Framework for Managing Uncertain Distributed Categorical Data

Adel Benaissa, Mustapha Yahmi, Yassine Jamil
Université Paris Cité Sorbonne
Laboratoire informatique Paris Descartes
45, Rue des saints pères
Paris, France

***Abstract*—In recent years, data has become uncertain due to the flourishing advanced technologies that participate continuously and increasingly in producing large amounts of incomplete data. Often, many modern applications where uncertainty occurs are distributed in nature, e.g., distributed sensor networks, information extraction, data integration, social network, etc. Consequently, even though the data uncertainty has been studied in the past for centralized behavior, it is still a challenging issue to manage uncertainty over the data *in situ*. In this paper, we propose a framework to managing uncertain categorical data over distributed environments that is built upon a hierarchical indexing technique based on inverted index, and a distributed algorithm to efficiently process queries on uncertain data in distributed environment. Leveraging this indexing technique, we address two kinds of queries on the distributed uncertain databases 1) a distributed probabilistic thresholds query, where its answers satisfy the probabilistic threshold requirement; and 2) a distributed top-k-queries, optimizing, the transfer of the tuples from the distributed sources to the coordinator site and the time treatment. Extensive experiments are conducted to verify the effectiveness and efficiency of the proposed method in terms of communication costs and response time.**

***Keywords*—Distributed uncertain data; Top-k query; threshold query; indexing; categorical data**

I. Introduction

In recent years, data has become uncertain due to the flourishing of advanced technologies that participate continuously and increasingly in producing large amounts of incomplete data, data with missing values and uncertain data. When we talk about these types of data, some questions that come to mind are: 1) What are uncertain, imprecise and incomplete data? 2) How can we represent these data? 3) What is the difference between them? 4) How can we manage these types of data? 5) Can we use the traditional relational database management systems to store and request these data?. Therefore, Researchers of the database community have been asking these questions since the early 1980's. In fact, managing uncertain data has seen a revival in recent years due to new methods and the emergence of applications that produce this type of data. This is what prompts many challenges in terms of modeling, storing, indexing and querying uncertain data. Thus, many efforts have been devoted to studying uncertain databases. These efforts yield different approaches and algorithms for modeling and representing uncertain data [1–3], indexing techniques and query processing over uncertain data [3–9].

Over the last decade, many cases where uncertainty arises have been distributed in nature, e.g. distributed sensor networks and multiple data sources for information integration [10–12]. Unfortunately, existing techniques that include indexing and query processing over uncertain data were mainly proposed in centralized environments and are not adaptable to distributed environments. As a result, it is still challenging to efficiently process queries over distributed uncertain data. Notable exceptions include recent work on indexing and query processing of distributed uncertain data [10–13]. These works have only considered top-k queries on uncertain real-valued attributes. Distributed top-k query processing focuses on reducing communication cost while providing high quality answers. However, in many domains, data records are composed of a set of descriptive attributes many of which are neither numeric nor inherently ordered in any way. In this paper, we address the problem of indexing and query processing on uncertain categorical data in distributed environments. We propose an original approach that efficiently answers queries on distributed uncertain data with minimum communication and processing costs.

Example 1.1: Let us consider that $Farm$ is a relation that stores bovine records of breeders and veterinary surgeons as it is explained in the motivating example of the general introduction. The Relation $Farm$ is specified by the schema $Farm(T_{id}, weight, illness)$ where the *illness* attribute specifies the illnesses that can affect a cow. *Illness* attribute is an uncertain one that takes its values from the categorical domain $\{mc, fa, nc, fs\}$ where *mc* means *mad cow*, *fa* means *fever accurate*, *nc* means *normal cow*, *fs* means *fever simple*. Let R be a relation instance of $Farm$. In Fig. 1, the relations R_1, R_2, R_3 and R_4 are the horizontal partitions of R on S_1, S_2, S_3 and S_4 distributed sites respectively. The first tuple in R_1 specifies that the cow with the identifier T_1 has a weight 700 kg and that its possible illness maybe *acute fever (fa)* with the probability 0.7 or maybe *simple fever (fs)* with the probability 0.3. The veterinary surgeons are interested in locating which farm is affected by a disease in order to begin preventive measures.

From the previous Example 1.1, two interesting queries for the breeders and insurers are as follows:

$Q1$: *Find the cows affected by an accurate Fever with a probability above 0.5*;

$Q2$: *Find the two cows affected by an accurate fever in all the farms.*

The query $Q1$ aims to identify the tuples where the illness attribute has the value *fa* and The query $Q2$ aims to return the first 2 tuples with the highest probability, where the

T_{id}	Weight	Illness
T_1	700	$\{(fa, 0.7); fs(0.3)\}$
T_2	710	$\{(fa, 0.9); fs(0.1)\}$
T_3	790	$\{(nc, 1)\}$
T_4	725	$\{(nc, 0.9); (fs, 0.1)\}$

(a) R_1

T_{id}	Weight	Illness
T_5	700	$\{(fa, 0.2); fs(0.8)\}$
T_6	710	$\{(fa, 0.9); fs(0.1)\}$
T_7	790	$\{(nc, 0.85); (fs, 0.15)\}$
T_8	725	$\{(nc, 0.9); (fs, 0.1)\}$

(b) R_2

T_{id}	Weight	Illness
T_9	749	$\{(mc, 0.8); nc(0.2)\}$
T_{10}	645	$\{(mc, 1)\}$
T_{11}	801	$\{(nc, 0.7); (mc, 0.3)\}$
T_{12}	799	$\{(nc, 0.5); (mc, 0.5)\}$

(c) R_3

T_{id}	Weight	Illness
T_{13}	711	$\{(mc, 0.18); nc(0.82)\}$
T_{14}	745	$\{(nc, 0.9); (mc, 0.1)\}$
T_{15}	901	$\{(nc, 0.85); (mc, 0.15)\}$
T_{16}	799	$\{(nc, 0.95); (mc, 0.05)\}$

(d) R_4

Fig. 1. Example of distributed uncertain relation R.

illness attribute has the value *nc*. The two queries $Q1$ and $Q2$ are distributed probabilistic threshold and distributed top-k queries, respectively .

A straightforward and naive approach to answer such a query is to answer the above queries is to forward them to all the sites so that local treatment and processing will be done on each site locally. Then, each site send its results to the coordinator site. Finally the query site merges the tuples received from all sites and computes the final result. The drawbacks of this approach is that all sites are asked even when their tuples are not involved in the final response. Also, transferring a large number of tuples consumes bandwidth in the network and takes time to process. Furthermore, this approach does not scale with number of distributed sites. Hence, In order to address these drawbaks, we propose an approach that use a *Local Uncertain Index* (***LUI***) for uncertain data on each local site, while a *Global Uncertain Index* (***GUI***) is used to summarizing the local indexes. The local and global uncertain indexes are inverted-index based structures. We show that these structures support a broad range of probabilistic queries over uncertain data, including distributed uncertain threshold and top-k queries. Specifically, we propose `DUTh` and `DUTk`, two distributed algorithms for processing probabilistic threshold and top-k queries on distributed uncertain data. The main contributions of this paper are as follows:

- We propose a distributed indexing of distributed uncertain categorical data based on a two-level hierarchical index: a local index ***LUI*** on each site [7] and a top-level global index on a coordinator site (***query site***). The global index(***GUI***) summarizes local indexes and determines which ones should be accessed.
- We propose two distributed algorithms `DUTh` and `DUTk` to respectively answer threshold and top-k queries. Our proposed algorithms use the proposed indexes LUI and GUI to perform distributed pruning and allow minimum communication and processing costs.
- We conduct an extensive experimental study to evaluate our proposed framework over syntactic data. The results of the study show the efficiency of our two proposed algorithms and indexing techniques.

The rest of this paper is organized as follows: Related work is presented in Section II. Then, Section III presents the problem definition. In Section IV, we present an overview of our proposed framework. Section V presents our distributed indexing technique. In Section VI, we describe query processing using the distributed index. We report a performance evaluation of our proposed framework in Section VII and finally we conclude the chapter in Section VIII.

II. Related Work

With the advent of the Internet and network technology [14], there is an important emergence and unprecedented flourishing of real world applications and devices that participated to produce large amounts of uncertain data daily. e.g. data collected from sensor networks [15], information extraction from the web [16, 17], data integration[18, 19], data cleaning [20–25], social networks [26, 27], radio frequency identification RFID [7]. Due to various reasons that differ from one application to another, the uncertainty is inherent in such applications. With the emergency of these applications, considerable research efforts have been made in into the field of managing uncertain data [6]. Existing work in this area provides new models for uncertain data, prototype implementations, specific indexing techniques and efficient query processing algorithms.

Indexing uncertain data was extensively studied in the literature of centralized uncertain databases. Many approaches and a variety of indexing techniques were proposed in this field. However, these are not suitable for distributed uncertain data. To the best of our knowledge, there is only the work in [12] that proposed indexing uncertain data, presented as moving objects over a peer to peer (P2P) environment based on Quad-Tree indexes structures. In our work, we aim to propose an original indexing technique for uncertain data in general distributed environments and not specifically P2P environments.

As the same, many query processing techniques on distributed certain data have been studied with particular interest in top-k queries [5, 15]. However, these techniques are not adaptable to uncertain distributed environments. As a result, it is still challenging issue to efficiently process queries on distributed uncertain data. Thus, we aim to study query processing over uncertain distributed data based on efficient index structures.

Most existing work on top-k query processing in distributed environments performs with several rounds of communication between the query site and the other distributed sites. As a result, it is still challenging to efficiently process top-k queries in a minimum communication rounds between sites. In our work, we seek to reduce the communication cost of query processing in distributed environments and reduce it to one round of communication, in addition to dealing with uncertainty.

In the literature, the majority of works considered top-k queries on uncertain real-valued data. However, in many domains, data records are composed of a set of descriptive attributes, many of which are neither numeric nor inherently ordered in any way. Thus, we focus on indexing and querying distributed uncertain data presented as qualitative (or categorical) data.

III. Problem Definition

In this section, we describe the data model and query classes which we consider in our work. Then, we define the problem of query processing over uncertain categorical data on distributed environments.

A. Data Model

Uncertainty can be identified both at the tuple level [1, 3] and the attribute level [2]. In this work, we consider the attribute-level uncertainty model. The uncertain data in our work is modelled as a probabilistic database horizontally distributed over a set of sites $\mathcal{S} = \{S_1, S_2, ..., S_m\}$. A database consists of a set of relations R partitioned into (R_1,..., R_m) such that $R = \bigcup_{i\in[1,m]} R_i$ and $R_i \cap R_j = \emptyset$, for $i, j \in [1..m], i \neq j$. Consequently, each database on a local site S_i consists of a relation R_i, with n tuples each of which has uncertain attribute values. For the sake of simplicity and without loss of generality, we limit the discussion to relations with a single uncertain attribute. In our work, we focus on uncertain attributes that are drawn from categorical domains. Let $\mathcal{S}.a$ be a particular attribute in $\mathcal{S}$ which is uncertain. $\mathcal{S}.a$ takes values from the categorical domain D with cardinality $|D| = N$. For a traditional (certain) relation, the value of an attribute a for each tuple would be a single value in D. In the case of an uncertain relation, $t_k.a$ is a probability distribution over D instead of a single value. Let $D = \{d_1, d_2, ..., d_N\}$, then $t_k.a$ is given by the probability distribution $Pr(t_k.a = d_i)$ for $i \in \{1, ..., N\}$. We illustrate our uncertainty model with the following example:

B. Basic Queries on Uncertain Data

We specifically examine two types of queries: distributed uncertain query threshold and top-k queries. They are defined below:

Distributed Uncertain Threshold Query (`DUTh`): Threshold queries can be used in different settings [28]. In the literature related to databases, there has been much interest in studying such queries [29]. Generally, most of these work said that the goal of this type of query is to detect all tuples (or objects) whose likelihood (or score) exceeds a given threshold. In our work, we defined a distributed threshold query as follows[1]:

Definition 3.1: Given a set of distributed uncertain relations $\mathcal{S} = \bigcup_{i\in[1,m]} S_i$, `DUTh` returns a set of tuples $t_k \in S_i$ such that $Pr(t_k.a = d_i) > \tau$, where τ is a probability threshold and $t_k.a$ is a probability distribution over D where $D = \{d_1, d_2, ..., d_N\}$.

Example 3.1: From the previous Example 1.1, an interesting query for the breeders and insurers is as follows:

$Q1$: *Find the cows affected by an accurate Fever with a probability above 0.5*

The query $Q1$ is distributed probabilistic threshold queries that aim to identify the tuples where the illness attribute has the value *fa* with probability above 0.5 (cf. Fig. 1) respectively, from the whole distributed relation $\mathcal{S}$.

Distributed Uncertain Top-k Query (`DUTk`): A lot of effort has been devoted to studying top-k query in the uncertain database [30]. The majority of these works considered two aspects for ranking (top-k) uncertain databases. These were the score and the likelihood of objects (or tuples). In our work, we consider only the likelihood attached to the tuples in the ranking process. Hence, the definition of the top-k query in our work is as follows:

Definition 3.2: Given a set of distributed uncertain relations $\mathcal{S} = \bigcup_{i\in[1,m]} S_i$, `DUTk` returns the k first tuples $t_k \in S_i$ with the highest probability among all distributed sites of $\mathcal{S}$.

Example 3.2: From the previous example 1.1, an interesting query for the breeders and insurers is as follows:

$Q2$: *Find the two cows affected by an accurate fever in all the farms*

The query $Q2$ is distributed probabilistic Top-k queries. The aim of query $Q2$, is to return the first 2 tuples with the highest probability, where the illness attribute has the value *fa*.

IV. Proposed Framework

Having presented the uncertain model and the query classes that we aim to process, let us now present, the main components of our framework. The principal aim of our approach is to efficiently answer queries on uncertain distributed data with minimum communication and processing costs. Our framework is performed in two main phases (Fig. 2):

Offline Phase

We propose a distributed indexing technique based on a two-level hierarchical index:

- ***Local uncertain indexes(LUI)*** : we build a local index at each local site. To do so, we adopt the inverted uncertain index structure proposed by Shin et al. [7].
- **Global uncertain index(GUI)**: This index summarizes the local indexes on the distributed environments.

[1]In our work, we do not use the family of threshold algorithms [28] for the reason that we treat queries with threshold probability in distributed environments in different contexts.

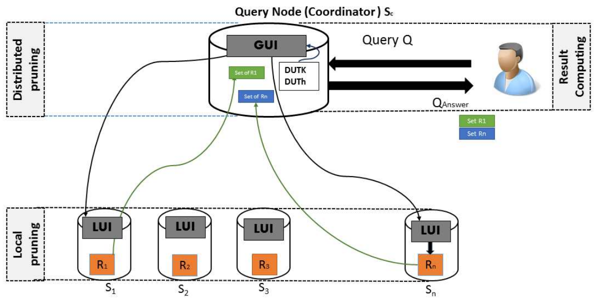


Fig. 2. Our proposed framework.

Its structure is the same as the local index structures LUI (Inverted list).

Online Phase

Using the local (LUI) and global uncertain indexes (GUI), we propose two distributed algorithms DUTh and DUTk to respectively answer distributed uncertain threshold and top-k queries. The main steps of the proposed algorithms are as follows:

- **DUTh (Distributed Uncertain Threshold Query)**: In this algorithm, only the sites whose tuples are involved in a final response are asked. DUTh performs the following main steps in one round of communication:
  - Distributed Pruning on the coordinator site.
  - Local pruning on each site.
  - Result computation on the coordinator site.
- **DUTk (Distributed Uncertain Top-k Query)** The main objective of this algorithm is to reduce the communication cost and time processing of the execution of the Top-k query processing. To do this, the top-k query will only be concerned with a subset from $\mathcal{S}$. The main steps of this algorithm are:
  - Distributed Pruning on the coordinator site.
  - Probability threshold tuning.
  - Local pruning and query ranking.
  - Distributed query ranking.

V. Distributed Uncertain Indexing

To index distributed uncertain data, we propose a two-level hierarchical index: a local index on each site as proposed in [7] and a top-level global index to determine which local indexes should be accessed. Hence, our indexing process is performed in two stages. In the first stage, each local site builds its local uncertain index. In the second stage, summaries from local indexes are merged to build a global index on a coordinator site where the query will be executed. In what follows we describe the main process used to build local and global uncertain indexes in detail.

A. *Local Uncertain Index Structure*

For the local uncertain index on each site we adopt the inverted index based structure proposed in [7]. The main build of this index is as follows:

Each value d_i of an uncertain attribute a is stored in the list of the inverted index, which consists of a set of pairs including the T_{id} tuple of the local relation R_i and the probability of $d_i.a$ attached to T_{id}. Therefore, the component of the list of a given $d_i \in D$ is a pair of (t_{id}, P_i). This list is organized in decreasing order. In practice, such a list is organized in the memory with a dynamic structure such as B+ tree. The main advantage of the local indexing is that most parts of the query processing can be performed by the site containing the required index data. The site responsible for solving the query is only required to broadcast the query to the coordinator site, which then combines the returned results.

Example 5.1: Returning to Example 1.1 of the uncertain distributed database R in Fig. 1. The local uncertain index LUI of R_1 (the local relation of R on S_1) is depicted in Fig. 3. It is built on the *illness* uncertain attribute. Notice that each entry of LUI corresponds to a value from the categorical domain of *illnesses*: $\{mc, fs, fa, nc\}$. For each of those values, a set of pairs is stored. For instance, the $\boldsymbol{nc}$ value is associated with the following pairs $(t_3, 1)$ and $(t_4, 0.9)$. The pair $(t_3, 1)$ specifies that the tuple with the t_3 identifier includes the value $\boldsymbol{nc}$ with the probability 1 in its *illness* uncertain attribute.

$$\begin{tikzcd}
\boxed{\boldsymbol{nc}} \arrow[r] & \boxed{(\boldsymbol{t_3}; \mathbf{1.0})} & \boxed{(\boldsymbol{t_4}; \mathbf{0.9})} & \\
\boxed{\boldsymbol{fa}} \arrow[r] & \boxed{(\boldsymbol{t_2}; \mathbf{0.9})} & \boxed{(\boldsymbol{t_1}; \mathbf{0.7})} & \\
\boxed{\boldsymbol{fs}} \arrow[r] & \boxed{(\boldsymbol{t_1}; \mathbf{0.3})} & \boxed{(\boldsymbol{t_2}; \mathbf{0.1})} & \boxed{(\boldsymbol{t_4}; \mathbf{0.1})}
\end{tikzcd}$$

Fig. 3. Local inverted index of R_1.

B. *Global Uncertain Index Structure*

Now, we describe the structure of the global uncertain index which we refer to as GUI. The GUI determines which local index(es) should be accessed. Hence, GUI should be stored in the coordinator site.

```
input  : $D = \{d_1, d_2..., d_n\}$;$\mathcal{S} = \{S_1, S_2..., S_n\}$;
output: GUI
for each site $\in S$ do
    Goes through the index LUI
    for each $d \in \{D \wedge LUI\}$ do
        $P_{max} \leftarrow$ maximal probability of d in LUI
        $Ld \leftarrow (S, P_{max}))$
    end
end
for each $d \in D$ do
    organize $ld$ as inverted list in decreasing order
end
return(GUI)
```
Algorithm 1: Global Index (GUI) Construction

Given a distributed relation R on $\mathcal{S}$, GUI is an inverted index based structure on the uncertain attribute a of R. It summarizes information of local indexes LUIs. More specifically, it stores the sites of each LUI. As for the local indexes, entries in

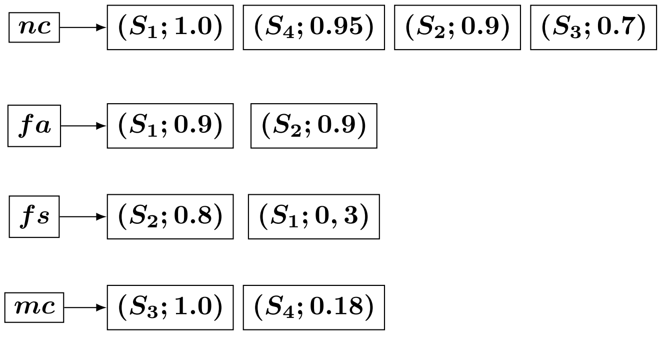


Fig. 4. Global inverted index (GUI).

a GUI correspond to the categorical domain values of a, i.e. $\{d_1, d_2..., d_n\}$. For each entry d_j, GUI collects information from the local indexes as shown below:

- Given a LUI stored on a local site S_i, for each entry d of LUI, GUI stores one pair (S_i, P_{max}) where $S_i \in \mathcal{S}$ and $P_{max}(R.a = d)$ is the maximum probability over the set of pairs (t_{id}, P_{max}) associated with d on the LUI of site S_i.
- To complete the list of pairs for each entry d_j in the GUI, all the LUI's are explored to obtain the corresponding pair for d_j as performed above. Once the list is completed, the pairs are ordered in decreasing order of their probabilities.

Algorithm 1 depicts the main steps of creating the GUI.

Example 5.2: Returning to Example 1.1 of the distributed database R in Fig. 1. The global uncertain index of our distributed environments is depicted in Fig. 4. In our distributed environment we have four values of the domain $D = \{nc, mc, da, fs\}$, For each of those values, a set of pairs is stored. Hence each value from domain D has an inverted list. For instance, the $\boldsymbol{fa}$ value is associated with the following pairs: $(S_1, 0.9)$ and $(S_2, 0.9)$. The pair $(S_2, 0.9)$ specifies that the illness $\boldsymbol{fa}$ is located in site S_2 with maximal probability 0.9. The Inverted list of $\boldsymbol{fa}$ indicates that this illness is only located within site S_1 and site S_2.

In what follows, we will discuss how to provide two kinds of queries on uncertain distributed sites using the proposed index structures.

VI. Distributed Uncertain Query Processing

In our uncertain distributed environment, the query can be initiated by the coordinator site, also called query site (S_c). The naive approach[2] to processing a distributed query requires transferring an excessive amount of data and processing the distributed query at each site. Instead, we present two algorithms DUTh and DUTk which improve the overall performance by reducing both the processing and communication cost of the threshold and top-k queries, respectively.

[2]We will call this approach NAIV in our experiments.

A. Distributed Uncertain Threshold Algorithm

The Distributed Uncertain Threshold Algorithm, referred to as DUTk, is depicted in Algorithm 2. The aim of our algorithm is to reduce the communication cost by pruning the sites that are not concerned by the query. Let us consider an uncertain

```
input : Query $Q$,Index $GUI$ ,Threshold:$\tau$
output: $DUTh_{response}()$
$DUTh_{response}() = \{\}$;
$Q_{answer} = \{\}$
Execute query Q in coordinator site;
Goes through the index GUI;
for each $S_i$ in index GUI do
    if $p_{max} > \tau$ then
        Execute query Q ;
        $Q_{answer} \leftarrow$ Answer of $Q$
    end
    $DUTh_{response}() \leftarrow DUTh_{response}() \cup Q_{answer}$
end
return($DUTh_{response}()$)
```
Algorithm 2: The DUTh Algorithm

database R distributed over a set of site $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ and query Q on $\mathcal{S}$. Note that the query Q will be initiated in the coordinator site with a given threshold probability (τ). The main steps of the DUTh algorithm execution are as follows:

- The query Q will be executed in the coordinator site $S_c \in \mathcal{S}$. The first step of DUTh, is to go through the global index (GUI) and visit the inverted list of associated value driven from the domain of categorical data initiated in the query Q.
- In this step, we get only the list of sites which have maximal probability greater than the defined threshold ($p_{max} > \tau$). All sites that cannot satisfy this condition will not be concerned by the query Q and will be pruned.
- The query will be sent to the concerned sites where it will be executed locally. Notice, that in each concerned site, pruning is performed locally.
- Each site forwards its results to the coordinator site, which will merge all the received results.

Example 6.1: We illustrate the main steps of DUTh by considering the query Q_1 from Example 3.1 that we will execute on the farms data depicted in Example 1.1 and Fig. 1:

$Q_1 : Select{*}from\ \mathcal{S}\ hereW\ illness =' fa'$ And $P > 0.5$

We suppose that we would have all cows that are affected by fa illness with probability ($p > 0.5$).

- The query Q_1 is executed in the coordinator site. In the first step of our DUTh, the GUI will be visited to obtain a list of sites(farms) where fa illness is present.
- Then, we visit the inverted list of fa. All sites that had maximal probability of illness q above $p < 0.5$ will be pruned. Consequently, in our case the query Q_1 will be concerned with farm S_1 and S_2.

- Next, we send the query Q_1 to S_1 and S_2 with threshold probability $\tau > 0.5$
- In the farms S_1 and S_2 the query Q will be executed and local pruning will be performed in each sites (R_1 and R_2).
- Only tuples with probability attached to value fa above 0.5 will be returned to the coordinator site S_c as presented in Table I.

TABLE I. RESULTS OF DUTH QUERY Q_1.

$R_i.T_{id}$	weight	illness
$R_1.T_1$	700	(fa,0.7)
$R_1.T_2$	710	(fa,0.9)
$R_2.T_6$	710	(da,0.9)

B. *Distributed Uncertain Top-k Algorithm*

Given an uncertain relation $R = \underset{i\in[1,n]}{\cup} R_i$ distributed on $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ where R_i is located on S_i. Each S_i stores a local index LUI and a coordinator site S_c stores a global index GUI. Given a top-k query Q issued from S_c, the algorithm DUTk will efficiently and effectively answer Q. We consider the following query Q on the distributed relation R. The class of queries we treat are obviously those with an uncertain attribute in the where-clause. The main steps of the execution of DUTk are as follows:

```
input : Query Q,Index GUI ,DUTh, k
output: $DUTk_{response}()$
$DUTk_{response}() = \{\}$;
Execute query Q in coordinator site;
crossing the index GUI;
$Li \leftarrow$ list of site from GUI
for each $S_i$ in liste $Li$ do
    explore LUI;
    $Lp_{min} \leftarrow$ get the probability of $k_{th}$ of site from
      Local index;
end
$\tau \leftarrow$ Max of $Lp_{min}$
//Execute Threshold algorithm DUTh($\tau$)
Global_response$\leftarrow$ Duth($\tau$)
//Merge and Rank the results, then choose the top-$k$
  response
$DUTk_{response}() \leftarrow$ Order(Global_response(limitk))
return($DUTk_{response}()$)
```
Algorithm 3: DUTk Query Algorithm

- **Distributed pruning:** On the coordinator S_c where Q is executed, the algorithm explores the global index GUI to find the uncertain attribute value of the where-clause, and then obtain the distributed sites in the corresponding list of that value. Let $S_Q = \{S_1, S_2, ..., S_h\}$ where $h \leq m$, is the list of the distributed sites where Q should be executed. The coordinator site S_c broadcasts the query Q to those sites. Notice that the GUI allows distributed pruning, i.e. only sites where Q should be executed are considered and the others are pruned. The main advantages of this pruning is reducing the query processing time and cost.
- **Probability threshold tuning:** On each site S_i in S_Q from the previous step, the algorithm explores its local indexes LUI. The aim of this step is to search the k^{th} probability P_i (i.e. the maximal probability in the LUI inverted list) from each selected site. Then, this probability value is sent to the query node where a threshold list Th_{List} stores all the probability values sent by the distributed sites. The maximal probability, $\tau = Max_{i\in[1,m]}P_i$ (maximal probability of the list (Th_{List}), is considered as the new probability threshold, then a new pruning is performed from the global index GUI according to τ. Indeed, for each pair (S_i, P_{max}) from the GUI inverted list, if $P_{max} \leq \tau$, then the corresponding site S_i is pruned. The result of this step is a new list S_{Q_τ} of distributed sites where Q should be executed where $S_{Q_\tau} = \{S_1, S_2, ..., S_l\}$ where $l \leq h \leq m$. Consequently, the coordinator site S_c sends the query Q to the sites in S_{Q_τ}.
- **Local query ranking:** On each site S_i in S_{Q_τ}, the algorithm executes Q. It explores each local index LUI and obtains the tuples from the corresponding inverted list of the uncertain attribute value. Each node ranks its own tuples in decreasing probability order, stopping once no more tuples are likely to satisfy k. Then the ranked k^{th} tuples are sent to the coordinator site.
- **Distributed query ranking:** On the coordinator site, the algorithm computes the final query result. Another ranking is performed on the whole top-k tuples of each site.

Example 6.2: Returning again to Example 1.1, we illustrate the main steps of DUTk using the distributed environments presented in Fig. 1. Let us consider the query Q_2 of Example 3.2 that we can rewrite as follows:

Q_2: Find the two cows affected by an accurate fever (fa) in all the farms.

Let us show the execution of the example step by step:

- **Distributed pruning:** First, the GUI is explored to find the value fa of the uncertain attribute $illness$. Only the farms S_1 and S_2 are concerned by the query Q, because the value fa is present in these farms. The farm S_3 and S_4 will not be concerned by query Q_2 for the reason that value fa does not exist in these farms, so distributed pruning is first conducted in this step. As presented in Fig. 4, the GUI will contain the highest probabilities for each node, that is $max(S_1) = 0.9$, $max(S_2) = 0.9$,
- **Probability threshold tuning:** Second, the query site sends the query to the concerned farms S_1 and S_2. The aim is to get the 2^{nd} $(k = 2)$ probability value fa after consulting their LUIs. Next, the farms S_1 and S_2 send the values of the 2^{nd} probability to the query site(($P_{2^{nd}ofS_1} = 0.7$) and ($P_{2^{th}ofS_2} = 0.2$)), which will define the maximal received value ($P_{max} = 0.7$) from S_1 as the new threshold probability. Then, given this threshold probability $\tau = 0.7$ the GUI will be

visited again and only the farm S_1 will be concerned by the query Q_2, S_2 will be pruned.

- **Locally query ranking:** Third, S_1 and S_2 will process the query locally and send the tuples that satisfy the threshold $\tau = 0.7$ to the query site. In this step only t_1,t_2 from S_1 and t_6 from S_2 will be retained.
- **Distributed query ranking:** Fourth, the query site will rank the received tuples in decreasing order of their probability and will finally choose the first two tuples as the result of top-2 of the query Q_2 (cf. Table II).

TABLE II. RESULT OF DUTK QUERY Q_2

$R_i.T_{id}$	Weight	Illness
$R_1.T_1$	700	(fa,0.9)
$R_2.T_6$	710	(fa,0.9)

From these steps, we have the following Lemma:

The DUTk algorithm effectively provides an effective top-k tuples with minimum cost communication and reduces the transferring of tuples between the coordinator site and the other sites. It executes the process in two rounds of communications between the query site and the other sites.

VII. Experimental Study

We experimentally evaluate the performance of our proposed algorithms DUTh and DUTk. We study the effect of the different parameters for both of the proposed algorithms and we compare their results with the NAIV approach to show the efficiency of the pruning phase.

We conduct an extensive experiment on a synthetic data sets. In particular, we have generated uncertain database *Farm* with three attributes (Fig. 1). The uncertain attribute *Illness* has 60 possible domain values ($|D| = 60$). The data sets follow two different distributions 1) a *pairwise* distribution where the probabilities for an illness are chosen randomly from [0,1]; and 2) a *Zipf* distribution over probabilities with the default skewness 1.2.

Next, we distributed the database horizontally over 50 sites where each site contains more than 230 000 tuples. To better show the performance of our algorithms and the efficiency of their pruning phases, we ensure that some categorical values do not appear in all sites. For this, we ensure that the distribution of the maximal probabilities of these categorical values are different in each site and their probabilities are in the range [0.1, 1].

We implemented DUTh,DUTk and NAIV in R and C# and tested them using a simulated distributed environment where each node has an Intel Core TM processor CPU 3.40 GHZ with 8 GO RAM. We measure the total communication cost in terms of number of transferred tuples. Then, we measure the response time as the time elapsed between sending the query from the query site and receiving all the responses in the query site. Each experiment is repeated 10 times in order to calculate an average time.

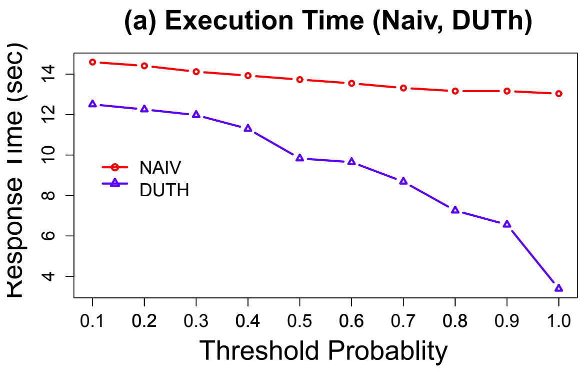
Table III shows the experimental setting.

TABLE III. EXPERIMENTAL SETTING

Symbol	Meaning	Range
m	number of nodes	$\{10, 20, 30, 40, 50\}$
k	size of top-k	$[10, 100]$ and $[100.1000]$
τ	threshold value	$[0, 1]$ and $[0.91, 1]$
N	size of database on each node	above 230000 tuples

A. Efficiency of DUTh

In this subsection, we show the efficiency of the proposed DUTh algorithm. In particular, we compare the DUTh algorithm with the NAIV algorithm. We conducted experiments on the two types of data sets and we consider the response time of DUTh compared with the naive algorithm (NAIV) that sends the query Q to all nodes.



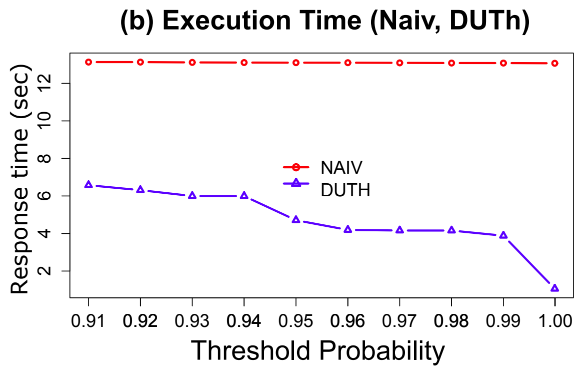
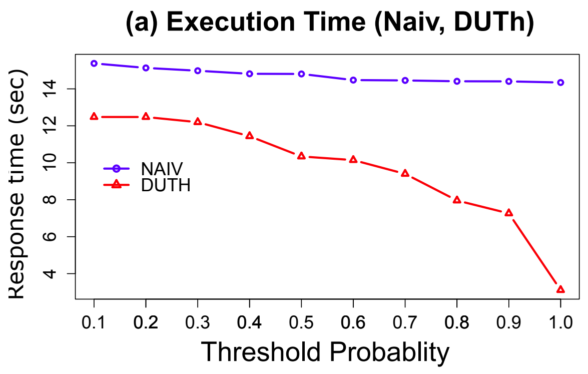


Fig. 5. Efficiency of DUTh with pairwise distribution by varying τ.

In these experiments, we varied the value of threshold τ (Table III). In Fig. 5(a) and 6(b), the threshold τ is in the range $[0.1, 1]$. For Fig. 6(b) and 5(b), it is in the range $[0.9, 1]$ with varying step 0.01. We observe that execution time for the DUTh algorithm decreases with the increase of the value of τ. For the highest value $\tau = 1$, the total response time is about 1.65 sec for DUTh and 13.79 sec for NAIV.

We note that the difference between the response time of DUTh and NAIV is due to the fact that the number of sites pruned are more important when the threshold probability is greater. When a threshold probability increases the number of pruned sites increases as well for the two data distributions.



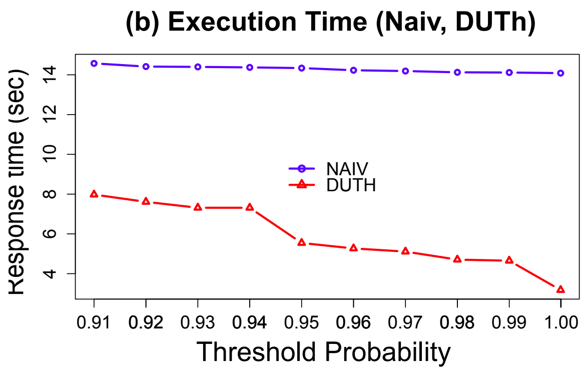


Fig. 6. Efficiency of `DUTh` with Zipf distribution by varying τ.

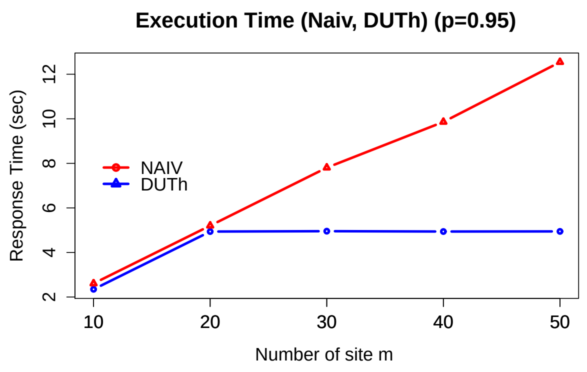
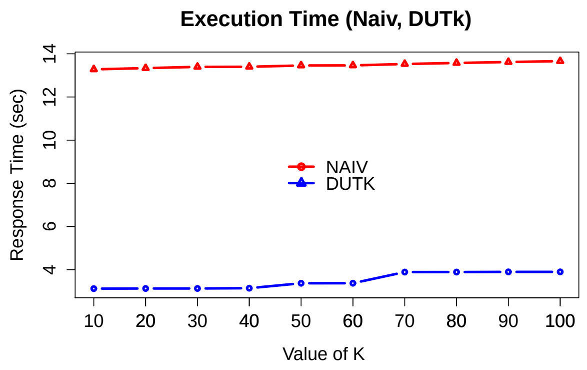


Fig. 7. Scalability of `DUTh` with Zipf distribution by varying m.



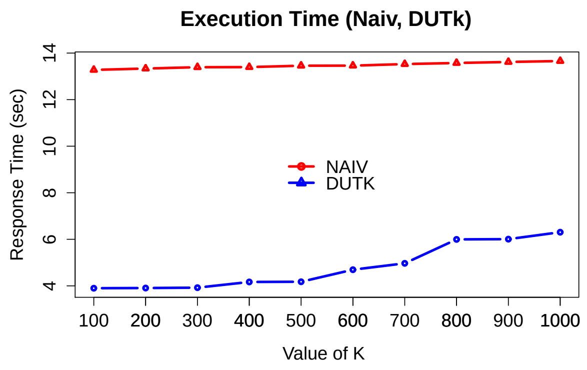


Fig. 8. Efficiency of `DUTk` with pairwise distribution by varying k.

B. *Scalability of* `DUTh`

In this experiment, we study the scalability of the `DUTh` algorithm with respect to the number of distributed nodes using Zipf distribution. To do so, we fixed the size of the whole distributed data and we varied the number of distributed sites from 10 to 50. Fig. 7 shows the results of the fixed threshold probability ($\tau = 0.95$).

We note that the gap between `DUTh` and `NAIV` when $m > 20$ increases as m increases due to the fact that only a subset of sites is requested based on GUI (only 20 sites have maximal probability of value d above 0.96). Consequently, the number of sites who have a maximal probability less than threshold τ, do not effect the query processing. This clearly shows the benefit of the GUI index in terms of scalability.

C. *Efficiency of* `DUTk`

In this subsection, we investigate the computation time of the `DUTk` algorithm. More precisely, we compare the computation time of `DUTk` and `NAIV` over different values of k. In Fig. 8(a) and 9(a), k is in the range $[10, 100]$, for the (b) figures, it is in the range $[100, 1000]$. As these figures clearly show, `DUTk` is substantially faster than `NAIV`. Furthermore, the computation time of `DUTk` is highly stable relative to the variations of k. This demonstrates the advantages of the GUI index in terms of time processing.

D. *Effectiveness of* `DUTk`

We compared the proposed `DUTk` algorithm with the `NAIV` method, over different k, i.e. $k \in [10, 100]$ and $k \in [100, 1000]$. Fig. 10 (a) and 11 (a) show the communication cost of the two algorithms for different values of k over the data sets generated, respectively with the Pairwise and Zipf distributions. Interpreting our findings is straightforward: from the experimental results, we can see that `DUTk` involves substantially fewer tuples than the `NAIV`. Moreover, we note that the number of returned tuples is greater than k which demonstrates the effectiveness of `DUTk`.

Furthermore, for different value of k, the communication cost of `DUTk` algorithm is smaller then `NAIV`. Consequently, using the GUI index yields a better communication cost and ensures returning the requested tuples. The second important point observed in this experiment is that the number of tuples transferred using `DUTk` is always greater than the value of k

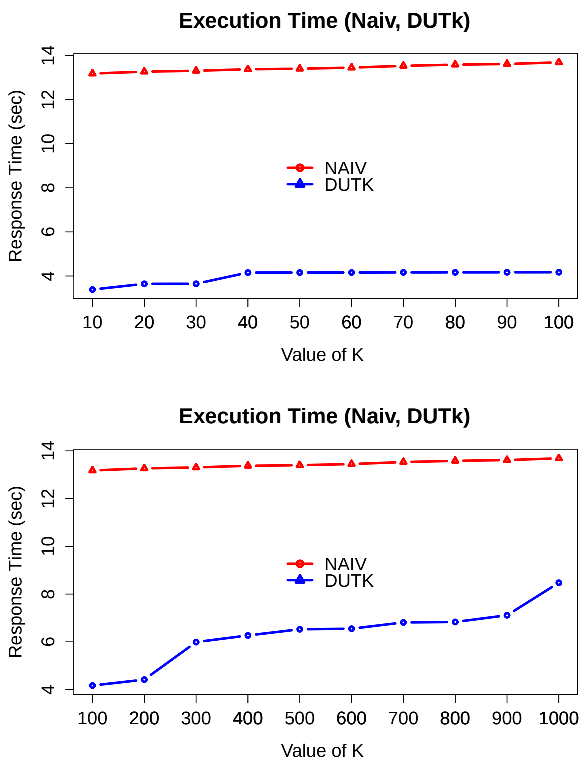


Fig. 9. Efficiency of `DUTk` with Zipf distribution by varying k.

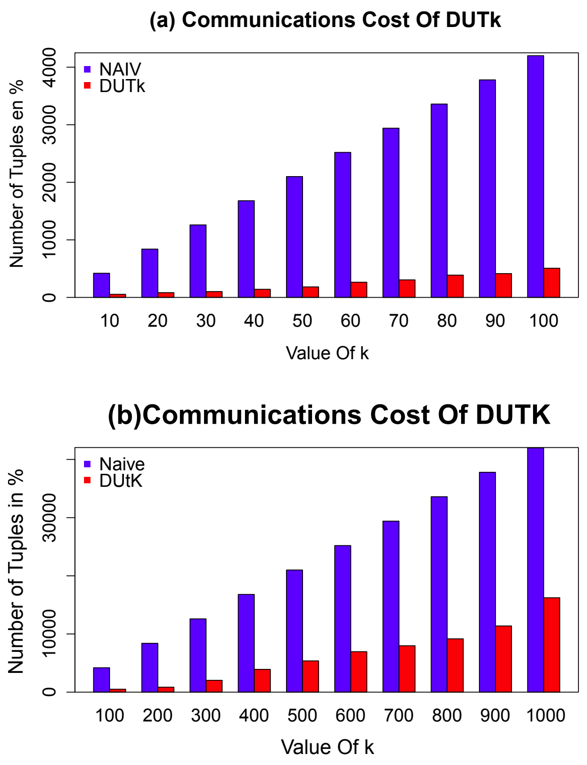


Fig. 11. `DUTk` with Zipf distribution by varying k.

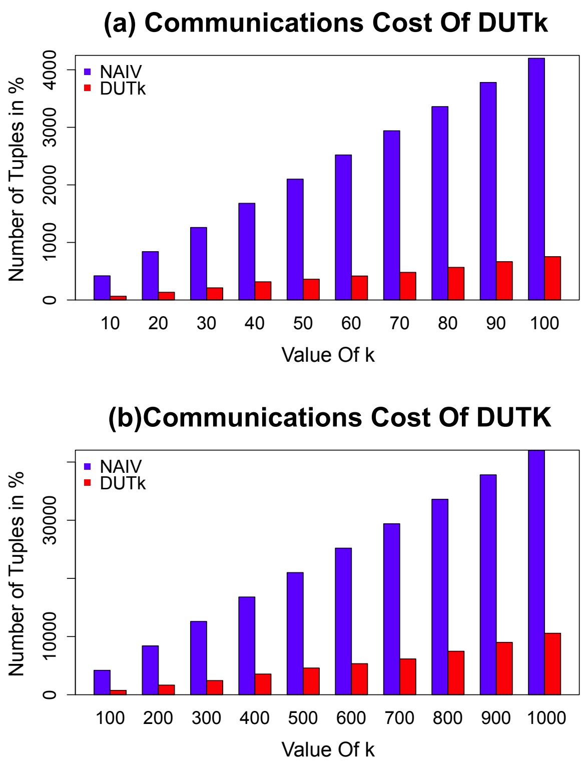


Fig. 10. `DUTk` with pairwise distribution by varying k.

which proves the effectiveness of our results.

E. Scalability of `DUTk`

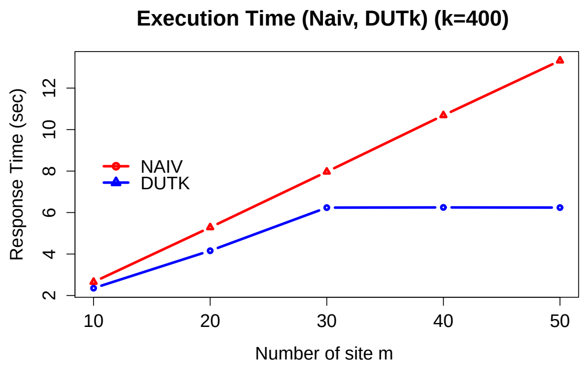


Fig. 12. Scalability of `DUTh` with Zipf distribution by varying m.

In this experiment, we study the scalability of our algorithm with respect to the number of distributed sites m. We keep the size of the whole distributed data unchanged and vary the number of sites from 10 to 50. We use the Zipf Data set in this experiment. Fig. 12 shows the results for $k = 400$. `DUTk` scales well with the number of sites m and outperforms `NAIV` substantially.

We note that the gap between `DUTk` and `NAIV` increases as m increases from $m > 30$. Fig. 12 clearly shows that the execution time of `DUTk` for $m > 30$ is constant.

This result is expected due to the fact that the only a subset of sites are queried based on the GUI. Consequently, the number of sites does not effect the query processing. This clearly shows the benefits of the index structures in terms of scalability.

VIII. Conclusion

In this paper, the problem of indexing and query processing over uncertain categorical data in distributed environments was addressed. An original approach was proposed to efficiently answers queries over distributed uncertain data using a distributed top-level index with *Local Uncertain Indexes (LUIs)* at local nodes and a *Global Uncertain Index (GUI)* summarizing local indexes. Leveraging the distributed indexing technique, we proposed a framework integrating two distributed algorithms `DUTh` and `DUTk` to process distributed uncertain threshold and top-k queries respectively.
An extensive experiment was conducted to illustrate the performance of the proposed framework, which performs distributed query processing and allows for minimal communication and processing costs as opposed to the naive(`NAIV`) approach.

We have seen that our proposed `DUTk` enhanced the naive approach that effectively and efficiently answers top-k queries over distributed uncertain data with minimum communication by using the proposed hierarchical index. However, during the ranking phase on the coordinator site, the tuples of some queried sites cannot be considered in the final result as shows in the effectiveness experiments. Hence, these sites should not be accessed and should be pruned to further reduce the communication cost and processing time.

References

[1] R. Cavallo and M. Pittarelli, “The theory of probabilistic databases,” in *VLDB*, 1987, pp. 71–81.
[2] D. Barbará, H. Garcia-Molina, and D. Porter, “The management of probabilistic data,” *TKDE*, pp. 487–502, 1992.
[3] N. N. Dalvi and D. Suciu, “Efficient query evaluation on probabilistic databases,” in *VLDB*, 2004, pp. 864–875.
[4] M. Hua, J. Pei, W. Zhang, and X. Lin, “Ranking queries on uncertain data: a probabilistic threshold approach,” in *SIGMOD*. ACM, 2008, pp. 673–686.
[5] M. A. Soliman, I. F. Ilyas, and K. C. Chang, “Top-k query processing in uncertain databases,” in *ICDE*. IEEE Computer Society, 2007, pp. 896–905.
[6] P. K. Agarwal, S. Cheng, Y. Tao, and K. Yi, “Indexing uncertain data,” in *PODS*, 2009, pp. 137–146.
[7] S. Singh, C. Mayfield, S. Prabhakar, R. Shah, and S. E. Hambrusch, “Indexing uncertain categorical data,” in *ICDE*, 2007, pp. 616–625.
[8] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, “Efficient indexing methods for probabilistic threshold queries over uncertain data,” in *VLDB*, 2004, pp. 876–887.
[9] A. Benaissa, S. Benbernou, M. Ouziri, and S. Sahri, “Indexing uncertain categorical data over distributed environment,” in *IFSA-EUSFLAT*, 2015.
[10] X. Li, Y. Wang, and J. Yu, “An efficient scheme for probabilistic skyline queries over distributed uncertain data,” *Telecommunication Systems*, vol. 60, no. 2, pp. 225–237, 2015.
[11] F. Li, K. Yi, and J. Jestes, “Ranking distributed probabilistic data,” in *SIGMOD*, 2009, pp. 361–374.
[12] Y. Sun, Y. Yuan, and G. Wang, “Top-k query processing over uncertain data in distributed environments,” *WWW*, vol. 15, pp. 429–446, 2012.
[13] Y. M. AbdulAzeem, A. I. El-Desouky, H. A. Ali, and M. M. Salem, “Ranking distributed database in tuple-level uncertainty,” *Soft Comput.*, vol. 19, no. 4, pp. 965–980, 2015.
[14] N. A. Othman, A. S. Eldin, and D. S. E. Zanfaly, “Handling uncertainty in database: An introduction and brief survey,” *Computer and Information Science*, vol. 8, no. 3, pp. 119–133, 2015.
[15] M. Ye, X. Liu, W. Lee, and D. L. Lee, “Probabilistic top-k query processing in distributed sensor networks,” in *ICDE*, 2010, pp. 585–588.
[16] H. Ji, H. Deng, and J. Han, “Uncertainty reduction for knowledge discovery and information extraction on the world wide web,” *WWW*, 2012.
[17] E. Michelakis, R. Krishnamurthy, P. J. Haas, and S. Vaithyanathan, “Uncertainty management in rule-based information extraction systems,” in *SIGMOD*, 2009, pp. 101–114.
[18] X. L. Dong, A. Y. Halevy, and C. Yu, “Data integration with uncertainty,” in *VLDB*, 2007, pp. 687–698.
[19] E. Altareva and S. Conrad, “The problem of uncertainty and database integration,” in *EFIS*, 2001, pp. 92–99.
[20] R. J. Miller, “Management of inconsistent and uncertain data,” in *VLDB*, 2007, p. 7.
[21] ——, “Efficient management of inconsistent and uncertain data,” in *BIRTE*, 2008.
[22] A. Fuxman, E. Fazli, and R. J. Miller, “Conquer: Efficient management of inconsistent databases,” in *SIGMOD*, 2005, pp. 155–166.
[23] M. Chen, G. Yu, Y. Gu, Z. Jia, and Y. Wang, “An efficient method for cleaning dirty-events over uncertain data in wsns,” *J. Comput. Sci. Technol.*, vol. 26, no. 6, pp. 942–953, 2011.
[24] R. Cheng, J. Chen, and X. Xie, “Cleaning uncertain data with quality guarantees,” *PVLDB*, vol. 1, no. 1, pp. 722–735, 2008.
[25] R. Cheng, “Querying and cleaning uncertain data,” in *QuaCon*, 2009, pp. 41–52.
[26] F. Jiang and C. K. Leung, “A data analytic algorithm for managing, querying, and processing uncertain big data in cloud environments,” *Algorithms*, vol. 8, no. 4, pp. 1175–1194, 2015.
[27] E. Adar and C. Ré, “Managing uncertainty in social networks,” *IEEE Data Eng. Bull.*, vol. 30, no. 2, pp. 15–22, 2007.
[28] R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” in *SIGMOD*, 2001.
[29] J. Li, B. Wang, G. Wang, and Y. Zhang, “Probabilistic threshold query optimization based on threshold classification using ELM for uncertain data,” *Neurocomputing*, vol. 174, pp. 211–219, 2016.
[30] I. F. Ilyas, G. Beskales, and M. A. Soliman, “A survey of top-k query processing techniques in relational database systems,” *ACM Comput. Surv.*, vol. 40, no. 4, 2008.