AUTHENTICATION OF UNCERTAIN DATA BASED ON K-MEANS CLUSTERING

by

Levent Unver

B.S., Computer Engineering, Izmir Institute of Technology, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2011

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Taflan Gundem for his guidance, support and ideas throughout the preparation of this thesis.

I would also like to thank Fikret Gurgen and Ismail Ari for participating in the thesis jury and giving me feedback in this work.

I would like to express my deepest gratitude to the people who were really helpful and supportive during these hard times. Those people are my teammates in Netas, Yasemin Nisanoglu, Emrah Dincer, Gorkem Sayar, Asli Karatekelioglu, Ufuk Ozbay, Yavuz Selim Cakir, Sevda Ozdemir, Gokhan Yildirim, Yalcin Guney and Aykut Arda. They always shared my greatest pain while sitting in front of my LCD monitor during the craziest hours. Their never-ending love and friendship made me feel extremely comfortable, preventing me from losing my sanity. Also my special thanks go to Ceyda Ulker, my manager. She had a very understanding attitude and gave me all the free time that I needed throughout this journey. She always trusted me in this work, even in the times that I lost my self-confidence. This thesis would be undone without these guys. There is no better way to express my feelings.

I would also like to thank the closest people to me, Mustafa Ersahin, Emrah Bilgin, Silan Avsar, Gokhan Gulgezen and Arda Unvan. All the joyful moments shared together helped me progress even more. Their motivational support and energy kept me up all the time.

Finally, I want to mention about two special people in my life, Nuri Ermis and Tatlihan Tuncel. Words will always be less than their actual values, no matter how much I tell.

ABSTRACT

AUTHENTICATION OF UNCERTAIN DATA BASED ON K-MEANS CLUSTERING

Probabilistic databases are beginning to expand in the database literature because of the upcoming challenges of uncertainty. It is a very new topic for the community and there are still some open problems for the researchers. Outsourcing probabilistic databases has never been worked before since there are no commercial probabilistic database management systems yet. The aim of this research is to introduce authenticated query processing in outsourced probabilistic databases. In order to proceed with the authentication, indexing methods should be analyzed first. We have surveyed the existing structures for this purpose and decided to use pdr-Tree as the indexing method, because it works very efficiently on probabilistic databases and fits really well with the authentication techniques. We have proposed a novel authenticated data structure (ADS) called PH-Tree, which is an hybrid model of pdr-Tree and MH-Tree. Straightforward approach is not competent for hybridization and produce very poor results. By this reason, we have also implemented k-means clustering as a preprocessor. We have compared our algorithm with an existing ADS called MR-Tree and proved that PH-Trees outperform MR-Trees significantly.

ÖZET

BELİRSİZ VERİLERİN K-ORTA KÜMELEME TEKNİĞİYLE DOĞRULANMASI

Verilerdeki belirsizliklerin zorluklarından ötürü olasılıksal veritabanları, akademik literatürde yavaş yavaş kabul görmeye başladı. Henüz çok yeni bir araştırma konusu olduğu için hala birçok açık problem bulunmakta. Olasılıksal veritabanlarının dış kaynak olarak kullanılması daha önce hiç çalışılmadı, çünkü henüz ticarileşmiş bir olasılıksal veritabanı yönetim sistemi yok. Bu çalışmanın amacı dış kaynak olarak kullanılmış olasılıksal veritabanlarındaki sorguların doğrulanması. Doğrulama işlemlerine geçebilmek için öncelikle indeksleme yöntemlerinin incelenmesi gerekir. Bu amaçla literatür taraması yaptık ve pdr-Ağaç üzerinde çalışmaya karar verdik, çünkü bu yapılar hem olasılıksal veritabanları üzerinden çok etkili çalışıyor, hem de doğrulama yöntemleriyle başarılı bir biçimde örtüşüyor. Bu çalışmada, pdr-Ağaç ile MH-Ağaç'ları birleştirerek PH-Ağaç diye adlandırılan yeni bir doğrulama veri yapısı önerdik. Bu birleştirmeyi daha da geliştirmek amacıyla önişlemci olarak k-orta kümeleme tekniğini kullandık. Bu sayede algoritmalarımızın düşük performanslı çalışmasını engelledik. Önerdiğimiz veri yapısını MR-Ağaç ile karşılaştırdık ve PH-Ağaç'ların MR-Ağaç'lardan çok daha iyi çalıştığını deney sonuçlarıyla ispatladık.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	attribute
b_n^i	cluster label of a data item
C_i	cluster
d_i	data item
D	categorical domain
E	Reconstruction Error
$E(u, v)$	Euclidian distance
$F(u, v)$	Divergence function
h_i	hash value
h_r	hash value of the root node
$H(x)$	Hash function
$KL(u, v)$	Kullback-Leibler distance
m_n	mean value
$M(u, v)$	Manhattan distance
MBR	Minimum bounding rectangle
N_i	Node
p_i	probability value
P	probability vector
q	query
R	relation
RS	result set
t	tuples
u, v	uncertain attributes
τ	Threshold value

LIST OF ACRONYMS/ABBREVIATIONS

ADS	Authenticated Data Structure
ADTree	Alternating Decision Tree
DO	Data Owner
DSP	Database Service Provider
EM	Expectation Maximization Algorithm
MH-Tree	Merkle Hash-Tree
MR-Tree	Merkle R-Tree
pdr-Tree	Probabilistic Distribution R-Tree
PH-Tree	Probabilistic Hash-Tree
VO	Verification Object

1. INTRODUCTION

The need for uncertain data management and probabilistic queries are arising in many areas of the computer applications. Most common examples are sensor networks, linguistic collections, data cleaning applications and bioinformatics. In order to manage the data with the underlying nature of the uncertainty, several methods have been proposed for query processing and efficient indexing [1-4]. All of these techniques have been developed in order to handle the outgrowing need for managing uncertain data.

Another important subject of the database systems is outsourcing which is being developed over the last decade. This concept has been proposed by *Hacıgümüş et al.* [5] and gained some popularity since then. In this work, we are focusing on outsourcing the uncertain data which needs some specialization for this purpose.

Database outsourcing has been achieved by authenticated query processing and related structures. 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 DSP responds with the result set of this query. While processing the range query DSP should also deliver a Verification Object (VO) to the DO. DO authenticates the result set with the given public key and the incoming VO. In order to achieve this functionality DSP stores the data in a different structure which is called Authenticated Data Structure (ADS). Figure 1.1 illustrates this model.

DO is checking for the soundness and the completeness with the given VO and

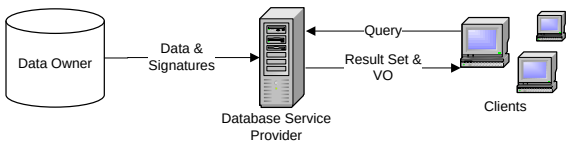

Figure 1.1. Authentication model in outsourced databases.

Table 1.1. 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

public key. Soundness means the result set is correct and not modified. Completeness means all the valid results are included within the result set. These two criteria are explained with an example.

Consider that we have a sample data given in Table 1.1. An example of a simple query can be the entries which have over 65K income. Then the actual result set would be $RS = \{(d_1, Jim, Tech-Support), (d_2, Brian, Exec-Managerial), (d_3, Nancy, Sales)\}$. $RS = \{(d_1, Jim, Tech - Support), (d_6, Brian, Exec - Managerial), (d_3, Sally, Sales)\}$ will not be sound because it contains altered data. $RS = \{$*(d_1, Jim, Tech-Support), (d_2, Brian, Exec-Managerial)*$\}$ also is not complete because it has missing data.

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, so minimizing VO size and verification time is much more important than minimizing query processing time [6]. These two concepts increase the overhead at the client side.

In this thesis, we are implementing a probabilistic database as a service. In order to achieve this, a new type of authenticated data structure called probabilistic hash-tree (PH-Tree) is proposed. PH-Trees are hybridized model of the probabilistic distribution R-trees (PDR-Trees) [7], designed for probabilistic databases and Merkle-Hash Trees

[8]. PH-Trees also gain benefit from machine learning techniques such as clustering. k-means clustering is implemented as a preprocessor and the results are stored within the root node of the PH-Tree. This algorithm fits excellently in our model and greatly reduces the overhead at the client side, which is the main objective of the authenticated query processing algorithms as mentioned above.

1.1. Outline

The following sections are organized as follows: Chapter 2 gives information about the related algorithms and some preliminary work. PH-Tree is explained in detail with the pseudocodes and figures in Chapter 3. Chapter 4 contains the experiments and their corresponding results. Finally, Chapter 5 discusses the subject for the future work and concludes the thesis.

2. RELATED WORK AND BACKGROUND

2.1. Uncertain Data Model

Before going into the authentication techniques, it is important to address the problem of uncertainty and how it is handled in the literature. Probabilistic databases have been developed for this purpose and several works have been published recently [1, 9].

In probabilistic databases, possible worlds are generated with the given imprecise data through all the set of outcomes. In these possible worlds, we are introduced with the uncertain attributes which have probability values instead of certain data. The sum of all probability values in a given attribute should not be higher than one. Table 2.1 gives much more detail about the data model and uncertain attributes.

In this model a relation can have attributes which are allowed to take probabilistic values. These attributes are called *uncertain attributes*. In Table 2.1, an approach is made towards this model by a sample data similar to *Adult* dataset [10]. Entries have certain information such as *name*, *income*, *age*, *sex* and an uncertain attribute *occupation*. The values of *occupation* are estimated by the prior data. A query can be made on this data by getting all the tuples which are highly likely to work on *Sales*. An

Table 2.1. 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)}

uncertain attribute can be formally defined as follows.

Definition 2.1: *An uncertain attribute u can be defined as a probability distribution over D, which is a discrete categorical domain* $D = \{d_1, d_2, d_3, ..., d_N\}$. *The probability vector* $u.P =< p_1, p_2, p_3, ..., p_N >$ *can be defined as* $Pr(u = d_i) = u.p_i$.

Uncertainty comes from the lack of knowledge of the exact value, however we describe the uncertain attribute with an exact probability value. This way, operators can be used as if they are dealing with certain attributes.

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

Manhattan distance (city-block distance) between two distributions can be calculated as:

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

Euclidian distance 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 is useful when we are focusing on distributional divergence:

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

KL distance is based on cross-entropy measure in information theory and is very useful during clustering implementation. Clustering is detailed in Chapter 2.4.

After the data model and primitives are defined, we proceed with the basic query

operations. The most common type of queries in probabilistic databases are top-k query processing. Several research studies have been made for the implementation of top-k query processing over uncertain data [2, 11, 12]. Note that these applications are not identical with the traditional top-k query processing.

In this work, we will be focusing on probabilistic equality threshold and distributional similarity threshold queries.

Definition 2.2: *Probabilistic Equality Query q can be defined via relation R over attribute a such that q returns all the tuples t implying $Pr(q = t.a) > 0$.*

Most of the tuples with very low probabilities can satisfy this query, so it is not always a good approach to use probabilistic equality queries. In this sense, threshold mechanisms are more meaningful than probabilistic equality queries.

Definition 2.3: *Probabilistic Equality Threshold Query q can be defined via a relation R over attribute a such that the result set of q is all tuples t satisfying $Pr(q = t.a) \geq \tau$, where τ value is the given threshold.*

For example, a probability equality threshold query can be given as "which people are potentially working on the same job with a given minimum probability?". Similarly, we can find out which people can work for a given occupation with a pre-defined minimum probability. This method can also enable top-k query mechanisms in which the most similar k tuples to q can be returned as a result set.

Definition 2.4: *Distributional Similarity Threshold Query q can be defined via relation R over attribute a such that the result set of the q is all tuples t satisfying $F(q, t.a) \leq \tau$, where F is a divergence function and τ is the given threshold value.*

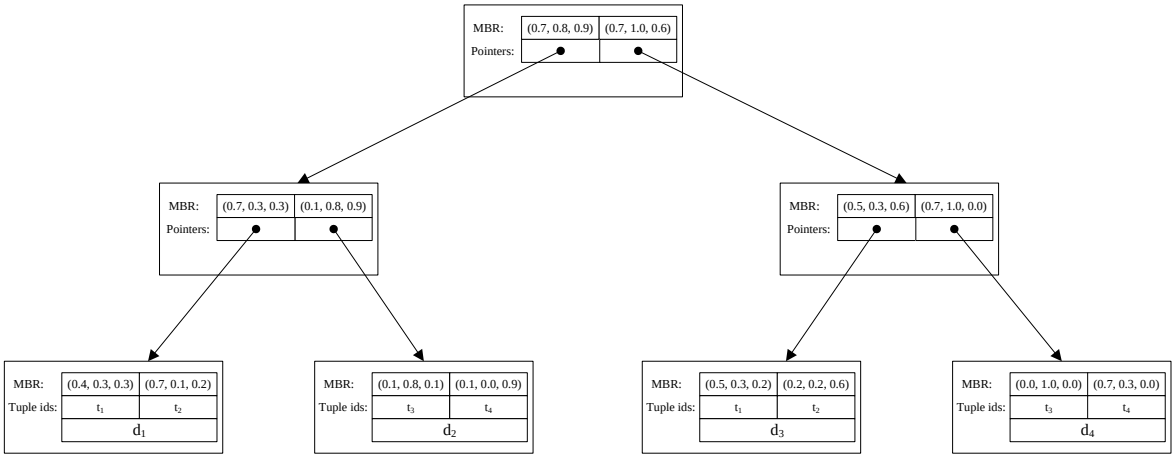

Figure 2.1. A sample probabilistic distribution R-tree.

2.2. PDR Tree

Indexing methods are also beginning to arise in the field of uncertainty. *Kanagal et al.* [4] proposed a structure for correlated probabilistic data based on junction trees *aka* clique trees [13]. *Cheng et al.* [3] and *Singh et al.* [7] developed another indexing method which are highly based on traditional R-Trees [14]. These trees are also called probabilistic distribution R-Trees (pdr-Tree) and they have different query processing mechanisms from top-k query processing. Our proposed authenticated data structure (PH-Tree) is based on pdr-Trees therefore we will detail this subject further.

In pdr-trees, each uncertain attribute is stored in a memory page with other similar attributes 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. Each page contains more than one uncertain attribute and items similar to the R-tree approach.

Figure 2.1 illustrates an example of pdr-tree. t_i are the tuple ids that are contained in the leaf pages only. MBRs are formed by the probability vectors. These vectors are included with their actual values instead of p_i notations. MBR boundaries describe the nodes of the pdr-tree. These boundaries are determined by vector $v =< v_1, v_2, v_3..., v_N >$

in R^N and v_i is the maximum probability of item d_i in any of the nodes indexed in the children of the current page. Left internal node for instance contains the maximum probabilities of d_1- d_2 and d_3- d_4 pairs respectively. $d_1.v =< 0.4, 0.3, 0.3 >$, $d_2.v =< 0.7, 0.1, 0.2 >$ and maximum values of these vectors are $< 0.7, 0.3, 0.3 >$ which is also the MBR boundary of their parent node. Uncertain attributes of the whole tree can be calculated by this way in a bottom-up manner, starting from the leaf nodes up to the root node. The pruning property of R-trees are maintained this way. If the MBR boundary does not satisfy the incoming query for any of the internal nodes, then we can assume that none of its children will qualify for the query. Also, the area of the MBR can be calculated by using any of the distance functions described in Chapter 2.1.

To insert a new entry into a page, MBR information is updated first according to the uncertain attribute. The subtrees of the page are investigated and the node which ends up with the minimum area increase and most similar MBR is chosen.

Distributional similarity threshold query is implemented in pdr-trees in a straightforward manner. A depth-first search is carried out via pruning the MBR boundaries. If the query has been satisfied for a given MBR, then the subtrees of that page are also entered, otherwise the page is pruned out. When we go down to the leaf level, the pages which qualifies for the query are included in the result set.

2.3. Authenticated Query Processing

Authentication has been studied before by *Merkle et al.* [8] and he proposed a novel method called Merkle-Hash Tree (MH-Tree). This research is considered as the fundamental work for a wide range of authentication methods and structures [6]. Merkle B-Tree has been proposed by *Marios et al.* which is the combination of MH-Tree and B+tree [15]. Authentication also has been extended to spatial data domain by several works recently [16, 17]. These researches are also important for the database outsourcing literature because they enable the implementation of range query processing in a multi-attribute environment.

Hash functions are essential to the authenticated query processing. A hash function H can be described as easy to compute but difficult to invert. If $H(x) = y$ then it is easier to compute y with the given H and x, but very difficult to compute x with given y and H [18, 19]. Also H should be able to compress a large input into a smaller and fixed output size. H should also satisfy the following properties.

- H function should be applied to any size of arguments.
- 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 x, it should be computationally infeasible to compute y for $H(x) = y$.

In order to execute authentication, hash functions are vital in this manner. It is possible to authenticate x by computing $H(x) = y$, considering the properties described above. No other input can generate y value and even a large value of x can produce a small fixed output of y. This is very crucial for the authentication of large amounts of data.

After hash functions are examined in detail, we continue with MH-Trees which are the state-of-the-art indexing structures for the authentication literature. Figure 2.2 illustrates an example of the MH-Tree.

Hash values h_i are computed as $H(d_i)$ such as $h_1 = H(d_1)$, $h_2 = H(d_2)$ etc. Internal node hash values are computed by concatenating the child nodes' hash values. 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 h_r 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 and efficient.

While executing the queries, the tree is being pruned out starting from the root. If the query qualifies for the given node, then it goes one level down for the subtree, otherwise hash value is included in the verification object. The traversing order is

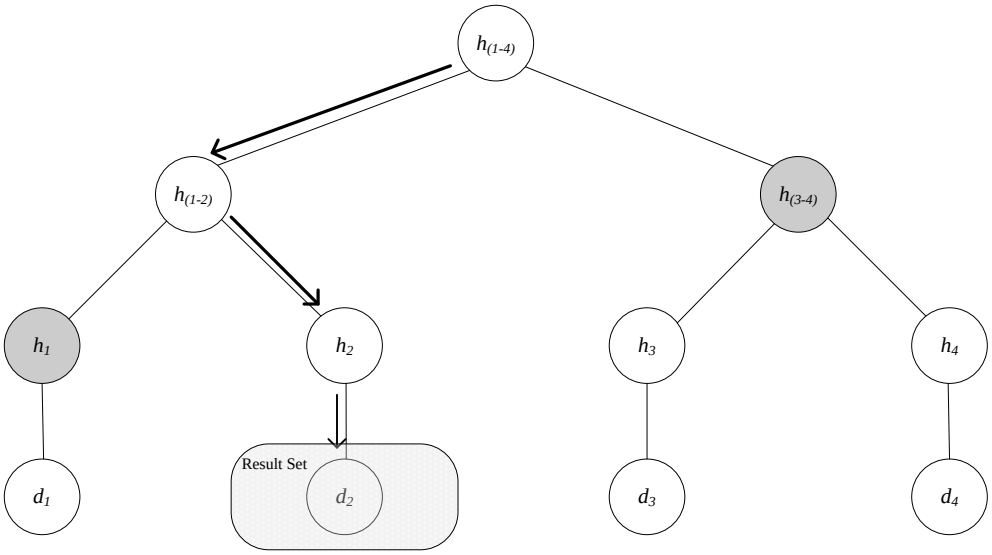

Figure 2.2. Merkle hash tree.

important in order to achieve a successful authentication, because concatenation is made during the tree construction. Verification object should provide the hash values exactly in the same order while the tree is being built. Only by this way we can provide the same input into the hashing functions. Therefore, the tree is traversed in a depth-first approach with preordering satisfied.

For the given query, assume that the result set only includes d_2 from Figure 2.2. Arrows are included to show the path of the query from the root to the leaf. Algorithm starts from the root node and qualifies for the left child node containing the hash value $h_{(1-2)}$. After that, depth-first search encounters with h_1 and includes that value to the VO, because result set does not contain d_1. d_2 is included to VO later, and algorithm finishes by adding $h_{(3-4)}$. The final state of $VO = [[h_1, [d_2]], h_{(3-4)}]$. In the client side, authentication will be done with 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 h_r which means authentication has been done successfully at the client side.

2.4. k-Means Clustering

The last subject that should be mentioned here is k-means clustering [20], which is also the essential part of our algorithm. Basically, clustering methods are unsupervised machine learning techniques, but they can also be used as preprocessors for different types of applications. The most common type of clustering methods are k-means clustering and expectation-maximization algorithm [21]. For the sake of simplicity, we will be concentrating on k-means clustering method.

Intuitively, this algorithm is trained to find out the k amount of points that represents data most. These data points are mean values (denoted as m_n) of the clusters relying under the data. In order to gather these mean values, an iterative approach should be carried out to minimize the error. This *reconstruction error* is proportional to the distance, $||d_i - m_n||$. Before going into detail of the algorithm, the primitives of this technique should be defined.

Reconstruction error can be calculated as follows:

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

where:

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

b_n^i are the labels for any point in the dataset. The idea here is to find the closest mean value to the data item so we are looking for the $m_n$value that minimizes the Euclidian distance. The best m_n values are those that minimize the overall reconstruction error. Solving this optimization problem analytically via above equations is not feasible, because both b_n^i and m_n values are dependant on each other. That is why iterative procedures are used for k-means clustering. First, m_n values are randomly picked up from the initial dataset. 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.

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

When we calculate m_n values, we recompute b_i^n values according to the new mean values, which is the start of second iteration. These two steps are repeated until m_n values converge. After convergence, all the centers cover the underlying clusters of the data. Figure 2.3 illustrates this algorithm in detail.

In this example, two different labels are denoted as dot character "." and plus character "+". m_n values are shown as star characters "*". Initially, some random data items are picked up as mean values. Here we have only two mean values which means in the end we are going to have just two clusters. At each iteration m_n values are updated and b_n^i labels are re-estimated according to the updated means. At first, we had only few data items labelled as dot "." and most of the data were labelled as "+". However, at fourth iteration we can see that the amount of both labels are nearly balanced. The pseudocode of k-means is given in Figure 2.4.

Each iteration consists of steps from two to nine. Our equations that are told before are implemented in step four and step seven. In step four, b_n^i is updated according to the m_n and in step seven, m_n values are updated with respect b_n^i. As it is told before, these iterations are performed until m_n values are converged.

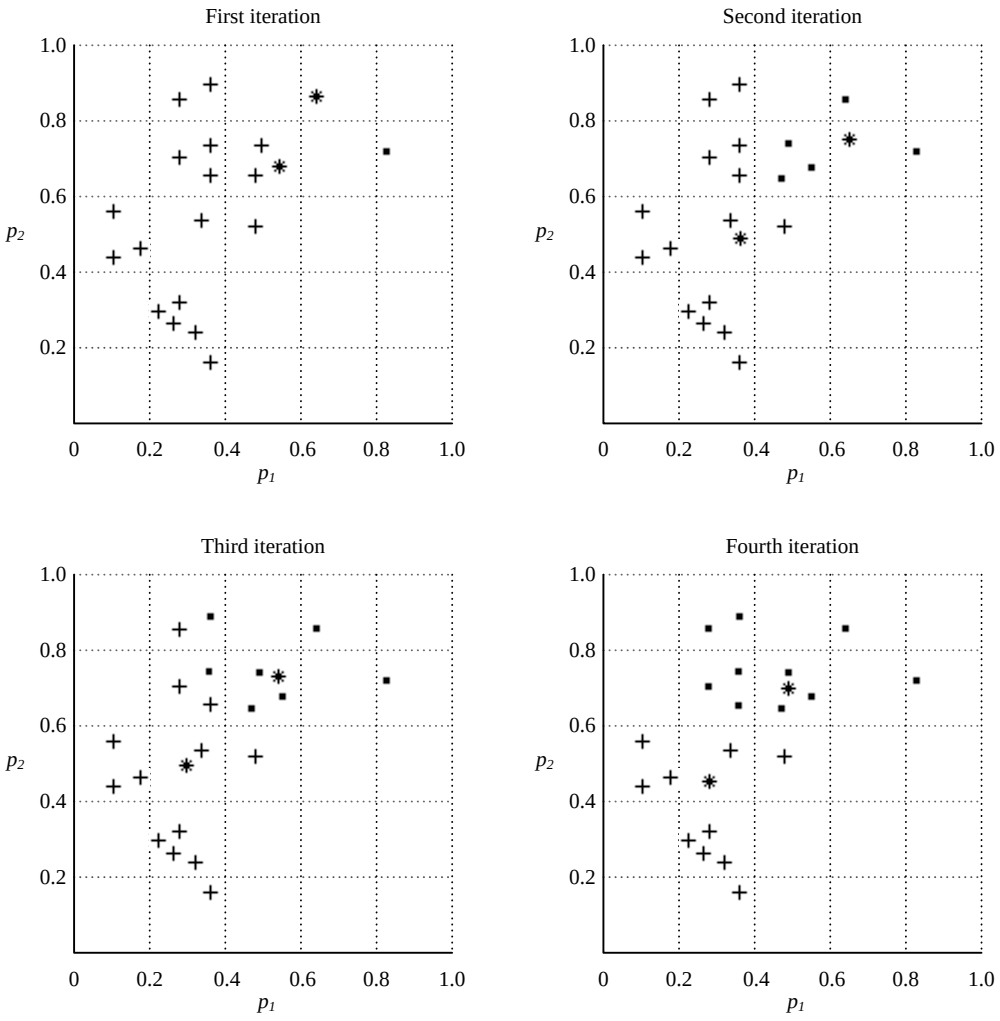

Figure 2.3. k-means clustering.

kmeans(k)

1. initialize m_i values, where i = *1,2...k*
2. while m_i is not converged
3. for each d_i
4. update b_n^i values with respect to m_i
5. end-for
6. for each m_i
7. $m_n = \sum_i b_n^i d_i / b_n^i$
8. end-for
9. end-while

Figure 2.4. k-means algorithm.

3. PH-TREE

3.1. Tree Structure

In this thesis, we are presenting a novel structure called Probabilistic Hash-Tree. Basically PH-Tree is the combination of MH-Tree and pdr-Tree with a preprocessing mechanism of clustering. This clustering algorithm also plays an essential role for the tree creation. As it is mentioned before, the mean values gathered from clustering are stored in the root node of the PH-Tree. Below figure shows an example of PH-tree and its internal node structure.

In the figure the clouds represent the clusters and their corresponding sub PH-Trees. These clusters are notated as C_1, C_2, C_3, C_4 respectively. However, due to the window size problem, only C_1 and C_4 are included in the figure. Each cluster is created as a sub PH-Tree. Note that the root of these subtrees are not the same as the root of the whole tree. Mean values of each cluster are notated as m_n and stored in the root node. These m_n values are the output of k-means clustering algorithm. After all the clusters are formed as a sub ph-tree and m_n values are included in the root, all subtrees (clusters) are connected to the root node.

In PH-Tree, only leaf nodes hold the data, just like pdr-Tree and MH-Tree. Data values are notated as d_i and i corresponds to the node index in the figure. Similarly hash values are contained inside the leaf nodes with the notation of h_i. We use Tiger Hashing algorithm to compute $h_i = H(d_i)$. Probability values are stored within $p_{i,j}$ variables. These probability vectors also determine the MBR of that node.

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, we look for the child $p_{i,1}$ values and the retrieve the maximum of them to find the value of $p_{6,1}$. All the probabilities up to the root are evaluated by this way. This ensures that if the query q does not qualify for the given

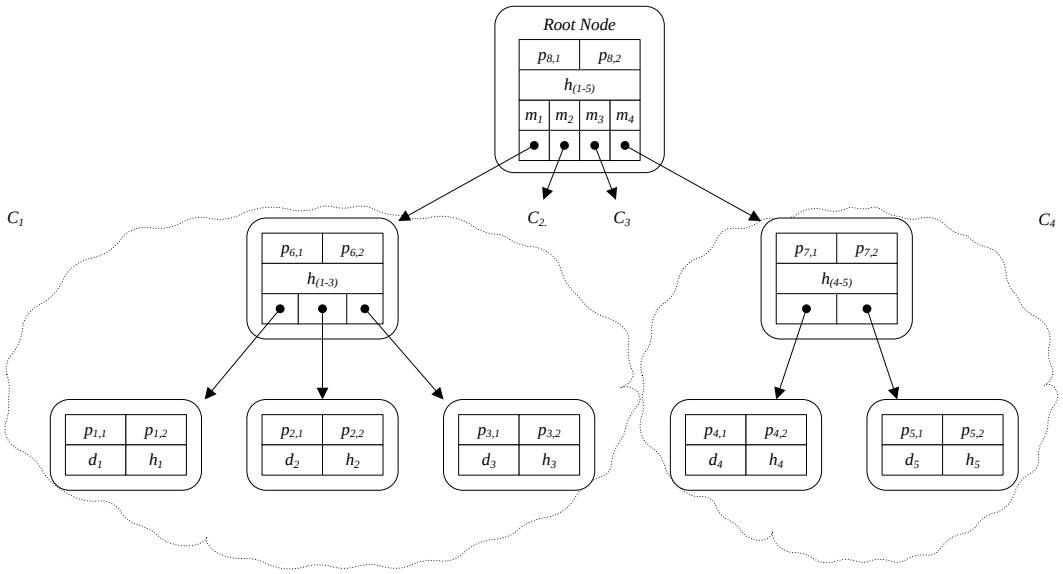

Figure 3.1. Probabilistic hash tree.

MBR, then it will also not qualify for the child nodes' MBR [7].

Hash values of internal nodes are computed similar to the MH-Tree computation. All the binary hash strings in the child nodes are concatenated in a breadth-first manner and re-hashed. E.g., $h_{(1-3)} = H(h_1|h_2|h_3)$ where "|" is the concatenation. Similarly, $h_{(4-5)} = H(h_4|h_5)$ and $h_{(1-5)} = H(h_{(1-3)}|h_{(4-5)})$.

3.2. Range Query Algorithm

The query processing algorithm in PH-Tree differs slightly from the other methods. The algorithm has been modified according to the tree structure based on clustering.

RangeQueryRoot Algorithm processes the incoming query q by calculating the Kullback-Leibler divergence with the mean values [22]. The least divergent subtree is the nearest cluster to the 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 of 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 let *RangeQuery* algorithm to go into

```
RangeQueryRoot(Query q)
1.   for each child N_i of root
2.       d = KL(q, m_i) // Kullback Leibler Divergence, m_i is mean value of N_i
3.       if ( d < SmallestDistance)
4.           SmallestDistance = d
5.           ClosestNode = N_i
6.           ClosestNodeIndex = i
7.       end-if
8.   end-for
9.   i = 0
10.  while ( i < ClosestNodeIndex)
11.      N_i = i^th child of root
12.      Append h_i to VO // h_i is hash value of N_i
13.  end-while
14.  Append "(" to VO
15.  RangeQuery(q, ClosestNode)
16.  Append ")" to VO
17.  i = ClosestNodeIndex
18.  while ( i < SizeOfRoot) // until we reach the last child of root node
19.      N_i = i^th child of root
20.      Append h_i to VO // Again this is the hash value of N_i
21.  end-while
```

Figure 3.2. Range query root algorithm.

any unnecessary subtrees at the root level. There are two distinct advantages of this method. (i) Decreases the VO size significantly, (ii) Improves verification time at the client side. The results of the experiments also validate our arguments.

The order of the concatenated hash bytes is important so 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 of *RangeQuery* procedure, we should include the final item in VO, hash value of N_4.

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

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

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

3.3. Root Authentication

Figure 3.4 illustrates the authentication in the client side. The hash value computed with the verification object should match the given h_r to verify both soundness.

Basically, there are no radical changes in the authentication procedure. The hashing is done after the "-" character appears in the VO. Otherwise it is concatenated to the hash string. If the entry is "+" character, then RootAuthentication is called again

```
RangeQuery(Query q, Node N)
1.    if N is leaf node
2.        for each child node C_i of N
3.            if ≪ C_i.p, q ≫< τ // C_i.p is probability vector of C_i
                // τ is predefined threshold
4.                Append C_i.d to the VO // C_i.d is the data item in C_i
5.            else
6.                Append C_i.h to the VO // C_i.h is the hash value in C_i
7.            end-if
8.        end-for
9.    else
10.       for each child node C_i of N
11.           if ≪ C_i.p, q ≫< τ
12.               Append "(" to VO
13.               RangeQuery(q, C_i)
14.               Append ")" to VO
15.           else
16.               Append C_i.h to the VO
17.           end-if
18.       end-for
19.   end-if
```

Figure 3.3. Range query algorithm.

```
RootAuthentication(VerificationObject VO)
1.   h_c = null // concatenated hash values will be stored in this variable
2.   while VO is not empty
3.       Entry e_i = next entry in VO
4.       if (e_i is data item)
5.           h_c = h_c|H(e_i) // "|" denotes concatenation operator
6.       if (e_i is hash value)
7.           h_c = h_c|e_i
8.       if (e_i is "+") // "+" query goes one level down here
9.           h_c = h_c| RootAuthentication(VO) // returns hash value
10.      if (e_i is "-") // "-" query goes one level up here
11.          break
12.      end-if
13.  return H(h_c)
```

Figure 3.4. Root authentication algorithm.

to enable recursion inside the VO.

Proof 3.1: *Assume that a data item d is bogus or modified in the result set. Because the hash-function is collision-resistant and d is used by RootAuthentication algorithm, the recomputed value of h_{root} will be the different than the one in DO and it cannot be verified by the client side.*

4. EXPERIMENTS & RESULTS

We have tested our algorithms by using a synthetic dataset and real dataset. The programming language used in the implementation is Java 1.6 and experiments are performed on a 2.1GHZ dual processor machine with a 2GB physical memory. Page size has been fixed to 8KB in all of the experiments. Additionally, we gained benefit from GNU Crypto which is a free cryptography library written in Java [23]. It enabled us to implement Tiger Hashing in range query and authentication algorithms.

Synthetic Data contains probability values generated from a uniform distribution. There are only two uncertain attributes in this dataset. Data fields are also randomly generated as 100 byte strings. Note that the focus of this work is the authentication part, so the relations between the data are not evaluated in the experiments. The cardinalities of this dataset are 10000, 20000, 30000, 40000 and 50000 respectively.

Adult is a real dataset that has been taken from UCI Machine Learning Repository [10]. This dataset has been created for classification purposes and cited in various papers. The attributes of *Adult* are 14 different social attributes such as *age*, *workclass*, *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. In this experiments, we are not interested in predictive methods, however we have to somehow introduce the uncertainty in the data. Since the *income* itself is not a certain attribute, posterior probabilities are calculated according to the prior attributes. We have chosen ADTree [24] to estimate these probabilities, because ADTree is very useful and efficient when the data contains both continuous and discrete attributes. Weka Data Mining Software [25] is used during the implementation of ADTree. The cardinalities of the adult dataset in the experiments are defined as 5000, 10000, 15000, 20000 and 25000 respectively.

We have compared our method with MR-Trees, which is another multi attribute R-tree variant authentication technique. The difference between MR-Tree and PH-Tree is that MR-Tree is based on spatial data while PH-Tree concentrates on uncertain data.

Another important notion here is optimization of k in clustering. Since it is a hyperparameter, repeating the experiments with different values of k 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. *Adult* has an optimum of 12 as k and synthetic dataset has an optimum of 30. Because of the uniform distribution of the synthetic data, it is very normal to see a large k for this dataset.

4.1. Verification Object Size

Figure 4.1 shows the results of our experiments in terms of VO size. It can be easily seen that PH-Tree outperforms MR-Tree. Because of the clustering mechanism used in PH-Trees, we have been able pre-eliminate the obsolete entries in the VO. Another important detail in this figure is that the slope (tangent) of PH-Tree line is smaller than MR-tree line. By this argument, we can deduce that if we increase the dataset cardinality, then the difference between PH-Tree and MR-Tree in terms of VO size will increase.

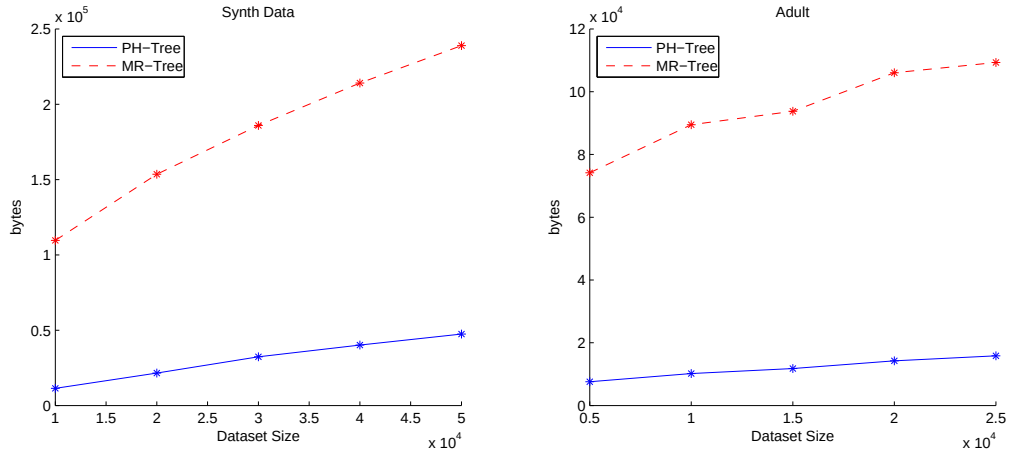

Figure 4.1. VO size.

4.2. Verification Time

Here, we have compared the results of MR-Tree and PH-Tree in terms of verification time at the client side. We are expecting the results to be in parallel with the VO size and Figure 4.2 proves our assumption. The advantage of smaller VO size makes it easier to authenticate the data for the client. The only difference between PH-Tree and MR-Tree in this comparison is the VO size, so Figure 4.1 is very similar to the Figure 4.2. PH-Tree again outperforms MR-Tree and it is becoming much more evident when the dataset cardinality increases for both synthetic and real data. Recall that the most important criteria for the authentication methods were VO size and verification time, because they were related with the client side. Up to this time, we have successfully demonstrated that PH-Tree works better than MR-Tree for the client.

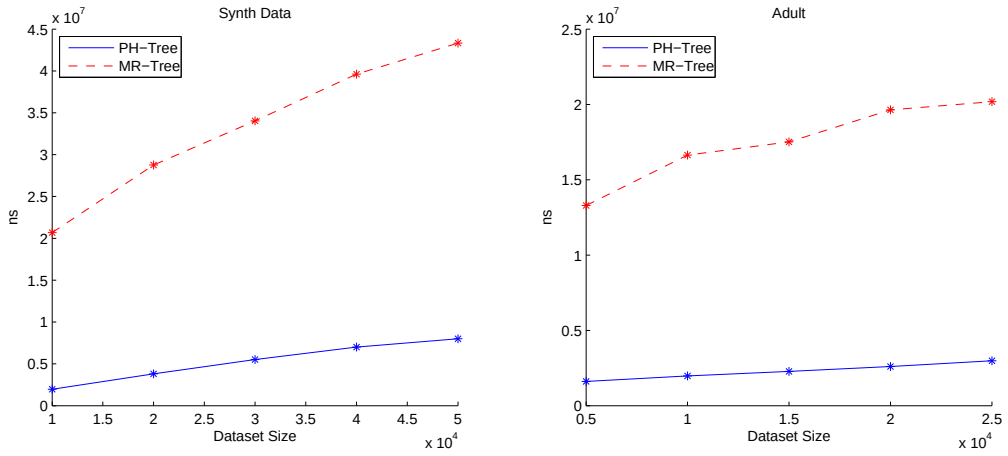

Figure 4.2. Verification time.

4.3. Query Processing Time

Figure 4.3 illustrates the query processing time for these two methods. Actually querying tests highly depend on the given dataset so we had to repeat the queries with different values and take the average of the total cost. In terms of query processing time, PH-Trees work better than MR-Trees which is an important criterion for the database service provider. The growth of the query cost does not change by the dataset cardinality for both of the algorithms. Nevertheless, PH-Tree query processing always works more efficiently than MR-Tree's.

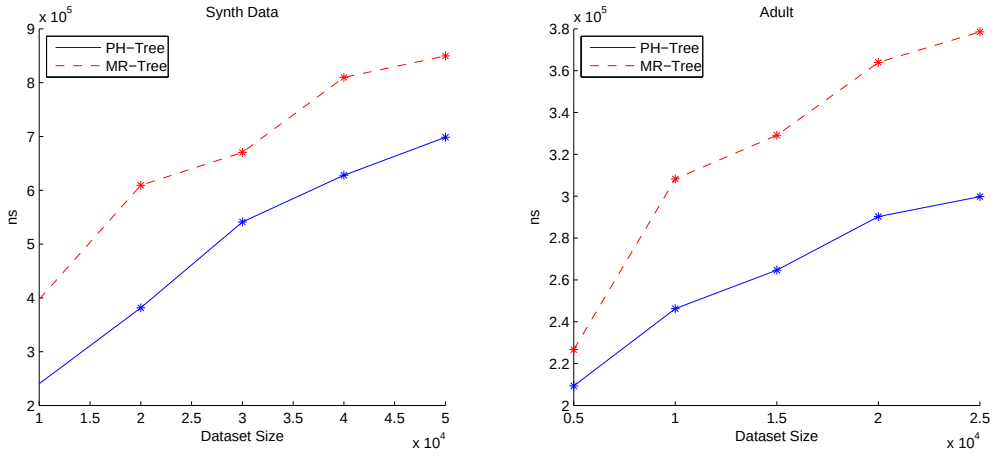

Figure 4.3. Query cost.

4.4. Tree Construction Time

Finally, Figure 4.4 investigates the tree construction time of these two methods. This metric includes both the time to create the initial tree and the time to compute the hash values of the nodes. It can be clearly seen that total construction time is exponential to the dataset cardinality. While building the initial PH-Tree, we have divided the whole dataset into k amount of clusters. By this way, we have 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 efficiencies. When we increase the data cardinality, there is a huge increase on the cost of MR-Tree because of the exponentiality. PH-Trees also grow exponentially but they are not as vulnerable as MR-Trees, because PH-Trees never start constructing the whole dataset.

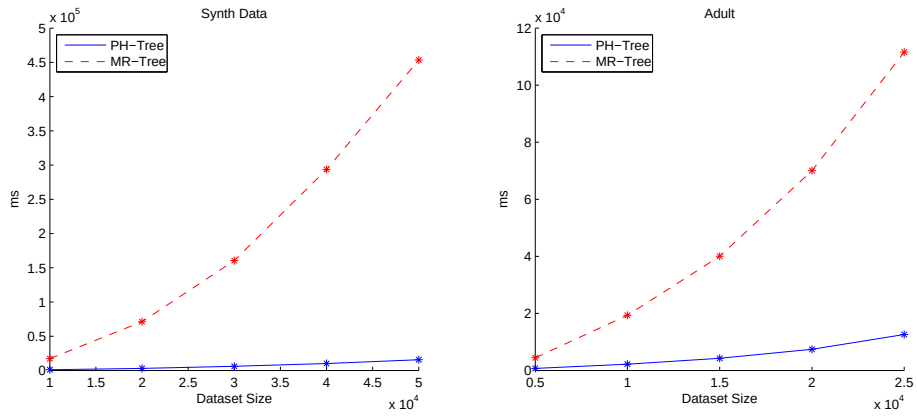

Figure 4.4. Tree construction time.

5. CONCLUSION & FUTURE WORK

In this work, we have proposed a novel ADS called PH-Tree for outsourced probabilistic databases which hasn't been covered before. In order to achieve this, we have created a hybrid of pdr-Tree and MH-Tree structures. A similar approach called MR-Trees do not work well on uncertain data. For this reason, we have embedded clustering mechanism into our algorithm. By including the k-means clustering, final structure of PH-Tree has been formed.

Three criteria are very important for the ADSs. These are query cost, verification object size and verification time. We have built our experiments on both synthetic and real data and showed that PH-Tree outperforms MR-Tree in all of the three criteria. Especially in terms of VO size and verification time, PH-Tree works much better than MR-Tree because the growth of these metrics by the dataset cardinality is smaller. As we increase the dataset size, the difference between the two methods becomes significant. Note that these two criteria are the ones that effect the client side. Additionally, we have also measured the tree construction time for DSP and showed that there is also significant difference between MR-Tree and PH-Tree in that metric.

The most vital future work will be changing the index structure for probabilistic data. pdr-Trees assume that the underlying uncertain data are independent. This work can be extended to the correlated data and include junction trees instead of pdr-Trees. Keep in mind that junction trees are more complex than pdr-Trees so authentication within them will not be an easy task to do.

The underlying clustering mechanism can also be improved. The weakness of k-means is that we pre-define the k value before executing the algorithm. *Leader cluster algorithm* [26], which is also a clustering variant, defines the k value dynamically. By this way we can predefine the number of subtrees more efficiently than the proposed fine-tuning.

Another idea about clustering would be changing the algorithm completely. We have used k-means clustering, the most prominent method for unsupervised learning. The research can be improved by also trying the expectation-maximization (EM) algorithm which uses Mahalanobis distance instead of the Euclidian distance.

REFERENCES

1. Antova, L., C. Koch, and D. Olteanu, "10^(10^6) Worlds and Beyond: Efficient Representation and Processing of Incomplete Information", *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, Turkey, 2007.

2. Soliman, M. A., I. F. Ilyas, and K. C. C. Chang, "Top-k Query Processing in Uncertain Databases", *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, Turkey, 2007.

3. Cheng, R., Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data", *Proceedings of the 30th International Conference on Very Large Data Bases*, Ontario, Canada, 2004.

4. Kanagal, B., and A. Deshpande, "Indexing Correlated Probabilistic Databases", *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, Rhode Island, USA, 2009.

5. Hacigumus, H., B. Iyer, and S. Mehrotra, "Providing Database as a Service", *Proceedings of the 2002 IEEE 18th International Conference on Data Engineering*, California, USA, 2002.

6. Yang, Y., D. Papadias, S. Papadopoulos, and P. Kalnis, "Authenticated Join Processing in Outsourced Databases", *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, Rhode Island, USA, 2009.

7. Singh, S., C. Mayfield, S. Prabhakar, R. Shah, and S. E. Hambrusch, "Indexing Uncertain Categorical Data", *Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, Turkey, 2007.

8. Merkle, R. C., "A Certified Digital Signature", *Proceedings of the 9th International*

Conference on Advances in Cryptology, 1989.

9. Dalvi, N., and D. Suciu, "Management of Probabilistic Data Foundations and Challenges", *Proceedings of the 2007 ACM SIGMOD 26th Symposium on Principles of Database Systems*, 2007.

10. Frank, A., and A. Asuncion, "UCI Machine Learning Repository: Adult Data Set", 2010, http://archive.ics.uci.edu/ml/datasets/Adult, Irvine, CA: University of California, School of Information and Computer Science, May 2011.

11. Ge, T., S. B. Zdonik, and S. Madden, "Top-k Queries on Uncertain Data: On Score Distribution and Typical Answers", *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, Rhode Island, USA, 2009.

12. Li, F., K. Yi, and J. Jestes, "Ranking Distributed Probabilistic Data", *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, Rhode Island, 2009.

13. Jensen, F. V., and F. Jensen, "Optimal Junction Trees", *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, 1994.

14. Guttman, A., "R-Trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, Massachusetts, USA, 1984.

15. Li, F., M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic Authenticated Index Structure for Outsourced Databases", *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, Illinois, USA, 2006.

16. Cheng, W., H. Pang, and K. L. Tan, "Authenticating Multi Dimensional Query Results in Data Publishing", *International Federation for Information Processing Workshop on Database Security*, 2006.

17. Yang, Y., S. Papadopoulos, D. Papadias, and G. Kollios, "Spatial Outsourcing for

Location Based Services", *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, Cancun, Mexico, 2008.

18. Evans, A., W. Kantrowitz, and E. Weiss, "A User Authentication Scheme Not Requiring Secrecy in the Computer", *Communications of the ACM*, Vol. 17, No. 8, pp. 437-442, 1974.

19. Wilkes, M. V., *Time Sharing Computer Systems*, Elsevier Science, Inc., New York, NY, USA, 1975.

20. Lloyd, S. P., "Least squares quantization in PCM", *IEEE Transactions on Information Theory*, Vol. 28, pp. 129-137, 1982.

21. Dempster, A. P., N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm", *Journal of the Royal Statistical Society*, Vol. 39, No. 1, pp. 1-38, 1977.

22. Kullback, S., and R. A. Leibler, "On Information and Sufficiency", *The Annals of Mathematical Statistics*, Vol. 22, No. 1, pp. 79-86, 1951.

23. The GNU Project, *GNU Crypto - GNU Project - Free Software Foundation*, 2001 http://www.gnu.org/software/gnu-crypto/, March 2011.

24. Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update", *SIGKDD Explorations*, Vol. 11, Issue 1, pp. 10-18, 2009.

25. Freund, Y., and L. Mason, "The Alternating Decision Tree Learning Algorithm", *Proceedings of the International Conference on Machine Learning*, 1999.

26. Alpaydin, E., *Introduction to Machine Learning*, MIT Press, Cambridge, MA, USA, 2004.