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-values 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 \(\text{Farm}(T_{id}, \text{weight}, \text{illness})\) where the \text{illness} attribute specifies the illnesses that can affect a cow. \text{Illness} attribute is an uncertain one that takes its values from the categorical domain \{\text{mc}, \text{fa}, \text{nc}, \text{fs}\} where \text{mc} means mad cow, \text{fa} means fever accurate, \text{nc} means normal cow, \text{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 \text{acute fever} (\text{fa}) with the probability 0.7 or maybe \text{simple fever} (\text{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:

\(Q_1\) : Find the cows affected by an accurate Fever with a probability above 0.5;

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

The query \(Q_1\) aims to identify the tuples where the illness attribute has the value \text{fa} and The query \(Q_2\) aims to return the first 2 tuples with the highest probability, where the
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 forward the query to all the sites so that local treatment and processing will be done on each site locally. Then, each site sends 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 drawbacks, 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 \( S = \{S_1, S_2, ..., S_m\} \). 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 \( S.a \) be a particular attribute in \( S \) which is uncertain. \( S.a \) 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:

**Definition 3.1:** Given a set of distributed uncertain relations \( S = \bigcup_{i \in [1, m]} S_i \), \( \text{DUTH} \) returns a set of tuples \( t_k \in S_i \) such that \( Pr(t_k.a = d_i) > \tau \), where \( \tau \) 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:

\[ Q_1 : \text{Find the cows affected by an accurate Fever with a probability above 0.5} \]

The query \( Q_1 \) 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 \( S \).

**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 \( S = \bigcup_{i \in [1, m]} S_i \), \( \text{DUTk} \) returns the \( k \) first tuples \( t_k \in S_i \) with the highest probability among all distributed sites of \( S \).

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

\[ Q_2 : \text{Find the two cows affected by an accurate fever in all the farms} \]

The query \( Q_2 \) is distributed probabilistic Top-k queries. The aim of query \( Q_2 \), 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.

---

1In 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.
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 \( d_i \cdot a \) 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 \( nc \) 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 \( nc \) with the probability 1 in its **illness** uncertain attribute.

\[
\begin{align*}
nc & \rightarrow (t_3; 1.0) \quad (t_4; 0.9) \\
fa & \rightarrow (t_2; 0.9) \quad (t_1; 0.7) \\
fs & \rightarrow (t_1; 0.3) \quad (t_2; 0.1) \quad (t_4; 0.1)
\end{align*}
\]

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 | \( D = \{d_1, d_2, \ldots, d_n\} \); \( S = \{S_1, S_2, \ldots, S_n\} \); |
| output | GUI |
| for each site \( \in S \) do |
| \( P_{max} \leftarrow \) maximal probability of \( d \) in LUI |
| \( Ld \leftarrow (S, P_{max}) \) |
| end |
| for each \( d \in D \) do |
| organize \( ld \) as inverted list in decreasing order |
| end |

Algorithm 1: Global Index (GUI) Construction

Given a distributed relation \( R \) on \( S \), 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
Given a LUI stored on a local site $S_i$, for each entry $d_j$ of LUI, GUI stores one pair $(S_i, P_{max})$ where $S_i \in S$ 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 LUIs 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 $fa$ 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 $fa$ is located in site $S_2$ with maximal probability 0.9. The Inverted list of $fa$ 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 $\text{DUT}_h$ and $\text{DUT}_k$ which improve the overall performance by reducing both the processing and communication cost of the threshold and top-$k$ queries, respectively.

#### A. Distributed Uncertain Threshold Algorithm

The Distributed Uncertain Threshold Algorithm, referred to as $\text{DUT}_k$, 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 database $R$ distributed over a set of site $S = \{S_1, S_2, ..., S_m\}$ and query $Q$ on $S$. Note that the query $Q$ will be initiated in the coordinator site with a given threshold probability ($\tau$). The main steps of the $\text{DUT}_h$ algorithm execution are as follows:

- The query $Q$ will be executed in the coordinator site $S_c \in S$. The first step of $\text{DUT}_h$, 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 $\text{DUT}_h$ 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:

$Q_1 : Select * from S where illness = 'fa' and P > 0.5$

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 $\text{DUT}_h$, 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 $S_2$.

---

\(^2\)We will call this approach $\text{NAIV}$ in our experiments.
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 1.

**Table I. Results of \(\text{DUT}_h\) Query \(Q_1\).**

<table>
<thead>
<tr>
<th>(R_i, fa)</th>
<th>weight</th>
<th>illness</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1, fa)</td>
<td>700</td>
<td>(fa,0.9)</td>
</tr>
<tr>
<td>(R_1, fa)</td>
<td>710</td>
<td>(fa,0.7)</td>
</tr>
<tr>
<td>(H_2, fa)</td>
<td>740</td>
<td>(fa,0.9)</td>
</tr>
</tbody>
</table>

### B. Distributed Uncertain Top-k Algorithm

Given an uncertain relation \(R = \bigcup_{i \in [1,n]} R_i\) distributed on \(S = \{S_1, S_2, \ldots, S_n\}\) 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 \(\text{DUT}_k\) 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 \(\text{DUT}_k\) are as follows:

**Algorithm 3: \(\text{DUT}_k\) Query Algorithm**

input: Query \(Q\), Index GUI .\(\text{DUT}_h\), \(k\)

output: \(\text{DUT}_k^{\text{response}}()\)

\(\text{DUT}_k^{\text{response}}() = \{\}\); Execute query \(Q\) in coordinator site;

crossing the index GUI;

\(L_i \leftarrow\) list of site from GUI

for each \(S_i\) in list \(L_i\) do

\(L^p_{\min} \leftarrow\) get the probability of \(k_{th}\) of site from Local index;

end

\(\tau \leftarrow\) Max of \(L^p_{\min}\)

//Execute Threshold algorithm \(\text{DTH}(\tau)\)

Global\_response\leftarrow Duth(\tau)

//Merge and Rank the results, then choose the top-k response

\(\text{DUT}_k^{\text{response}}() \leftarrow\) Order(Global\_response(limit_k))

return(\(\text{DUT}_k^{\text{response}}()\))

- **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 \(S_Q = \{S_1, S_2, \ldots, S_h\}\) 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 \(\text{TH}_L\) stores all the probability values sent by the distributed sites. The maximal probability, \(\tau = \text{Max}_{i \in [1,m]} P_i\) (maximal probability of the list \(\text{TH}_L\)), is considered as the new probability threshold, then a new pruning is performed from the global index GUI according to \(\tau\). 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 \(S_{Q_2}\) of distributed sites where \(Q\) should be executed where \(S_{Q_2} = \{S_1, S_2, \ldots, S_i\}\) where \(l \leq h \leq m\). Consequently, the coordinator site \(S_c\) sends the query \(Q\) to the sites in \(S_{Q_2}\).

- **Local query ranking:** On each site \(S_i\) in \(S_{Q_2}\), 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 \(\text{DUT}_k\) 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 \(\text{max}(S_1) = 0.9, \text{max}(S_2) = 0.9\),

- **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 \((P_{2nd_{fa}}(S_1) = 0.7)\) and \(P_{2nd_{fa}}(S_2) = 0.2\)), 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>
<thead>
<tr>
<th>( R_i, T_i )</th>
<th>Weight</th>
<th>Illness</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1, T_1 )</td>
<td>700</td>
<td>( (a, 0.9) )</td>
</tr>
<tr>
<td>( R_2, T_2 )</td>
<td>710</td>
<td>( (a, 0.9) )</td>
</tr>
</tbody>
</table>

From these steps, we have the following Lemma:

The DUT\( k \) 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 DUT\( h \) and DUT\( k \). We study the effect of the different parameters for both of the proposed algorithms and we compare their results with the NAI\( V \) 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 DUT\( h, DUT\( k \) and NAI\( V \) 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>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>number of nodes</td>
<td>{10, 20, 30, 40, 50}</td>
</tr>
<tr>
<td>( k )</td>
<td>size of top-k</td>
<td>{10, 100} and {100, 1000}</td>
</tr>
<tr>
<td>( \tau )</td>
<td>threshold value</td>
<td>([0, 1]) and ([0.9, 1])</td>
</tr>
<tr>
<td>( N )</td>
<td>size of database on each node</td>
<td>above 230000 tuples</td>
</tr>
</tbody>
</table>

A. Efficiency of DUT\( h \)

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

![Graph](attachment:graph.png)

**Fig. 5.** Efficiency of DUT\( h \) with pairwise distribution by varying \( \tau \).

In these experiments, we varied the value of threshold \( \tau \) (Table III). In Fig. 5(a) and 6(b), the threshold \( \tau \) 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 DUT\( h \) algorithm decreases with the increase of the value of \( \tau \). For the highest value \( \tau = 1 \), the total response time is about 1.65 sec for DUT\( h \) and 13.79 sec for NAI\( V \).

We note that the difference between the response time of DUT\( h \) and NAI\( V \) 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 $\tau$.

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 $\tau$, 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 than 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. 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.
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