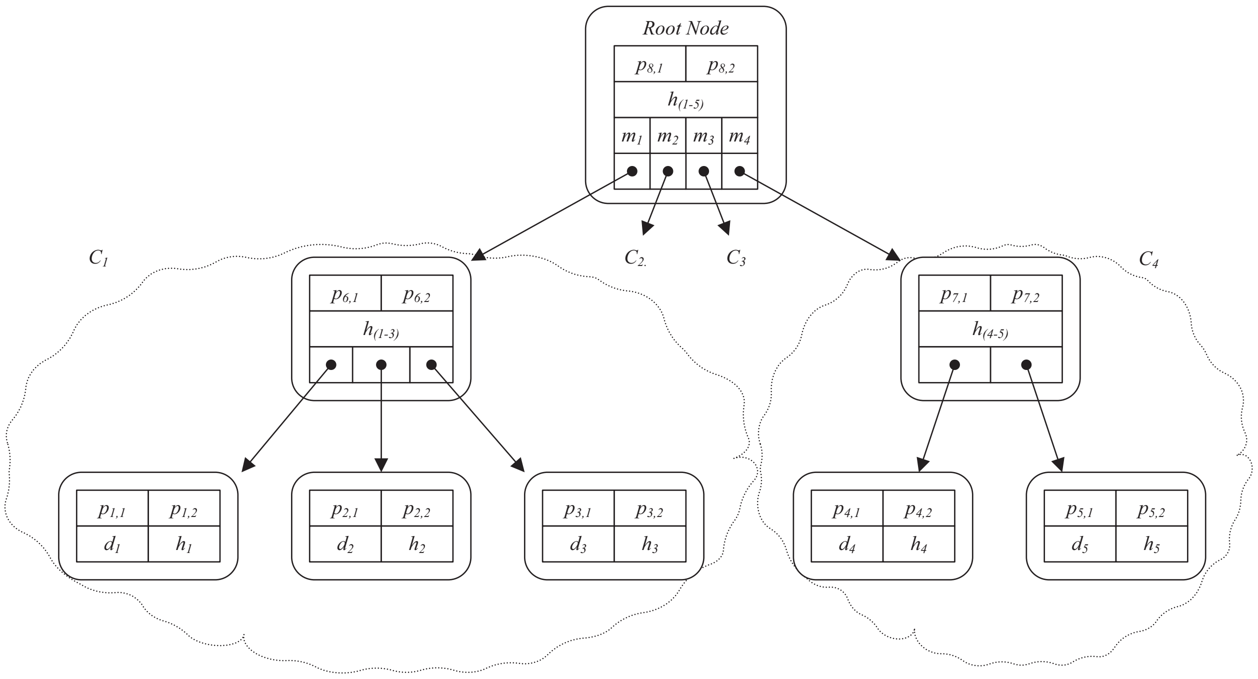
Root Node
$p_{8,1}$
$p_{8,2}$
$h_{(1-5)}$
m_1
m_2
m_3
m_4
C_1
C_2
C_3
C_4
$p_{6,1}$
$p_{6,2}$
$h_{(1-3)}$
$p_{7,1}$
$p_{7,2}$
$h_{(4-5)}$
$p_{1,1}$ $p_{1,2}$ d_1 h_1
$p_{2,1}$ $p_{2,2}$ d_2 h_2
$p_{3,1}$ $p_{3,2}$ d_3 h_3
$p_{4,1}$ $p_{4,2}$ d_4 h_4
$p_{5,1}$ $p_{5,2}$ d_5 h_5

Figure 4. Probabilistic hash tree.

Internal nodes only contain MBRs, hash values, and pointers. In order to evaluate the probability values of an internal node, we look at the maximum values of the child nodes. For the given example in Figure 4, we look for the children $p_{i,1}$ values, where $i = 1, 3$. We retrieve the maximum of them and store it as $p_{6,1}$. All the probabilities up to the root are evaluated in this manner. This ensures that if a node's MBR does not satisfy a query, then the MBRs of the node's children will not either [10].

Hash values of the internal nodes are computed in the same way as in MH-tree computation. All the binary hash strings in the child nodes are concatenated in a breadth-first manner and rehashed (e.g., $h_{(1-3)} = H(h_1|h_2|h_3)$, $h_{(4-5)} = H(h_4|h_5)$ and $h_{(1-5)} = H(h_{(1-3)}|h_{(4-5)})$).

The differences between PH-trees, MH-trees, and PDR-trees in terms of node structure and tree structure are shown briefly in Tables 3 and 4.

Table 3. Comparison in terms of node structure.

	Node structure
MH-tree	• Root node and internal nodes are structurally similar. • Does not contain MBR. • Does not contain probability vectors. • Nodes contain corresponding hash values.
PDR-tree	• Root node and internal nodes are structurally similar. • Nodes contain MBR. • Nodes have probability vectors. • Does not hold hash values.
PH-tree	• Root node contains cluster mean values. • Nodes contain MBR. • Nodes have probability vectors. • Nodes contain corresponding hash values.

Table 4. Comparison in terms of tree construction.

	Tree construction
MH-tree	• Does not implement preprocessing. • Binary tree.
PDR-tree	• Does not implement preprocessing. • Multiway tree.
PH-tree	• K-means clustering implemented as preprocessing. • Multiway tree.

3.2. Query processing

The query processing algorithm is implemented according to the tree structure based on clustering.

3.2.1. Query processing at root level

The *RangeQueryRoot Algorithm* processes the incoming query q by calculating the Euclidian distance with the mean values. The subtree with the smallest distance is the nearest cluster to q. Then we call the recursive *RangeQuery* function to go deeper into the tree and extract the result set inside this cluster. Other subtrees' hash values are included in the VO since we know that they do not hold any data item inside the result set. This is the key part of our method. Obsolete entries in the verification object are eliminated at the beginning. We do not allow the *RangeQuery* algorithm to go into any unnecessary subtrees at the root level. There are two distinct advantages of this method: (i) decreasing the VO size significantly and (ii) improving the verification time at the client side. The results of the experiments also validate these arguments.

The details of the *RangeQueryRoot* function are given in Algorithm 1. In this algorithm, "d" denotes the temporary distance variable, and "N_i" denotes the child node of the root node, where "i" is the index value. "m_i" corresponds to the mean value and "h_i" corresponds to the hash value stored within node "N_i". "*Euclidian*" is the function that is used to calculate the Euclidian distance between two vectors.

Algorithm 1

```
RangeQueryRoot(Query $q$)
1. for each child $N_i$ of root
2.     $d$ = Euclidian($q$, $m_i$)        // calculates the euclidian distance between $q$ and $m_i$
3.     SmallestDistance = 9999999          // initialize SmallestDistance to a large value
4.     if ( $d$ < SmallestDistance)        // check if $d$ is the smallest distance
5.         SmallestDistance = $d$
6.         ClosestNode = $N_i$             // assign the ClosestNode to $N_i$
7.         ClosestNodeIndex = $i$
8.     end-if
9. end-for        // at this point, the cluster which holds the result set is found (ClosestNode).
10. $i$ = 0
11. while ( $i$ < ClosestNodeIndex)     // until we reach the ClosestNode
12.     $N_i$ = ith child of root
```

```
13.         Append $h_i$ to $VO$          // all the subtrees' hash values are included inside VO
14.         i++
15. end-while
16. Append "(" to $VO$                    // "(" is appended to indicate that algorithm goes one level down
17. RangeQuery($q$, ClosestNode)          // query processing inside the cluster
18. Append ")" to $VO$                    // ")" is appended to indicate that algorithm goes one level up
19. $i$ = ClosestNodeIndex
20. while ( $i$ < SizeOfRoot)   // until we reach the last child of root node
21.         $N_i$ = $ith$ child of root
22.         Append $h_i$ to $VO$   // all the subtrees' hash values are included inside VO
23.         i++
24. end-while
```

The characters "("and")" are also appended to the VO. These markers tell the client that the query goes one level down or up respectively. This is very useful for authentication because the client is informed when the hash function must be called.

The order of the concatenated hash bytes is important. In the following, we explain it with an example. Suppose that we have four different clusters C_1, C_2, C_3, C_4 consecutively and C_3 (also N_3) contains the result set. First, we must include the hash values of N_1, N_2 into the VO then we call the *RangeQuery*(N_3, VO). After VO is updated inside the *RangeQuery* procedure, we should include the final hash value of N_4 into the VO.

Figure 5 shows a sample PH-tree with the probability vectors that are included in the internal nodes. In this example, clusters C_1 and C_4 are formed according to the probability values of their subtrees. Assume that a query q, which has probability values of <0.17, 0.83>, is being processed for this sample PH-tree. Since $q$is closer to cluster C_1 than other clusters in terms of distance, C_2, C_3, C_4 have been pruned out and the traversal of the tree has been minimized.

Figure 6 illustrates an implementation similar to that of this example with the exception that C_1 also contains the result set.

3.2.2. Query processing inside the cluster

The *RangeQuery* algorithm is called when the right cluster is found for the query q. If the MBR boundary qualifies for q, either it goes one level down or includes the result set item in the VO. Otherwise the hash value is included in the VO.

Algorithm 2 gives the pseudo code for the RangeQuery procedure. In this pseudo code, "C_i" denotes a child node of "N", where "N" is an internal node. "$C_i.p$" denotes the probability vector of C_i, "$C_i.d$" denotes the data item of C_i, and "$C_i.h$" denotes the hash value stored within the C_i. "*KL*" denotes the Kullback–Leibler distance function and "T" denotes the threshold value.

Threshold T also plays an essential role here. It determines the objects in the result set. If we increase T too much, then we have the risk of having too many items in the result set. More than one cluster should be pruned in such a situation. If T is over-decreased, then we can end up having few or no items in the result set. The T value should be fine tuned with experiments to get optimum results.

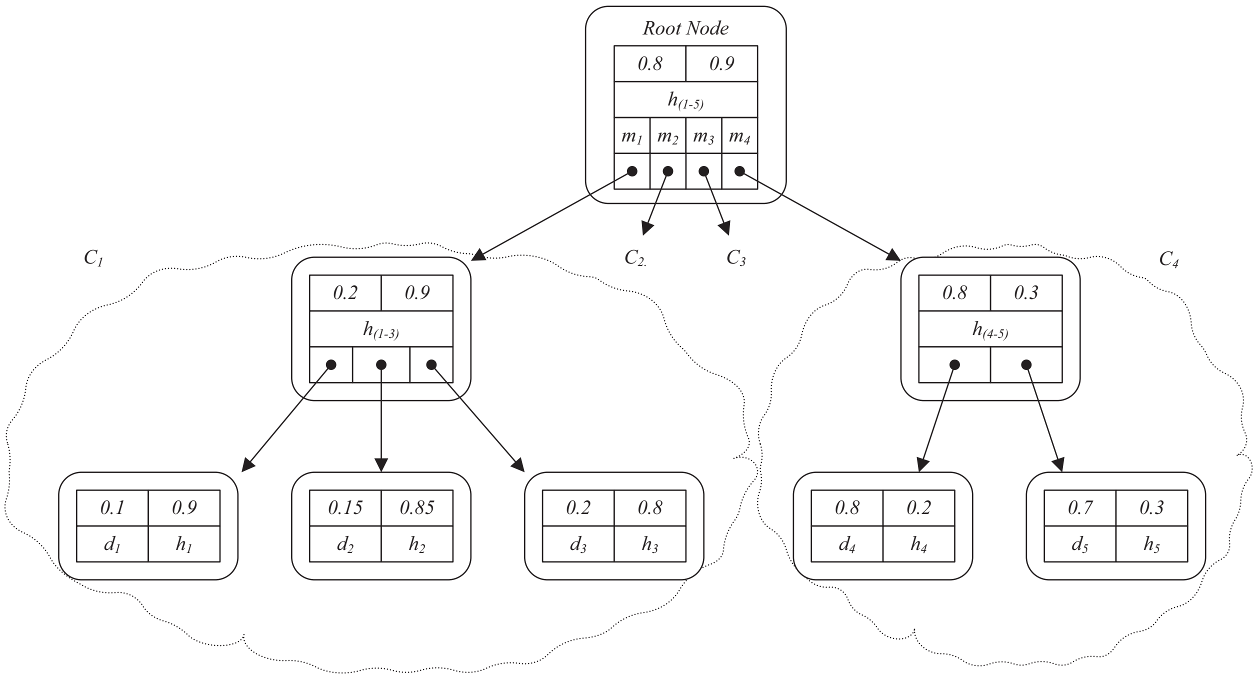

Figure 5. Probabilistic hash tree.

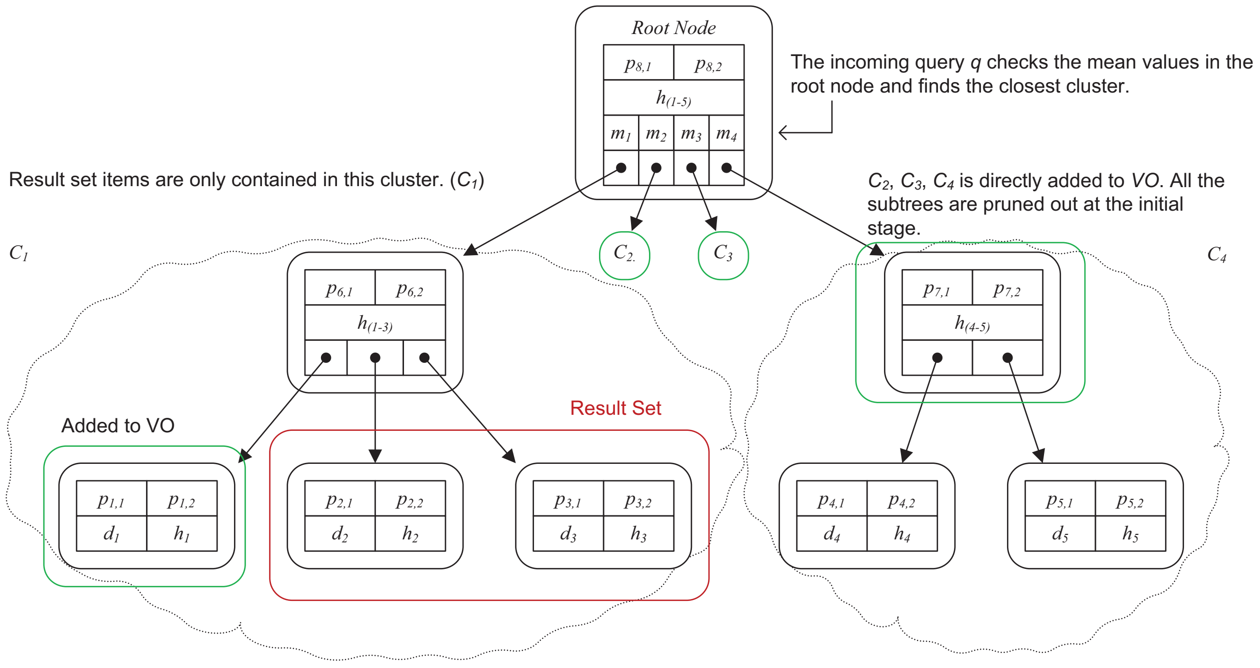

Figure 6. Probabilistic hash tree.

3.3. Authentication in PH-trees

Authentication on the client side is handled by the *RootAuthentication* procedure. This procedure fetches all the entries inside the VO and processes them according to their data type. The final hash value computed with the VO should match the given h_r.

Algorithm 2

```
RangeQuery(Query q, Node N)
1. if N is leaf node
2.     for each cluster $C_i$ of N
3.         if $KL(C_i.p, q) < T$            // if query q qualifies for $C_i.p$
4.             Append $C_i.d$ to the VO     // $C_i.d$ is a result set item
5.         else                             // if query q does not qualify
6.             Append $C_i.h$ to the VO     // hash value is appended to VO
7.         end-if
8.     end-for
9. else                                     // N is an internal node
10.    for each cluster $C_i$ of N
11.        if $KL(C_i.p, q) < T$            // if query q qualifies for $C_i.p$
12.            Append "(" to VO
13.            RangeQuery(q, $C_i$)          // we go one level down with a recursion
14.            Append ")" to VO
15.        else                             // if the query q does not qualify
16.            Append $C_i.h$ to the VO     // hash value is appended to VO
17.        end-if
18.    end-for
19. end-if
```

Algorithm 3 illustrates *RootAuthentication* function in detail. "—" denotes the concatenation operator in the algorithm, e_i denotes the entry item, h_c denotes the concatenated hash string, and $H(x)$ denotes the hashing function.

Algorithm 3

```
RangeQuery(Query q, Node N)
1. if N is leaf node
2.     for each cluster $C_i$ of N
3.         if $KL(C_i.p, q) < T$            // if query q qualifies for $C_i.p$
4.             Append $C_i.d$ to the VO     // $C_i.d$ is a result set item
5.         else                             // if query q does not qualify
6.             Append $C_i.h$ to the VO     // hash value is appended to VO
7.         end-if
8.     end-for
9. else                                     // N is an internal node
```

```
10.      for each cluster $C_i$ of $N$
11.          if $KL(C_i.p, q) < T$              // if query q qualifies for $C_i.p$
12.              Append "(" to $VO$
13.              RangeQuery($q$, $C_i$)              // we go one level down with a recursion
14.              Append ")" to $VO$
15.          else                                // if the query $q$ does not qualify
16.              Append $C_i.h$ to the $VO$      // hash value is appended to VO
17.          end-if
18.      end-for
19. end-if
```

Basically, the hashing function is called after the ")" symbol appears in the VO. Otherwise it is concatenated to the hash string. An entry can be either a data item in the result set or a hash value. If it is a data item it is hashed before concatenation; otherwise it is directly appended to h_c. If the entry is a "(" character, then the *RootAuthentication* is called again to enable recursion inside the VO.

The differences between PH-trees, PDR-trees, and MH-trees in terms of query processing and authentication are shown briefly in Table 5.

Table 5. Comparison in terms of query processing.

	Query processing
MH-tree	• Implements authenticated processing. • Does not handle uncertainty. • Does not implement range querying. • Does not include distance functions. • Constructs verification object during processing.
PDR-tree	• Does not implement authentication methods. • Handles uncertainty. • Range queries are used for query processing. • Includes distance functions. • Does not contain verification object.
PH-tree	• Implements authenticated processing. • Handles uncertainty. • Range queries are used for query processing. • Includes distance functions. • Constructs verification object during processing.

4. Experiments and results

We have tested our algorithms using both synthetic and real data. The programming language used in the implementation is Java 1.6 and experiments are performed on a 2.1 GHZ dual processor machine with a 2 GB physical memory. Page size is fixed to 8 KB in all of the experiments. Additionally, we used GNU Crypto, which is a cryptography library written in Java. It enabled us to implement Tiger hashing [30] in range query and authentication algorithms.

Synthetic data contain probability values generated from a uniform distribution using the random number generator of Java Platform Standard Edition 6. This built-in class of Java returns the pseudorandom numbers each time, which are uniformly distributed between the values 0 and 1. There are only two uncertain attributes associated with the tuples in this dataset. Data fields are also randomly generated as 100 byte strings created by generating long values and converting them into strings. The cardinalities (sizes) of this dataset are 10,000, 20,000, 30,000, 40,000, and 50,000.

Adult is a real dataset that is obtained from the UCI Machine Learning Repository [14]. This dataset is for classification purposes and is cited in various papers in the literature [42–44]. The attributes of *Adult* are 14 different social attributes including *age*, *work class*, *race*, *sex*, *native country* etc. and the associated task is to predict whether the income of a person exceeds $50K income per year based on census data. Since the *income* itself is an uncertain attribute, we have chosen ADTree [45] to estimate the probabilities values for this dataset. ADTree is very useful and efficient when the data contains both continuous and discrete attributes. Weka Data Mining Software [46] is used during the implementation of ADTree. The cardinalities of the adult dataset in the experiments are defined as 5000, 10,000, 15,000, 20,000, and 25,000.

Another important notion here is the optimization of k in clustering. Since it is a hyperparameter, repeating the experiments for each value of k from 1 to 50 is the best way to find the optimum value. Each dataset may have different underlying distributions so it has been optimized for both of the datasets. The optimum value of k is 12 and 30 for Adult and our synthetic dataset, respectively. Because of the uniform distribution of our synthetic data, it is not surprising to see a big value of k.

We compare our method with MR-trees, which is another multiattribute R-tree variant authentication technique. The results are analyzed according to VO size, verification time, query processing time, and tree construction time.

4.1. Verification object size

Figure 7 shows the results of our experiments in terms of VO size. It can be easily seen that the PH-tree outperforms the MR-tree. Because of the clustering mechanism used in PH-trees, we are able to pre-eliminate the obsolete entries in the VO. Another important detail in Figure 7 is that the slope (tangent) of the PH-tree line is smaller than that of the MR-tree line. From Figure 7, we can deduce that if we increase the dataset size, then the difference between the PH-tree and MR-tree in terms of VO size will increase.

4.2. Verification time

Here we compare the results of the MR-tree and PH-tree in terms of verification time (nanoseconds) at the client side. We were expecting the results to be analogous to those in Section 4.1. The results displayed in Figure 8 verify our expectation. The advantage of smaller VO size makes it easier to authenticate the data for the client. The only difference between the PH-tree and MR-tree in this comparison is the VO size. Consequently, Figure 7 is very similar to Figure 8. The PH-tree again outperforms the MR-tree. This becomes much more evident when the dataset cardinality increases for both synthetic and real data. Please recall that the most important criteria for the authentication methods are VO size and verification time, because they are related to the client side efficiency.

4.3. Query processing time

Figure 9 illustrates the query processing time (nanoseconds) using the MR-tree and PH-tree. Actually query tests highly depend on the given dataset and so we repeated the queries with different values and took the average. PH-trees require less time in processing queries than MR-trees do. This is important for the database

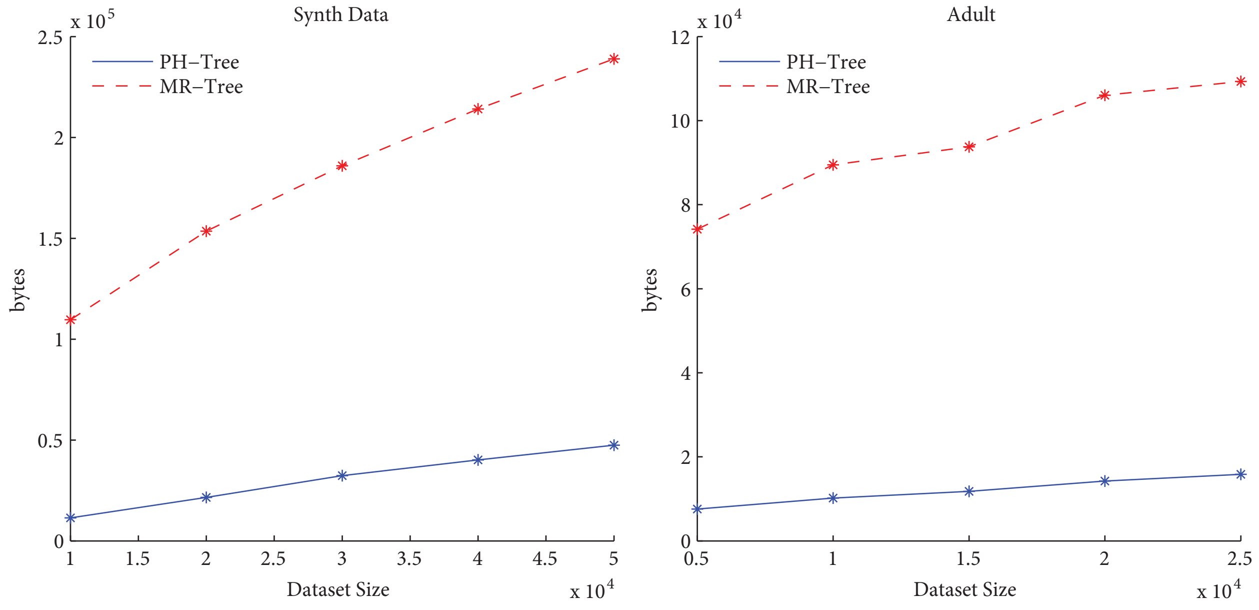

Figure 7. VO size.

service provider. The growth of the query processing cost does not change with the dataset cardinality for either of the algorithms.

4.4. Tree construction time

Finally, Figure 10 illustrates the construction time of the trees in terms of milliseconds. Time includes both the time to create the initial tree, k-means clustering implementation for the PH-tree, and the time to compute the hash values of the nodes. We see that total construction time is exponential in the dataset cardinality.

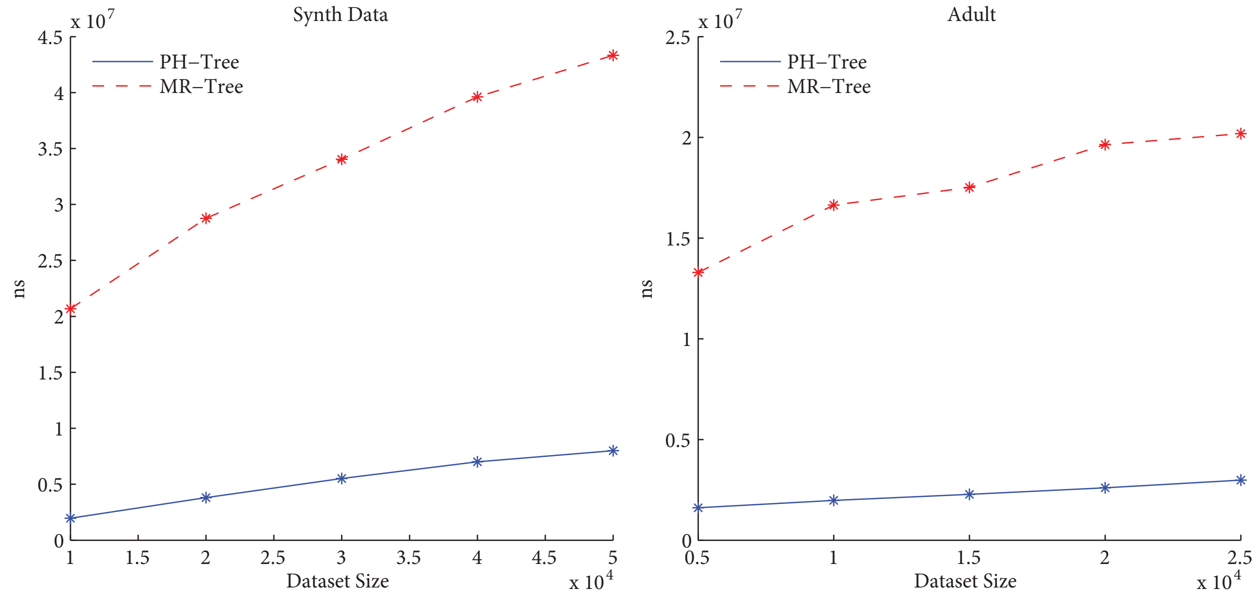

Figure 8. Verification time.

While building the initial PH-tree, we divided the whole dataset into k clusters. In this way, we constructed smaller sub-PH-trees instead of one big PH-tree, which reduces the total cost significantly. For smaller data cardinalities, both algorithms have similar requirements. When we increase the data cardinality, there is a huge increase in the cost of the MR-tree. PH-trees also grow exponentially but not as much as MR-trees, because PH-trees never start constructing the whole dataset.

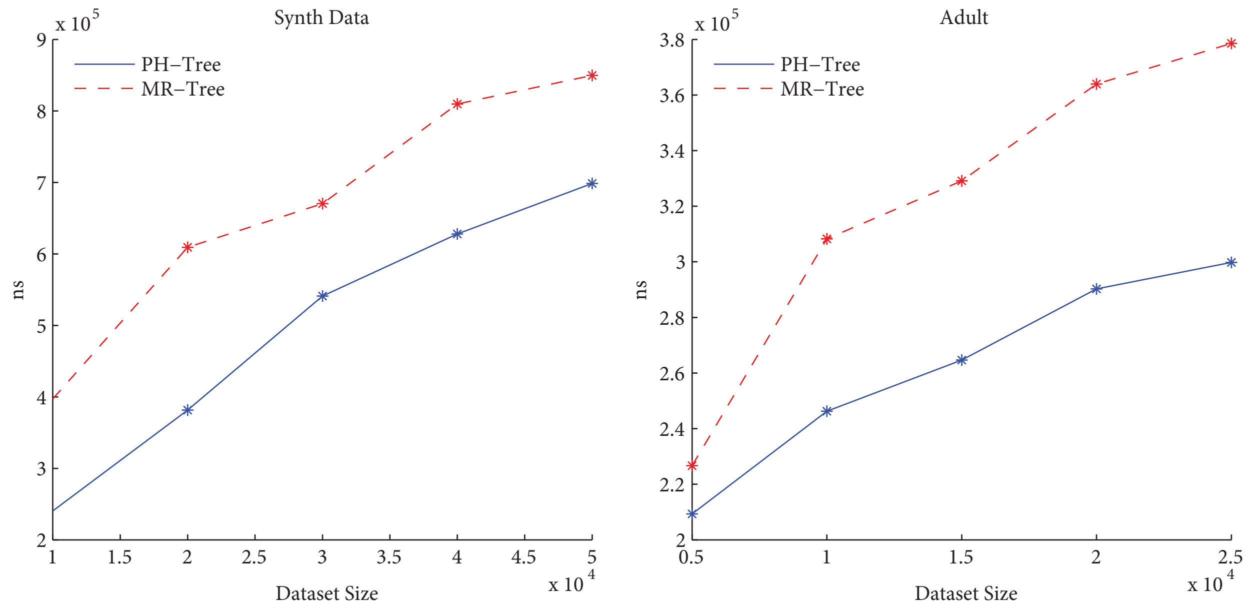

Figure 9. Query cost.

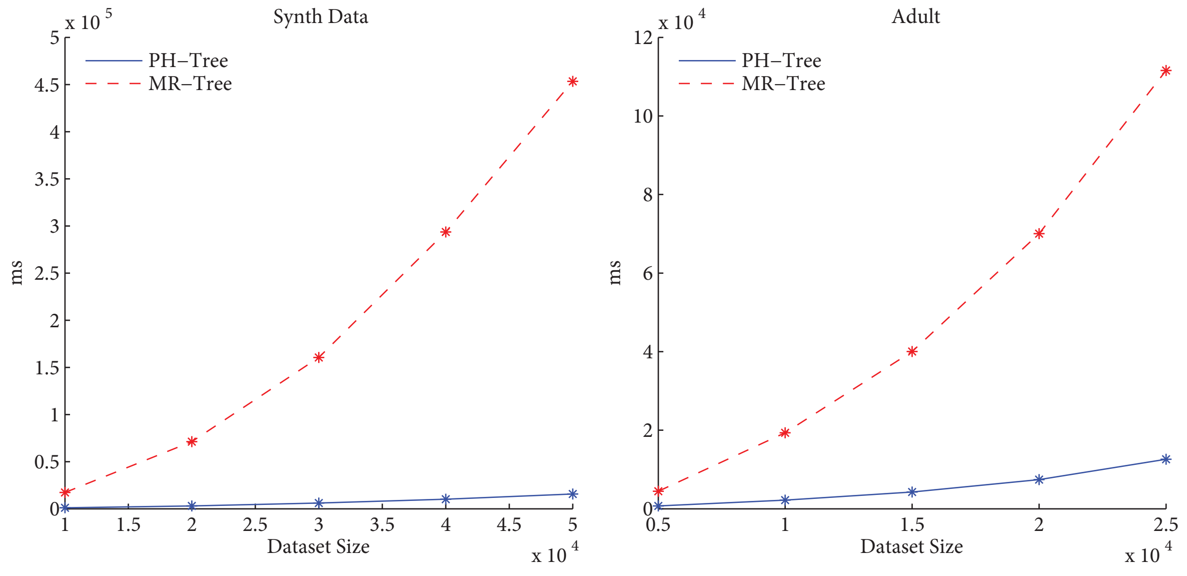

Figure 10. Tree construction time.

5. Conclusion and future work

In this paper, we propose a novel ADS called the PH-tree. The PH-tree is a hybrid model of the PDR-tree and MR-tree with a preprocessing mechanism based on k-means clustering. The main contribution of the PH-tree is its applicability on uncertain data and probabilistic databases. Existing methods in the literature such as MR-trees do not work well on uncertain data in terms of query processing cost, verification object size, and verification time. Our experiments show that PH-trees outperform MR-trees in these three criteria. PH-trees also improve the performance of tree construction, which is an important benefit for database service providers. PH-trees assume that the underlying uncertain data are independent. As future research, this work can be extended to correlated data and to include authentication of junction trees.

References

[1] Almeida TV, Güting RH. Supporting uncertainty in moving objects in network databases. In: 2005 Annual International Workshop on Geographic Information Systems; 31 October–05 November 2005; Bremen, Germany. New York, NY, USA: ACM. pp. 31-40.

[2] Andritsos P, Fuxman A, Miller RJ. Clean answers over dirty databases: a probabilistic approach. In: IEEE 2006 International Conference on Data Engineering; 3–7 April 2006; Georgia, USA. New York, NY, USA: IEEE. pp. 30.

[3] Silberstein AS, Braynard R, Ellis C, Munagala K, Yang J. A sampling-based approach to optimizing top-k queries in sensor networks. In: IEEE 2006 International Conference on Data Engineering; 3–7 April 2006; Georgia, USA. New York, NY, USA: IEEE. pp. 68.

[4] Antova L, Koch C, Olteanu D. (2007) 10ˆ(10ˆ6) Worlds and beyond: efficient representation and processing of incomplete information. In: IEEE 2007 International Conference on Data Engineering; 15–20 April 2007; Istanbul, Turkey. New York, NY, USA: IEEE. pp. 606-615.

[5] Lian X, Chen L. Probabilistic top-k dominating queries in uncertain databases. Inform Sciences 2013: 23-46.

[6] Kanagal B, Deshpande A. Indexing correlated probabilistic databases. In: ACM SIGMOD 2009 International Conference on Management of Data; 29 June–02 July 2009; Rhode Island, USA. New York, NY, USA: ACM. pp. 455-468.

[7] Soliman MA, Ilyas IF, Chang KCC. Top-k query processing in uncertain databases. In: IEEE 2007 International Conference on Data Engineering; 15–20 April 2007; Istanbul, Turkey. New York, NY, USA: IEEE. pp. 896-905.

[8] Hacigumus H, Iyer B, Mehrotra S. Providing database as a service. In: IEEE 2002 International Conference on Data Engineering; 26 February–1 March 2002; Istanbul, Turkey. New York, NY, USA: IEEE. pp. 29-38.

[9] Yang Y, Papadias D, Papadopoulos S, Kalnis P. Authenticated join processing in outsourced databases. In: ACM SIGMOD 2009 International Conference on Management of Data; 29 June–02 July 2009; Rhode Island, USA. New York, NY, USA: ACM. pp. 5-18.

[10] Singh S, Mayfield C, Prabhakar S, Shah R, Hambrusch SE. Indexing uncertain categorical data. In: IEEE 2007 International Conference on Data Engineering; 15–20 April 2007; Istanbul, Turkey. New York, NY, USA: IEEE. pp. 616-625.

[11] Merkle RC. A certified digital signature. In: 1989 International Conference on Advances in Cryptology; 20–24 August 1989; California, USA. pp. 218-238.

[12] Dalvi N, Suciu D. Management of probabilistic data foundations and challenges. In: ACM SIGMOD 2007 Symposium on Principles of Database Systems; 12–14 June 2007; Beijing, China. New York, NY, USA: ACM. pp. 1-12.

[13] AbdulAzeem Y, ElDesouky AI, Ali HA. A framework for ranking uncertain distributed database. Data Knowl Eng 2014; 92: 1-19.

[14] Frank A, Asuncion A. UCI Machine Learning Repository: Adult data set, 2010.

[15] Kullback S, Leibler RA. On information and sufficiency. Ann Math 1951; 22: 79-86.

[16] Pereira FCN, Tishby N, Lee L. Distributional clustering of English words. In: 1993 Meeting of the Association for Computational Linguistics; 22–26 June 1993; Ohio, USA. pp. 183-190.

[17] Ge T, Zdonik SB, Madden S. Top-k queries on uncertain data: on score distribution and typical answers. In: ACM SIGMOD 2009 International Conference on Management of Data; 29 June–02 July 2009; Rhode Island, USA. New York, NY, USA: ACM. pp. 375-388.

[18] Li F, Yi K, Jestes J. Ranking distributed probabilistic data. In: ACM SIGMOD 2009 International Conference on Management of Data; 29 June–02 July 2009; Rhode Island, USA. New York, NY, USA: ACM. pp. 361-374.

[19] Liu D, Wan C, Xiong N, Park JH, Rho S. Top-k entities query processing on uncertainly fused multi-sensory data. Pers Ubiquit Comput 2013; 17: 951-963.

[20] Jensen FV, Jensen F. Optimal junction trees. In: 1994 Conference on Uncertainty in Artificial Intelligence; 29–31 July 1994; Washington, USA. pp. 360-366.

[21] Angiulli F, Fassetti F. Indexing uncertain data in general metric spaces. IEEE T Knowl Data En 2012; 24: 1640-1657.

[22] Zhang Y, Zhang W, Lin Q, Lin X, Shen HT. Effectively indexing the multidimensional uncertain objects. IEEE T Knowl Data En 2013; 26: 608-622.

[23] Guttman A. R-Trees: a dynamic index structure for spatial searching. In: ACM SIGMOD 1984 International Conference on Management of Data; 18–21 June 1984; Massachusetts, USA. New York, NY, USA: ACM. pp. 47-57.

[24] Li F, Hadjieleftheriou M, Kollios G, Reyzin L. Dynamic authenticated index structure for outsourced databases. In: ACM SIGMOD 2006 International Conference on Management of Data; 27–29 June 2006; Illinois, USA. New York, NY, USA: ACM. pp. 121-132.

[25] Cheng W, Pang H, Tan KL. Authenticating multi dimensional query results in data publishing. In: 2006 International Federation for Information Processing Workshop on Data and Applications Security; 31 July–2 August 2006; Sophia Antipolis, France. pp. 60-73.

[26] Yang Y, Papadopoulos S, Papadias D, Kollios G. Spatial outsourcing for location based services. In: IEEE 2008 International Conference on Data Engineering; 7–12 April 2008; Cancun, Mexico. New York, NY, USA: IEEE. pp. 1082-1091.

[27] Ku WS, Hu L, Shahabi C, Wang H. A query integrity assurance scheme for accessing outsourced spatial databases. Geoinformatica 2013; 17: 97-124.

[28] Evans A, Kantrowitz W, Weiss E. A user authentication scheme not requiring secrecy in the computer. Commun ACM 1974; 17: 437-442.

[29] Wilkes MV. Time sharing computer systems. New York, NY, USA: Elsevier Science, 1975.

[30] Anderson R, Biham W. Tiger: A fast new hash function. In: 1996 International Workshop on Fast Software Encryption; 21–23 February 1996; Cambridge, UK. pp. 89-97.

[31] Mendel F. Two passes of Tiger are not one-way. In: 2009 International Conference on Cryptology; 21–25 June 2009; Gammarth, TUN. pp. 29-40.

[32] Ishai Y, Kushilevitz E, Ostrovsky R. Sufficient conditions for collision-resistant hashing. In: 2005 Theory of Cryptography Conference; 10–12 February 2005; Cambridge, MA, USA. pp. 445-456.

[33] Rogaway P, Shrimpton T. Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: 2004 International Workshop on Fast Software Encryption; 5–7 February 2004; Delhi, India. pp. 371-388.

[34] Lloyd SP. Least squares quantization in PCM. IEEE T Inform Theory 1982; 28: 129-137.

[35] Cai X, Li W. A spectral analysis approach to document summarization: Clustering and ranking sentences simultaneously. Inform Sciences 2011; 181: 3816-3827.

[36] Ghosh A, Mishra NS, Ghosh S. Fuzzy clustering algorithms for unsupervised change detection in remote sensing images. Inform Sciences 2011; 181: 699-715.

[37] Huang X, Zheng X, Yuan W, Wang F, Zhu S, Witten IH. Enhanced clustering of biomedical documents using ensemble non-negative matrix factorization. Inform Sciences 2011; 181: 2293-2302.

[38] Tsai YS, Tsai P. Adaptive data hiding for vector quantization images based on overlapping codeword clustering. Inform Sciences 2011; 181: 3188-3198.

[39] Oh SK, Kim WD, Pedrycz W, Joo SC. Design of k-means clustering-based polynomial radial basis function neural networks (pRBF NNs) realized with the aid of article swarm optimization and differential evolution. Neurocomputing 2012; 78: 121-132.

[40] An F, Mattausch HR. K-means clustering algorithm for multimedia applications with flexible HW/SW co-design. J Syst Architect 2013; 59: 155-164.

[41] Dempster AP, Laird NM, Rubin DB. Maximum likelihood from incomplete data via the EM algorithm. J Roy Stat Soc B Met 1977; 39: 1-38.

[42] Agrawal R, Ikant R, Thomas D. Privacy preserving OLAP. In: ACM SIGMOD 2005 International Conference on Management of Data; 14–16 June 2005; Maryland, USA. New York, NY, USA: ACM. pp. 251-262.

[43] Zadrozny B. Learning and evaluating classifiers under sample selection bias. In: 2004 International Conference on Machine Learning; 4–8 July 2004; Alberta, Canada. New York, NY, USA: ACM. pp. 114.

[44] Saharon R. Model selection via the AUC. In: 2004 International Conference on Machine Learning; 4–8 July 2004; Alberta, Canada. New York, NY, USA: ACM. pp. 89.

[45] Freund Y, Mason L. The alternating decision tree learning algorithm. In: 1999 International Conference on Machine Learning; 27–30 June 1999; Bled, Slovenia. New York, NY, USA: ACM. pp. 124-133.

[46] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA Data Mining Software: An Update. SIGKDD Explor 2009; 11: 10-18.