TOP-K ANSWERING UNDER UNCERTAIN SCHEMA MAPPINGS

A Thesis

by

SHANXIAN MAO

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Texas A&M University - Corpus Christi

Corpus Christi, Texas

August 2012

Major Subject: Computer Science
ABSTRACT

The data sources of information systems running on various hardware and software platforms are independent to each other and mutually closed, which makes data exchange difficult. With the evolvement of the information application technology, data sharing between internal departments or external enterprises is necessarily required. Finally, data integration has been developed. The data integration is an application providing a bridge of communication between isolated sources and offering a platform for information exchange. However, due to the need of markets nowadays, the big-data sources become one of main burdens on the transaction rates for data integration systems. There are two semantics, by-table and by-tuple, which are developed to capture top-k answering in the data integration system. Both semantics are developed to attempt to enhance the performance when the system encounters uncertain queries or obscure schema mappings between local sources and their centralized system. However, although the current algorithms support some features to capture accurate top-k answering and try to avoid accessing all data from sources, they cannot effectively minimize the number of traversed items in most cases. Consequently, we are trying to propose our solutions to improve the efficiency for the data integration with uncertainty. In our research, we apply histogram-based approximation to capture an estimated list of top-k results in order to improve the ability of processing a large amount of data more efficiently. Histogram-based approximation is used to generate approximate values from histograms provided by sources, and the approximate values are summarized for
calculating a confidence of top-

$k$ candidates. In our algorithm, the confidence is able to
control termination of processing data in both by-table and by-tuple semantics. Finally,

traditional by-table and by-tuple methods could be applied to present true top-

$k$ outputs, whose result can be utilized to evaluate our new approaches
ACKNOWLEDGMENTS

Thanks for my parents’ support that I could study in the Texas A&M University – Corpus Christi for the Master’s degree of Computer Science. In addition, I would like to thank Dr. Longzhuang Li who paid a lot of effort for my research. He provided suggestions and ideas on designing new algorithms. Dr. Ahmed M. Mahdy offered me a lot of comments and help to complete my report. Dr. Dulal C. Kar offered much help and guidance in the completion of the thesis.
Table of Contents

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background and Rationale</td>
<td>5</td>
</tr>
<tr>
<td>1.1.1 Uncertainty in Data Integration</td>
<td>6</td>
</tr>
<tr>
<td>1.1.2 System Architecture</td>
<td>10</td>
</tr>
<tr>
<td>1.2 Related Work</td>
<td>14</td>
</tr>
<tr>
<td>1.2.1 Threshold Algorithm</td>
<td>14</td>
</tr>
<tr>
<td>1.2.2 Handling Uncertainty in Mappings</td>
<td>15</td>
</tr>
<tr>
<td>1.2.2.1 By-Table Semantics</td>
<td>19</td>
</tr>
<tr>
<td>1.2.2.2 Top-(k) Query Answering Using By-Table Semantics</td>
<td>20</td>
</tr>
<tr>
<td>1.2.2.3 By-Tuple Semantics</td>
<td>25</td>
</tr>
<tr>
<td>1.2.2.4 Top-(k) Query Answering Using By-Tuple Semantics</td>
<td>26</td>
</tr>
<tr>
<td>1.2.3 PDFs Using Histograms</td>
<td>26</td>
</tr>
<tr>
<td>1.2.4 Histogram-Based Approximation</td>
<td>31</td>
</tr>
<tr>
<td>1.3 Challenges Motivating Proposed Work</td>
<td>35</td>
</tr>
<tr>
<td>1.3.1 Issues on Data Accuracy</td>
<td>35</td>
</tr>
<tr>
<td>1.3.2 Issues on Query Processing</td>
<td>36</td>
</tr>
<tr>
<td>1.3.3 Issues on Time Consuming</td>
<td>37</td>
</tr>
</tbody>
</table>
2 SYSTEM DESIGN OR RESEARCH .............................................................. 38
  2.1 Motivations and Objectives of Proposed Research ......................... 38
  2.2 New Algorithms for By-Table Query Answering .............................. 41
    2.2.1 Heuristic-Based Approach ......................................................... 44
     2.2.1.1 Pre-Conditions Required ...................................................... 44
     2.2.1.2 Extracting Top-k Candidate Tuples ........................................ 46
     2.2.1.3 Computing Confidence for Top-k Candidates ............................. 49
     2.2.1.4 An Example ........................................................................... 49
    2.2.2 Histogram-Based Approximate Approach in By-Table Semantics .... 57
     2.2.2.1 Pre-Conditions Required ...................................................... 57
     2.2.2.2 Extracting Top-k Candidate Tuples ........................................ 58
     2.2.2.3 Computing Confidence for Top-k Candidates ............................. 61
     2.2.2.4 An Example ........................................................................... 66
  2.3 New Algorithms for By-Tuple Query Answering ............................... 68
    2.3.1 Histogram-Based Approximate Approach in By-Tuple Semantics .... 69
     2.3.1.1 Pre-Conditions Required ...................................................... 70
     2.3.1.2 Extracting Top-k Candidate Tuples ........................................ 71
     2.3.1.3 Computing Confidence for Top-k Candidates ............................. 75
     2.3.1.4 An Example ........................................................................... 80
    2.3.2 Maximum Likelihood Approach .................................................. 81
  3 EVALUATION AND RESULTS ................................................................. 84
    3.1 Heuristic-Based Approach ............................................................ 84
    3.2 Histogram-Based Approximate Approach in By-Table Semantics ....... 89
    3.3 Histogram-Based Approximate Approach in By-Tuple Semantics ........ 93
4 FUTURE WORK .................................................................................................................96
5 CONCLUSION ..................................................................................................................97
REFERENCES .....................................................................................................................99
APPENDIX A ......................................................................................................................102
APPENDIX B .....................................................................................................................105
APPENDIX C .....................................................................................................................109
APPENDIX D .....................................................................................................................120
APPENDIX E .....................................................................................................................130
# LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>A source $S$ for running the schema mapping with target $T$.</td>
</tr>
<tr>
<td>II</td>
<td>Schema mapping with different probabilities between $S$ and $T$.</td>
</tr>
<tr>
<td>III</td>
<td>Sorted schema mappings $m_1, m_2, m_3,$ and $m_4$ with their tuples.</td>
</tr>
<tr>
<td>IV</td>
<td>The price range of five buckets for distributions of source $S$.</td>
</tr>
<tr>
<td>V</td>
<td>Summary of buckets for generated schema mappings $m_1, m_2, m_3,$ and $m_4$.</td>
</tr>
<tr>
<td>VI</td>
<td>Histogram-based approximate values generated for $m_1, m_2, m_3,$ and $m_4$.</td>
</tr>
<tr>
<td>VII</td>
<td>Sorted and distinct tuples in each mapping $m_i$ ($1 \leq i \leq 4$).</td>
</tr>
<tr>
<td>VIII</td>
<td>A list of all tuples’ probabilities in $S$.</td>
</tr>
<tr>
<td>IX</td>
<td>Result of equi-joins between histograms of $m_3$ and $m_2$.</td>
</tr>
<tr>
<td>X</td>
<td>Result of all approximate after equi-join SQL query.</td>
</tr>
<tr>
<td>XI</td>
<td>The summary of an output using both Luna’s algorithm and Approach $B$.</td>
</tr>
<tr>
<td>XII</td>
<td>Comparison between histogram-based approach and the old by-tuple method.</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sub-queries based on different probabilistic schema mapping between $S$ and $T$.</td>
</tr>
<tr>
<td>2</td>
<td>The architecture of a data integration system</td>
</tr>
<tr>
<td>3</td>
<td>The architecture of a data integration system supporting uncertain query</td>
</tr>
<tr>
<td>4</td>
<td>Flow chart of Luna’s by-table query answering</td>
</tr>
<tr>
<td>5</td>
<td>PDFs based on prices in each reformulated query mappings</td>
</tr>
<tr>
<td>6</td>
<td>Screenshot of interface for the histogram-based approach</td>
</tr>
<tr>
<td>7</td>
<td>Screenshot of a window showing the sorted data of mappings</td>
</tr>
<tr>
<td>8</td>
<td>The interface for Luna’s by-table query answering</td>
</tr>
<tr>
<td>9</td>
<td>The interface for the histogram-based approach</td>
</tr>
<tr>
<td>10</td>
<td>Sample output the histogram-based approach</td>
</tr>
<tr>
<td>11</td>
<td>Screenshot for inputting the number of buckets</td>
</tr>
<tr>
<td>12</td>
<td>The interface of histogram-based approach in By-Table Semantics</td>
</tr>
<tr>
<td>13</td>
<td>Output of generated distributed histograms for $m_i$</td>
</tr>
<tr>
<td>14</td>
<td>Groupbox for asking the user to input top-$k$, confidence, and lines to view</td>
</tr>
<tr>
<td>15</td>
<td>Global interface of histogram-based approximation in by-tuple semantics</td>
</tr>
<tr>
<td>16</td>
<td>Distributed histograms with 50 buckets generated based on $m_i$</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Mediated data integration systems provide a uniform interface model and the capability of accessing to multiple heterogeneous data sources [Lenzerin 2002]. These systems have been verified successfully [Halevy 2003][Li 2006] and recently attracted an increasing number of organizations and enterprises from different fields to make use of such a virtual data integration architecture, which is a critical advance and extension of the traditional database conception. However, most researchers argue that the knowledge of query mappings from data sources to the core is still troubling the global mediator. One reason is that the autonomous remote sources are likely to offer objects with similar but distinct data format, attributes, and structures, or even distinct software applications. Another reason is to rank important items and return top-k answers to end users rather than showing a mess. Therefore, the recent challenges need us to contrive improved analyses to model top-k processing techniques from multiple uncertain data, mappings, or schema matching [Chai 2008][Dong 2009][Gal 2009][Heo 2010][Ilyas 2008][Jayram 2007][Li 2005][Shmueli-Scheuer 2009].

Among the optimal aggregation algorithms [Fagin 2001], the Fagin’s Algorithm, or FA, is optimal for some monotone aggregation functions but not for all of them. Another simple algorithm Threshold Algorithm, or TA [Akbarinia 2007] [Arai 2007][Dong
TA application can provide a boundary to extract top-\(k\) objects from local sources, which can reduce the number of objects that have to be accessed from multiple data sources. However, this algorithm considers real values placed in databases rather than the threshold mentioned in the Luna’s article [Dong 2009]. In addition, in uncertain schema mappings, we concern about values with top-\(k\) probabilities instead of the top-\(k\) values with top grades, which also differs from the TA application. Luna in her research mainly focuses on solving the problem on uncertain queries that may happen between centralized mediator and data sources. When uncertainty occurs, the system could automatically reformulate multiple queries based on target as well as the sources. The Luna’s method is trying to capture top-\(k\) answering from sources without traversing all generated queries. The threshold is a minimum bound of top-\(k\) results, which means threshold is the value that equals to the top-\(k^{th}\) minimum probability.
According to the data integration under uncertainty [Dong 2009], each reformulated query is assigned with a specific probability. If the sum of probabilities is 1.0, all generated queries have been considered. There are two possible semantics, \textit{by-table} and \textit{by-tuple}, applied to compute probability for each viewed item placed in one or different queries. Top-$k$ items would be found according to their highest probabilities. Handling uncertainty in schema mappings, by-table semantics, which uses a threshold value as a condition, can successfully reduce the number of viewed columns (or query mappings). However, for each viewed mapping, by-table algorithm has to access the whole columns from the first tuple till the end. By-tuple semantics is more complicated than by-table semantics, which has to concern about both the frequency of each tuple that has been viewed and locations of the tuple that is placed in different reformulated queries. Using by-tuple query answering to capture top-$k$ results, there is no condition to control termination at an earlier stage, and all tuples from all generated queries have to be extracted and analyzed, which wastes a large amount of time and payment. Finally, in our proposed methods, we are concerned with both objectives:

1. For by-table semantics, reducing the number of traversed mappings, in which the number of viewed items also needs to be minimized;

2. For by-tuple semantics, proposing a condition to stop traversing tuples in the middle of rows no matter if all generated queries have to be considered simultaneously;
We propose top-k answers approaches based on the top-k query answering under uncertain schema mappings [Dong 2009]. The background and rationale we presented in Chapter 2 and describe th used in a variety of top-k functions but differs from the one employed in Luna’s theory; both by-table and by-tuple semantics are explained by presenting how they work in the data integration system; approximating Probability Density Functions (PDFs) using Histograms was proposed in [Arai 2007] to present distributions for each list of unseen tuples; histogram-based approximation [Ioannidis 1999] was employed to our improved methods, which simplifies complicated algorithms and generates approximate values according to known distributions instead of analyzing real data that are record (Chapter 1.2).

There are challenges that we should consider when use previous approaches (Chapter 1.3). Therefore, according to the problems or shortcomings, we attempt to propose our improvements with new objectives (Chapter 2.1).

In the section of our proposed methods, we firstly give an overview of our viewpoints and the process of our analysis, as well as showing a structure for the following two chapters. Specifically, we elaborate our enhanced algorithms for both by-table (Chapter 2.2) and by-tuple (Chapter 2.3) semantics to realize our objectives. Furthermore, in both chapters, we apply PDFs with generated approximate values to compute the confidence
for current candidate top-$k$ tuples and control termination of the program.

In order to easily test and prove our proposed algorithms, all methods are implemented in a Graphic User Interface (GUI), which has the feature to ask the users to submit requirements and return a sorted top-$k$ answering list back to the interface (Chapter 3). Finally, we describe future work (Chapter 4) and summarize our research in the conclusion (Chapter 5).

1.1 Background and Rationale

With the rapid development of science and technology and the advancement of information technology in recent decades, the amount of data accumulated by the human society has been more than the sum of the past five thousand years. At the same time, the number of data collection, storage, processing and dissemination keeps growing. Data sharing enables more existing data resources that can be applied by more people so as to reduce duplication of data collection, data acquisition, and the corresponding costs. However, during the process of implementation for data sharing, data may be from different users in different ways. Data content, data format, and data quality vary widely. Commonly, users may encounter the problem that data format cannot be converted or loss of information after the data converting process, which becomes a serious impediment for the data transaction and data sharing among various departments and
different software systems. Therefore, how to effectively integrate and manage data becomes an inevitable choice to enhance business competitiveness.

As the rapid development, modern enterprises have gradually grown up and become a connection to exchange information through the network and an entity of business affairs, rather than an isolated node. Meanwhile, data exchange has been evolved to occur among enterprises instead of simply internal exchange. However, data uncertainty and frequent changes induce that if application changes or physical data change, the entire system may have to be modified subsequently. Hence, data integration will be faced with challenges such as how to adapt to the complex requirements of the development of modern society, how to effectively expand the application fields, how to separate technology and application requirements, and how to adequately describe the various data formats and types.

1.1.1 Uncertainty in Data Integration

Data integration and exchange systems have the ability to provide a uniform interface and logically and physically centralize multiple data sources from different format and characteristics. Data integration techniques offer comprehensive data sharing for enterprises and have a lot of mature framework, of which we can take advantage. When the local schemas in different sources map to the centralized schema in a data integration system, the system should be able to model uncertainty at their core. Uncertainty may
occur in data or queries. The reason for data uncertainty is that data are automatically extracted from unstructured or semi-structured data sources in most cases. In addition, data stored in sources may be not reliable or have not been updated before captured. In our research, we mainly focus on uncertainty that happens in queries.

In a data integration system, queries will be submitted as keywords rather than a structured query with well defined schemas. The system has to translate queries into a structured form so as to satisfy a structured format for data sources. Therefore, the system may create multiple candidate structures. The generated queries are reformulated based on the submitted query, each of them with some probability. For example, as illustrated in Table I, there is a data source $S$ providing information on houses for sale in Texas. $S$ has five attributes, house identity, original price, reduced price, appraisal price, and sale price. The local schema of the source $S$ is defined as follows:

$$S = (house\_id, list\_price, reduced\_price, appraisal\_price, sale\_price).$$

**Table I** A source $S$ for running schema mapping with target $T$

<table>
<thead>
<tr>
<th>House_id</th>
<th>list_price</th>
<th>reduced_price</th>
<th>appraisal_price</th>
<th>sale_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>260K</td>
<td>250K</td>
<td>230K</td>
<td>240K</td>
</tr>
<tr>
<td>1002</td>
<td>280K</td>
<td>270K</td>
<td>230K</td>
<td>230K</td>
</tr>
<tr>
<td>1003</td>
<td>180K</td>
<td>150K</td>
<td>150K</td>
<td>130K</td>
</tr>
</tbody>
</table>
The target schema $T$ in centralized system has two attributes, house number and price:

$$T = (\text{house\_number, price}).$$
Assume that the system receives a query to select prices from the target:

\[ Q: \text{SELECT } \text{price} \text{ FROM } T \]

According to the attributes of \( S \) mapping to \( T \), there are four possible sub-queries that are reformulated, \( Q_1, Q_2, Q_3, \) and \( Q_4 \) from a semi-automatic schema-mapping tool:

\[ Q_1: \text{SELECT } \text{sale_price} \text{ FROM } S \]

\[ Q_2: \text{SELECT } \text{reduced_price} \text{ FROM } S \]

\[ Q_3: \text{SELECT } \text{appraisal_price} \text{ FROM } S \]

\[ Q_4: \text{SELECT } \text{list_price} \text{ FROM } S \]

As illustrated in Figure 1, the generated queries \( Q_1, Q_2, Q_3, \) and \( Q_4 \) represent that attributes, \( \text{sale_price}, \text{reduced_price}, \text{appraisal_price}, \) and \( \text{list_price} \) in source \( S \) are mapped to the attribute \( \text{price} \) of \( T \) respectively.
1.1.2 System Architecture

There is a data integration architecture illustrated in Figure 2. The system involves combining data in different data sources. The mediator provides users a global interface to let them pose queries and view results captured from multiple data sources. As shown in Figure 2, global schema is designed in the mediator, and local schema is located in each of the three local sources. When a user submit a query, according the schema mapping between the target $T$ and source $S$, global schema will be mapped to local schema from the source $S$, $S'$, or $S''$ and then submit a structured query to each of the three. Because sources may export data in different formats, each local source is connected with a wrapper. The wrapper is customer-built programs, which can translate
data from source format to some acceptable structure to the mediator.

Figure 2 The architecture of a data integration system

However, when we concern about uncertainty in schema mapping that is likely to happen, the system becomes more complicated and different from the traditional data integration architecture.
Figure 3 The architecture of a data integration system

that can work out any uncertain query

The data integration system, in Figure 3, has the three data source $S, S', \text{ and } S''$, each of which has its local schema. If a user submits a structured query from the global interface, the query will be reformulated into a single query for each source, which is the same as the traditional data integration.
However, if the user poses a semi-structured or keyword query, the system in Figure 3 will generate a set of probable queries in the process of *Query Reformulation A*, for example, $Q_1, Q_2, Q_3,$ and $Q_4$ that are presented in Figure 1. *Query Reformulation A* is not a traditional step of reformulating the query to satisfy and access to the data sources. This step is not to directly reach the data sources. All the possible mappings that may meet the user's query will be generated.

*Query Reformulation B* is the second step, which differs from the first step, but it is the same as the traditional query reformulation. Each of the generated queries $Q_1, Q_2, Q_3,$ and $Q_4$ will be translated to some acceptable format to each data source. For instance, as for the data source $S$, the query $Q_1$ (or $Q_2, Q_3, Q_4$) will be converted to $Q_{11}$ (or $Q_{12}, Q_{13}, Q_{14}$); for the data source $S'$, the query $Q_1$ (or $Q_2, Q_3, Q_4$) will be converted to $Q_{21}$ (or $Q_{22}, Q_{23}, Q_{24}$); for data source $S''$, there are reformulated queries $Q_{31}$ (or $Q_{32}, Q_{33},$ and $Q_{34}$).

In the uncertain data integration system, it is essential to provide an adaptable and flexible query processor. We realize that all generated queries may have different probabilities, such as $Q_1$ and $Q_4$. The schema mapping in $Q_1$ is *sale_price* mapped to *price*, and $Q_4$ shows that *list_price* is mapped to *price*. If the user prefers viewing the price of houses on sale, the probability of $Q_1$ should be bigger than that of $Q_4$. Therefore,
the values listed in attribute sale_price will be more correct than the values in the column list_price (See Table I). Because of complexity of the uncertain query mapping, there would be multiple queries generated from the first step. When processes query, the system has to waste a large amount of time. Instead of processing all possible mappings in each source, the query processor needs the feature to select the most probable queries and return the answers with higher probabilities.

Returning the result from sources, if the user requires all the answers to the query, all generated queries from Query Reformulation B could be considered equally. In this situation, the reformulated query is just a grouping and aggregation query. When the user only prefers to check top-k answers from the sources, both the reliability of different sources and the correctness of the reformulated queries should be concerned. The top-k answering will be returned with highest probabilities.

To describe our research more easily and clearly, we simplify the system (Figure 3) and analyze the single source S. In source S, we mainly focus on the generated queries $Q_1, Q_2, Q_3, \text{and } Q_4$ and return top-k answers back to the user.

1.2 Related Work
1.2.1 Threshold Algorithm

The threshold Algorithm (TA) is itself based on FA, which is more optimal in all top-k query analyses over sorted lists. Best-Position Algorithm (BPA) is an improvement over TA and can stop at an earlier stage, by ignoring the system to examine all data stored in each list. In an example shown in [Akbarinia 2007], each object is one particular local score, and all objects in the same list are sorted by their local scores. The first step in the algorithm is to access each of the m sorted lists, which is critical for calculating TA that is used to make a decision whether traversing process should continue or terminate.

The algorithms using TA process real items from distributed systems. Each item listed in each sub-system is assigned with a different grade. Top-k items are extracted and ranked according to their grades. Threshold in the algorithms is a boundary to determine top-k answering and terminate data processing. However, it is different from the threshold that is mentioned in the data integration handling uncertainty even though it still represents a bound to compare top-k answering with other items and control stop processing. We should keep in mind that the threshold in the uncertain data integration system equals the top-$k^{th}$ minimum probability, which will be explained in the next paragraph.

1.2.2 Handling Uncertainty in Mappings

The architecture of a data integration system that supports uncertainty differs from
traditional data integration systems, which reformulates the user’s keyword query into a set of probable sub-queries. Because of the ambiguities of schema mapping between the sources and their core, it is impossible to guarantee which data mapped from local sources to the mediated schema are exactly correct. In practice, uncertain queries would be generated when a keyword query and unstructured query is posed by the user. At the first step, we assume that the system may generate multiple candidate structured queries created by a semi-automatic schema-mapping tool. Depending on the program designer, all generated probable queries can be considered with different degrees of possibility, and each of them is assigned a probability, as shown in Table II.

**Table II** Schema mapping with different probabilities between $S$ and $T$

<table>
<thead>
<tr>
<th>possible mapping</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$ ${(house_id, house_number), (sale_price, price)}$</td>
<td>0.4</td>
</tr>
<tr>
<td>$m_2$ ${(house_id, house_number), (reduced_price, price)}$</td>
<td>0.3</td>
</tr>
<tr>
<td>$m_3$ ${(house_id, house_number), (appraisal_price, price)}$</td>
<td>0.2</td>
</tr>
<tr>
<td>$m_4$ ${(house_id, house_number), (list_price, price)}$</td>
<td>0.1</td>
</tr>
</tbody>
</table>
There are mappings \( m_1, m_2, m_3, \) and \( m_4 \), and probabilities of the four schema mappings are \( p_1 = 0.4, \ p_2 = 0.3, \ p_3 = 0.2, \) and \( p_4 = 0.1 \). Recall that the four generated queries \( Q_1, Q_2, Q_3, \) and \( Q_4 \) are ranked in descending order, which is different from the initial sequence shown in the table I. When queries have been assigned with different probabilities, we rank them in descending order for our future work. In this example, when generate queries, suppose that we assume that we have already known their probabilities. To make our description clearly, we rank the reformulated queries before we give them probabilities, so the ranked mappings \( m_1, m_2, m_3, \) and \( m_4 \) have the same sequence as queries \( Q_1, Q_2, Q_3, \) and \( Q_4 \) in this paper. However, in our real life, we simply generate a random sequence of possible queries by following the attributes placed in the source entity and then rearrange them into different order based on their assigned probabilities.

From the source schema to the mediated schema, the by-table semantics concern a single mapping that would be the correct one. The second interpretation is by-tuple semantics mapping the candidate queries based on each tuple in the source, instead of assuming one as a correct mapping.
Table III Sorted schema mappings $m_1, m_2, m_3,$ and $m_4$ with their tuples

<table>
<thead>
<tr>
<th>house_id</th>
<th>$m_1 (p_1 = 0.4)$</th>
<th>$m_2 (p_2 = 0.3)$</th>
<th>$m_3 (p_3 = 0.2)$</th>
<th>$m_4 (p_4 = 0.1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>240K</td>
<td>250K</td>
<td>230K</td>
<td>260K</td>
</tr>
<tr>
<td>1002</td>
<td>230K</td>
<td>270K</td>
<td>230K</td>
<td>280K</td>
</tr>
<tr>
<td>1003</td>
<td>130K</td>
<td>150K</td>
<td>150K</td>
<td>180K</td>
</tr>
<tr>
<td>1004</td>
<td>230K</td>
<td>240K</td>
<td>240K</td>
<td>240K</td>
</tr>
<tr>
<td>1005</td>
<td>30K</td>
<td>30K</td>
<td>30K</td>
<td>30K</td>
</tr>
<tr>
<td>1006</td>
<td>180K</td>
<td>200K</td>
<td>170K</td>
<td>200K</td>
</tr>
<tr>
<td>1007</td>
<td>90K</td>
<td>120K</td>
<td>100K</td>
<td>140K</td>
</tr>
<tr>
<td>1008</td>
<td>200K</td>
<td>220K</td>
<td>190K</td>
<td>260K</td>
</tr>
<tr>
<td>1009</td>
<td>90K</td>
<td>80K</td>
<td>90K</td>
<td>90K</td>
</tr>
<tr>
<td>10010</td>
<td>210K</td>
<td>220K</td>
<td>150K</td>
<td>280K</td>
</tr>
<tr>
<td>10011</td>
<td>100K</td>
<td>110K</td>
<td>90K</td>
<td>120K</td>
</tr>
<tr>
<td>10012</td>
<td>280K</td>
<td>260K</td>
<td>270K</td>
<td>260K</td>
</tr>
<tr>
<td>10013</td>
<td>230K</td>
<td>230K</td>
<td>250K</td>
<td>240K</td>
</tr>
<tr>
<td>10014</td>
<td>80K</td>
<td>80K</td>
<td>150K</td>
<td>140K</td>
</tr>
<tr>
<td>10015</td>
<td>100K</td>
<td>150K</td>
<td>160K</td>
<td>230K</td>
</tr>
<tr>
<td>10016</td>
<td>170K</td>
<td>200K</td>
<td>190K</td>
<td>210K</td>
</tr>
<tr>
<td>10017</td>
<td>160K</td>
<td>220K</td>
<td>230K</td>
<td>220K</td>
</tr>
<tr>
<td>10018</td>
<td>30K</td>
<td>30K</td>
<td>90K</td>
<td>50K</td>
</tr>
<tr>
<td>10019</td>
<td>240K</td>
<td>270K</td>
<td>240K</td>
<td>280K</td>
</tr>
<tr>
<td>10020</td>
<td>130K</td>
<td>150K</td>
<td>190K</td>
<td>180K</td>
</tr>
</tbody>
</table>
1.2.2.1 By-Table Semantics

The by-table semantics simply consider that only one particular mapping list of items is accurate. Let $pM = (S, T, m)$ be a p-mapping. $S$ is a relation in a source, $T$ is a relation in the target, and $m$ is one particular relational mapping under a Select-Project-Join (SPJ) query $Q$. The probability $P_p(m)$ is attached to every certain answer under $m$. If an object becomes an answer to $Q$ under possible mappings in $m$, then the probabilities of this object will be summed from multiple seen mappings.

There is an example involving four uncertain query mappings $m_1, m_2, m_3,$ and $m_4$ with probability $p_1 = 0.4$, $p_2 = 0.3$, $p_3 = 0.2$, and $p_4 = 0.1$ respectively, as shown in Table II. The generated mappings are ordered based on their probabilities, each of which provides a list of prices as illustrated in Table III. Note that the prices (or tuples) are objects rather than the expression of numerals. The probability of each viewed tuple $t$ is calculated by the formula $p_t = \sum_{i=1}^{n} p_{i(t)}$.

For example, when the system traverses the first row of house information, the probability of price $230K$ is $p_{(230K)} = 0.2$. When the second of information is viewed, the probability of price $230K$ is updated to $p_{(230K)} = 0.4 + 0.2 = 0.6$. In the by-table
semantics, the frequency of the item in the same column is not considered.

1.2.2.2 Top-k Query Answering Using By-Table Semantics

\( p_{\text{min}} \) and \( p_{\text{max}} \). The variable \( p_{\text{min}} \) represents the minimum probability of each tuple. \( p_{\text{max}} \) indicates the maximum probability of the tuple. The real probability of this object must be lower or equal to its \( p_{\text{max}} \). Meanwhile, the real probability of the object should be higher or equal to its \( p_{\text{min}} \). The objects listed in the same mapping(s) have the same \( p_{\text{min}} \), which equals to the probability of the mapping \( p_i \) (or \( \sum p_i \)). When an object has been viewed in different mappings, \( p_{\text{min}} \) of this tuple has to be updated by adding this traversed mapping’s probability \( p_i \) according to by-table semantics. The significance of \( p_{\text{min}} \) is to obtain top-k candidates and update \( th \) that equals to a current top-\( k \)\(^{th} \) \( p_{\text{min}} \). For all the unseen tuples, their \( p_{\text{max}} \) equals 1.0 if there is no reformulated query that has been viewed thoroughly. When a mapping \( m_i \) has been viewed from the first tuple to the last one, \( p_{\text{max}} \) of the tuple that has not found in \( m_i \) equals previous \( (p_{\text{max}} - p_i) \).

The pseudo code of Luna’s method on extracting top-k answering using by-table semantics [Dong 2009] is presented in Appendix A. In this approach, all generated queries are ranked in descending order based on their probabilities, such as the four uncertain query mappings \( m_1, m_2, m_3, \) and \( m_4 \) with probability \( p_1 = 0.4, p_2 = 0.3, \)
The flow chart of returning top-$k$ query answering using by-table semantics is illustrated in Figure 4. The first step is to initialize the variables $i = 1$ (processing starts from $m_4$), $P_{\text{MAX}} = 1$ (the maximum probability of unseen sub-queries), $th = 0$ (the top-$k^{th}$ minimum probability). The second step is to determine if both conditions can be satisfied. If yes, the system halts and return top-$k$ answers. If not, the system starts viewing data from $m_i$, and update each viewed tuple’s $p_{\text{min}}$ and $p_{\text{max}}$. According to the $p_{\text{min}}$ of all viewed tuples, select top-$k$ tuples with highest probabilities, and set $th$ to the top-$k^{th}$ $p_{\text{min}}$. Before checking results with the two conditions, set $i$ to $(i + 1)$, which means if the answering cannot meet either one of the conditions, the system will continue to retrieve next data from the mapping list.
For example, if the number $k$ is 2 (top-2), and the user ask for top-$k$ prices found from the Table III, the four mappings will be traversed one by one.
Iteration 1:

- Processing $m_1$ with 20 prices
- Top-1: 240K, Top-2: 230K
- Threshold ($th$) = 0.4
- The maximum probability of all unseen sub-queries ($P_{MAX}$) = 0.6
- Because $P_{MAX} > th$, which the unseen prices only appear in the unseen sub-queries may be greater than $th$, the system continues the second process.

Iteration 2:

- Processing $m_2$ with 20 prices (The total number of viewed tuples is 40 with duplicates)
- Top-1: 240K, Top-2: 230K
- Threshold ($th$) = 0.7
- $P_{MAX}$ = 0.3
- $th > P_{MAX}$, but some viewed prices such as 30K whose maximum probability are still greater or equal $th$. The system continues the third process.

Iteration 3:

- Processing $m_3$ with 20 prices (The total number of viewed tuples is 60 with duplicates)
Top-1: 240K, Top-2: 230K

Threshold ($th$) = 0.9

$P_{MAX} = 0.1$

$th > P_{MAX}$, but some viewed prices such as 30K whose maximum probability are still greater or equal $th$. The system continues the fourth process.

Iteration 4:

Processing $m_4$ with 20 prices (The total number of viewed tuples is 80 with duplicates)

Top-1: 240K, Top-2: 230K

Threshold ($th$) = 1.0

$P_{MAX} = 0.0$

All the prices have been processed, and top-$k$ answers have been found. Actually, not all the top-$k$ candidates whose probabilities equal to the top-$2^{th}$ probability, like the tuple 30K.

In this example, the by-table query answering using Luna’s approach has viewed all generated queries if top-2 prices are returned. The reason is that there are many overlapped values placed in the mappings. If given different data in Table III, it may happen that after executing Iteration 2, the maximum probabilities of all viewed non-top-2 tuples are smaller than $th$, the system halts and return top-2 candidates. In this case, the
rest mappings will not be viewed.

1.2.2.3 By-Tuple Semantics

Using by-tuple semantics, the correct mapping depends on one particular tuple among each line of record based on uncertain queries. Because by-table semantics assume that a single mapping between target and source is correct but cannot identify which one it is, the objects recorded in the source may present different probabilities. Related to multiple uncertain queries generated automatically according to one practical query from the target, the source using by-tuple method to return top-$k$ results is more complicated than by-table semantics. The by-tuple semantics effectively reveal each seen object with the number of occurrences that has appeared in viewed mappings.

If there are four generated queries $Q_i \ (i \in [1,4])$ independent to each other, and each of them has a probability $p_i \ (i \in [1,4])$, the probability of tuple $t$ would be calculated through the combination formula $p(t) = 1 - \prod_{j=1}^{l}(1 - \sum_{i=1}^{4}p_i(t))$, where, $l$ is the number of lines. For example, we use the same data shown in Table III. As for the tuple 230K, when the system traverses the first row of house information, the probability of price 230K is $p_{(230K)} = 1 - (1 - 0.2) = 0.2$. When the second of instance is viewed, the probability of price 230K is updated to $p_{(230K)} = 1 - (1 - 0.2)(1 - (0.4 + 0.2)) = 0.68$. Obviously, this semantics concerns
about both the number of occurrences for each viewed item and the location placed in each of the query mappings. If by-tuple semantics associate four uncertain mappings with \( n \) tuples, there will be \( 4^n \) different sequences to assign mappings to objects.

1.2.2.4 Top-\( k \) Query Answering Using By-Tuple Semantics

Considering different locations and frequencies that may generate different probabilities, all mappings \( m_1, m_2, m_3, \text{ and } m_4 \) have to be viewed together. Therefore, the system has to access tuples from the first row of four prices shown in Table III, which are 240K, 250K, 230K, and 260K. Following the formula of by-tuple semantics, the current top-2 answers are 240K and 250K with \( th = 0.3 \). \( P_{MAX} \) in this process is 1.0 and will keep the probability until the system scans the last row of data (the price information of the 20\(^{th} \) house in Table 3). The process of each iteration result is shown in Appendix B, and the result of top-2 answering is 30K (probability is 1.0) and 230K (probability is 0.959).

1.2.3 PDFs Using Histograms

The Probabilistic Density Function (PDF) is employed to illustrate how a continuous random variable is distributed, but in the real-world applications, the PDF is unknown. Given \( n \) observations \( x_1, x_2, ..., x_n \) of a variable \( X \), the histogram is utilized to estimate the PDF. The area under the histogram always adds up to 1. Histograms neither impose any restrictions nor need any assumptions on distributions that are approximated in
Threshold algorithms. Arai et.al [Arai 2007] connected their top-$k$ algorithm to approximating PDFs through histograms to predict the possible distribution of tuples. The form $\text{Prob}(k\text{MinScore} > \text{MaxOthers})$ is the probability of a condition that the current top-$k$ tuples are likely to be the true answers. In this algorithm, each tuple has a minimum score and maximum score. The condition $k\text{MinScore} > \text{MaxOthers}$ constrains that the minimum scores of all candidate top-$k$ tuples must be greater than maximum scores of the tuples that are not contained in the top-$k$ list.

In our research, we assume that the histograms of all the source are available for us, and we can directly view the distributions of real data to do some estimation. The histogram presents values by equally dividing values into buckets. The number of buckets can be chosen by the user.

The buckets have distinct ranges (minimum value and maximum value of each bucket), so the values could be clustered into a corresponding bucket. For example, if the number of buckets is 5, and the price range of the source $S$ in Table I is from 30K to 280K in $m_1$ and $m_4$, and from 30K to 270K in $m_2$ and $m_3$, the system divide the values into 5 equal-width buckets for each schema mapping, as illustrated in Table IV. All the items placed in the same mapping list should be included in any of the 5 buckets.
According to the price range of the buckets, the next step is to place the prices of the query mapping $m_1, m_2, m_3, \text{ or } m_4$ (Table III) into the related field and record the number of items (including duplicates) in each bucket. The histogram of each probable schema mapping is drawn independently to others. Figure 4 illustrates the result of distributed histograms for the four schema mappings $m_1, m_2, m_3, \text{ and } m_4$ (Table III) based on the generated buckets (Table IV).

**Table IV** The price range of five buckets for distributions of source $S$

<table>
<thead>
<tr>
<th>bucket</th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$m_3$</th>
<th>$m_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(30K, 80K)</td>
<td>(30K, 78K)</td>
<td>(30K, 78K)</td>
<td>(30K, 80K)</td>
</tr>
<tr>
<td>4</td>
<td>(183K, 233K)</td>
<td>(177K, 225K)</td>
<td>(177K, 225K)</td>
<td>(183K, 233K)</td>
</tr>
</tbody>
</table>
(a) Histogram of $m_1$

(b) Histogram of $m_2$
Figure 5 PDFs based on prices in each reformulated query mapping
In reality, the number of items in the data source is very huge, which is different from the example S. In the experiment, the program is going to randomly generate 1,000,000 prices (250,000 for each of the four generated queries) that are stored.

1.2.4 Histogram-Based Approximation

Histogram-based approximate answers to the set-valued query could effectively process data distributions demanding fast responses [Ioannidis 1999]. The method converts complex SQL query into algebraic operations on histograms. Originally, a histogram on an attribute X is summarized into disjoint buckets based on a partitioning rule. Each bucket has a unique range, so each of data could be only placed into a corresponding bucket, the same as the example shown in Figure 4. Approximate values of a histogram \( H \) that would be created rely on the values lower bound \( (l_{0i}) \) and high bound \( (h_{0i}) \) of a bucket with an index \( i \), the total number of values that are placed into this bucket \( (\text{tot}_i) \), the number of distinct values in the bucket \( (\text{count}_i) \), the average spread along the dimension \( (\text{sp}_i = \frac{h_{0i} - l_{0i}}{\text{count}_i - 1}) \), and the average frequency for the bucket \( (\text{avg}_i = \frac{\text{tot}_i}{\text{count}_i}) \).

The index \( i \) is the bucket number.

We use the same example described before to explain the method shown in this part.
According to Table III and Table IV, there are 20 equal-width buckets with additional information for all the four query mappings $m_1$, $m_2$, $m_3$, and $m_4$. The result is shown in Table V.

**Table V** Summary of buckets for generated schema mappings $m_1$, $m_2$, $m_3$, and $m_4$

<table>
<thead>
<tr>
<th>mapping #</th>
<th>bucket #</th>
<th>lo</th>
<th>hi</th>
<th>tot</th>
<th>count</th>
<th>sp</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30K</td>
<td>80K</td>
<td>3</td>
<td>2</td>
<td>50K</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>81K</td>
<td>131K</td>
<td>6</td>
<td>3</td>
<td>25K</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>132K</td>
<td>182K</td>
<td>3</td>
<td>3</td>
<td>25K</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>183K</td>
<td>233K</td>
<td>5</td>
<td>3</td>
<td>25K</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>234K</td>
<td>284K</td>
<td>3</td>
<td>2</td>
<td>50K</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>30K</td>
<td>78K</td>
<td>2</td>
<td>1</td>
<td>50K</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>79K</td>
<td>127K</td>
<td>4</td>
<td>3</td>
<td>24K</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>128K</td>
<td>176K</td>
<td>3</td>
<td>1</td>
<td>48K</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>177K</td>
<td>225K</td>
<td>5</td>
<td>2</td>
<td>12K</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30K</td>
<td>78K</td>
<td>1</td>
<td>1</td>
<td>48K</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>79K</td>
<td>127K</td>
<td>4</td>
<td>2</td>
<td>24K</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>128K</td>
<td>176K</td>
<td>5</td>
<td>3</td>
<td>16K</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>177K</td>
<td>225K</td>
<td>3</td>
<td>1</td>
<td>16K</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>226K</td>
<td>274K</td>
<td>7</td>
<td>4</td>
<td>16K</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>30K</td>
<td>80K</td>
<td>2</td>
<td>2</td>
<td>50K</td>
<td>1</td>
</tr>
</tbody>
</table>
For the value $avg_i$ and $sp_i$ in the bucket $i$, when we compute these values, they may be not divisible. In this situation, we can choose either the flooring value (an approximate integer smaller than but next to the real result) or ceiling value (an approximate integer greater than but next to the real result). In our example, we use the flooring values to remove the remainder.

The approximate relation can be calculated using a SQL query.

$$\text{SELECT } (H.lo_i + I_C.idx \times H.sp_i)$$

$$\text{FROM } H, I_C, I_A$$

$$\text{WHERE } I_C.idx < H.count_i \text{ and } I_A.idx \leq H.avg_i$$

In the SQL query commands, $I_A$ and $I_C$ are auxiliary relations. $I_A$ represents integers from 0 to $A$, and $A$ is the largest average of the bucket $i$. $I_C$ contains integers 0, 1, ..., $C$. 
The value $C$ is the largest $count_i$. The query uses $I_C$ to generate positions of values within each bucket. Based on the values $lo_i$ and $sp_i$, approximate values are computed with an equal interval and the largest average of frequency.

**Table VI** Histogram-based approximate values generated for $m_1$, $m_2$, $m_3$, and $m_4$

<table>
<thead>
<tr>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$m_3$</th>
<th>$m_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>approx.value</td>
<td>avg</td>
<td>approx.value</td>
<td>avg</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
<td>30K</td>
<td>2</td>
</tr>
<tr>
<td>80K</td>
<td>1</td>
<td>79K</td>
<td>1</td>
</tr>
<tr>
<td>81K</td>
<td>2</td>
<td>103K</td>
<td>1</td>
</tr>
<tr>
<td>106K</td>
<td>2</td>
<td>127K</td>
<td>1</td>
</tr>
<tr>
<td>131K</td>
<td>2</td>
<td>128K</td>
<td>3</td>
</tr>
<tr>
<td>132K</td>
<td>1</td>
<td>177K</td>
<td>2</td>
</tr>
<tr>
<td>157K</td>
<td>1</td>
<td>225K</td>
<td>2</td>
</tr>
<tr>
<td>182K</td>
<td>1</td>
<td>226K</td>
<td>1</td>
</tr>
<tr>
<td>183K</td>
<td>1</td>
<td>238K</td>
<td>1</td>
</tr>
<tr>
<td>208K</td>
<td>1</td>
<td>250K</td>
<td>1</td>
</tr>
<tr>
<td>233K</td>
<td>1</td>
<td>262K</td>
<td>1</td>
</tr>
<tr>
<td>234K</td>
<td>1</td>
<td>274K</td>
<td>1</td>
</tr>
<tr>
<td>284K</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Approximate values are generated with their approximate averages are shown in Table VI. From the output, we can see that each schema mapping has a set of approximate items, some of which are the same as the true prices placed in Table III, but the rest are different. Moreover, the number of generated prices in each probable query mapping differs from each other. Although approximation has some deviation compared to the real items, it can effectively simplify the original complicated SQL query, which has been proved in article [Ioannidis 1999].

1.3 Challenges Motivating Proposed Work

1.3.1 Issues on Data Accuracy

After analysis, we have concluded that the existing top-k query answering approach based on by-table semantics considers neither the frequencies of currently viewed items nor the positions of them. Furthermore, from the above example explained before, we know that the calculated top-k results are not necessarily the only answer because the tuples that are not considered as top-k answering may have the same probabilities as the top-kth probability. Obviously, the results calculated using this method is not 100% accurate. According to the formula of by-tuple semantics, the probabilities of all seen items tend to be different. However, the existing by-tuple query answer approach is very expensive and time consuming in practical applications; this method does not provide
any of the conditions to prompt the system to stop running.

1.3.2 Issues on Query Processing

In theory, Luna's method on the by-table query answering could reduce the number of viewed query mappings. However, when we are looking for top-\(k\) answers, according to the stop condition of the by-table algorithm, the largest probabilities of the unseen non-top-\(k\) tuples should not exceed the minimum probability of the top-\(k^{th}\) tuple. Therefore, if there are a lot of overlapped values among the mapping lists, then the top-\(k\)-answering may be not unique due to a large number of top-\(k\) candidates that can be chosen as one of the top-\(k\) results. When this state happens, the system has to continue reading data from the next query mapping until can meet the conditions. This is the problem of the traditional by-table algorithm.

For the by-tuple query answering, the same object, which has the different number of occurrences, will be calculated with different probabilities. In addition, when the object appears in different locations of each row among reformulated mappings, the results of probabilities are also different. Hence, when the system starts scanning each line of tuples, all reformulated queries should be read. Only in this way, the real top-\(k\) results with correct probabilities using by-tuple semantics can be calculated and returned. Therefore, the existing by-tuple query answering cannot reduce the number of viewed
1.3.3 Issues on Time Consuming

Although the by-table algorithm in some cases can reduce the number of query mappings that have to be seen, for each of the traversed queries, all lines of data must be viewed because only if any reformulated query has been viewed thoroughly, then the value $P_{\text{MAX}}$ can be changed so as to be compared to $th$. $P_{\text{MAX}}$ is the largest probability of all unread query mappings. The conditions of by-table algorithm have limitations that all of the unseen and non-top-$k$ seen tuples’ maximum probabilities must be smaller than the minimum top-$k$ probabilities. In most cases, the system cannot effectively decrease the number of viewed queries. In short, because of traversing a large number of useless tuples, the current by-table algorithm is not efficient enough.

For the by-tuple approach, there is only a formula available to us. When the system processes queries, it does not have constraints to stop running, so it has to keep running until all data have been processed. In addition, the calculation of the by-tuple semantics is very complicated for each viewed item. Obviously, the existing method does not save time at all.
CHAPTER 2

SYSTEM DESIGN OR RESEARCH

2.1 Motivations and Objectives of Proposed Research

The top-$k$ mechanism supporting uncertain schema mappings in the data integration still needs to be improved. Accessing to remote data sources is very time consuming and expensive. A new enhanced approach should be more efficient and profitable compared to the current methods. Normally, most data sources have unlimited data, so the designed algorithms should not simply review all data. To reduce the number of traversed items in the uncertain data integration, there are two directions we can choose, horizontal and vertical. Vertical means to minimize the number of viewed items in each generated query mapping. Horizontal means to reduce the number of viewed queries. Usually, we do not have to view all the data, and we only need to identify the top-$k$ values with certain probabilistic guarantee.

Our first objective is to improve the by-table algorithm, which is easier to analyze compared to the by-tuple computation. The classical by-table query answering [Dong 2009], to some extent, can only reduce the number of traversed queries, and also its conditions cannot efficiently help the system extract top-$k$ answers. Our proposed
improvement on by-table semantics will minimize the number of reformulated queries that have to be traversed, and also we attempt to reduce the number of viewed tuples in each seen query mapping, which means reducing the number of viewed items from both horizontal and vertical directions.

The second objective is to develop an algorithm for the by-tuple semantics when it is applied in the uncertain query processing of data integration systems. Due to the precision of by-tuple calculation for extracting top-$k$ answering, we should make use of this semantics. Instead of processing all tuples stored in the source, we are trying to find any conditions that can effectively terminate the system. Meanwhile, when we attempt to improve the efficiency of the by-tuple algorithm, a high accuracy should also be guaranteed. Because the number of traversed query mappings cannot be minimized, we consider reducing the number from the vertical direction.

We have mentioned that if the user asks for all prices from $T$, the query can be processed as a typical one as grouping or aggregation. Instead, if the user waits for top-$k$ prices that can be found in the source $S$, the system should consider the probabilities of reformulated queries as well as the probability of each viewed price. In our research, we mainly focus on returning top-$k$ answering under uncertain schema mappings. Especially, we employ the confidence value to estimate that the current top-$k$ candidate tuples are likely to be the true results. Based on the estimation of confidence, the traversing process
can stop at an earlier stage.

For the by-table algorithms, both the number of searched mappings and the number of viewed tuples in each accessed mapping list could be minimized. Originally, we tried to use the simplest data to design our first algorithm. Suppose we know nothing about the original items, and only analyze the processed data. The so-called processed data in each generated mapping is a set of values that are ranked in ascending or descending order and each of them is unique without repeats. A prerequisite for our first algorithm is very cumbersome. However, the algorithm is not stable, and the accuracy cannot be guaranteed. We also propose another approach, which does not require such conditions sorting and removing all duplicate values. To be more specific, Histogram-based approximation is used to predict top-\(k\) answers of the real values, which is called the probability of confidence. The detailed description for the two by-table algorithms is in Chapter 2.2.

There are two by-tuple algorithms described in Chapter 2.3. According to the probability of confidence used in the second by-table algorithm, we make use of histogram-based approximation to estimate top-\(k\) results to avoid viewing the whole mapping lists. In addition, our research is working on the maximum likelihood method to improve the correctness of histogram-based estimation because the frequency of each approximate value will be estimated closely to the true data.
2.2 New Algorithms for By-Table Query Answering

To improve the efficiency of data processing, the centralized system ranks the generated queries in descending order, and the most possible mappings with higher probabilities are able to be traversed earlier than the rest ones with lower probabilities. The mappings would be traversed by following the descending order, and items in the viewed query mapping(s) start to be captured from the first row and read with a fixed number of lines at each time until a calculated confidence for top-$k$ answering satisfies the client’s requirement.

According to current seen items, a candidate result of top-$k$ tuples can be revealed through the comparison between the threshold value ($th$) and probabilities of real tuples (approach A) or approximate values (approach B). Each item has an upper-bound probability ($p_{max}$) and lower-bound probability ($p_{min}$).

**The Number of Lines to View** ($numOfLinesToView$): the data processing in our improvements would extract a single row of item(s) each time. To improve efficiency of query processing, our central global interface provides a textbox to let the user input any number of lines that the program traverses data at each temporary break point. That
means, the system reads a fixed number of rows each time while it is running. If the probability of confidence for top-\(k\) answering meets the user’s requirement, the system will halt. If the confidence does not satisfy the condition, the system continues accessing the same number of rows.

**Expected Confidence (\(\text{minConf}\)).** Confidence of top-\(k\) candidates is calculated in the estimating step. Therefore, we can make a benefit from a flexible user’s input to let our clients conveniently select an expected confidence that they demand. It is possible that if the \(\text{minconf}\) becomes lower, the number of traversed data would be smaller. The less data have been traversed, the faster and cheaper the process could be. Meanwhile, accuracy would be substituted by efficiency, and percentage of precision keeps the same tendency along with an updated \(\text{minconf}\).

In addition, we concern about returning the top-\(k\) query answering, which are the \(k\) distinct tuples with the highest probabilities. The most challenging to design an algorithm for by-table semantics is to perform the only necessary steps and reformulations in the program.

Assume that each possible query mapping provides a list of objects, and in most cases, the generated queries are likely to be assigned distinct probabilities, such as the example
shown in Table III. At the first step, the set of sub-queries are ranked in descending order based on their probabilities, and the most possible mappings will be traversed before the other mapping lists. The second step is to pick up and start traversing the top two possible mappings simultaneously.

**Theorem 1:** For a big-data source with a large number of items (greater than \( k \)), at least two probabilistic match queries need to be executed to identify elements with the top-\( k \) probabilities.

**Proof.** It is obvious that by executing only one query with the largest probability \( p_1 \), we cannot determine the top-\( k \) probabilities because all the elements of the query results have the same minimum probability \( p_{\text{min}} = p_1 \) and maximum probability \( p_{\text{max}} = 1.0 \).

According to the algorithm, we have \( \text{threshold} = p_{\text{min}} \) and \( \text{threshold} < p_{\text{max}} \).

After running the query with the second largest probability \( p_2 \), we have three cases:

**Case 1:** We find \( k \) elements with \( p_{\text{min}} = p_1 + p_2 \) and \( p_{\text{max}} = 1.0 \). Then \( \text{threshold} = p_{\text{min}} \), and the temporary maximum probability of unseen mappings \( P_{\text{MAX}} = 1 - (p_1 + p_2) \); if \( (p_1 + p_2) > (1 - p_2) \) and \( (p_1 + p_2) > P_{\text{MAX}} \), and if \( t \geq p_{\text{max}} \) (here, \( p_{\text{max}} \) is the highest maximum probability for all viewed tuples that are not placed in top-\( k \) list), those \( k \) elements are the real top-\( k \) elements, as a result, \( p_1 + p_2 > \frac{1}{2} \) and \( p_1 + 2p_2 > 1 \).

Otherwise, the next query mapping \( m_3 \) has to be accessed.
Case 2: There are no duplicate elements in the results of first two queries, all the elements have either (1) $p_{\text{min}} = p_1$ and $p_{\text{max}} = 1 - p_2$ or (2) $p_{\text{min}}' = p_2$ and $p_{\text{max}}' = 1 - p_1$. If any element with $p_{\text{min}} > p_{\text{max}}'$ and $p_{\text{max}}' > P_{\text{MAX}}$, the element must be in the top-$k$ answer. We have $p_1 > 1 - p_2$ and $p_2 > 1 - (p_1 + p_2)$, which is $p_1 > 1/2$ and $2p_1 + p_2 > 1$. Otherwise, the next query mapping $m_3$ has to be accessed.

Case 3: There are $n$ ($n < k$) elements with $p_{\text{min}} = (p_1 + p_2)$ and $p_{\text{max}} = 1.0$. The other elements have either (1) $p_{\text{min}}' = p_1$ and $p_{\text{max}}' = 1 - p_2$ or (2) $p_{\text{min}}'' = p_2$ and $p_{\text{max}}'' = 1 - p_1$. If there are $k - n$ elements with $p_{\text{min}}' > p_{\text{max}}''$ and $p_{\text{min}}'' > P_{\text{MAX}}$, those elements will be the $(k - n)$ of top-$k$ candidate results. Because the $n$ tuples with $p_{\text{min}} = (p_1 + p_2)$ have the property of $p_{\text{min}} > p_{\text{min}}'$ and $p_{\text{min}} > p_{\text{min}}''$, then $p_{\text{min}} > P_{\text{MAX}}$, and the top-$k$ answering can be achieved. Otherwise, the next query mapping $m_3$ has to be accessed.

2.2.1 Heuristic-Based Approach (Approach A)

2.2.1.1 Pre-Conditions Required

When the approach $A$ is applied, the data source is required to provide an ordered data structure in each reformulated query. That is, in each column (or mapping $m_i$), the values should be sorted in descending order. Meanwhile, all duplicate tuples in $m_i$ should be removed, such as the example shown in Table VII.
Moreover, to make the program more flexible, the systems will ask a user to input a confidence and \( k \) values. The value \( k \) is the number of top items that will be returned.

The confidence is a percentage value from 0\% (the lowest value) to 100\% (the highest value), which is an expected probability of confidence that a current top-\( k \) list provides the correct answers. Hence, the confidence is named as minconf in the program, which is

<table>
<thead>
<tr>
<th>( m_1 ) (( p_1 = 0.4 ))</th>
<th>( m_2 ) (( p_2 = 0.3 ))</th>
<th>( m_3 ) (( p_3 = 0.2 ))</th>
<th>( m_4 ) (( p_4 = 0.1 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>270K</td>
<td>270K</td>
<td>280K</td>
</tr>
<tr>
<td>240K</td>
<td>260K</td>
<td>250K</td>
<td>260K</td>
</tr>
<tr>
<td>230K</td>
<td>250K</td>
<td>240K</td>
<td>240K</td>
</tr>
<tr>
<td>210K</td>
<td>240K</td>
<td>230K</td>
<td>230K</td>
</tr>
<tr>
<td>200K</td>
<td>230K</td>
<td>190K</td>
<td>220K</td>
</tr>
<tr>
<td>180K</td>
<td>220K</td>
<td>170K</td>
<td>210K</td>
</tr>
<tr>
<td>170K</td>
<td>200K</td>
<td>160K</td>
<td>200K</td>
</tr>
<tr>
<td>160K</td>
<td>150K</td>
<td>150K</td>
<td>180K</td>
</tr>
<tr>
<td>130K</td>
<td>120K</td>
<td>100K</td>
<td>140K</td>
</tr>
<tr>
<td>100K</td>
<td>110K</td>
<td>90K</td>
<td>120K</td>
</tr>
<tr>
<td>90K</td>
<td>80K</td>
<td>30K</td>
<td>90K</td>
</tr>
<tr>
<td>80K</td>
<td>30K</td>
<td></td>
<td>50K</td>
</tr>
<tr>
<td>30K</td>
<td></td>
<td></td>
<td>30K</td>
</tr>
</tbody>
</table>
the minimum probability of confidence that should be guaranteed.

For example, if the user select \( k = 2 \), and \( \text{minconf} = 90\% \). When the system has found the top-2 answers and then calculate a real confidence that is equal to or greater than 90\%, the system stops running and returns top-2 answers.

2.2.1.2 Extracting Top-\( k \) Candidate Tuples

Step 1: When the system starts running, we generate a list of tuples’ probabilities and initialize objects to empty or set values to 0.

Step 1.1: Generate a list of tuples’ probabilities, including all possible mappings for any of the data. In the example described before, if a tuple \( t_1 \) is mapped in \( m_1 \), the probability of \( t_4 \) is 0.4; if another tuple \( t_2 \) is placed in \( m_4 \) and \( m_4 \), its probability equals 0.5. For all possible mappings of data, we only record unique probabilities without duplicates even though the tuples are mapped in different list(s). The list of all tuples’ probabilities is in Table VIII.
### Table VIII
A list of all tuples’ probabilities in $S$

<table>
<thead>
<tr>
<th>probability</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
</table>

Step 1.2: Set objects or values to initial value.

- Set $th$ to 0;
- Set index $i$ to 2; //At the beginning, the system starts viewing the first two mappings
- Set dictionary $Dict_L$ to empty; //the dictionary used to record viewed objects with their updated $p_{min}$, $p_{max}$, and all expected probabilities from $p_{min}$ to $p_{max}$;
- Set arrayList $top_k$ to empty; //the arrayList recording the top-$k$ tuples with highest probabilities

The expected probabilities mentioned in $Dict_L$ are the probabilities an unseen tuple could be. Because of the principles of $p_{min}$ and $p_{max}$, we know that the tuple’s real probability must be in the range of $[p_{min}, p_{max}]$. According to the list shown in Table, all the possible probabilities will be found and recorded into $Dict_L$.

Step 2: If $i = 2$, access a line of tuples in $m_1$ and $m_2$, process the two tuples, and place them into $Dict_L$; Find all expected probabilities for all seen tuples; if the number of
viewed tuples in $Dict_L$ is smaller than the number $k$, continue accessing the next line of tuples.

If $i > 2$, start viewing a tuple each time from $m_i$ until the number of viewed tuples in $m_i$ is the same as previous viewed mapping(s); when the number of viewed tuples in $m_i$ equals the number of seen tuples in $m_1$ to $m_{i-1}$, capture a line of tuples from $m_1$ to $m_i$.

Update processed data

Step 3: Find top-$k$ objects with top $p_{min}$, and add these objects to top_k arrayList; set $th$ to the top-$k^{th}$ $p_{min}$.

Step 4: Calculate probability of confidence that guarantees the top-$k$ objects as the real output (Chapter 2.2.1.3).

If the computed confidence is smaller than minconf, set arrayList top_k to empty; if confidence decreases compared to the last Process and the number of viewed tuples in $m_i$ is equal to that in $m_1$ to $m_{i-1}$, then set $i$ to $(i + 1)$; return back to Step 2.

Else, the computed confidence that is equal to or greater than minconf, the system terminates accessing data and return top-$k$ answers.
2.2.1.3 Computing Confidence for Top-k Candidates

In this method, to compute a confidence for top-k answering is based on the expected probabilities recorded in Dict_L. The system will summarize all distinct expected probabilities that appear in the dictionary, and record the number of occurrences of them, including duplicates.

Because we know a current th, we calculate the value A (the sum of the number of all expected probabilities that are smaller or equal to th), and then compute the value B (number of all expected probabilities recorded in Dict_L, including duplicates).

To calculate the confidence, we use the formula \( currentConf = \frac{A}{B} \).

2.2.1.4 An Example

We apply the same example shown before to help readers clearly understand the query processing by using this approach. At the beginning, we know that \( k = 2 \), and \( minconf = 90\% \), a list of probabilities shown in Table VIII, and the Table VII that provides sorted data in each of the four schema mappings.

Iteration 1:
The number of tuples viewed: 2 (total: 2)

Top-1: 280K, top-2: 270K

th = 0.3

Dict_L:

<table>
<thead>
<tr>
<th>Object</th>
<th>$p_{\text{min}}$</th>
<th>$p_{\text{max}}$</th>
<th>expected probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4, 0.5, 0.6, 0.7</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
<td>1</td>
<td>0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0</td>
</tr>
</tbody>
</table>

currentConf = 1/12 = 8.33%. Because currentConf $< \minconf$, continue the second process.

Iteration 2:

The number of tuples viewed: 2 (total: 4)

Top-1: 280K, top-2: 240K

th = 0.4

Dict_L:

<table>
<thead>
<tr>
<th>Object</th>
<th>$p_{\text{min}}$</th>
<th>$p_{\text{max}}$</th>
<th>expected probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4, 0.5, 0.6, 0.7</td>
</tr>
</tbody>
</table>
currentConf = 6/19 = 31.58%. Because currentConf < minconf, continue the next process.

Iteration 3:

- The number of tuples viewed: 2 (total: 6)
- Top-1: 280K, top-2: 240K
- th = 0.4
- Dict_L:

<table>
<thead>
<tr>
<th>Object</th>
<th>( p_{\min} )</th>
<th>( p_{\max} )</th>
<th>expected probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4, 0.5, 0.6, 0.7</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3, 0.4, 0.5, 0.6</td>
</tr>
<tr>
<td>260K</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3, 0.4, 0.5, 0.6</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
<td>0.6</td>
<td>0.3, 0.4, 0.5, 0.6</td>
</tr>
<tr>
<td>240K</td>
<td>0.4</td>
<td>1.0</td>
<td>0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0</td>
</tr>
</tbody>
</table>


\[
\begin{array}{|c|c|c|c|}
\hline
\text{Object} & p_{\text{min}} & p_{\text{max}} & \text{expected probability} \\
\hline
280K & 0.4 & 0.7 & 0.4, 0.5 \\
270K & 0.5 & 0.6 & 0.5, 0.6 \\
260K & 0.3 & 0.6 & 0.3, 0.4, 0.5, 0.6 \\
250K & 0.3 & 0.6 & 0.3, 0.4, 0.5, 0.6 \\
240K & 0.4 & 1.0 & 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 \\
230K & 0.4 & 1.0 & 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 \\
\hline
\end{array}
\]

\[
currentConf = 9/30 = 30\%. \text{ Compared to the } currentConf \text{ computed in Process 2, the probability of } currentConf \text{ decreases, so set the index } i \text{ to } (i + 1), \text{ and continue Process 4.}
\]

Iteration 4:

\[
\text{The number of tuples viewed: 1 (total: 7)}
\]

\[
\text{Top-1: 270K, top-2: 280K}
\]

\[
\text{th} = 0.4
\]

\[
\text{Dict}_L:
\]

\[
currentConf = 7/26 = 26.92\%. currentConf < \text{minconf}, \text{ continue the next process.}
\]
Iteration 5:

- The number of tuples viewed: 1 (total: 8)
- Top-1: 270K, top-2: 250K
- \( th = 0.5 \)
- \( Dict_L: \)

<table>
<thead>
<tr>
<th>Object</th>
<th>( p_{\text{min}} )</th>
<th>( p_{\text{max}} )</th>
<th>expected probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4, 0.5</td>
</tr>
<tr>
<td>270K</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5, 0.6</td>
</tr>
<tr>
<td>260K</td>
<td>0.3</td>
<td>0.4</td>
<td>0.3, 0.4</td>
</tr>
<tr>
<td>250K</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5, 0.6</td>
</tr>
<tr>
<td>240K</td>
<td>0.4</td>
<td>1.0</td>
<td>0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0</td>
</tr>
<tr>
<td>230K</td>
<td>0.4</td>
<td>1.0</td>
<td>0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0</td>
</tr>
</tbody>
</table>

\[ currentConf = 10/22 = 45\%. \] \( currentConf < minconf \), continue the next process.

Iteration 6:

- The number of tuples viewed: 1 (total: 9)
- Top-1: 240K, top-2: 270K
-- th = 0.5

-- Dict_L:

<table>
<thead>
<tr>
<th>Object</th>
<th>( p_{\min} )</th>
<th>( p_{\max} )</th>
<th>expected probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>0.4</td>
<td>0.7</td>
<td>{0.4, 0.5}</td>
</tr>
<tr>
<td>270K</td>
<td>0.5</td>
<td>0.6</td>
<td>{0.5, 0.6}</td>
</tr>
<tr>
<td>260K</td>
<td>0.3</td>
<td>0.4</td>
<td>{0.3, 0.4}</td>
</tr>
<tr>
<td>250K</td>
<td>0.5</td>
<td>0.6</td>
<td>{0.5, 0.6}</td>
</tr>
<tr>
<td>240K</td>
<td>0.6</td>
<td>1.0</td>
<td>{0.6, 0.7, 0.8, 0.9, 1.0}</td>
</tr>
<tr>
<td>230K</td>
<td>0.4</td>
<td>1.0</td>
<td>{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}</td>
</tr>
</tbody>
</table>

-- currentConf = 8/20 = 40%. currentConf < minconf, continue the next process.

Iteration 7:

-- The number of tuples viewed: 3 (total: 12)

-- Top-1: 240K, top-2: 230K

-- th = 0.6

-- Dict_L:

<table>
<thead>
<tr>
<th>Object</th>
<th>( p_{\min} )</th>
<th>( p_{\max} )</th>
<th>expected probability</th>
</tr>
</thead>
</table>
\[ \text{currentConf} = \frac{12}{22} = 54.54\%. \text{currentConf} < \minconf, \text{continue the next process.} \]

**Iteration 8:**

- The number of tuples viewed: 3 (total: 15)
- Top-1: 240K, top-2: 230K
- \( th = 0.9 \)
- \( \text{Dict}_L: \)

<table>
<thead>
<tr>
<th>Object</th>
<th>( p_{\text{min}} )</th>
<th>( p_{\text{max}} )</th>
<th>expected probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>280K</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4, 0.5</td>
</tr>
<tr>
<td>270K</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5, 0.6</td>
</tr>
<tr>
<td>260K</td>
<td>0.3</td>
<td>0.4</td>
<td>0.3, 0.4</td>
</tr>
<tr>
<td>250K</td>
<td>0.5</td>
<td>0.6</td>
<td>0.5, 0.6</td>
</tr>
<tr>
<td>240K</td>
<td>0.9</td>
<td>1.0</td>
<td>0.9, 1.0</td>
</tr>
<tr>
<td>230K</td>
<td>0.6</td>
<td>1.0</td>
<td>0.6, 0.7, 0.8, 0.9, 1.0</td>
</tr>
<tr>
<td>210K</td>
<td>0.4</td>
<td>1.0</td>
<td>0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0</td>
</tr>
</tbody>
</table>
\[ \begin{array}{|c|c|c|c|}
\hline
\text{Price} & \text{Price} & \text{Prob.} & \text{Prob.} \\
\hline
260K & 0.3 & 0.4 & 0.3, 0.4 \\
\hline
250K & 0.5 & 0.6 & 0.5, 0.6 \\
\hline
240K & 0.9 & 1.0 & 0.9, 1.0 \\
\hline
230K & 0.9 & 1.0 & 0.9, 1.0 \\
\hline
210K & 0.4 & 0.8 & 0.4, 0.5, 0.6, 0.7, 0.8 \\
\hline
200K & 0.4 & 0.8 & 0.4, 0.5, 0.6, 0.7, 0.8 \\
\hline
190K & 0.2 & 1.0 & 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 \\
\hline
\end{array} \]

\[ \text{currentConf} = \frac{29}{31} = 93.5\%. \text{ currentConf} > \text{minconf}, \text{ satisfy the user's} \]

expectation, so return the top-2 answers.

From the result of query processing by using approach A, the both number of traversed mappings and the number of seen tuples in viewed mappings can be minimized. There are three mappings \( m_1, m_2, \text{ and } m_3 \) that have been viewed. In the mapping \( m_1 \), the first 5 tuples have been seen. In the mapping \( m_2 \), the first 5 tuples have been seen. In the mapping \( m_3 \), the first 5 tuples have been seen. The total number of viewed tuples is 15.

Compared to the result of Luna’s by-table algorithm, this approach has improved.

Because of sorted prices in \( m_4 \), the probabilities for viewed tuples can be correctly determined.
2.2.2 Histogram-Based Approximate Approach in By-Table Semantics (Approach B)

2.2.2.1 Pre-Conditions Required

This approach using a different method to estimate the probability of confidence for by-table top-$k$ query answers does not require that the tuples must be ranked and duplicates must be removed before execution of the system. In most real sources, it is seldom that they provide sorted data and remove replicates, which needs a large amount of time and also does not need to be done. However, it is common that most sources provide the histograms for their data and are probably available to public, such as the example illustrated in Figure 4.

Finally, we attempt to make use of the histogram-based approximation to do the estimation in the confidence calculation. To apply this algorithm, the information of histograms should be available to us, such as a summary provided in Table V. In our program, before the execution starts, we ask the user to input the number of buckets for generating a histogram of $m_i$, and then the program automatically processes real data. After data processing, approximate values will be created for each reformulated query mapping $m_p$, as shown in Table VI.
In approach B, the confidence is computed based on the generated approximate values rather than directly analyzing the seen tuples.

2.2.2.2 Extracting Top-$k$ Candidate Tuples

Recall that the by-table query answering does not consider duplication that happens in the same mapping list, and all mappings are independent to each other. Therefore, the probability of an answer would be the sum of probabilities of generated queries when the same tuple is mapped in different mappings. The purpose of approach B is to reduce the number of viewed tuples in both horizontal and vertical directions without sorting tuples of $m_i$ and getting rid of duplicates.

At first, for each tuple $t$, we keep tracking its upper bound $p_{\text{max}}(t)$ and lower bound $p_{\text{min}}(t)$ probability. The program halts when a list of top-$k$ candidates has been found and the confidence can meet the user’s requirement. Before the process starts, the values $k$, $\text{minconf}$, and the number of lines to view at each break point $(\text{numOfLinesToView})$ will be input by the end-user. There are four main steps in this algorithm.

Step 1: Initialize local variables. The variable $q$ is the index of mappings used to record
the viewed reformulated queries. Initially, we set \( q = 2 \), which means that the top two generated queries \( Q_1 \) and \( Q_2 \) with highest probabilities \( m_1 \) and \( m_2 \) would be traversed simultaneously according to Theorem 1. We also initialize \( \text{currentConf} \) to 0;

Step 2: Execute generated queries \( Q_1, Q_2, \ldots, Q_n \) in descending order of their probabilities except the first top two queries \( Q_1 \) and \( Q_2 \). If after executing step 4 but \( q < n \) and \( \text{currentConf} < \text{minConf} \), then set \( q \) to \((q + 1)\), which means moving to the next query mapping.

Step 3: If \( \text{currentConf} > \text{minConf} \), halts and return top-\( k \) answering. Otherwise, find top-\( k \) answers from the mappings while \( q \leq n \) and \( \text{currentConf} < \text{minConf} \); set a list of top-\( k \) answers \( \text{topKList} \) to empty; initialize \( i \) to \( q \). If \( q \) equals 2, set \( i \) to 1. At the beginning, we allow the first two queries to be viewed together so as to improve the efficiency of performance. If \( q = 2 \), both tuples in \( Q_1 \) and \( Q_2 \) will be traversed from the first line. If \( q > 2 \) and \( q \leq n \), then \( i \) still equals \( q \), and when the query \( Q_i \) has been searched, the tuples will be captured from the first row until the top-\( k \) list has been fulfilled or until the number of viewed lines in \( Q_i \) equals previous viewed mapping queries \( Q_1, \ldots, Q_{i-1} \); if the top-\( k \) answering still has not been found completely, all the next line of tuples \( t_1, t_2, \ldots, t_i \) will be viewed simultaneously;
Step 3.1: At each traversing step, the number of $numOfLinesToView$ tuples have been analyzed. After processing the bunch of tuples from one or more than one reformulated queries, the tuples with top-$k$ highest probabilities in the dictionary $mappValueDict$ are placed into $topKList$; meanwhile, set $th$ to the top-$k^{th}$ $prob$.

Within the loop, each tuple $t_i$ is captured from $Q_i$:

— If the dictionary $mappValueDict$ does not contain $t_i$, set $prob(t_i)$ to $Q_i$.

and add $t_i$ with $prob(t_i)$ into $mappValueDict$. The dictionary $mappValueDict$ was defined to record all viewed tuples and their probabilities

— If the dictionary $mappValueDict$ contains $t_i$, set $prob(t_i)$ to the sum of $Q_i$ and $mappValueDict[t_i]$. $prob(t_i)$; remove and add $t_i$ with updated $prob(t_i)$ into $mappValueDict$; here, $mappValueDict[t_i]$. $prob(t_i)$ is the previous recorded probability of tuple $t_i$ stored in the dictionary $mappValueDict$. 
Step 3.2: Check the confidence of the current top-\(k\) list \(mappValueDict\). If all top-\(k\) tuples can be completely found, the confidence \(currentConf\) will be computed (Chapter 2.2.2.3). Otherwise, set \(currentConf\) to 0, which means that the top-\(k\) answering is not complete.

Step 4: Calculate the percentage of accuracy if execute our estimation by-table algorithm, 

\[
\text{accuracy} = \frac{\text{nume} \text{rOfRealTopKTuples}}{k}.
\]

Originally, the value of \(\text{nume} \text{rOfRealTopKTuples}\) is 0. In this step, we create a function \(\text{FindRealTopK}\) to record all top-\(k\) candidates derived from the traditional by-table query answering [Dong 2009]; for each \(t\) in \(\text{topKL}i\text{s}\), if \(t\) also appears in the list of top-\(k\) candidates from \(\text{FindRealTopK}\), then \((\text{nume} \text{rOfRealTopKTuples} + 1)\).

2.2.2.3 Computing Confidence for Top-\(k\) Candidates

Histogram-based approximation can be used to translate the SQL query into algebraic operations, which was exploited to determine the probability of confidence and control termination of data extraction. The confidence of top-\(k\) answering in approach \(B\) is computed from generated approximate values. In this method, we use projection with no duplicate elimination in the by-table semantics. Equi-joins of distributed histograms are implemented to calculate probabilities of each approximate value. The SQL query of Equi-joins applied in our method is defined to output three attributes, the approximate
value \( v \), its probability \( p_{(v)} \), and its frequency \( avg_{(v)} \).

\[
\text{SELECT } (H_1.lo_i + I_{N_1}.idx \times H_1.sp_i) \text{ as } v,
\]

\[
(H_1.prob + H_2.prob) \text{ as } p_{(v)},
\]

\[
(H_1.avg_i + H_2.avg_j) \text{ as } avg_{(v)}
\]

\text{FROM } H_1, H_2, I_{N_1}, I_{N_2}

\text{WHERE } (H_1.lo_i + I_{N_1}.idx \times H_1.sp_i) = (H_2.lo_j + I_{N_2}.idx \times H_2.sp_j)

\text{and } (I_{N_1}.idx < H_1.count_i) \text{ and } (I_{N_2}.idx < H_2.count_j);

In the SQL query commands of equi-joins, the histogram \( H_1 \) is joint to another histogram \( H_2 \). The index \( i \) is the bucket \( i \) appearing in the histogram \( H_1 \), and \( j \) represents the bucket \( j \) that is involved in \( H_2 \). \( I_{N_1} (I_{N_2}) \) is a set of integers 1, 2, ..., \( N_1 (N_2) \). The histogram-based approximation keeps tracking all approximate values in a histogram and compared it to another histogram. When an approximate value appears in both histograms, this value’s average and its probability will be updated.

For example, Table VI lists the generated approximate values. We use the first two
mappings \( m_1 \) and \( m_2 \) to explain how the SQL query works. The result is shown in Table IX.

Table IX Result of equi-joins between histograms of \( m_1 \) and \( m_2 \)

<table>
<thead>
<tr>
<th>( v )</th>
<th>( p_{(v)} )</th>
<th>( \text{avg}_{(v)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>30K</td>
<td>0.7</td>
<td>3</td>
</tr>
</tbody>
</table>

The SQL query shown above only contains two distinct histograms. In our example, there are four distributed histograms derived from mappings \( m_1, m_2, m_3, \) and \( m_4 \).

Furthermore, the SQL query only retains the joint values. Actually, in our algorithm, we consider all joint or non-joint approximate results, as described in Table X.

Table X Result of all approximate after equi-join SQL query

<table>
<thead>
<tr>
<th>( v )</th>
<th>( p_{(v)} )</th>
<th>( \text{avg}_{(v)} )</th>
<th>( v )</th>
<th>( p_{(v)} )</th>
<th>( \text{avg}_{(v)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>30K</td>
<td>1</td>
<td>5</td>
<td>199K</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>79K</td>
<td>0.5</td>
<td>3</td>
<td>208K</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>80K</td>
<td>0.5</td>
<td>2</td>
<td>215K</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>81K</td>
<td>0.5</td>
<td>3</td>
<td>225K</td>
<td>0.3</td>
<td>2</td>
</tr>
<tr>
<td>103K</td>
<td>0.3</td>
<td>1</td>
<td>226K</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>106K</td>
<td>0.4</td>
<td>2</td>
<td>231K</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>127K</td>
<td>0.5</td>
<td>3</td>
<td>233K</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>---</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>128K</td>
<td>0.5</td>
<td>4</td>
<td>234K</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>131K</td>
<td>0.5</td>
<td>3</td>
<td>238K</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>132K</td>
<td>0.5</td>
<td>3</td>
<td>242K</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>152K</td>
<td>0.2</td>
<td>1</td>
<td>250K</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>157K</td>
<td>0.4</td>
<td>1</td>
<td>258K</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>176K</td>
<td>0.2</td>
<td>1</td>
<td>259K</td>
<td>0.1</td>
<td>2</td>
</tr>
<tr>
<td>177K</td>
<td>0.5</td>
<td>5</td>
<td>262K</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>182K</td>
<td>0.5</td>
<td>3</td>
<td>274K</td>
<td>0.5</td>
<td>2</td>
</tr>
<tr>
<td>183K</td>
<td>0.5</td>
<td>2</td>
<td>284K</td>
<td>0.5</td>
<td>3</td>
</tr>
</tbody>
</table>

The form $currentConf = 1 - \frac{numOfObjsGreaterThanTh_{(unseen)}}{numOfObjs_{(unseen)}}$ is applied in the by-table algorithm. The variable $numOfObjsGreaterThanTh_{(unseen)}$ is the number of unviewed approximate objects whose probabilities are greater than $th$. When the value $numOfObjsGreaterThanTh_{(unseen)}$ equals 0, which means there is no any unseen tuple whose probability is greater than $th$, $currentConf$ becomes 100% (or 1.0). The value $numOfObjs_{(unseen)}$, the number of all approximate objects that have not been viewed yet, will be modified frequently. When a tuple has been viewed, the program will search for an equal or closely related approximate value. If an approximate value has been identified, the result of $numOfObjs_{(unseen)}$ will be $(numOfObjs_{(unseen)} - 1)$. 
The value \textit{currentconf} is an output of estimation whether a temporary list of top-\textit{k} answers is the real answering. \textit{currentConf} is compared with the \textit{minconf}. If the result \textit{currentConf} is greater than or equal to expected confidence (\textit{currentConf} \geq \textit{minconf}), the program will stop traversing new lines of data and start calculating accuracy and prove the final top-\textit{k} list. Otherwise, if \textit{currentConf} < \textit{minconf}, the program continues traversing the next line of items or jump to the next sorted mapping list in which begins from the first line.

Before we execute the first step of calculating confidence, initialize \textit{currentConf} to 0.

There are four main steps in this process.

Step 1: If the number of tuples in \textit{topKList} equals the number \textit{k}, execute step 2; otherwise, skip the function of checking confidence (from step 2 to step 4);

Step 2: Set $numOf\text{Obj}sthanTh_{(unseen)}$ to 0;

Step 3: For each approximate value \textit{v} in the list of all distinct approximate values, if the probability of \textit{v} is greater than \textit{th}, $num\text{OfObj}sthanTh_{(unseen)}$ is added by
The value $total_v$ is the number of frequency of $v$ that has not been viewed.

When a tuple $t$ (a real object) has been captured, if there is $v$ that equals to $t$, we will minus $total_v$ with 1; if there is no $v$ that is the same as $t$, we will modify the $total_u$ of $u$ whose value is close to $v$;

Step 4: Use the equation to calculate the result of $currentConf$.

2.2.2.4 An Example

At first, there is an interface to ask the user input the numbers $k$, $minconf$, and $numOfLinesToView$. Assume that $k = 2$, $minconf = 90\%$, and $numOfLinesToView = 4$.

Iteration 1:

- The viewed tuples: 240K, 230K, 130K, 30K (total 4)
- Top 1: 240K, top-2: 230K
- $th = 0.4$
- $currentConf = 1 - 45/62 = 27.42\%$
- Because $currentConf < minconf$, continue the next process.

Iteration 2:
The viewed tuples: 250K, 270K, 150K, 240K (total 8)

Top 1: 240K, top-2: 230K

\( th = 0.4 \)

\( currentConf = 1 - \frac{44}{58} = 24.14\% \)

Because \( currentConf < minconf \), continue the next process.

Iteration 3:

The viewed tuples: 30K, 30K, 180K, 200K (total 12)

Top 1: 240K, top-2: 30K

\( th = 0.7 \)

\( currentConf = 1 - \frac{3}{54} = 94.44\% \)

Because \( currentConf > minconf \), terminate and return the top-2 answers.

From the process of execution using the approach B, we see that the computation is not as complicated as the approach A. There are only two mappings \( m_1 \) and \( m_2 \) that have been viewed. In the mapping \( m_1 \), the first 6 tuples have been seen. In the mapping \( m_2 \), the first 6 tuples have been seen. In addition, this method does not require sorted data of generated mappings.
2.3 New Algorithms for By-Tuple Query Answering

By-tuple semantics have the form to calculate a probability of each seen item, which is $p(t) = 1 - \prod_{j=1}^{l}(1 - \sum_{i=1}^{n} p_i(t))$. Developing an algorithm that can offer any condition to terminate query processing in the by-tuple mechanism is a challenge. When there is no condition that limits the viewed number of tuples, the whole data from all generated mappings have to be viewed. The result of top-$k$ answering is definitely accurate, which needs a large amount of time and cost.

The first step of both approaches $C$ and $D$ is to use by-tuple semantics to compute $p_{\text{min}}$ of all traversed items and choose top-$k$ candidates. Due to the property of by-tuple semantics, any row of all mappings should be considered simultaneously. For instance, if a seen obj $o_1$ appears in the same row of mappings $m_1$ and $m_2$, $p_{\text{min}}$ of $o_1$ has a probability of $p(o_1) = 1 - (1 - p_{m_1}(o_1) - p_{m_2}(o_1))$. On the other hand, if $o_1$ appears in different rows of mappings $m_1$ and $m_2$, $p_{\text{min}}$ of $o_1$ has another probability of $p(o_1) = 1 - (1 - p_{m_1}(o_1))^2(1 - p_{m_2}(o_1))$. Meanwhile, $p_{\text{min}}$ of any object is tightly related to the frequency that have been viewed in mappings. When $o_1$ appears in mapping $m_1$ twice, $p_{\text{min}}$ of $o_1$ will be $p(o_1) = 1 - (1 - p_{m_1}(o_1))^2$.

Furthermore, the by-tuple based algorithm determines current top-$k$ candidates by $p_{\text{min}}$. 
of viewed objects. In our algorithm, we let program traverse items from each of all mappings together. To enhance efficiency of the program, we also ask the user to input any number of lines going to be viewed \((\text{numOfLinesToView})\) at each break point. When a calculated confidence satisfies the required confidence selected by the user \((\text{currentConf} \geq \text{minconf})\), the process will terminate in the middle rather than keep tracking until all items have been viewed. In addition, if there is no mapping that has been viewed completely, the value \(p_{max}\) of each tuple will not be changed, which always equals 1.0. therefore, \(P_{MAX}\), the maximum probability of all unseen mappings, is useless, and \(th \geq p_{max}\) (non-top-\(k\) tuples) cannot be applied here.

2.3.1 Histogram-Based Approximate Approach in By-Tuple Semantics (Approach C)

In approach \(C\), we still make use of the histogram-based approximation to estimate the confidence for a list of top-\(k\) answering and be a condition to control the termination. Considering the property of by-tuple semantics that different locations and occurrences of tuples may generate different probabilities, we cannot directly use the same way in approach \(B\) to figure out the problem of confidence estimation in the by-tuple algorithm. In this situation, we attempt to cluster the viewed tuples into different scenarios.

**Mapping-based scenarios.** There are many possible ways to group viewed objects into different cases. For example, data can be grouped into three possible cases.
case 1: Group of objects that are mapped in only one mapping

case 2: Group of objects that are mapped in all mappings

case 3: Group of objects that are mapped in more than one but less than all mappings

In another way, if the number of generated mappings is $n$, the viewed items can be clustered into $n$ cases. At the beginning, we use this method to analyze the probable confidence of top-$k$ candidates.

case 1: Group of objects that are mapped in only one mapping

case 2: Group of objects that are mapped in any two mappings

... ...

case $n$: Group of objects that are mapped in any ($n$) mappings

2.3.1.1 Pre-Conditions Required

This approach offers a set of steps to estimate the probability of confidence for by-tuple top-$k$ query answering and a condition to control if the program keeps running or terminates. This method does not require that the tuples must be ranked and duplicates must be removed before query processing.
To apply the histogram-based approximation to do estimation in the confidence calculation, the information of histograms should be available to us, such as a summary provided in Table V. In our program, we ask the user to input the number of buckets for generating a histogram for each of \( m_i \), and then the program automatically processes real data. Before the process starts, we use the function to generate all approximate values and record their average frequencies, which is the same as the SQL query and equi-joins described in the by-table semantics. After data processing, approximate values will be created for each reformulated query mapping \( m_i \), as shown in Table VI.

2.3.1.2 Extracting Top-k Candidate Tuples

The by-tuple query answering that calculates probability \( \text{prob}(t) \) of each viewed tuple \( t \) should consider both the frequency of \( t \) that has been viewed and the location of \( t \) that appears among the reformulated queries. According to the previous research [Dong 2009], the by-tuple query answering could not reduce the number of viewed queries, and even worse that all tuples in generated queries have to be viewed. Instead, although our method of by-tuple query answering has to traverse a line of tuples from all reformulated queries, we can make use of histogram-based analysis on approximate values to calculate an estimated confidence, which can be applied as a condition to control termination of accessing lines of data at an earlier stage.
Because the program extracts a line of tuples from all queries $Q_1, Q_2, \ldots, Q_n$, sorting the reformulated queries based on their probabilities $m_1, m_2, \ldots, m_n$ will not affect the efficiency of data processing. In addition, the values $k, \text{minconf}$, and $\text{numOfLinesToView}$ will be defined by the end-user. There are six main steps in this process.

Step 1: If $(\text{currentconf} < \text{minconf})$ and (at least one tuple $t$ has not been viewed), go to step 2;

Step 2: Set the top-$k$ list $\text{topKList}$ to empty, and set $th$ to 0;

Step 3: For each line $l$ of tuples $t_1, t_2, \ldots, t_n$ in $\text{numOfLinesToView}$, analyze $t$ respectively, as shown in step 3.1; add or update the record of $t$ as described in step 3.2;

Step 3.1: Initialize the dictionary $\text{currLineDict}$ to empty, which is a dictionary used to record each viewed line of tuples with their probabilities;
— If `currLineDict` does not contain `t_i`, set the probability of `t_i` to `m_i`, which means `prob(t_i) = m_i`; `i` is the index of a reformulated Query `Q_i`, and `m_i` is the probability of `Q_i`; finally, add the tuple `t_i` with its probability into `currLineDict`;

— If `currLineDict` contains `t_i`, update `prob(t_i)` that equals the sum of `Q_i` and `currLineDict[t_i].prob`; here, `currLineDict[t_i].prob` is the probability of `t_i` recorded in the dictionary `currLineDict`; remove and add the tuple `t_i` with its updated probability into `currLineDict`;

Step 3.2: For each tuple `t` in `currLineDict`, `t` is added or updated in another dictionary `L`, which contains all viewed tuples, their probabilities, and the number of mappings they are placed and found (this value would be used in the step of case classification);

— If dictionary `L` does not contain `t`, the probability of `t` is calculated by the form of by-tuple semantics, which is 

\[ prob(t) = 1 - (1 - currLineDict[t_i].prob) \] ; meanwhile, record the number of mappings `t` has been located so far; add the new tuple `t` with its probability
and the number of mappings into \( L \);

--- If dictionary \( L \) contains \( t \), then the probability of tuple \( t \) is

\[
prob(t) = 1 - (1 - L[t].prob) \times (1 - currLineDict[t_i].prob);
\]

in the equation, \( L[t].prob \) is the previous record of \( prob(t) \) recorded in \( L \); if \( t \) is placed in a new mapping, add the number of mappings by 1; remove and add the record of \( t \) in \( L \);

Step 4: If the number of tuples in \( topKList \) is less than the value \( k \) (top-\( k \) answers have not been found completely) and there are tuples that have not been viewed yet in reformulated queries, return the execution back to step 3; if top-\( k \) answers have not been searched but all the tuples have been viewed, program halts and return a warning of incomplete research; else, if \( topKList \) is fulfilled, continue the step 5;

Step 5: For each \( t \) in the dictionary \( L \), classify \( t \) into different cases based on the number of mappings that the viewed \( t \) appears; Simply, we define the cases \( C_1, C_2, \ldots, C_n \) relying on the number of mappings any tuples is located; for example, if \( t \) appears in query \( Q_1 \) and \( Q_2 \), \( t \) will be grouped into case \( C_2 \); in this step, \( realConf \) will be calculated by the process described in the section 2.3.1.3;
Step 6: If \((currentConf < minconf)\) and (at least one tuple \(t\) has not been viewed), return back to step 1. Otherwise, calculate accuracy to proof the precision of the estimated top-\(k\) tuples.

### 2.3.1.3 Computing Confidence for Top-\(k\) Candidates

When a bunch of seen tuples have been placed into separate cases, and the program is invoked to start estimating a confidence of the current top-\(k\) list, the formula

\[
currentConf = 1 - \frac{numOfObjsGreaterThanTh_{(unseen)}}{numOfObjs_{(unseen)}}
\]

will be used. Although the equation is similar to the one used in by-table algorithm, the method to calculate \(numOfObjs_{(unseen)}\) and \(numOfObjsGreaterThanTh_{(unseen)}\) are different. The value \(numOfObjs_{(unseen)}\) is the number of distinct approximate values. For example, the Table \(X\) presents all distinct approximate values generated from distributed histograms. The sum of distinct values \(v\) equals to \(numOfObjs_{(unseen)}\) (\(numOfObjs_{(unseen)} = 32\)). The result of \(numOfObjsGreaterThanTh_{(unseen)}\) is the total number of approximate objects that are greater than \(th\), which is calculated and summarized from all distributed cases. There are five main steps to analyze the value \(numOfObjsGreaterThanTh_{(unseen)}\) in any respective case.
Step 1: Find the percentage of seen items that are placed in case $i$

$$\text{percent}(\text{case } i) = \frac{\text{numOfSeenObjsInCase}_i}{\text{numOfAllSeenObjs}}.$$ Here, $\text{numOfSeenObjsInCase}_i$ is the number of viewed objects that are grouped into the case $i$. $\text{numOfAllSeenObjs}$ is the number of all viewed objects. Duplicates are not included. Note that $\text{percent}$ is not the probability of items or case $i$. For example, there are four items $A$, $B$, $C$, and $D$, suppose that $B$ and $C$ belong to case 1, and then $\text{percent}(\text{case 1}) = \frac{2}{4} = 50\%$

Step 2: Apply the formula $\text{Prob} = 1 - \pi_{k=1}^{M} (1 - x_i)^M > \text{th}$ to find the value $M$. Here, the value $x_i$ could be an optimistic probability, pessimistic probability, or expected probability of all seen tuples that are contained in the case $i$. (Optimistic probability is the largest probability of a tuple involved in the case. Pessimistic probability is derived by contrasting which tuple’s probability is the smallest. Logically, expected probability should be more precise, which is calculated based on all viewed tuples in case $i$).

Step 3: Use the histograms to estimate the number of approximate values that may greater than $\text{th}$. $M$ shown in step 2 is an approximate average frequency, and the frequency of any approximate value $k$ should satisfy the requirement ($\text{avg}_{k} > M$), which will be recorded for step 4.
Step 4: Adjust approximate values with removing duplicate ones (only consider the distinct approximate values).

Step 5: Calculate the value $numOfObjsGreaterThanTh_i$ in case $i$.

The five steps will be repeated $n$ times if there are $n$ reformulated mappings. After processing all cases, $numOfObjsGreaterThanTh_{(unseen)}$ will be generated by $numOfObjsGreaterThanTh_{(unseen)} = \sum_{i=1}^{n} numOfObjsGreaterThanTh_i$. The value $n$ is the total number of cases.

**Optimistic probability, pessimistic probability, or expected probability.** It should be noted that in Step 2, when the value $x_1$ needs to be computed, it could be an optimistic probability, pessimistic probability, or expected probability of all seen tuples that are contained in the case $i$.

The probability of the values belonging to case $i$ in the seen data is the same as the value of $p_{(c_0)}$, which is used to estimate the number of case $i$ values in the unseen data.
For example, as shown in Table III, if the program has accessed the first three instances with `house_id` 1001, 1002, and 1003, there are 12 prices that have been captured. According to the four cases, the tuples are grouped into one of them.

Case 1 (tuple that is placed in one column): 240K, 250K, 260K, 270K, 280K, 130K, 180K
Case 2 (tuple placed in two columns): 230K, 150K
Case 3 (tuple placed in three columns): 0
Case 4 (tuple placed in four columns): 0

Case 3 and Case 4 are empty, so in this situation `num0fObjsGreaterThanTh_5` and `num0fObjsGreaterThanTh_4` are zero, and we do not need to process the five steps in these two cases.

**Optimistic probability** is the largest probability of a tuple involved in a case.

In Case 1, there are tuples 240K, 250K, 260K, 270K, 280K, 130K, and 180K. The optimistic probability among these prices is 0.4. Therefore, $x_{i=1} = 0.4$. 
In Case 2, there are two tuples 230K and 150K. For the 230K,

\[ \text{Prob} = 1 - (1 - 0.2)^M(1 - 0.6)^N \leq 1 - (1 - 0.6)^{M+N}, \text{ and } x_{i=2} = 0.6; \]

for 150K,

\[ \text{Prob} = 1 - (1 - 0.5)^M, \text{ and then } x'_{i=2} = 0.5. \]

For optimistic probability, \( x_{i=2} = 0.6 \).

**Pessimistic probability** is derived by contrasting which tuple’s probability is the smallest.

In Case 1, there are tuples 240K, 250K, 260K, 270K, 280K, 130K, and 180K. The pessimistic probability among these prices is 0.1. Therefore, \( x_{i=1} = 0.1 \).

In Case 2, the price 230K has the probability \( \text{Prob} = 1 - (1 - 0.2)^M(1 - 0.6)^N \geq 1 - (1 - 0.2)^{M+N}, \text{ and } x_{i=2} = 0.2; \) for 150K, \( \text{Prob} = 1 - (1 - 0.5)^M, \text{ and then } x'_{i=2} = 0.5. \) For optimistic probability, \( x_{i=2} = 0.2 \).

**Expected probability** is calculated based on all viewed tuples in case \( i \).

In Case 1, there are tuples 240K, 250K, 260K, 270K, 280K, 130K, and 180K. The tuples 240K and 130K are mapped in \( m_1 \), 250K and 270K are mapped in \( m_2 \), and 260K, 280K, and 180K are mapped in \( m_4 \). Therefore,

\[ x_{i=1} = \frac{2}{7} \times 0.4 + \frac{2}{7} \times 0.3 + \frac{3}{7} \times 0.1 \approx 0.24. \]

In Case 2, \( x_{i=2} = \frac{1}{5} \times 0.4 + \frac{1}{5} \times 0.3 + \frac{3}{5} \times 0.2 = 0.26. \)
2.3.1.4 An Example

We use the same example shown in Table III and summary of buckets presented in Table V. In the example, we use calculating expected probability in each case. Assume that the user set values to variables

\[ \text{numOfLinesToView} = 4, \minconf = 90\%, k = 2 \]

When the system starts running, initializes \( \text{currentconf} \) to 0.

Iteration 1:

- The number of lines viewed: 4
- The number of mappings viewed: 4 (total: 16)
- Top-1: 230K, \( p_{(230K)} = 0.808 \); Top-2: 240K, \( p_{(240K)} = 0.76 \)
- \( \text{th} = 0.76 \)
- Case 1- \( \text{percent} = \frac{6}{9}, \text{expectedProb} = 0.22, M = 5, \)
  \[ \text{numOfObjsGreaterThanTh}_1 = 0 \]
- Case 2- \( \text{percent} = \frac{2}{9}, \text{expectedProb} = 0.28, M = 4, \)
  \[ \text{numOfObjsGreaterThanTh}_2 = 0 \]
- Case 3- \( \text{percent} = 0, \text{expectedProb} = 0 \)
- Case 4- \( \text{percent} = \frac{1}{9}, \text{expectedProb} = 0.25, M = 4, \)
  \[ \text{numOfObjsGreaterThanTh}_4 = 0 \]
Because \( \text{numOfObjs}_{\text{unseen}} = 32 \)

\[ \text{currentconf} = 1 - \frac{0}{32} = 1.0 = 100\% \]

Because \( \text{currentconf} > \text{minconf} \), the system halts and return top-2 answers.

From the example, we can conclude that although all generated queries have to be viewed simultaneously, Approach C can minimize the number of traversed tuples from the horizontal direction. That is, the method can effectively reduce the number of traversed rows among the schema mappings shown in Table III.

2.3.2 Maximum Likelihood Approach (Approach D)

Because of using the histogram-based estimation to summarize the distributed histograms of schema mappings is not accurate, such as shown Table V and Table VI. The number of occurrences for different approximate values in the bucket is the same, which means that the average is computed and assigned to each approximate value that appears in the same bucket. To improve accuracy of information provided by the histogram, we generate random tuples for each bucket of a histogram by using the normal distribution.

According to the equations designed for the normal distribution using the maximum
likelihood method, the two variables $\mu$ and $\sigma$ can be estimated.

$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$, the average of real tuples generated in the bucket. $n$ is the number of tuples in the bucket, and $x_i$ is the value of the tuple $i$.

$\sigma = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2$

It was proved that, in the normal distribution, the region $\mu \pm 3\sigma$ contains 99.7% of the data.

When the system generates approximate values such as shown in Table VI based on the SQL query, instead of using the average of each bucket assigned to the approximate values, we use the following three steps to calculate the distinct number of approximate values that may happen.

Step 1: Use the formula $p(v) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(v-\mu)^2}{2\sigma^2}}$ to compute the probability of each approximate value in the bucket. $v$ is an approximate value, and $p(v)$ is the likelihood that value may appear.
Step 2: After calculating $p_{(v)}$ for all generated values $v$ within a bucket, we compute the sum of these probabilities, $\sum p_{(v)}$.

Step 3: For the bucket, if the total number of occurrences is $M$, then the frequency for each $v$ in the bucket can be computed by the formula $num_{(v)} = \frac{p_{(v)}}{\sum p_{(v)}} \times M$. The value $num_{(v)}$ is the number of occurrence of approximate value $v$.

The maximum likelihood method used in approach $D$ could improve the correctness of top-$k$ answering because it is capable to improve the accuracy of generated histograms based on schema mappings. When approximation is closer to the real data, the calculated probability of top-$k$ answers’ confidence ($currentConf$) will be more accurate.
CHAPTER 3

EVALUATION AND RESULTS

The proposed algorithms are implemented on C# Console Application platform. Microsoft Visual Studio 2010 is applied to create programming projects with multiple classes, input files and output files.

To do the implementation, we use the library Random. Next to make the program randomly generate items (e.g., prices) for each of all possible schema mappings $m_i$. In reality, most data sources are very massive, which may contain millions or even billions of data. Therefore, we let our algorithms process the total 200,000 (50,000 for each of four mappings) tuples and attempt to find out top-100 answers. In our methods, we can flexibly modify the number of generated queries as well as the probabilities of schema mappings $p_i$. In our implementation, we use the same schema mappings illustrated in Table II.

3.1 Heuristic-Based Approach

Approach A applies by-table semantics to find top-$k$ answering. The interface of
approach $A$ is shown in Figure 6. The button Dong Xin’s Algorithm is the by-table query answering designed by Luna. The button New Method is to process our approach $A$.

Figure 7 is a window providing the sorted tuples in each schema mapping. Because this new approach requires the sorted lists, when the program runs, the data tuples are ranked according to their values (eg. prices). To sort data in each mapping, it took 43 minutes and 14 seconds. If the content of data source becomes bigger, the cost time will be doubled. Figure 8 shows an interface for the Luna’s by-table algorithm. In Luna’s method, the number $k$ can be selected. Because the Luna’s method can provide answering with 100% accuracy, so we know that method offered by her has 100% confidence.

![Screenshot of interface for Approach $A$](image_url)

**Figure 6** Screenshot of interface for Approach $A$
Figure 7 Screenshot of a window showing the sorted data of mappings

Figure 8 The interface for Luna’s by-table query answering
In Figure 9, this is an interface created for the new approach \( A \). The groupbox asks the user to input two numbers, \( k \) and confidence. A sample output is shown in Figure 10.
In the output information, the top-\(k\) results are shown on the screen. Meanwhile, the number of traversed tuples is shown on the screen. The number of mappings that need to be seen using the new approach depends on the confidence selected by the user.

We have tried this method, Approach A is time consuming and even spends longer time than the previous algorithm, even though it can provide top-\(k\) answering with high accuracy, and when the user select \(\text{minconf} = 100\%\), the correct results are guaranteed to be returned. Moreover, the method has constraints that the data in generated queries should be sorted and do not have duplicates.
3.2 Histogram-Based Approximate Approach in By-Table Semantics

The approach $B$ using by-table semantics does not require the ranked and non-duplicate data that are placed in each mapping, so the time on sorting tuples can be saved, compared to Approach $A$. Before the system starts to execute the second approach for top-$k$ by-table query answering, the screen asks for the number of buckets for the generated histograms, which is shown in Figure 11.

**Figure 11** Screenshot for inputting the number of buckets
The interface of Approach $B$ is shown in Figure 12. By clicking the button Histograms, the generated histograms will be shown in the screen, as shown in Figure 13.

**Figure 12** The interface of Approach $B$

**Figure 13** Output of generated distributed histograms for $m_i$
As presented in Figure 14, to improve the flexibility of using the new method, we let the user input three numbers in the groupbox for Approach B.

![Figure 14 Groupbox for asking the user to input top-k, confidence, and lines to view](image)

The summary of an output using both Luna’s algorithm and Approach B is shown in Table XI.
Table XI: The summary of an output using both Luna’s algorithm and Approach B

<table>
<thead>
<tr>
<th></th>
<th>tuples#</th>
<th>time cost</th>
<th>accuracy</th>
<th>top-k</th>
<th>Lonestoview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luna’s by-table</td>
<td>200000</td>
<td>0:3:3:158</td>
<td>1</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Approach B</td>
<td>tuples#</td>
<td>time cost</td>
<td>accuracy</td>
<td>top-k</td>
<td>conf</td>
</tr>
<tr>
<td></td>
<td>53260</td>
<td>0:0:16:964</td>
<td>1</td>
<td>100</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>53280</td>
<td>0:0:8:46</td>
<td>1</td>
<td>100</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>53280</td>
<td>0:0:5:311</td>
<td>1</td>
<td>100</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>53280</td>
<td>0:0:4:878</td>
<td>1</td>
<td>100</td>
<td>100%</td>
</tr>
</tbody>
</table>

Compared to the previous by-table query answering, Approach save a large amount of time, and also can effectively reduce the number of traversed tuples. In addition, when the number of lines to view (numOfLinesToView) increases, time cost will decrease, which saves more time.

3.3 Histogram-Based Approximate Approach in By-Tuple Semantics

The same as Approach B, before the system starts running the algorithm, the screen will prompt to ask the user to input the number of buckets created for histograms. It takes some time to do the summary for different scenarios. To improve the efficiency of execution, we also let the user input the numbers 

\[ k, \text{minconf}, \text{and numOfLinesToView}. \]
The global interface of using this approach is shown in Figure 15. The generated histograms are with 50 buckets for each are illustrated in Figure 16.

**Figure 15** Global interface of using histogram-based approximation in the by-tuple semantics
When the program executes, we provide the Luna’s by-tuple semantics to test our top-$k$ answering because the correctness of her output has been proved before. However, the by-tuple semantics takes long time on its complex calculation.

Table XII is a sample output using the histogram-based approximate approach using optimistic probability in each case, which is compared with the old by-tuple query answering. From the table, we conclude that our method is much more efficient than the
old one, and it only takes around a second to process queries. In addition, the number of viewed tuples can be minimized among all the viewed mappings. Third, the user-select confidence makes the system flexible. Fourth, the $minconf$ can effectively provide a condition and let the system stop running at an earlier stage rather than viewing all data in mappings. Finally, along with the increasing number of $numOfLinesToView$, the accuracy tends to increase, the time cost reduces, and the number of viewed tuples increases.

**Table XII** Comparison between histogram-based approach and the old by-tuple query answering

<table>
<thead>
<tr>
<th>Luna's by-tuple</th>
<th>Tuples#</th>
<th>timecost</th>
<th>accuracy</th>
<th>top-k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>200000</td>
<td>0:31:19:362</td>
<td>100%</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Approach C (Optimistic)</th>
<th>Tuples#</th>
<th>timecost</th>
<th>accuracy</th>
<th>top-k</th>
<th>conf</th>
<th>numLinesView</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3440</td>
<td>0:0:1:34</td>
<td>80%</td>
<td>100</td>
<td>95%</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>3520</td>
<td>0:0:0:696</td>
<td>82%</td>
<td>100</td>
<td>95%</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3600</td>
<td>0:0:0:525</td>
<td>87%</td>
<td>100</td>
<td>95%</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>3520</td>
<td>0:0:0:480</td>
<td>82%</td>
<td>100</td>
<td>95%</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>3600</td>
<td>0:0:0:419</td>
<td>87%</td>
<td>100</td>
<td>95%</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>4000</td>
<td>0:0:0:392</td>
<td>100%</td>
<td>100</td>
<td>95%</td>
<td>200</td>
</tr>
</tbody>
</table>
At present, we use the uniform (C# default) distribution to generate data for each reformulated mapping. In the future, we are going to test other distributions and check whether the by-table algorithms will be affected. Second, the second by-tuple (Approach D) using maximum likelihood approach will be developed and evaluated to improve the confidence estimation and effectively control the query processing. Finally, more test experiments are needed to evaluate new approaches.
CHAPTER 5

CONCLUSION

In this paper, we outlined the development for extracting top-k answering in uncertain schema mappings of the data integration system. Our purpose is to improve the methods of by-table and by-tuple tp-k query answering.

The first approach uses the viewed tuples to do the estimation on the probability of confidence how the top-k results are guaranteed to be correct. Compared to the Luna’s by-table algorithm, this method is better because it can minimize the number of viewed tuples in viewed mappings and also can reduce the number of traversed sub-queries. However, the pre-conditions of using this approach is to sort data and remove duplicate tuples before the system starts running. The second problem of using this method is time consuming on sorting data, removing repetitions, and computing top-k results.

Therefore, we provide our second by-table algorithm, which neither requires ranking data in the mappings nor asks for the mapping list to delete duplicates. This approach uses the histogram-based algorithm to estimate the probability of confidence for current top-k candidates. Moreover, the second algorithm has the same characteristics that both
the number of tuples in viewed mappings as well as the number of accessed mappings can be reduced. This method is much more efficient than both previous methods (Luna’s and Approach A).

For the by-tuple semantics, it is unlikely to reduce the number of viewed mappings because of by-tuple’s property, so we try to minimize the number of seen tuples in the mappings to enhance our by-tuple algorithm. The by-tuple algorithm is according to the histogram-based approximation, and approximate values are applied to calculate the confidence of top-k answering. The confidence in the by-tuple approach is a condition that controls the program to stop running without accessing the whole data from all the reformulated mappings. In addition, the histogram-based approximation still can be improved to make approximate values with their frequencies closer to the real data.

Therefore, we make use of the maximum likelihood method to estimate the probability for each approximate value in order to compute the number of occurrences the value may have.
REFERENCES


APPENDIX A

PESUDO CODE OF LUNA’S BY-TABLE QUERY ANSWERING

The $k$ answers are chosen from the seen tuple list, which are the objects with top probabilities. A pseudocode of returning top-$k$ by-table answering is shown in Appendix A(a). In the algorithm $ByTable$, $p_{min}$ is a variable representing the lower-bound of each tuple’s probability; $p_{max}$ is the upper-bound of the tuple’s probability; $T$ is an arraylist of $k$ buckets that will be occupied by top-$k$ candidate tuple(s) from each application; the variable $allViewedTuples$ is an arraylist recording all viewed non-duplicated objects (obj); $L$ is a Dictionary used to record each seen obj with its $p_{min}$ and $p_{max}$ values; $Q[q]$ is probability of mapping $q$, and $q \in [1, n]$; The value $n$ is the number of multiple mappings under $m$ ($m_1, m_2, m_3, \ldots, m_n$). There is a sub-method $CheckListL$ called by the task $ByTable$, which has two value parameters $th$ and $PMax$, as illustrated in Appendix A(b).

**Appendix A (a) ByTable**

1:  $P\_MAX := 1$ //The highest prob. for objects that have not been viewed yet
2:  $th := 0$ //Threshold value
3:  Rewrite array $Q$ into a set of queries $Q[0], \ldots, Q[n]$ in descending order
4:  for $q := 1$ to $n$ do // $q \in [1, n]$, $n$ is the number of multiple mappings under $m$
5:  
   foreach obj in $Q[q]$
6: \textbf{if} \( \text{obj is not in allViewedTuples} \)
7: \hspace{1em} \textbf{add} \( \text{obj into allViewedTuples} \)
8: \textbf{end if}
9: \textbf{foreach} \( \text{tuple in allViewedTuples} \) \text{ //analyze seen tuples in the list}
10: \hspace{1em} \textbf{if} \( \text{L contains tuple and Q} [q] \) \text{ contains tuple //case 1}
11: \hspace{2em} \textbf{add} \( Q[q] \) \text{ to tuple's } \( p_{min} \) \text{ in L}
12: \hspace{1em} \textbf{end if}
13: \hspace{1em} \textbf{if} \( \text{L contains tuple and Q} [q] \) \text{ not contain tuple //case 2}
14: \hspace{2em} \textbf{subtract} \( Q[q] \) \text{ from tuple's } \( p_{max} \) \text{ in L}
15: \hspace{1em} \textbf{end if}
16: \hspace{1em} \textbf{if} \( P_{MAX} \geq th \) \text{ and L not contain tuple}
17: \hspace{2em} \text{and} \( Q[q] \) \text{ contains tuple}
18: \hspace{2em} \textbf{add} \( \text{tuple in L} \)
19: \hspace{1em} \textbf{end if} \text{ //case 3}
20: \textbf{end foreach}
21: \text{//record next } \( p_{max} \text{ value for unseen tuples}
22: \hspace{1em} P_{MAX} := P_{MAX} - Q[q];
23: \text{//record kth largest } \( p_{min} \text{ values}
24: \hspace{1em} \text{Record top-k candidate tuple(s) into } T
25: \hspace{1em} th := \text{top-kth tuple's } \( p_{min} \text{ //the tuple is in } T
26: \hspace{1em} \text{if CheckListL(th, P_{MAX}) = true //Call the subtask CheckList()}
27: \hspace{2em} q := n + 1; \text{//end loop}
28: \textbf{end foreach}
29: \textbf{end for}
The pseudocode of CheckListL is applied to examine whether the current candidate top-$k$ results can satisfy two requirements. In the first place, threshold $th$ should be greater than $P_{MAX}$. Second, the $p_{max}$ value of any item that is allocated in the $L$ dictionary but not involved in the $T$ arraylist should be smaller than the current $th$. If the current answering can meet the two conditions, CheckListL will return true to the calling method ByTable. Finally, the Appendix A(a) will stop processing the rest unseen data to return the selected top-$k$ tuples to end users.

---

**Appendix A (b) CheckListL($th$, $P\_MAX$)**

1: \textbf{if} $th \leq P\_MAX$

2: \hspace{1cm} \textbf{return} false

3: \hspace{1cm} \textbf{end if}

3: \hspace{1cm} \textbf{foreach} item in $L$

4: \hspace{2cm} \textbf{if} item not in $T$

5: \hspace{3cm} \textbf{if} $L[item].p_{max} \geq th$ \text{// if the $p_{max}$ value of item listed in $L \geq th$}

6: \hspace{3cm} \textbf{return} false

7: \hspace{3cm} \textbf{end if}

8: \hspace{1cm} \textbf{end if}

9: \hspace{1cm} \textbf{end foreach}

10: \textbf{return} true
APPENDIX B

PSEUDO CODE OF LUNA’S BY-TUPLE QUERY ANSWERING

Appendix B (a) is the method to traverse data from multiple mappings and find top-$k$ answers based on the tuples’ minimum probabilities.

---

Appendix B (a) ByTuple

```java
    // <Step 1 Initialize variables>
1:    allViewedTuples := new ArrayList(); // list of all viewed non-duplicated objects
2:    L := new Dictionary<int, double>(); // key: item, value: probability
3:    Dictionary<int, double> currLineDict; // key: item, value: p_min
4:    double PMax := 1; // the maximum unseen mappings' probability
5:    bool checkresult := false;
6:    threshold := 0;

    // <Step 2 Try to open the input file>
7:    inFile := new StreamReader("PossibleMappingLists.txt"); // Open the file
8:    Rewrite Q into a set of queries Q[0], ...Q[n] in descending order

    // <Step 3: Execute mappings line by line>
9:    string arraydata; // Read the first line
10:   int eachdata; // each integer object
11:   string[] worddata; // a line of data in mappings
12:   EachData anobj; // an obj
13:   ArrayList currlst; // the current line of objects
14:   double prob; // probability of each obj
```
arraydata = inFile.ReadLine(); //Read the first data

while (arraydata != null && arraydata.Length != 0 && checkresult == false)

//Split the arraydata
worddata := arraydata.Split();
currLineDict := new Dictionary<int, double>(); //key: item, value: p_min
currList := new ArrayList();
//Process each object column by column
for (int i = 1; i < worddata.Length - 1; i++)
eachdata := int.Parse(worddata[i]);
if (viewedTupleNumber[i - 1] == NUMBER_OF_SAMPLES)
PMax := Q[i - 1]; //update PMax
anobj := new EachData(eachdata);
if (!currLineDict.ContainsKey(eachdata))
currLineDict.Add(eachdata, Q[i - 1]); //add distinct object
currList.Add(anobj);
else //the dictionary currLineDict contains the object
    //duplicate object in the same line
    //change the p_min of the object
double p_min := currLineDict[eachdata] + Q[i - 1];
currLineDict.Remove(eachdata);
currLineDict.Add(eachdata, p_min);
    //update L, allViewedTuples
    //L: all the viewed objects, the value: current prob. of each object
    //allViewedTuples: all viewed distinct object list
foreach (EachData obj in currList)
    //insert obj into L if not exist
if (!L.ContainsKey(obj.item))
    prob := 1 - (1 - currLineDict[obj.item]);
L.Add(obj.item, prob);
allViewedTuples.Add(obj);
else //contains the object, update prob.
    prob := 1 - (1 - L[obj.item]) * (1 – currLineDict[obj.item]);
L.Remove(obj.item);
L.Add(obj.item, prob);

//update top-K candidate list
//T: top-K candidate objects
T := new ArrayList(); //a list of k tuples with top p_min values
//Find top-k answers with higher probabilities
foreach top-kth tuple:
    T.Add(tuple);
threshold := topkthpmin;
checkresult := CheckListL(threshold, PMax, outFile);
arraydata := inFile.ReadLine(); //Read next line of data
end for loop
//Close the input file
inFile.Close();
end while

The method Appendix B (b) requires two parameters, th and PMax. The parameter th is the current top-k th probability. PMax is the current maximum probability of unseen
mappings. The method returns true if the result can satisfy all conditions, and returns false if any condition cannot be satisfied.

---

**Appendix B (b) ByTuple(double th, double PMax)**

1:   bool result := true;
2:   if (th <= PMax)
3:       outFile.WriteLine("* Error: th <= PMax");
4:   result := false;
5:   ArrayList ObjInLNotInT: = new ArrayList();
6:   foreach (EachData item in allViewedTuples)
7:       Boolean haveObj := false;
8:   foreach (EachData obj in T)
9:       if (obj.item == item.item)
10:          haveObj := true;
11:   if (haveObj == false)
12:      ObjInLNotInT.Add(item);
13:   foreach (EachData item in ObjInLNotInT)
14:      if (Math.Round(L[item.item], 4) >= Math.Round(th, 4))
15:         outFile.WriteLine("Error happens in {0} >= th({1})", L[item.item], th);
16:   result := false;
17:   return result;
## APPENDIX C

### EXECTION AND RESULT OF TOP-K BY-TUPLE QUERY ANSWERING

<table>
<thead>
<tr>
<th>Process: 1</th>
<th>The number of viewed tuples: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( P_{\text{MAX}} = 1 )</td>
</tr>
<tr>
<td></td>
<td>( L ) list</td>
</tr>
<tr>
<td></td>
<td>Object  Probability</td>
</tr>
<tr>
<td></td>
<td>240K   0.4</td>
</tr>
<tr>
<td></td>
<td>250K   0.3</td>
</tr>
<tr>
<td></td>
<td>230K   0.2</td>
</tr>
<tr>
<td></td>
<td>260K   0.1</td>
</tr>
<tr>
<td></td>
<td>( T ) list (Candidate top-2 objects)</td>
</tr>
<tr>
<td></td>
<td>240K</td>
</tr>
<tr>
<td></td>
<td>250K</td>
</tr>
<tr>
<td></td>
<td>( P_{\text{MAX}} = 1, \ th = 0.3 )</td>
</tr>
<tr>
<td></td>
<td>* Error: ( th \leq P_{\text{MAX}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process: 2</th>
<th>The number of viewed tuples: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( P_{\text{MAX}} = 1 )</td>
</tr>
<tr>
<td></td>
<td>( L ) list</td>
</tr>
<tr>
<td></td>
<td>Object  Probability</td>
</tr>
<tr>
<td></td>
<td>240K   0.4</td>
</tr>
<tr>
<td></td>
<td>250K   0.3</td>
</tr>
<tr>
<td></td>
<td>230K   0.68</td>
</tr>
<tr>
<td></td>
<td>260K   0.1</td>
</tr>
<tr>
<td></td>
<td>270K   0.3</td>
</tr>
<tr>
<td></td>
<td>280K   0.1</td>
</tr>
<tr>
<td></td>
<td>( T ) list (Candidate top-2 objects)</td>
</tr>
<tr>
<td></td>
<td>230K</td>
</tr>
<tr>
<td></td>
<td>240K</td>
</tr>
<tr>
<td></td>
<td>( P_{\text{MAX}} = 1, \ th = 0.4 )</td>
</tr>
<tr>
<td></td>
<td>* Error: ( th \leq P_{\text{MAX}} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process: 3</th>
<th>The number of viewed tuples: 12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( P_{\text{MAX}} = 1 )</td>
</tr>
<tr>
<td></td>
<td>( L ) list</td>
</tr>
<tr>
<td></td>
<td>Object  Probability</td>
</tr>
<tr>
<td></td>
<td>240K   0.4</td>
</tr>
<tr>
<td></td>
<td>250K   0.3</td>
</tr>
<tr>
<td></td>
<td>230K   0.68</td>
</tr>
<tr>
<td></td>
<td>260K   0.1</td>
</tr>
<tr>
<td></td>
<td>270K   0.3</td>
</tr>
<tr>
<td></td>
<td>280K   0.1</td>
</tr>
<tr>
<td></td>
<td>130K   0.4</td>
</tr>
<tr>
<td></td>
<td>150K   0.5</td>
</tr>
<tr>
<td></td>
<td>180K   0.1</td>
</tr>
</tbody>
</table>
T list (Candidate top-2 objects)
230K
150K
\( P_{\text{MAX}} = 1, \ th = 0.5 \)
* Error: \( th \leq P_{\text{MAX}} \)

Process: 4
The number of viewed tuples: 16
\( P_{\text{MAX}} = 1 \)
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.1</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
</tr>
<tr>
<td>280K</td>
<td>0.1</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.5</td>
</tr>
<tr>
<td>180K</td>
<td>0.1</td>
</tr>
</tbody>
</table>
T list (Candidate top-2 objects)
230K
240K
\( P_{\text{MAX}} = 1, \ th = 0.76 \)
* Error: \( th \leq P_{\text{MAX}} \)

Process: 5
The number of viewed tuples: 20
\( P_{\text{MAX}} = 1 \)
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.1</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
</tr>
<tr>
<td>280K</td>
<td>0.1</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.5</td>
</tr>
<tr>
<td>180K</td>
<td>0.1</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
</tbody>
</table>
T list (Candidate top-2 objects)
30K
230K
\( P_{\text{MAX}} = 1, \ th = 0.808 \)
* Error: \( th \leq P_{\text{MAX}} \)

Process: 6
The number of viewed tuples: 24
\( P_{\text{MAX}} = 1 \)
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.1</td>
</tr>
<tr>
<td>Object</td>
<td>Probability</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
</tr>
<tr>
<td>280K</td>
<td>0.1</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.5</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.4</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
</tbody>
</table>

T list (Candidate top-2 objects)

30K
230K

P_MAX = 1, th = 0.808
* Error: th <= P_MAX

Process: 7
The number of viewed tuples: 28
P_MAX = 1
L list

<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.1</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
</tr>
<tr>
<td>280K</td>
<td>0.1</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.5</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.4</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
<tr>
<td>90K</td>
<td>0.4</td>
</tr>
<tr>
<td>120K</td>
<td>0.3</td>
</tr>
<tr>
<td>100K</td>
<td>0.2</td>
</tr>
<tr>
<td>140K</td>
<td>0.1</td>
</tr>
</tbody>
</table>

T list (Candidate top-2 objects)

30K
230K

P_MAX = 1, th = 0.808
* Error: th <= P_MAX

Process: 8
The number of viewed tuples: 32
P_MAX = 1
L list

<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.19</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
</tr>
<tr>
<td>280K</td>
<td>0.1</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.5</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>Object</td>
<td>Probability</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.64</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
<tr>
<td>90K</td>
<td>0.4</td>
</tr>
<tr>
<td>120K</td>
<td>0.3</td>
</tr>
<tr>
<td>100K</td>
<td>0.2</td>
</tr>
<tr>
<td>140K</td>
<td>0.1</td>
</tr>
<tr>
<td>220K</td>
<td>0.3</td>
</tr>
<tr>
<td>190K</td>
<td>0.2</td>
</tr>
</tbody>
</table>

T list (Candidate top-2 objects)
30K
230K

P_MAX = 1, th = 0.808
* Error: th <= P_MAX

Process: 9
The number of viewed tuples: 36
P_MAX = 1
L list
Object         Probability
240K           0.76
250K           0.3
230K           0.808
260K           0.19
270K           0.3
280K           0.1
130K           0.4
150K           0.5
180K           0.46
30K            1
200K           0.64
170K           0.2
90K            0.82
120K           0.3
100K           0.2
140K           0.1
220K           0.3
190K           0.2
80K            0.3

T list (Candidate top-2 objects)
30K
90K

P_MAX = 1, th = 0.82
* Error: th <= P_MAX

Process: 10
The number of viewed tuples: 40
P_MAX = 1
L list
Object         Probability
240K           0.76
250K           0.3
230K           0.808
260K           0.19
270K           0.3

<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.19</td>
</tr>
<tr>
<td>270K</td>
<td>0.3</td>
</tr>
<tr>
<td>280K</td>
<td>0.19</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.6</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.64</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
<tr>
<td>90K</td>
<td>0.856</td>
</tr>
<tr>
<td>120K</td>
<td>0.37</td>
</tr>
<tr>
<td>100K</td>
<td>0.52</td>
</tr>
<tr>
<td>140K</td>
<td>0.1</td>
</tr>
<tr>
<td>220K</td>
<td>0.51</td>
</tr>
<tr>
<td>190K</td>
<td>0.2</td>
</tr>
<tr>
<td>80K</td>
<td>0.3</td>
</tr>
<tr>
<td>210K</td>
<td>0.4</td>
</tr>
<tr>
<td>110K</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Process: 11  The number of viewed tuples: 44
P_MAX = 1
L list

Process: 12  The number of viewed tuples: 48
P_MAX = 1
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.76</td>
</tr>
<tr>
<td>250K</td>
<td>0.3</td>
</tr>
<tr>
<td>230K</td>
<td>0.808</td>
</tr>
<tr>
<td>260K</td>
<td>0.514</td>
</tr>
<tr>
<td>270K</td>
<td>0.44</td>
</tr>
<tr>
<td>280K</td>
<td>0.514</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.6</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.64</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
<tr>
<td>90K</td>
<td>0.856</td>
</tr>
<tr>
<td>120K</td>
<td>0.37</td>
</tr>
<tr>
<td>100K</td>
<td>0.52</td>
</tr>
<tr>
<td>140K</td>
<td>0.1</td>
</tr>
<tr>
<td>220K</td>
<td>0.51</td>
</tr>
<tr>
<td>190K</td>
<td>0.2</td>
</tr>
<tr>
<td>80K</td>
<td>0.3</td>
</tr>
<tr>
<td>210K</td>
<td>0.4</td>
</tr>
<tr>
<td>110K</td>
<td>0.3</td>
</tr>
</tbody>
</table>

T list (Candidate top-2 objects)
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>90K</td>
<td>0.856</td>
</tr>
</tbody>
</table>

P_MAX = 1, th = 0.856
* Error: th <= P_MAX

Process: 13
The number of viewed tuples: 52
P_MAX = 1
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.784</td>
</tr>
<tr>
<td>250K</td>
<td>0.44</td>
</tr>
<tr>
<td>230K</td>
<td>0.9424</td>
</tr>
<tr>
<td>260K</td>
<td>0.514</td>
</tr>
<tr>
<td>270K</td>
<td>0.44</td>
</tr>
<tr>
<td>280K</td>
<td>0.514</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.6</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.64</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
<tr>
<td>90K</td>
<td>0.856</td>
</tr>
<tr>
<td>120K</td>
<td>0.37</td>
</tr>
<tr>
<td>100K</td>
<td>0.52</td>
</tr>
<tr>
<td>140K</td>
<td>0.1</td>
</tr>
<tr>
<td>220K</td>
<td>0.51</td>
</tr>
<tr>
<td>190K</td>
<td>0.2</td>
</tr>
<tr>
<td>80K</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Process: 14  The number of viewed tuples: 56
P_MAX = 1
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.784</td>
</tr>
<tr>
<td>250K</td>
<td>0.44</td>
</tr>
<tr>
<td>230K</td>
<td>0.9424</td>
</tr>
<tr>
<td>260K</td>
<td>0.514</td>
</tr>
<tr>
<td>270K</td>
<td>0.44</td>
</tr>
<tr>
<td>280K</td>
<td>0.514</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.68</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.64</td>
</tr>
<tr>
<td>170K</td>
<td>0.2</td>
</tr>
<tr>
<td>90K</td>
<td>0.856</td>
</tr>
<tr>
<td>120K</td>
<td>0.37</td>
</tr>
<tr>
<td>100K</td>
<td>0.52</td>
</tr>
<tr>
<td>140K</td>
<td>0.19</td>
</tr>
<tr>
<td>220K</td>
<td>0.51</td>
</tr>
<tr>
<td>190K</td>
<td>0.2</td>
</tr>
<tr>
<td>80K</td>
<td>0.79</td>
</tr>
<tr>
<td>210K</td>
<td>0.4</td>
</tr>
<tr>
<td>110K</td>
<td>0.3</td>
</tr>
</tbody>
</table>

T list (Candidate top-2 objects)
30K
230K
P_MAX = 1, th = 0.9424
* Error: th <= P_MAX

Process: 15  The number of viewed tuples: 60
P_MAX = 1
L list
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.784</td>
</tr>
<tr>
<td>250K</td>
<td>0.44</td>
</tr>
<tr>
<td>230K</td>
<td>0.94816</td>
</tr>
<tr>
<td>260K</td>
<td>0.514</td>
</tr>
<tr>
<td>270K</td>
<td>0.44</td>
</tr>
<tr>
<td>280K</td>
<td>0.514</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.776</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
</tbody>
</table>
200K 0.64
170K 0.2
90K 0.856
120K 0.37
100K 0.712
140K 0.19
220K 0.51
190K 0.2
80K 0.79
210K 0.4
110K 0.3
160K 0.2

T list (Candidate top-2 objects)
30K
230K

P_MAX = 1, th = 0.94816
* Error: th <= P_MAX

Process: 16
The number of viewed tuples: 64
P_MAX = 1
L list
Object Probability
240K 0.784
250K 0.44
230K 0.94816
260K 0.514
270K 0.44
280K 0.514
130K 0.4
150K 0.776
180K 0.46
30K 1
200K 0.748
170K 0.52
90K 0.856
120K 0.37
100K 0.712
140K 0.19
220K 0.51
190K 0.36
80K 0.79
210K 0.46
110K 0.3
160K 0.2

T list (Candidate top-2 objects)
30K
230K

P_MAX = 1, th = 0.94816
* Error: th <= P_MAX

Process: 17
The number of viewed tuples: 68
P_MAX = 1
L list
Object | Probability
--- | ---
240K | 0.784
250K | 0.44
230K | 0.958528
260K | 0.514
270K | 0.44
280K | 0.514
130K | 0.4
150K | 0.776
180K | 0.46
30K | 1
200K | 0.748
170K | 0.52
90K | 0.856
120K | 0.37
100K | 0.712
140K | 0.19
220K | 0.706
190K | 0.36
80K | 0.79
210K | 0.46
110K | 0.3
160K | 0.52

T list (Candidate top-2 objects)
30K
230K

P\_MAX = 1, th = 0.958528
* Error: th <= P\_MAX

---

Process: 18
The number of viewed tuples: 72
P\_MAX = 1

L list
Object | Probability
--- | ---
240K | 0.784
250K | 0.44
230K | 0.958528
260K | 0.514
270K | 0.44
280K | 0.514
130K | 0.4
150K | 0.776
180K | 0.46
30K | 1
200K | 0.748
170K | 0.52
90K | 0.8848
120K | 0.37
100K | 0.712
140K | 0.19
220K | 0.706
190K | 0.36
80K | 0.79
210K | 0.46
<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.9136</td>
</tr>
<tr>
<td>250K</td>
<td>0.44</td>
</tr>
<tr>
<td>230K</td>
<td>0.958528</td>
</tr>
<tr>
<td>260K</td>
<td>0.514</td>
</tr>
<tr>
<td>270K</td>
<td>0.608</td>
</tr>
<tr>
<td>280K</td>
<td>0.5626</td>
</tr>
<tr>
<td>130K</td>
<td>0.4</td>
</tr>
<tr>
<td>150K</td>
<td>0.776</td>
</tr>
<tr>
<td>180K</td>
<td>0.46</td>
</tr>
<tr>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.748</td>
</tr>
<tr>
<td>170K</td>
<td>0.52</td>
</tr>
<tr>
<td>90K</td>
<td>0.8848</td>
</tr>
<tr>
<td>120K</td>
<td>0.37</td>
</tr>
<tr>
<td>100K</td>
<td>0.712</td>
</tr>
<tr>
<td>140K</td>
<td>0.19</td>
</tr>
<tr>
<td>220K</td>
<td>0.706</td>
</tr>
<tr>
<td>190K</td>
<td>0.36</td>
</tr>
<tr>
<td>80K</td>
<td>0.79</td>
</tr>
<tr>
<td>210K</td>
<td>0.46</td>
</tr>
<tr>
<td>110K</td>
<td>0.3</td>
</tr>
<tr>
<td>160K</td>
<td>0.52</td>
</tr>
<tr>
<td>50K</td>
<td>0.1</td>
</tr>
</tbody>
</table>

* Error: th <= P_MAX

The number of viewed tuples: 76

Process: 20

The number of viewed tuples: 80

<table>
<thead>
<tr>
<th>Object</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>240K</td>
<td>0.9136</td>
</tr>
<tr>
<td>250K</td>
<td>0.44</td>
</tr>
<tr>
<td>230K</td>
<td>0.958528</td>
</tr>
<tr>
<td>260K</td>
<td>0.514</td>
</tr>
<tr>
<td>270K</td>
<td>0.608</td>
</tr>
<tr>
<td>280K</td>
<td>0.5626</td>
</tr>
<tr>
<td>130K</td>
<td>0.64</td>
</tr>
</tbody>
</table>

* Error: th <= P_MAX
<table>
<thead>
<tr>
<th>Size</th>
<th>Value</th>
<th>Size</th>
<th>Value</th>
<th>Size</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>150K</td>
<td>0.8432</td>
<td>180K</td>
<td>0.514</td>
<td>30K</td>
<td>1</td>
</tr>
<tr>
<td>200K</td>
<td>0.748</td>
<td>170K</td>
<td>0.52</td>
<td>90K</td>
<td>0.8848</td>
</tr>
<tr>
<td>120K</td>
<td>0.37</td>
<td>100K</td>
<td>0.712</td>
<td>140K</td>
<td>0.19</td>
</tr>
<tr>
<td>220K</td>
<td>0.706</td>
<td>190K</td>
<td>0.488</td>
<td>80K</td>
<td>0.79</td>
</tr>
<tr>
<td>210K</td>
<td>0.46</td>
<td>110K</td>
<td>0.3</td>
<td>160K</td>
<td>0.52</td>
</tr>
<tr>
<td>50K</td>
<td>0.1</td>
<td>30K</td>
<td>0.46</td>
<td>230K</td>
<td>P_MAX = 1, th = 0.958528</td>
</tr>
</tbody>
</table>

**T list (Candidate top-2 objects)**

30K
230K

---

The processing time:
Hours : Minutes : Seconds : milliseconds
0 : 0 : 0 : 15
APPENDIX D

PSEUDO CODE OF THE HEURISTIC-BASED APPROACH

Initially, the centralized mediator starts to capture data from the top-k query mappings $Q[1]$ and $Q[2]$, as shown in Algorithm 3. There is a condition of the iteration: if the number of viewed mappings ($viewedMappings$) is 2 and the estimated confidence ($estimConf$) is equal or greater than its previous estimated confidence ($prevEstimConf$), the program will continue catching the next line of tuples from $Q[1]$ and $Q[2]$. In the other words, if either condition cannot be satisfied, the program will stop to capture data from the next mapping $Q[3]$. There are variables defined and initialized before the Algorithm 3: $anObj[q]$ is an item caught from mapping $Q[q]$ $q \in [1,n]$, and $n$ is the number of multiple mappings under $m$. $Dict_L$ is a dictionary for recording the viewed elements with current $p_{\text{min}}$ and $p_{\text{max}}$ probabilities. $objBound[q]$ has two variables $p_{\text{min}}$ and $p_{\text{max}}$, which is used be calculate and record $anObj[q]$’s current $p_{\text{min}}$ and $p_{\text{max}}$; the original of $objBound[q].p_{\text{max}}$ equals 1 for each being viewed tuple, and $objBound[q].p_{\text{min}}$ equals $Q[q]$. $LLastObj$ is set to null at the beginning, which is an element that is the smallest seen tuple stored in $Dict_L$; $LLastObj$ has three components, $obj$ that is the smallest object, $mappingi$ that is the mapping number of $obj$, and $objNum$ tracks the row number of $obj$ in $maappingi$. $viewLine$ is set to 1, which means that the algorithm start viewing data from the first line of mapping lists. $Conf$ is false originally before the program begins, which will be derived from a sub-method named $CheckConf$. 
There are two references passed be CheckConf, anObj and the address of estimConf. estimConf will be computed through CheckConf.

Appendix D(a) ByTableTopK

1:  \textbf{if} Conf = false //estimConf < minconf
2:  \textbf{do}
3:      prevEstimConf := estimConf
5:  \textbf{if} anObj[1] \neq \text{null} \textbf{and} anObj[2] \neq \text{null}
//<step 1.1 Modify p_max of anObj[1] and anObj[2]>
//<step 1.2 Examine previous (seen) tuples in Dict_L>
10: \textbf{if} LLastObj.obj \neq \text{null}
//LLastObj in the mapping No.1
11: \textbf{if} LLastObj.mappingi = 1
12: \textbf{while} LLastObj.obj > anObj[2] \textbf{and}
    LLastObj.obj \neq anObj[1]
13: \textbf{if} LLastObj.obj’s p_min =
15:     Dict_L updates LLastObj.obj
16: \[LLLastObj.objNum := LLastObj.objNum + 1;\]
   //move to the next line
17: Get next Obj behind \(LLLastObj\) from mapping 1; \(LLLastObj := \text{Obj}\)
18: \textbf{else if} \(LLLastObj\).\textit{mapping} = 2 //\(LLLastObj\) is a tuple of mapping No.2
19: \[\text{//change mapping No.1 to 2 or 2 to 1;}\]
    \[\text{//other commands are the same as line:11 - 17}\]
///<Step 1.3 insert \(anObj[1]\) and \(anObj[2]\) into \(Dict_L\)>
20: \textbf{if} \(Dict_L\) not contains \(anObj[1]\)
21: \[Dict_L.Add(anObj[1], objBound[1]) //\text{insert obj}\]
    into dictionary
22: \textbf{else} //\(Dict_L\) contains \(anObj[1]\)
23: \[anObj[1]'s~p\_min += objBound[1].p\_min\]
24: Update \(anObj[1]\) in \(Dict_L\)
25: \[\text{//similar to line: 20 – 24; insert \(anObj[2]\) or update}\]
    \(anObj[2] \text{'s p\_min}\) in \(Dict_L\)
///<Step 1.4 update variable \(LLLastObj.obj\)>
26: \textbf{if} \(anObj[2] < anObj[1]\)
27: \[LLLastObj.obj := anObj[2]\]
28: \[LLLastObj.missing := 2;\]
29: \textbf{else} //\(anObj[1] \leq anObj[2]\)
30: \[LLLastObj.obj := anObj[1]\]
31: \[LLLastObj.missing := 1;\]
32: \( \text{LLastObj.objNum} := \text{viewLine} \)

///<Step 1.5 check estimated confidence>

33: \( \text{if viewedMappings} = 2 \)

34: \( \text{Conf} := \text{CheckConf(anObj, &estimConf)} \)

35: \( \text{viewLine} += 1 \)

36: \( \text{while estimConf} \geq \text{prevEstimConf} \text{ and viewedMappings} = 2 \)

37: \( \text{viewedMappings} += 1 \)

38: \( \text{end if} \)

When \( \text{estimConf} \geq \text{minconf} \) after Appendix D(a) stopped processing, the system will consider the current top-\( k \) elements as real answering and return back to end users. Otherwise, the top-3\(^{rd} \) mapping will be traversed, as in Appendix D(b). When the \( \text{estimConf} \) becomes decreasing after the iteration presented in Appendix D(a), as well as the estimated confidence still cannot meet the user’s requirement (\( \text{minconf} \)), the program will start traversing the next column with higher probability than the other unviewed query mappings. There are two variables used in Appendix D(b): \( \text{viewedObjNum} \) is set to be 0 initially, which keeps tracking the number of elements that have been traversed in the current list. \( \text{newObj} \) is the largest element stored in \( \text{Dict}_L \), applied to modify \( p_{max} \) probabilities of viewed elements stored in \( \text{Dict}_L \) when a new item is going to be added and recorded to the seen-data dictionary.

---

**Appendix D(b) ByTableTopK (Contd...)**

1: \( \text{if prevEstimConf} > \text{estimConf and Conf} = \text{false} \)
2:    if Conf = false
3:        do
4:            Traverse a new \( anObj[q] \) from \( Q[q] \)
5:            \( \text{viewedObjNum} += 1 \)
6:            \( \text{objBound}[q].p_{\text{max}} = 1; \)
7:            \( \text{objBound}[q].p_{\text{min}} = Q[q] \)
8:            if \( anObj[q] \) != null
9:                // Step 2.1: Compare \( anObj[q] \) with other viewed objects
10:               if \( \text{Dict}_L \) not contain \( anObj[q] \)
11:                if \( anObj[q] > \text{LLastObj}.obj \) /// \( \text{LLastObj} \): the smallest obj in \( \text{Dict}_L \)
12:                    for \( i := q ; i > 0 ; i-- \)
13:                        \( \text{objBound}[q].p_{\text{max}} = \text{Q}[i]; \)
14:               else /// \( \text{Dict}_L \) contains the object \( anObj[q] \)
15:                Remove \( anObj[q] \) from \( \text{Dict}_L \)
16:                \( \text{objBound}[q].p_{\text{min}} += Q[q] \)
17:                // Step 2.2 Record modified object’s bound in \( \text{Dict}_L \)
18:       while \( \text{newObj} > anObj[q] \)
19:            \( \text{newObj}.p_{\text{max}} = Q[q] \)
20:                Add updated \( \text{newObj} \) into \( \text{Dict}_L \)
21:                \( \text{newObj} := \) the next larger object compared to the other \( \text{objs} \) in \( \text{Dict}_L \)
22:  /// while loop until \( \text{newObj} < anObj[q] \) or \( \text{newObj} \) is the smallest \( \text{obj} \) in \( \text{Dict}_L \)
23:       /// Step 2.3 Insert new object \( anObj[q] \) in \( \text{Dict}_L \)
20: \[ \text{Dict}_L\text{.Add}(\text{anObj}[q], \text{objBound}[q]) \]

21: \///<Step 2.4 Check Confidence>

22: \[ \text{Conf} := \text{CheckConf}(\text{anObj}, \&\text{estimConf}) \]

23: \[ \textbf{while} \ \text{Conf} = \text{false} \ \textbf{and} \ \text{viewedObjNum} < \text{viewLine} \]

\///<Step 2.5 record the LLastObj>

24: \[ \textbf{if} \ \text{newObj} > \text{anObj}[q] \]

25: \[ \text{LLastObj}\text{.obj} := \text{anObj}[q] \]

26: \[ \textbf{else} \ \text{//newObj} < \text{anObj}[q] \]

27: \[ \text{LLastObj}\text{.obj} := \text{newObj}\text{.obj} \]

28: \[ \textbf{if} \ \text{LLastObj}\text{.obj} \text{ is not the smallest seen obj in Dict}_L \]

29: \[ \text{Set LLastObj to be the smallest element in Dict}_L \]

30: \[ \textbf{end if} \]

As illustrated in Appendix D(b), when the system starts traversing data of a query mapping extracted from the set of unseen generated queries, the mapping list will be viewed starting from the first (largest) element. The do-while iteration will halt when the \text{estimConf} \geq \text{minconf} or \text{viewedObjNum} < \text{viewLine}. \text{viewedObjNum} < \text{viewLine} means that the program continues traversing tuples in the same mapping until the tuple that is located at the same line as previous minimum objects in each seen mapping list. When the first three mappings have been traversed through Appendix D(a) and Appendix D(b), but the estimated confidence is smaller than \text{minconf}, this improved algorithm is going to process the first three mappings and meanwhile check \text{estimConf} after each line-traversing. If the \text{estimConf} \geq \text{prevEstimConf} as well as \text{Conf} = \text{false}, the 4\text{th} ranked
mapping will be viewed by following Appendix D(b), whose mapping index $q$ equals 4, and then repeat the processing mechanism described above. The searching process will be hindered as soon as $Conf = true$. That is, the current top-$k$ elements are predicted to be real top-$k$ query answering based on $minconf$ specified by users.

We take advantage of $prob. (estimconf)$ compared with $prob.(minconf)$ instead of computing $PMax$ to present $p_max$ of all unseen tuples. There are four different assumptions to extract top-$k$ objects and compute $estimConf$, using PC, OC, AC, or AVC to verify current top-$k$ answering. The four kinds of confidence provide answering with distinct precision.

**PC estimation:** use the upper bound $prob.$ value $p_max$ of viewed tuples to build top-$k$ list and estimate its confidence. Each tuple has an expected value that equals $p_max$. Only considering $p_max$ is a lopsided view. In the traditional logic, $th = top-k^{th} p_min$. However, the formula of $prob.(estimconf)$ is computed by expected values, each expected value of seen tuples keeps greater than $th$ until $top-k^{th} p_min = p_max$. Suppose that $th = top-k^{th} p_max$, if $k = 1$ (top-1), $prob.(estimconf) = 1$ only after traversing the first 2 items at the beginning, which is not accurate. Hence, neither assumptions on $th$ cannot work successfully in PC estimation.

**OC estimation:** use the lower bound $prob.$ value $p_min$ of viewed tuples to build top-$k$
list and estimate its confidence. Each tuple has an expected value that equals $p_{\text{min}}$. OC has the same problem as PC, inducing inaccurate top-$k$ results.

**AC estimation:** use the average of $p_{\text{min}}$ and $p_{\text{max}}$ of viewed tuples to build top-$k$ list and estimate its confidence. Each tuple has an expected value that equals the average, 
$\text{average} = (p_{\text{min}} + p_{\text{max}}) / 2$. Although both probabilities of lower and upper bounds are considered in AC estimation, when $k = 1$, the same problem occurs, as explained in PC estimation.

**AVC estimation:** use all the possible prob. values in $[p_{\text{min}}, p_{\text{max}}]$ for viewed tuples to build top-$k$ list and estimate its confidence. Each tuple has a group of expected values from $p_{\text{min}}$ up to $p_{\text{max}}$. In our experiment, inviting all possible probabilities of each viewed tuple, the algorithm can extract top-$k$ answering with high quality and accuracy. Furthermore, when $k = 1$, the problem mentioned in PC, OC, and AC never happens in this estimate of confidence. The sub-method of *CheckConf* using AVC estimation is shown in Appendix D(c).

---

**Appendix D(c) CheckConf (anObj, &estimConf)***

```
1: ///<Step 1 Find the candidate top-k and record th = ?>
for i := 1 to topK do
2: th := 0 //th = top-$k^{\text{th}}$ $p_{\text{min}}$
```
3:     \textbf{foreach} \ Obj in \ Dict\_L \\
4:         \textbf{if} \ Obj's \ p\_min > \ th \ \textbf{and} \ TopKObjs \ not \ contain \ Obj \\
5:             th := Obj's \ p\_min \\
6:             topKKey[i] = Obj \\
7:     \textbf{end foreach} \\
8:     \textbf{if} \ topKKey[i] = 0 \ //i^{th} \ \text{tuple cannot be found} \\
9:         i := topK \ //\text{stop for loop} \\
10:     \textbf{else} \\
11:         \text{Add} \ topKKey[i] \ \text{into} \ TopKObjs \\
12:     \textbf{end for} \\
13:     //\text{Step 2 Count all possible probabilities any tuples may have} \\
14:     //\text{Record each probable value into arrayList allProbValues} \\
15:     //\text{Step 3 Calculate estimated confidence} \\
16:     \textbf{foreach} \ Obj in \ Dict\_L \\
17:         \textbf{foreach} \ prob \ in \ allProbValues \\
18:             \textbf{if} \ Obj.p\_min \leq \ prob \leq \ Obj's \ p\_max \\
19:                 numOfExpValues += 1 \\
20:                 \textbf{if} \ prob \leq \ th \\
21:                     \text{numOf SmallerExpValues} += 1 \\
22:     \textbf{end foreach} \\
23:     \textbf{end foreach} \\
24:     estimConf := \text{numOf SmallerExpValues} / \text{numOfExpValues} \\
25:     \textbf{if} \ estimConf \geq \ minconf \\
26:         \textbf{return} \ true \\
27:     \textbf{return} \ false
There are two parameters in `CheckConf` method, a list of current seen tuples `anObj` and the address of `estimConf`. If the current estimConf is greater or equal to users’ requirement, return true; otherwise, return false back to the calling method. In this function call, `th` is set to zero at the beginning, and the real `th` will be found after the program finds top-k tuples; `TopKObjs` is the top-k buffer to record the current top-k tuples, and it should be cleared at the beginning of this function; `topKKey` is an array with size `k`, used to find top-k answering and put candidates into `TopKObjs`. `allProbValues` is an ArrayList to record all probabilities any element in the source may have; `numOfExpValues` equals zero initially, counting the number of total expected values in the seen-tuple dictionary `Dict_L`; `numOfSmallerExpValues` adds the number of expected values that are smaller or equal to `th` among all viewed tuples. As for `allProbValues`, for example, if there are three generated queries `Q = {0.5, 0.4, 0.1}`, then `allProbValues = {0.1, 0.4, 0.5, 0.6, 0.9, 1}`.
APPENDIX E

CODE OF HISTOGRAM-BASED APPROXIMATE APPROACH IN BY-TABLE SEMANTICS

Appendix E is the method to extract top-k answers, calculate confidence, and check accuracy with Luna’s top-k results.

Appendix E TopKByTable

///<Step 1: Initialize local variables>

//record for all viewed tuples

Dictionary<int, RealTupleValues> mappValueDict = new Dictionary<int, RealTupleValues>();

double th; //top-kth p_min (threshold value)

int q = 0; //index to record the viewed mapping number

double conf = 0; //Record the real confidence of current top-k answers

ArrayList valueLst = new ArrayList(); //record all distinct viewed data

int numOfAllUnseenGreaterThanTh = 0;

///<Step 2: Calculate approximate values and mapping times

///<Step 3: Arrange the mapping lists in descending order, sorted by probabilities

///<Step 4: find top-k answers from sorted mapping lists

//while loop: traverse from the 1st, 2nd, ... mapping by descending order

//until conf meets user's input confidence

while (q < NUMBER_OF_MAPPINGS && conf < confidence)

    ///<Step 4.1 : set variables to zero, and start finding a new top-k answering>
ArrayList currTopKLst = new ArrayList();

///<Step 4.2: Open and read a new (next) mapping list file Q[1], ... Q[n]
///<Step 4.3: Read data from the file Qi until top-k answers has been searched,
///<or until the number of viewed lines equals previous seen mapping lists;
///<if in the first situation, stop traversing and check confidence
///<else if in the first case, all the next line of data in viewed mapping lists
///<should be viewed simultaneously>
int i = q;

//find top-k results or read to the end of the file
//numOfTopKAnswers: the number of tuples mapped in all viewed mappings
while (numOfTopKAnswers < topK and viewedTupleNumber[q] < NUMBER_OF_SAMPLES and conf < confidence)
    ///<Step 4.4: set variables to zero, and start finding a new top-k answering>
    topKList = new ArrayList();

    ///<Step 4.5: Set threshold value th to the largest probability of
current viewed tuples
    ///<Step 4.6: Start viewing lines of tuples
for (int index = 0; index < numLinesToView; index++)
    //read a tuple (from the current or next mapping)
    arraydata = inFile[i].ReadLine();
    if (arraydata != null)
        add a new tuple into dictionary and set tuple’s prob to Q[i]
        //add the tuple with its values into the dictionary
        mappValueDict.Add(wordData, aTuple);
    else //the dictionary contains the tuple
update aTUPLE.prob += Q[i]

mappValueDict.Remove(aTUPLE) //remove the previous record
mappValueDict.Add(wordData, aTUPLE);

//check if continue read data from the next mapping list
if ((i < q) && (viewedTupleNumber[i + 1] < viewedTupleNumber[0])
and (viewedTupleNumber[i] == viewedTupleNumber[0]))
i += 1;
else if ((i == q) && (q != 0)
and (viewedTupleNumber[i] == viewedTupleNumber[0]))

//check if all mapping lists have been viewed in the same level
but top-k answering is not complete
i = 0;

//Check if the tuple is also an approximate value existing in the dictionary

///HistoValueDict. If yes, the value 'total' minus 1;
///if not, plus or minus the real value
///by 1, 2,...the largest spread
///return true if found, and numOfAllUnseen -= 1;
///return false if not found.

//End viewing lines of tuples

///<Step 4.7: Find top-k tuples and add into currTopKLst
///<Step 4.8: if all data of the viewed list(s) have been viewed
///<but top-k answers are not complete>
bool foundtopk = false

///<Step 4.9: Check confidence>
///<If all top-k data can be completely found,
///<check confidence of the current answering>
if (foundtopk == true)

//check confidence of the current top-k answers
numOfAllUnseenGreaterThanTh = 0;
    foreach (ApproxValue obj in valuelist)
        //valuelist: a list of all approx. values
        int item = obj.aValue;
        if (HistoValueDict.ContainsKey(item))
            if (HistoValueDict[item].prob > th)
                numOfAllUnseenGreaterThanTh += HistoValueDict[item].total;
        //Calculate confidence
        conf = (1 - ((double)numOfAllUnseenGreaterThanTh / numOfAllUnseen)) * 100;
else //Haven't found top-k candidates
    conf = 0;

//end while

///<Step 4.10 : move to the next mapping list file>
///<Open a new file if exists>
q++;

///<Step 5: Close all opened read-files>
///<Step 6: output top-k answers and the number of viewed tuples>
///<Step 7: Compute accuracy of the final top-k answers>
//end