THE NATIONAL UNIVERSITY
of SINGAPORE



School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRA1/10

On Tag-based Querying of Wide Tables

Bing Tian Dai, Meiyu Lu, Beng Chin Ooi and Anthony K. H. Tung

January 2010

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

On Tag-based Querying of Wide Tables

Bing Tian Dai, Meiyu Lu, Beng Chin Ooi, Anthony K. H. Tung
{daibingt,lumeiyu,ooibc,atung}@comp.nus.edu.sg

School of Computing
National University of Singapore



NUS SOC Technical Report
Number TRA1/10
January 15, 2010

School of Computing
National University of Singapore
13 Computing Drive
Singapore 117417
Telephone: +65 6516 2727
Fax: +65 6779 4580
Homepage: http://www.comp.nus.edu.sg

On Tag-based Querying of Wide Tables

Bing Tian Dai, Meiyu Lu, Beng Chin Ooi, Anthony K. H. Tung
{daibingt,lumeiyu,ooibc,atung}@comp.nus.edu.sg

School of Computing
National University of Singapore

National University of Singapore, School of Computing, TRA1/10

January 15, 2010

Abstract

The huge amount of heterogeneous data that are generated from Web 2.0 applications have resulted in the use of wide tables where tuples with different schemas may co-exist. Providing users with such freedom however poses new challenges such as difficulty in query formulation and query processing, and lack of guarantee in data quality.

In this paper, we propose an out-of-the-box approach that deals with challenges from a semantic perspective. We interpret the semantics of each data value by a set of similar tags. Such association between values and tags provides a layer that separates database operations from the database storage layer. This layer also serves as a platform for users to query the databases by a group of tags. We conducted a series of experiments and the results confirm its effectiveness.

1 Introduction

Schema is the core structure that defines the properties of a database, and therefore database design, implementation and query processing are all heavily dependent on the schema of the database. However, with the huge amount of data generated from Web 2.0 applications, it has become a necessity for database technologies to support heterogeneous data, which usually comes with no schema or a weak underlying schema. It is difficult and unrealistic to transform all forms of heterogeneous data into standard tuples in order to accommodate the pre-defined schema.

In order to accommodate data with no schema or weak schema, an emerging trend deployed by the database community is to relax the schema, so that data can be stored and processed in a self-organizing and self-explanatory manner, allowing data to be stored without specifying a schema. A successful case in point is Bigtable [7, 8], the data model of Google Base, which allows users to select names for the values they want to insert into the database. The Bigtable can be viewed as a wide table with thousands of columns, and each row can be a relation of any type, which relates only a few to a few dozens of columns. Thus Bigtable is extremely sparse, with many empty cells. Each value is stored in association to a column whose name is specified by users. For example, a user may insert a record in the format of {`make`: `Volvo`, `model`: `XC90`, `contact`: `123-4567`}, as shown in the last row of Figure 1. `make`, `model` and `contact` are user-defined column names for the values `Volvo`, `XC90` and `123-4567` respectively. Except for the three columns, all other columns are undefined. Other recent work that are similar in spirit include the structure-free query model [31, 40], the WebTable [5], and RDF tables[30].

With the relaxation on the constraints on data insertion, additional cost is incurred during query processing. This is because users tend to use different naming conventions for their own records and different attributes for similar data items. For instance, user A may name the make of a car as `make`, while user B may name the same thing as `manufacturer`. Now there are two columns with different names in the database which refer to the same context. Query processing is not straight forward. For example, to search a tuple by a specific value, query has to be broadcasted over all the columns with the same semantics. Another drawback is that users may define a column like `property` which is too generic for database system to know what it means and how the data in that column can be utilized for queries.

make	model	price	mileage	year	name	last_name	first _name	contact	email	phone	handphone
BMW	3	31900	190500			Smith	John		js@a.com	222-4567	222-5678
Ford	F150	15900	198405	2002	John			j@b.com			234-8765
Toyota	Camry		191278	2004		Yamada	Taro			345-6789	
Honda	Civic	16000		2004		Brown	John	9, 1st St	jb@c.com	567-1234	
Ford	Focus		189038		Tom	Davidson			td@d.com		678-2345
Volvo	XC90							123-4567			

Figure 1: An example of wide table on car database, including agents' details

TID	Data
026	BMW:{make,brand} | 3:{model} | Smith:{last_name} | John:{first_name} | 222-4567:{phone} | 222-5678:{handphone}
289	Ford:{make} | F150:{model,name} | John:{name,first_name} | j@b.com:{contact,email} | 234-8765:{handphone,contact}
354	Toyota:{make,manufacturer} | Camry:{name} | Yamada:{last_name} | Taro:{first_name} | 345-6789:{phone,contact}
571	Honda:{brand} | Civic:{model,name} | Brown:{last_name} | John:{first_name,name} | jb@c.com:{email,contact}
702	Ford:{manufacurer} | Focus:{model} | Tom:{name,first_name} | Davidson:{last_name,name} | td@d.com:{email,contact}
886	Volvo:{make} | XC90:{model} | 123-4567:{contact,phone,handphone}

Figure 2: Values tagged with multiple tags

Therefore, the free-and-easy way of inserting data into such databases without knowing the correct columns to insert into is undesirable. Ultimately, we would like to enjoy the benefits from databases with little schema constraint, but without paying for the cost incurred from careless operation. Therefore, a semi-supervised way of data management is preferred for databases with little schema constraint to deal with the following two challenges. First of all, we wish to provide a function which can help users insert data into databases so that the quality of databases does not degrade after insertions from different users. Second, the database system must at the same time provide efficient support for structure-free query model that allows users to pose queries without fully knowing the underlying schema. More importantly, users might specify their intention differently. For example, when a user specified `contact`, he/she could be referring to `phone`, `address` or `email`.

The problem posed by two these challenges is the lack of ability to precisely describe the semantics of data values. If we knew the semantics of data values, we would be able to solve the first challenge by reorganizing the data from users into semantically homogeneous columns, and we could also interpret the semantics of the queries users submitted. Therefore, our goal is to be able to describe the semantics of both data values and queries.

As discussed above, conventional database uses column names to describe the semantics of the values within this column, but it is pre-defined and restrict other means of describing the semantics. Inspired by the method of tagging multimedia object [1, 2] to facilitate searching, we use a set of words, which are called *tags*, to describe the semantics of each data value. Figure 2 shows the tags that are associated with selective values in Figure 1. Each value in a tuple can be associated with multiple tags which are semantically close. Likewise, a tag might be associated with multiple values in a tuple if these values are deemed similar.

The challenge here is to describe the semantics of values with an appropriate set of tags. For example, value `toyota` can be associated with both tags `make` and `manufacturer`, which is better than having a column with name `make` to describe `toyota`.

One may ask if the tags being used are equal likely to describe the semantics of data values, or if there is an order among the tags. Consider car agents' name in Figure 1, the first name `John` can be tagged by both `first_name` and `name`, but the tag `first_name` is closer than the tag `name` when describing `John`, thus `first_name` should be associated closer than `name` in the tag list of `John`. To deal with such kind of associability, a measure is to be proposed to measure how close a tag is from a given data value. This measure should be higher when a tag is closer to the given value, i.e. when a tag is used more frequently for tagging this value. Thus, the direct interpretation of such measure is on the likelihood of a tag being used to tag a given value.

Definition 1 (Probabilistic Tagging)**.** *Let $\mathcal{T}$ be the set of tags, for a value v and a tag $t \in \mathcal{T}$, probabilistic tagging refers to the act of associating a probability $\zeta_{v,t}$ for v and t, such that the value v is tagged by the tag t with probability $\zeta_{v,t}$. The probability $\zeta_{v,t}$ is called associated probability.*

For every tag list in Figure 2, there is a probability associated to each tag, representing the likelihood of this tag being used to tag the value. This is presented in Figure 3. Each value is identified with tuple-ID (TID) and location-ID (LID) to indicate which tuple it comes from and its location in the tuple. The type indicates the "type" of the value, if it is string (S) or numerical (N). The last column is a set of tag-probability pairs. For example, `John` can be tagged by `first_name` with probability 0.7, but tagged by `name` with only 0.5. We do not have the constraint that the associated probability sums up to be 1, as there is no mutual exclusion among the tags, a value is absolutely possible to be tagged by more than one tag.

Value	TID	LID	Type	Probabilistic Tagging
BMW	026	1	S	⟨make, 0.4⟩, ⟨brand, 0.3⟩
3	026	2	S	⟨model, 0.6⟩, ⟨name, 0.4⟩
31900	026	3	N	⟨price, 0.8⟩
Smith	026	5	S	⟨last_name, 0.8⟩
John	026	6	S	⟨first_name, 0.7⟩, ⟨name, 0.5⟩
Ford	289	1	S	⟨make, 0.8⟩, ⟨manufacturer, 0.4⟩
F150	289	2	S	⟨model, 0.4⟩, ⟨name, 0.2⟩
15900	289	3	N	⟨price, 0.3⟩
John	289	6	S	⟨name, 0.8⟩, ⟨first_name, 0.5⟩
Toyota	354	1	S	⟨make, 0.9⟩,⟨manufacturer, 0.7⟩
Camry	354	2	S	⟨model, 0.9⟩
Yamada	354	5	S	⟨last_name, 1.0⟩
Taro	354	6	S	⟨first_name, 0.9⟩
Honda	571	1	S	⟨manufacturer, 0.4⟩, ⟨brand, 0.2⟩
Civic	571	2	S	⟨model, 0.7⟩, ⟨name, 0.4⟩
16000	571	3	N	⟨price, 0.7⟩
Brown	571	5	S	⟨last_name, 0.4⟩, ⟨color, 0.5⟩
John	571	6	S	⟨first_name, 0.8⟩, ⟨name, 0.4⟩
Ford	702	1	S	⟨make, 0.7⟩, ⟨brand, 0.2⟩
Focus	702	2	S	⟨model, 0.5⟩, ⟨name, 0.2⟩
Tom	702	4	S	⟨first_name, 0.8⟩
Davidson	702	5	S	⟨last_name, 0.9⟩
Volvo	886	1	S	⟨make, 0.6⟩
XC90	886	2	S	⟨name, 0.5⟩

Figure 3: An illustration of probabilistic tagging

There are three tuples containing value `John` in Figure 3, it is because tuples come from different data sources with possibly different tags, and probabilities associated with these tags may also be different.

1.1 Our Contributions

Our main contribution is the proposal of probabilistic tagging, which allows data values tagged by multiple tags with different probabilities and to be queried based on the tags. In order to fully support such flexibility in querying based on tags, we contribute the following:

- A signature to sketch the values associated to a tag.
- A distance measure between tags, based on the clustering of their signatures. This distance measure enables the inference of associated probabilities between values and tags.
- A Tag-based query language that supports user queries database by a set of tags.
- A rapid online alignment between tags and data values during query processing.
- A customizable scoring function for arranging the result according to users' preferences.

1.2 Paper Outline

The following sections are organized as follows. Section 2 will give a big picture to our idea on probabilistic tagging, which includes the introduction to our data management and our query language. Along with this introduction, issues and challenges about probabilistic tagging will be discussed, which will be solved in Section 3 and Section 4. This is followed by Section 5, in which we validate probabilistic tagging and query processing. Section 6 will give a brief summaryof related works on database/query relaxation, information integration and probabilistic database. Lastly, we will conclude our results in Section 7.

2 Issues and Challenges

We will next discuss the issues and challenges involved in supporting a probabilistic tag-based approach to querying wide tables. This is separated into three parts. First, we will discuss the important issue on how we can associate tags and values together with the corresponding probabilities. Second, we will discuss our approach involved in query specification and processing of a probabilistically tagged table. Finally, we will highlight additional issues that need to be addressed including table updates and tag refinement.

2.1 Tag/Value Association

Since our processing is tag-centric, the first issues that need to be addressed is how tags can be associated correctly with the values in each tuple. Based on our analysis, there are at least three ways to do this:

1. Attribute Names: Since tuples in a wide table are typically inserted by users with a schema in mind, each of these values in the tuple will thus be associated with an attribute name. Thus, a trivial way to associate at least one tag with each of the values is to assign the attribute name as a tag.

2. User Specified: Alternatively, a user might want to simply specified a tag for a group of values. For example, he/she might issue a command tag(`BMW`, `manufacturer`,0.9) which will tag all values `BMW` with the tag `manufacturer`. Obviously, there exist many alternative approaches for users to specify association between the tags and values. Furthermore, checks should be in place to ensure that users do not add in spurious tags that make no meaning. Due to space constraint, we will not elaborate on this choice but leave it as a discussion topic in future work.

3. Tag Inference: Given a reasonably accurate tagged table, a third alternative is to apply data mining techniques to infer additional tag-value pairs. While there exist many mining algorithms each with different strength and weakness, in this paper, we propose the use of clustering algorithm for the purpose of tag inference.

However, it is not trivial to apply clustering method for tag inference. First, data values are heterogeneous, the space where clustering of data values is done already pose a problem. More importantly, even if there is such a space, tag inference is not simply decided by how close the tags are, it is also determined by the "scope" of the tags. The "scope" of a very specific tag is very small, the distance may look larger under such scenario. Therefore, tag inference must also cater to the modeling of each individual tag in the same space as the clustering algorithm.

2.2 Query Specification and Processing

We next describe our query model. We assume that we are querying against a single *TagTable*. A TagTable is constructed from a set of values $\mathcal{V}$, a set of tags $\mathcal{T}$ and the associated probabilities ζ. It has the following characteristics.

- The database consists of many tables with similar structure
- Each table is associated with one particular tag t, and stores all values v with probability $\zeta_{v,t}$ greater than a preset threshold. Each tuple of the tables is a quardruple, which will be discussed in detail in Section4.
- A mega table stores all values from the same tuple sequencially in a row. This helps retrive the entire row at one go, rather than retrieving them from many tables.

With TagTable, we now propose a query language that expresses a query based on tags, which we call *TagQL*. The general form of TagQL can be summarized as

```
select      ?$QTag_1, ?$QTag_2, ..., ?$QTag_m
from        -
where       $VTag_1 op_1 v_1
     and    $VTag_2 op_2 v_2
     and    ...
     and    $VTag_n op_n v_n
confidence  δ
```

TagQL is similar to a SQL select-project query. Since we are considering a wide table, there is no need to specify a table and no join operation is necessary. In TagQL, $\texttt{QTag}_1$, $\texttt{QTag}_2$, ..., $\texttt{QTag}_m$ are the tags which are associated with values that the users want to retrieve. To select tuples that satisfy certain conditions, a set of tags $\texttt{VTag}_1$, $\texttt{VTag}_2$, ..., $\texttt{VTag}_n$ are used to select values from the tuples and compare them against a set of values $v_1, v_2, \ldots, v_n$ using the logical operators represented by op_1, op_2, ..., op_n. These operators can be any of '=', '¡¿', '¡', '¿', '$\leq$' or '$\geq$' if the corresponding value is numerical, or be '=' or '¡¿' only if the corresponding value is a string. In addition, a probability threshold δ must be specified in order to make sure the confidence of the tag is at least δ.

Both `VTag` and `QTag` are disjunctions of a series of tags, e.g. {`make|manufacturer`}. We will refer to them as **composite tags**. This allow users to increase their chance of using the correct tag by selecting multiple tags to retrieve a value. Semantically, the users are also providing information and telling us that the set of tags that are enclosed within the same composite tag are in fact semantically close from their point of view. In or query processing, we capture this semantic by adopting the following rule:

Definition 2 (maximum probability rule(MPR))**.** *Given a composite tag, `VTag`/`QTag` and a value v within a tuple, the probability of the `VTag`/`QTag` being associate with v is $max\{\zeta_{v,t} | t \in \texttt{VTag}/\texttt{QTag}\}$.*

The maximum probability rule intuitively makes use of the fact that all tags in a composite tag are considered semantically equivalent and thus we can assign the highest probability when we are drawing an association between a value and a composite tag. As an example, let's consider the query and the probabilistic tagging in Figure 3.

```
select      ?${model|car_name},
            ?${contact|email}
from        -
where       ${name|first_name} = 'John'
     and    ${make|manufacturer} = 'Ford'
confidence  δ
```

The value "John" is tagged with both "name" and "first_name" but with different probability of 0.8 and 0.5 respectively. In this case, the user have indicate that "name" and "first_name" are considered to be semantically close and thus the probability of "John" being associate with {`name|first_name`} is set at 0.8.

Aside from the maximum probability rule, we also have a mutual exclusion rule.

Definition 3 (mutual exclusion rule(MER))**.** *Two composite tags that does not contain exactly the same set of tags cannot be instantiate with the same values from a tuple.*

Consider again our example query. When the user project on the two composite tags {`name|first_name`} and {`contact|email`}, he/she must have considered these two composite tags to be semantically different. Thus it does not make sense to assign the same value from a tuple to these two composite tags.

The reason behind defining the above two rules is simple. Since our tuples are all probabilistic in nature, a query in TagQL can generate many answer instances even from a single tuple. Applying these rules help us to capture certain semantics that can cut down the number of possible instances. Even with such rules, the number of possible instances can still be large. Thus we can only say that certain instances from the tuples have low likelihoods in answering the query and is not as interesting as tuples with high likelihoods. Thus, we focus our effort in answer top-k queries.

Definition 4 (Top-k query answering)**.** *Given (i) a TagTable, (ii) a query q written in TagQL, and (iii) a positive integer k, return a tuple set RS, such that $|RS| \leq k$, $\forall t \in RS$ and $t' \in \overline{RS}$, which is the complementary set of RS, $\iota(t) \geq \iota(t')$, where ι is the likelihood function given query q.*

In Section 4, we will look at processing TagQL in more details.

2.3 Updates and Refinement

The last issue that we will discuss in this paper concerns updates to the table and tag refinement. Again, due to space constraint, we will simply highlight these issues here but will not elaborate further.

Updates

Since tuples can be deleted and inserted into the TagTable, a natural challenge is to incrementally update the tag-value probability as tuples are inserted and deleted. Note that since users might have previously retrieved tuples using certain TagQL query statement, certain mechanism must be introduced to ensure the stability of the answer sets. One solution is to maintain the historical values of the probabilistic tag so that user can choose to adopt the probabilistic tagging within a period of choice.

In addition, when new tuples are inserted, a nice feature to have will be to have the ability to automatically assign tags with corresponding probabilities to the values in the new tuples. Doing so is entirely possible within our existing solution framework since we can essentially compare the values in the new tuple against our clusters and associated the corresponding tag of the clusters to these values.

Refinement

Since our retrieval is probabilistically in nature, it is possible that some of the returned tuples could be wrong. In such cases, mechanisms must be provided to (1) help the users to identify and fix the error and (2) refine the query result based on user's feedback and (3) improve the tagging accuracy.

There are many alternatives for implementing these mechanisms discussed above. For (1), since we are finding the most probable instances for each tuple given the query, it will be also useful to give users the flexibility of browsing the next best instance from a particular tuple so that the user can judge whether there are better instances that suite their purpose more. It will also be useful to list high probability tags that are associated with each value of an output instance so that users can judge the validity of the instances based on these tags. For (2), if the users specified that certain instances of the tuple are definitely wrong or correct, then certain statistical or machine learning techniques can be applied to extract rules which can subsequently be applied to fix results of other tuples. For example, if it is found that "selling price" must be also greater than "buying price", then such a rule can be enforced on all instances of the query result. Finally, for (3), if certain tags are found to be consistently the reason for correct or wrong results. Then it make sense to adjust their associated probability with the tuple values accordingly.

3 Tag Inference Processing

In this section, we will developed a clustering algorithm that allow us to perform tag inference. From the start, we simply have a set of value $\mathcal{V}$ and a set of tags $\mathcal{T}$ together with a set of associating probabilities $\zeta_{v,t}$ associating $v \in \mathcal{V}$ and $t \in \mathcal{T}$. Since $\zeta_{v,t}$ is only know some pair $v \in \mathcal{V}$ and $t \in \mathcal{T}$, we wish to infer more values of $\zeta_{v,t}$ which are originally unknown.

Intuitively, our algorithm need to group similar tags together in order to make tag inference i.e. if a value v is tagged with a tag t_1, then it also should be tagged with a tag t_2 with high probability if the distance between t_1 and t_2, $D(t_1, t_2)$ is close. Two tags are close if their semantics are close. Such closeness is reflected by the set of values they are associated with. For example, a tag associated with numerical values, like `price` or `year` is never close to a tag associated with string values like `make` or `model`. Based on these analysis, our procedure for tag inference can be separated into four steps:

1. Extract "signatures" from values $v \in \mathcal{V}$.
2. Cluster values in $\mathcal{V}$ based on extracted "signatures" from Step (1).
3. Compute the distance between tags in $\mathcal{V}$, by utilizing the clustering result from Step (2). Two tags are deemed to be close if the distribution of their associated values among the clusters are highly similar.
4. For $\zeta_{v,t}$ which are not available previously, estimate $\zeta_{v,t}$ by utilizing a set of tags t_1, ... ,t_n which are associated with v and which are close to t.

The first step is essentially a feature extraction process so that clustering can be perform on the values in $\mathcal{V}$. The "signature" generated for each value $v \in (V)$ is dependent on the tags that it is associated with. Assuming v is associated with tags t_1,...,t_n, then it's "signature" will be determined by the union of the features (or dimensions) that are generated for each of the tag t_1,...,t_n. For each tag t_i, it's descriptive features is determine by the set of values from (V) that are associated with it. As an example, consider our running example where `John` with tuple ID 289 is associated with both `name` and `last_name`, then `John`'s "signature" will be determine by the set of features generated from the tags `name` and `last_name`. In this case, the features of `name` will be determine by all values like `F150`, `Camry` that are associate with it in Figure 3.

Depending on whether t is a numeric or string tag, different approach will be used to generate it's feature. To capture the characteristics of strings, q-grams [11, 12], which are substrings of length q or less, is a sub-structure of strings that are often used to represent the strings in a much lower dimension. Here, we try to find significant q-grams (*sigram*) which are long and frequent. More specifically, we say a q-gram A is a *sigram* if i) $Freq_A > \epsilon$ and ii) $\nexists B$ s.t. $Freq_B > \eta \cdot Freq_A$ where B is an immediate superstring of A, given a pair of parameter ϵ and η. For a pair of parameters ϵ and η, and a tag t, we can derive an algorithm to select all sigrams with respect to a tag t which is shown in Figure 1. The algorithm essentially enumerate through all the possible q-grams and detect sigrams by comparing the frequency between immediate substring/superstring. The weight of a sigrams is then computed as the sum of $\zeta_{v,t}$ where v represent all values that are associated with the tag t and contain the sigram as a substring. Note that, as q increases, the number of q-grams that pass ϵ test shrinks, so the number of sigrams is not large.

The second type of tag, i.e., numerical tag are associate with numerical values that cannot be treated as strings to make sigrams as their "signature". As numerical values are often continuous, which makes it difficult to map into a set of dimensions, thus, we take histograms as the "signature" of a numeric tag. Given a parameter m, we can represent a range of values by $2m$ dimensions.

Algorithm 2 builds a set of dimensions D_t from a set of values associated with numerical tag t. Each dimension is of width g on the numerical domain, spread from the median for at most m stripes towards both sides. Tag t is now represented by a probability histogram over D_t. Given two tags t_1 and t_2 whose values are distributed similarly, the histograms obtained from the numerical dimensions D_{t_1} and D_{t_2} will be similar too.

Together with the set of sigrams that are obtained from string tags, we now have a set of features/dimensions $\mathcal{D}$. For a value $v \in \mathcal{V}$, let $\psi(v)$ be the probability distribution representing v over $\mathcal{D}$, obtained as follows. For a string value v and a sigram d, $\psi(v, d) = k$ if the sigram d appears in value v for k times; for numerical value v and a numerical dimension d, $\psi(v, d) = 1$ if $d.\min \leq v < d.\max$. $\psi(v)$ is then normalized from $\psi(v, d) \forall d \in \mathcal{D}$, so that $\sum_{d \in \mathcal{D}} \psi(v, d) = 1$.

Once the probability distribution had been determined, we can move on to Step (2) where the values in V are then clustered using clustering algorithm that was presented in [37, 11]. Given an integer k, the fuzzy clustering algorithm

Algorithm 1 Construction the dimension, i.e. sigrams, from string values associated with a given tag t

Input: A set of string values V associated with tag t
A pair of parameters ϵ and η
Output: A set of sigram SQ_t as a set of dimensions

$\zeta_t \leftarrow \sum_{v \in V} \zeta_{v,t}$
$k \leftarrow 1$
while $k = 1$ or $A_{k-1} \neq \emptyset$ **do**
  $A_k \leftarrow \emptyset$
  for all string value $v \in V$ **do**
    for all k-gram Q appears in v **do**
      $A_k \leftarrow A_k \cup \{Q\}$
      $Q.w_i$ += $\zeta_{v,t}$;
  for all $Q \in A_k$ **do**
    for all Q', an immediate substring of Q, where $k > 2$ **do**
      $Q'.w_s = \max\{Q'.w_s, Q.w_i\}$
    Remove Q from A_k **if** $k > 1$ and $Q.w_i \leq \epsilon \cdot \zeta_t$
  for all $Q \in A_{k-1}$ where $k > 2$ **do**
    Remove Q from A_{k-1} **if** $Q.w_s > \eta \cdot Q.w_i$
  k ++;
Return $SQ_t = \bigcup_k A_k$

compute the probability of membership for each value v within the cluster set $\mathcal{C}$ that contain k clusters. Let $\tau(v)$ represent this distribution of cluster membership, we now define the cluster probability distribution for a tag t as the mean of the cluster probability distributions on all value v, weighted by $\zeta_{v,t}$, i.e.

$$\tau(t) = \sum_{v \in \mathcal{V}} \frac{\zeta_{v,t}}{\sum_{v' \in \mathcal{V}} \zeta_{v',t}} \cdot \tau(v)$$

The $\tau(t)$ defined above is the representation of tag t in the cluster space $\mathcal{C}$. There are two factors that determine the representation of tag t. The first factor is the set of values associated with t, i.e. $\tau(v)$, and the other factor is how these values are distributed. Generally speaking, two tags t_1 and t_2 are close if their associated values are similar and the distribution of the values are similar. $\tau(t)$ captures both the similarity from values, i.e. $\tau(v)$, and the similarity from the value distribution, i.e. ζ_t. Therefore, we can make use of $\tau(t)$ to measure the distances among tags.

To proceed with our formulation on tag distance between a pair of tags, t_1 and t_2(i.e. Step (3)), we define a random variable X_1 that takes values in the set of cluster probability distributions $\tau(\mathcal{V}) = \{\tau(v_i)\}_{i=1}^n$ following the probability measure $\{\pi_1(v_i)\}_{i=1}^n$ where $\pi_1(v_i) = \frac{\zeta_{v_i,t_1}}{\sum_v \zeta_{v,t_1}}$. Another random variable X_2 takes the same set of values $\tau(\mathcal{V}) = \{\tau(v_i)\}_{i=1}^n$, but follows a different probability measure $\{\pi_2(v_i)\}_{i=1}^n$ where $\pi_2(v_i) = \frac{\zeta_{v_i,t_2}}{\sum_v \zeta_{v,t_2}}$.

Now consider the two mutual information $I(\mathcal{X}_1; \mathcal{C}_1)$ and $I(\mathcal{X}_1; \mathcal{C}_2)$ where $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}$ are the set of clusters following probability distribution $\tau(t_1)$ and $\tau(t_2)$ respectively. If tag t_1 and t_2 are close to each other in the space of cluster distribution, the mutual information $I(\mathcal{X}_1; \mathcal{C}_1)$ should be close to $I(\mathcal{X}_1; \mathcal{C}_2)$ as $\mathcal{C}_1$ and $\mathcal{C}_2$ are close. We first prove the following proposition to show the relationship between the two mutual information.

Proposition 1 (Bregman Information Equality)**.** *Let X be a random variable that takes values in $\mathcal{X} = \{x_i\}_{i=1}^n \subseteq \mathbb{R}^d$ following the probability measure $\pi = \{\pi(x_i)\}_{i=1}^n$, given a Bregman divergence ϕ and another point y,*

$$\mathbb{E}_\pi[d_\phi(\mathcal{X}, y)] = \mathbb{E}_\pi[d_\phi(\mathcal{X}, \mu)] + d_\phi(\mu, y)$$

where $\mu = \mathbb{E}_\pi[\mathcal{X}]$.

Algorithm 2 Construction of numerical dimensions from numerical values associated with a given tag t

Input: A set of numerical values V associated with tag t
A granularity parameter m
Output: A set of numerical dimensions D_t

$\zeta_t \leftarrow \sum_{v \in V} \zeta_{v,t}$
Get median v_m where $\sum_{v<v_m} \zeta_{v,t} = \frac{1}{2}\zeta_t$, lower quartile v_l where $\sum_{v<v_l} \zeta_{v,t} = \frac{1}{4}\zeta_t$, upper quartile v_u where $\sum_{v<v_u} \zeta_{v,t} = \frac{3}{4}\zeta_t$, minimum value v_{min} and maximum value v_{max} from V
Let $g \leftarrow \frac{v_u - v_l}{m}$
Let $D_t \leftarrow \emptyset$: the set of dimensions constructed by t
Let $n_1 \leftarrow \min(m, ceil(\frac{v_m - v_{min}}{g}))$
for all $i \in \{1, 2, \ldots, n_1\}$ **do**
    Construct numerical dimension on interval $[v_m - i \cdot g, v_m - (i-1) \cdot g)$
    insert this dimension into D_t
Let $n_2 \leftarrow \min(m, ceil(\frac{v_{max} - v_m}{g}))$
for all $i \in \{1, 2, \ldots, n_2\}$ **do**
    Construct numerical dimension on interval $[v_m + (i-1) \cdot g, v_m + i \cdot g)$
    insert this dimension into D_t
return D_t

Proof.

$$
\begin{aligned}
& \mathbb{E}_\pi[d_\phi(\mathcal{X}, y)] = \sum_{i=1}^{n} \pi(x_i) d_\phi(x_i, y) \\
= & \sum_{i=1}^{n} \pi(x_i)\{\phi(x_i) - \phi(y) - \langle x_i - y, \nabla\phi(y)\rangle\} \\
= & \sum_{i=1}^{n} \pi(x_i)\{\phi(x_i) - \phi(\mu) - \langle x_i - \mu, \nabla\phi(\mu)\rangle \\
& + \langle x_i - \mu, \nabla\phi(\mu)\rangle + \phi(\mu) - \phi(y) - \langle x_i - \mu + \mu - y, \nabla\phi(y)\rangle\} \\
= & \sum_{i=1}^{n} \pi(x_i)\{\phi(x_i) - \phi(\mu) - \langle x_i - \mu, \nabla\phi(\mu)\rangle\} \\
& + \sum_{i=1}^{n} \pi(x_i)\{\phi(\mu) - \phi(y) - \langle \mu - y, \nabla\phi(y)\rangle\} \\
= & \mathbb{E}_\pi[d_\phi(\mathcal{X}, \mu)] + d_\phi(\mu, y)
\end{aligned}
\tag{1}
$$

As $\sum_{i=1}^{n} \pi(x_i)(x_i - \mu) = 0$, both $\sum_{i=1}^{n} \pi(x_i)\langle x_i - \mu, \nabla\phi(\mu)\rangle = \sum_{i=1}^{n} \pi(x_i)\langle x_i - \mu, \nabla\phi(y)\rangle = 0$, that is how the second last equation comes from. □

By [4] and specifying $\mathcal{X} = \mathcal{X}_1$, $x_i = \tau(v_i)$, $\pi(x_i) = \pi_1(v_i)$, $y = \tau(t_2)$ and $\phi(p) = \sum_{j=1}^{d} p_j \log p_j$, proposition 1 shows

$$
\begin{aligned}
& I(\mathcal{X}_1; \mathcal{C}_2) = \sum_{i=1}^{n} \pi_1(v_i) KL(\tau(v_i) \| \tau(t_2)) \\
= & \sum_{i=1}^{n} \pi_1(v_i) KL(\tau(v_i) \| \tau(t_1)) + KL(\tau(t_1) \| \tau(t_2)) \\
= & I(\mathcal{X}_1; \mathcal{C}_1) + KL(\tau(t_1) \| \tau(t_2))
\end{aligned}
\tag{2}
$$

where $\tau(t_1) = \mathbb{E}_{\pi_1}[\tau(\mathcal{V})]$.

Thus $I(\mathcal{X}_1;\mathcal{C}_2) \geq I(\mathcal{X}_1;\mathcal{C}_1)$, if t_2 is close to t_1, $\frac{I(\mathcal{X}_1;\mathcal{C}_2)}{I(\mathcal{X}_1;\mathcal{C}_1)}$ is close to 1. However, this is not a symmetric measure, in order to make it symmetric, we may consider the random variable X_2 defined previously, which becomes

$$\frac{I(\mathcal{X}_1;\mathcal{C}_2)}{I(\mathcal{X}_1;\mathcal{C}_1)} + \frac{I(\mathcal{X}_2;\mathcal{C}_1)}{I(\mathcal{X}_2;\mathcal{C}_2)}$$

By applying Equation 2 again, we have

$$\begin{aligned} & \frac{I(\mathcal{X}_1;\mathcal{C}_2)}{I(\mathcal{X}_1;\mathcal{C}_1)} + \frac{I(\mathcal{X}_2;\mathcal{C}_1)}{I(\mathcal{X}_2;\mathcal{C}_2)} \\ = \quad & 2 + \frac{KL(\tau(t_1)\|\tau(t_2))}{I(\mathcal{X}_1;\mathcal{C}_1)} + \frac{KL(\tau(t_2)\|\tau(t_1))}{I(\mathcal{X}_2;\mathcal{C}_2)} \end{aligned}$$

Therefore, we define distance function between tag t_1 and t_2 as

$$D(t_1,t_2) = \frac{KL(\tau(t_1)\|\tau(t_2))}{I(\mathcal{X}_1;\mathcal{C}_1)} + \frac{KL(\tau(t_2)\|\tau(t_1))}{I(\mathcal{X}_2;\mathcal{C}_2)}$$

First, this distance is smaller when $\tau(t_1)$ is closer to $\tau(t_2)$, as shown in Figure 4, the closures are the contour with equal KL distance to their centers. This distance is the mutual information between $\mathcal{X}$ and $\mathcal{C}$ as defined in Proposition 1, so $I(\mathcal{X}_1;\mathcal{C}_1)$, $I(\mathcal{X}_2;\mathcal{C}_2)$ and $I(\mathcal{X}_3;\mathcal{C}_3)$ are all about the same. Since $\tau(t_3)$ is closer to $\tau(t_2)$ than $\tau(t_1)$, we have $D(t_1,t_2) > D(t_2,t_3)$. Second, when the distances are the same, distance increases as mutual information decreases, as it makes distance relatively larger. In Figure 4, $\tau(t_3)$ and $\tau(t_4)$ are having the same distance to $\tau(t_2)$. However, $D(t_2,t_3) < D(t_2,t_4)$ as $I(\mathcal{X}_3;\mathcal{C}_3) > I(\mathcal{X}_4;\mathcal{C}_4)$.

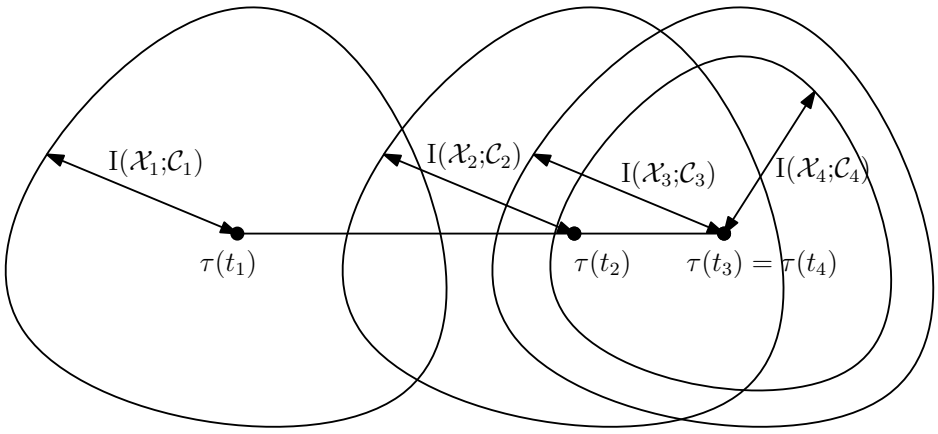


Figure 4: An illustration of the distance function

Note that the computation of this distance function only involves the weighted KL distance around each tag and pairwise KL distance between tags. Therefore, with pre-computation of cluster probability distribution and the weighted KL distance on all tags, the computation time for distance function only is two times as much as that for KL distance.

With tag distance defined above, we can now go to Step (4) to perform our tag inference by taking the power of the negative distance, i.e. $e^{-\alpha \cdot D(t_1,t_2)}$. α is a parameter which controls the power of inference. When α is set to be 0, every value is associated with every tag; when α is set to be infinity, there is no inference among tags.

Suppose value v is associated to tag t_1, t_2, ..., t_n with probability $\zeta_1, \zeta_2, \ldots, \zeta_n$ respectively, given a new tag t, the probability of v being tagged by t is thus $p = \max_{i=1}^{n} \zeta_i \cdot e^{-\alpha \cdot D(t,t_i)}$. In the application, only tags with probability above a certain threshold is worth being considered, therefore, when $D(t,t_i)$ is large for all t_i, the probability ζ will be very small, the association between v and t will thus not be recorded.

4 Tag-Based Query Processing

In order to process tag-based queries, we store both values and the associated probability according to tags, i.e., each individual tag is an individual table in our TagTable model, as shown in Figure 5.

This is inspired by the way wide-table[9] is implemented. Wide-table stores their data in column-wise manner, values are stored vertically rather than horizontally. This is to save their storage by getting rid of unnecessary empty cells, and to speed up pooling data from databases as there is no irrelevant columns occupying bandwidth in transferring.

Each table in our storage describes one tag. As an example, the table in Figure 5 stores all values associated with `make`. It has four attributes, `tid` (the tuple ID), `lid` (the location ID), `value` (the data value) and `probability` (the ζ association probability between values and tags).

TID	LID	Value	Probability
026	1	BMW	0.4
289	1	Ford	0.8
354	1	Toyota	0.9
702	1	Ford	0.7
886	1	Volvo	0.6

Figure 5: Table for tag `make`

Theoretically, for each tag $t \in \mathcal{T}$ and each value $v \in \mathcal{V}$, there is a $\zeta_{v,t}$ to associated with t and v to represent the fact that v is tagged by t with probability $\zeta_{v,t}$. But as each tag is only associates with a small proportion of the total data values, the majority of probability $\zeta_{v,t}$ is approximately 0. Therefore, only values with probabilities above a certain threshold are stored. threshold.

Similar to wide-table, TagTable does not store unnecessary empty cells, however TagTable may store more than one copy of the same value as the value can be associated with multiple tags. The larger the average number of tags that values are associated with, the larger storage the TagTable takes. Though this is not a plus point for TagTable, paying this cost is essential to support a flexible retrieval language like TagQL.

Next, we are going to elaborate on how we process tag-based query with TagTable. Throughout the discussion, we will use the following example query as illustration. Three values with tags `year`, `price` and `mileage` are to be retrieved from TagTable, by supplying constraints on tags {`make|manufacturer`}, {`model|name`} and {`price`}.

```
select      ?${year}, ?${price}, ?${mileage}
from        -
where       ${make|manufacturer} = 'Toyota'
     and    ${model|name} = 'Prius'
     and    ${price} ≥ 10,000
     and    ${price} ≤ 20,000
confidence  δ
```

Our query process consists of the following steps.

1. Parse the query according to the mutual exclusion rule, and group constraints together if they are under the same composite tag
2. Apply the maximum probability rule to select a list of values ordered by the descending of probabilities
3. Apply the TA algorithm on the generated lists until one of them is empty, or the estimated upper bound of likelihood is lower than that of the k^{th} result found so far
4. Apply Hungarian algorithm to find a matching between values and composite tags so that the likelihood is maximized while satisfying the mutual exclusion rule.

The entire algorithm is shown in Algorithm 3. Step 2 is then further expanded in Algorithm 4 while Step 4 is elaborated in Algorithm 5.

Algorithm 3 Top-k algorithm for TagQL queries

Input: A query Q written in TagQL
A positive integer k
A tradeoff parameter γ on instantiation
Output: A set of k most likely results

1: $\mathcal{S} \leftarrow \emptyset$
2: **for all** composite tag S appears in Q **do**
3: **if** $S \notin \mathcal{S}$ **then**
4: Insert S in $\mathcal{S}$
5: **if** S is a `VTag` **then**
6: add another value constraint to S
7: **for all** composite tag $S \in \mathcal{S}$ **do**
8: L_S =select(S): refer to Algorithm 4
9: $RS \leftarrow \emptyset$: the result set containing the top-k result with their scores
10: $S_{TID} \leftarrow \emptyset$
11: **while** $\nexists S \in \mathcal{S}$, where $|L_s| = 0$ **do**
12: $\iota_u \leftarrow 0$
13: highest $\leftarrow -\infty$
14: **for all** composite tag $S \in \mathcal{S}$ **do**
15: Let top$(L_s).p$ is the probability at the top of the list L_s whose TID is not in S_{TID}
16: **if** S is a `QTag` and a `VTag` at the same time **then**
17: $\iota_u \leftarrow \iota_u +$ top$(L_s).p$
18: highest $\leftarrow \max($highest, top$(L_s).p)$
19: **else if** S is a `VTag` **then**
20: $\iota_u \leftarrow \iota_u + \gamma \cdot$ top$(L_s).p$
21: highest $\leftarrow \max($highest, $\gamma \cdot$ top$(L_s).p)$
22: **else if** S is a `QTag` **then**
23: $\iota_u \leftarrow \iota_u + (1-\gamma) \cdot$ top$(L_s).p$
24: highest $\leftarrow \max($highest, $(1-\gamma) \cdot$ top$(L_s).p)$
25: **if** $|RS| = k$ and $\iota_u <$ the k^{th} score in RS **then**
26: **break**
27: Pop TID tid from L_S where highest is assigned from S, break tie randomly
28: Let score=calculate_score(tid): refer to Algorithm 5
29: **if** $|RS| < k$ or score$>$ the k^{th} score in RS **then**
30: pop RS if $|RS| = k$
31: insert (score, tid) in RS
32: insert tid in S_{TID}
33: **return** RS

We now take the above query as an example to illustrate our query processing. The first step identified the tag {`price`} in the `select` clause and the two tags {`price`} in the `where` clause are the same tag, therefore the two constraints are grouped together. By mutual exclusion rule, value `Toyota`, associated to `make` with probability 0.9 and `manufacturer` with probability 0.7, is now associated with composite tag {`make|manufacturer`} with probability 0.9, as shown in Figure 6. In step 3, the Threshold Algorithm (TA) [18, 17] selects the most likely under current estimation, and then build a bipartite graph as shown in Figure 6 to compute the best matching between tags and values.

The details are given in Algorithm 3. The parsing of the query is done from Line 1 to 6 after which the application of maximum probability rule is executed at Line 8 (by calling Algorithm 4). Line 9 to 33 then apply the TA algorithm. Finally, the Hungarian algorithm is apply in Step 4 by calling Algorithm 5 to find the tuple with maximum likelihood while satisfying the mutual exclusion principle.

However, one key issue is that the TA algorithm must be able to evaluate the so-called "likely tuples". As explained in Section 2, we study top-k queries as the association between values and tags are probabilistic. For a given value and

Algorithm 4 Select values with descending probability order by composite tag S

Input: A composite tag S
Output: A sorted list L_S containing value-probability pairs sorted by the descending order of probability

List $L_S \leftarrow \emptyset$ ordered by the descending order of the probabilities
for all tag $t \in S$ **do**
  if S is a `VTag` **then**
    Select the set of value-probability pair $\{(v, \zeta_{v,t})\}$) from table t with value constraints and probability at least δ, sorted by probability
  else
    Select value set V_t from table t with probability at least δ, sorted by probability
  for all $v \in V_t$ **do**
    if $v \notin L_S$ or $\zeta_{v,S} < \zeta_{v,t}$ **then**
      insert v in L_S and $\zeta_{v,S} \leftarrow \zeta_{v,t}$

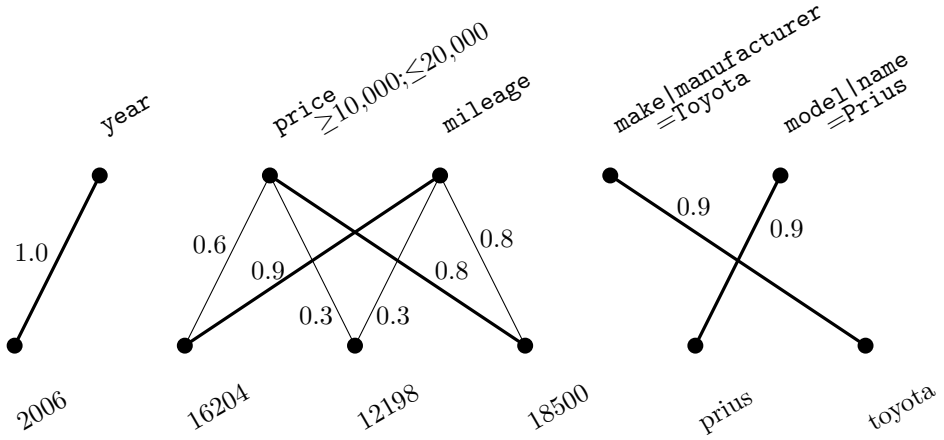


Figure 6: An illustration of matching

a given tag, there is no absolute match within our context, but a score representing how likely they can be matched, which we called *instantiation.*

Definition 5 (Instantiation)**.** *Given a set of composite tags $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ and a tuple with a sequence of values $V = \{v_1, v_2, \ldots, v_m\}$, where $n \leq m$, Let the set $F = \{f | f : \mathcal{S} \hookrightarrow V, Dom(f) = \mathcal{S}, f(S) = f(S') \Rightarrow S = S'\}$ be the set of bijections(matchings) from $\mathcal{S}$ to V, we define the instantiation ι to be $\iota = \max_{f \in F} \sum_{S \in \mathcal{S}} \log \zeta_{f(S),s}$. When $\mathcal{S}$ is taken to be the set of selection composite tags (`VTag`s), it is called selection instantiation, denoted as ι_s; when $\mathcal{S}$ is taken to be the set of projection composite tags (`QTag`s), it is called projection instantiation, denoted as ι_p.*

When we evaluate the likelihood of a given tuple, we have to consider both selection instantiation ι_s and projection instantiation ι_p. Therefore, a trade-off parameter γ is involved to balance the selection instantiation and projection instantiation by setting the overall instantiation to be $\iota = \gamma \cdot \iota_s + (1 - \gamma) \cdot \iota_p$.

The TA algorithm computes the upper bound ι_u in the following way. For composite tags which are both `QTag` and `VTag`, their instantiation should be included in both selection instantiation and projection instantiation, thus their instantiation takes the full weight of the logarithm of its associated probability. Otherwise, it takes γ of its associated probability if it only participates in selection instantiation, i.e. a `VTag`, or 1-γ if it only participates in projection instantiation, i.e. a `QTag`. The measure ι_u will cause the TA algorithm to terminate if it is no larger than the k^{th} result in the result set RS. Otherwise, we take the tuple whose value contributes most to ι_u, and try to match it with all composite tags, if it matches, we calculate the score ι, and replace the k^{th} result if ι is higher than the score of the k^{th} result in RS. The exact matching and ι calculation algorithm is presented in Algorithm 5.

Algorithm 5 Calculate the matching score of tuple tid

Input: A set of composite tag $\mathcal{S}$
A tuple id tid
Output: A score for tid matching $\mathcal{S}$
A matching between composite tags and values

1: $\iota \leftarrow 0$
2: $V \leftarrow$all values in tuple tid by one random access
3: **for all** composite tag $S \in \mathcal{S}$ **do**
4: **if** S is a `QTag` and a `VTag` at the same time **then**
5: $r \leftarrow 1$
6: **else if** S is a `VTag` **then**
7: $r \leftarrow \gamma$
8: **else if** S is a `QTag` **then**
9: $r \leftarrow 1 - \gamma$
10: **for all** value $v \in V$ **do**
11: **if** $\exists t \in S$ and t appears in the tag list of v **then**
12: edge$(S, v) \leftarrow r \cdot \log(\max \zeta_{v,t} \forall t \in S)$
13: **else**
14: edge$(S, v) \leftarrow -\infty$
15: Apply Hungarian algorithm to calculate the maximum matching score ι on $\mathcal{S}$ and V with the edge weight
16: **if** $\iota = -\infty$ **then**
17: **return** no matching
18: **else**
19: **return** score and matching between $\mathcal{S}$ and V

In line with the TA algorithm, we need to first do a random access to retrieve all values from the megatable belonging to the tuple, together with their tag list and associated probabilities. This is at Line 2 of Algorithm 5. With this set of values and the set of composite tags, we need to efficiently compute the best bijective function f that maximizes the instantiation ι as defined in Definition 5.

To do so, we construct a bipartite graph with one side representing the composite tags and the other side representing the values, as shown in Figure 6. The finding of the bijection function f has now become a problem of find the best matching in this bipartite graph. Therefore, we adopt the Hungarian algorithm [23] to find such a best matching efficiently. We now use the following tuple as shown in Figure 6 to illustrate the algorithm.

After the random access, the values {`2006, 16204, 12198, 18500, prius, toyota`} are retrieved from the megatable. Line 10 to 14 construct the weighted edge for the bipartite graph, this scalar r scales the edge weight depending on the composite tags. If the composite tag is only a `VTag`, by Definition 5, its instantiation must be adjusted by γ, therefore all edge weight from such composite tags are scaled by γ. Same reasoning goes for composite tags which is only a `QTag`, all edge weight are scaled by $1-\gamma$. However, if a composite tag is both a `QTag` and `VTag`, its weight will not be affected by γ, the overall instantiation will take the full instantiation from this composite tag.

Once the bipartite graph is built, we use the Hungarian algorithm to find out the matching which maximizes ι. We now give a proof on the correctness of the algorithm.

Property 1 (Optimality)**.** *Let $\iota_u = \sum_{S \in \mathcal{S}} r_s \cdot$top$(L_S).p$, where $r_S = 1$ for S being `QTag` and `VTag` at the same time, $r_S = \gamma$ for S being `VTag` and $r_S = 1 - \gamma$ for S being `QTag`, as defined in Algorithm 3, and let ι_k be the k^{th} score defined by Definition 5, if $\iota_u \leq \iota_k$, then $\forall t \in RS$, $\nexists t' \in \overline{RS}$, such that $\iota(t') > \iota(t)$.*

Proof. $\forall t' \in \overline{RS}$, $w(S, v) \leq$top$(L_S).p$ where $w(S, v)$ is the weight assigned to composite tag S and value v. As $w(S, v)$ is scaled by the same scale factor as top$(L_S).p$ in Algorithm 5, the selection instantiation $\iota_s(t') \leq \sum_{S \in \mathcal{S}_v} \log \zeta_{f(S),S}$ and $\iota_p(t') \leq \sum_{S \in \mathcal{S}_q} \log \zeta_{f(S),S}$ where $\mathcal{S}_v$ and $\mathcal{S}_q$ are the set of `VTag`s and `QTag`s respectively. Therefore, $\iota(t') = \gamma\iota_s(t') + (1-\gamma)\iota_p(t') \leq \iota_u$ by its definition, therefore, $\forall t \in RS$, $\iota(t') \leq \iota_u \leq \iota_k \leq \iota(t)$. □

This property shows that our method solves Problem 4 correctly.

5 Experimental Study

To understand our data model and query processing better, we conducted a bunch of experiments to validate our idea. We implemented our probabilistic tagging on top of MySQL, the interface is written in C++. It is set up on a PC with Intel(R) Core(TM)2 Duo CPU at speed 2.33GHz and 3.5GB memory.

Firstly, we are going to validate the tag distance function and the tag inference based on that, to demonstrate our distance function and tag inference are sound. The dataset we used is GoogleBase data set. We downloaded 15 vehicle related attributes with about 2 million data values. As GoogleBase provides options on attribute names for users to upload data, the entire is in fact quite well organized, and is unlikely that there are two attributes carrying the similar semantics. Therefore we pre-processed GoogleBase dataset as follows. For each of the attribute, we created a number of different attribute names with the same prefix. For example, we created another 9 attribute names `make_a`, `make_b`, ... `make_i` from the original attribute name `make`. These attribute names are used to simulate different tags from heterogeneous data sources, but with the same semantics. As we noticed that, the number of databases decreases as the size of database increases, these 10 tags are divided into four groups with different weighting scheme. Given a positive parameter p, the first group contains 4 attribute names, `make`, `make_a`, `make_b` and `make_c`, which are assigned with weight 1^p, the second group contains 3 attribute names, `make_d`, `make_e` and `make_f`, which are assigned with weight 2^p, the third group has only two attribute names, `make_g` and `make_h`, having weight 3^p, the last group, with the sole attribute `make_i`, is assigned with weight 4^p. With different weights assigned to each attribute name, we can calculate a probability distribution with probabilities proportional to their weights. The next is to assign every value to a tag according to this probability distribution. The assignment will give associated probability 1, associated probabilities with other tags are to be inferred. For example, if $p = 0$, the attributes names are uniformly randomly assigned, if $p = 1$, the probability for each one in the first group is 5%, for each one in the second group is 10%, 15% for the third group and 20% for the last group. In the experiment, we consider each attribute name as a tag, and then obtain the tag inference after clustering the values. The clustering ran 3 times with different initialization on a 2% sample ($\sim$40 thousand data values), while tags generated about 800 dimensions. We expect tags with the same ancestor to be high.

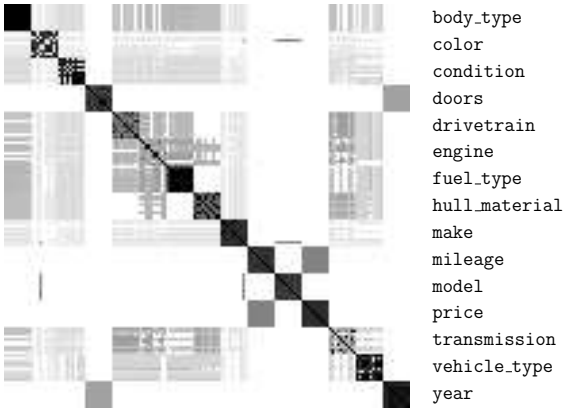


Figure 7: Tag distance bitmap: the closer two tags are, the darker the pixel is

Figure 7 visualizes the pairwise distance between tags. Each row and column represents a tag, the distance between the row tag and the column tag at a given pixel is inversely related to the darkness of the pixel. Note that the tags are arranged in the same order in both row orientation and column orientation, and tags with the same ancestor are grouped together. Therefore, the small squares on the diagonal are dark as the distances between tags with the same ancestor are small. The outlier is the fourth last attribute `price` and the sixth last attribute `mileage`, their off diagonal intersections still give a moderate greyscale, meaning these tags, although not having the same ancestor, are closer to one another.

Figure 8 shows how the maximum of the smallest-k all pair distances change as varying parameter p. Firstly, tag distance is very small when k is less than 1000. As shown in Figure 7, these small distance come from tags with the same ancestor. Secondly, the increment of the tag distance is stable regardless of the parameter p, i.e. regardless of how the distributions are. Therefore, our distance measure is robust.

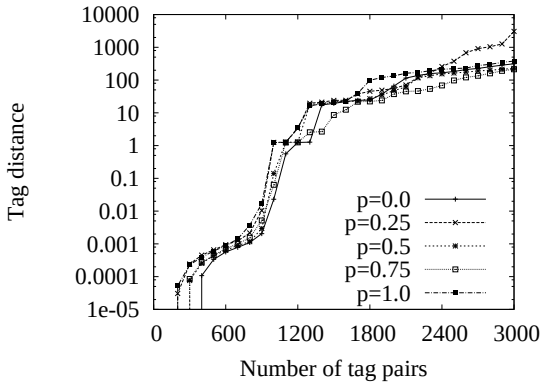


Figure 8: Tag distance steady increases regardless of the parameter p

We validated our top-k query with the original tags in Figure 9. All values that were assigned to other tags must be inferred back in order to be queried by the original tags. There are two sets of parameters we validated top-k queries against, one is k, the parameter introduced by the query itself, the other is T, the total number of composite tags appearing in the TagQL query. The precision of all queries, as shown in Figure 9(a), reaches as high as 1, this shows the result returned from our query engine are all valid ones.

As we purposely choose some query which returns around 100 tuples from the original data, therefore, Figure 9(b) shows a steady increase in the recall, while it reaches 80% to 90% when k is the same as the number of tuples in the result set. This shows there are some false negatives answered by our TagQL engine, which is caused by under inference, i.e. tags with the same semantics are not detected similar. This may due to the insufficient value overlapping. This usually happens on numerical values, as `price` is similar to `mileage` on their values, which caused false positives. We also plotted how the recall looks like when there is no inference, i.e. $\delta = 1$; this is shown in Figure 9(c). When the number of tags reaches 6, the recall is 0, thus comparison for recall when $T = 6$ and $T = 8$ is not presented here. Figure 9(c) shows how important it is to do tag inference when multiple tags are used for the same semantics.

The next set of experiments focuses on the CPU time. We have two choices of doing TA algorithm for query processing. The first method, TA-all, retrieves a list of values no matter it is a `QTag` or it is a `VTag`. However, the second method, TA-V, only retrieves lists for `VTag`s, and `QTag`s will be assigned at the matching step. Both figures in Figure 10 show that TA-V overtakes TA-all. The reason is that lists for `QTag`s are comparatively longer than lists for `VTag`s, as there is no value constraint. On the other hand, less number of lists frees more memory for buffering from megatable, which is expected to reduce random access. It is interesting that when fixing $k = 60$, the CPU time for queries with 4 composite tags is the minimum. The reason is this, when there are more composite tags, more lists are fed to TA algorithm, if there happens to be a list which is much shorter than others, this list will be exhausted rapidly, thus will make CPU time shorter.

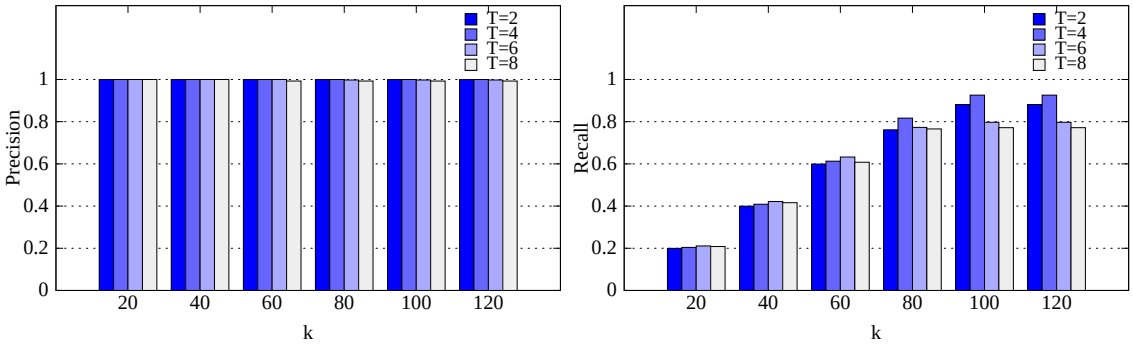
To confirm that TA-V method is better than TA-all method in term of I/O access, we used the number of access that TA has to retrieve tuple from megatable to simulate the number of random access, given page fault keeps constant. On the other hand, the number of sequential access is simulated by the number of times that TA algorithm poped the lists. The real number of random access and sequential access are not proportional to the two colors in Figure 11. This Figure once again showed the TA-V method can save as much as half of the time that TA-all needs.

Next, we are going to do an experiment that helps one integrate his or her own personal database with some other databases to broaden the result. This experiment used two dataset. One is homogeneous, collected by Cai-Nicolas Ziegler [1] (`BX` dataset), which consists of five attributes `ISBN`, `Book-Title`, `Book-Author`, `Year-Of-Publication` , `Publisher`, and 260 thousand tuples. The other dataset is from Freebase [2] (`FB` dataset), which is very heterogeneous. There are as many as 25 attributes, most of which are null; there are only 1.4 million values out of 280 thousand rows.

From Figure 12 selected all pairs of `BX` tags and `FB` tags with pairwise distance less than 10. We see that string val-

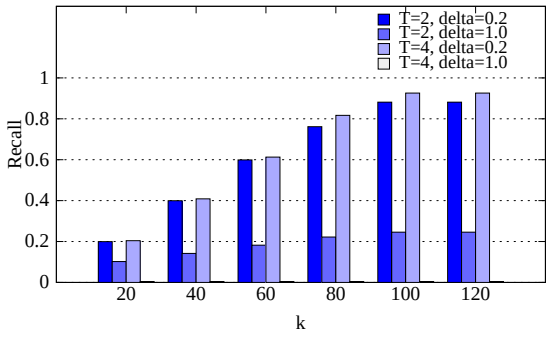
[1] http://www.informatik.uni-freiburg.de/ cziegler/BX/

[2] http://download.freebase.com/datadumps/



(a) Precision by varying k

(b) Recall by varying k

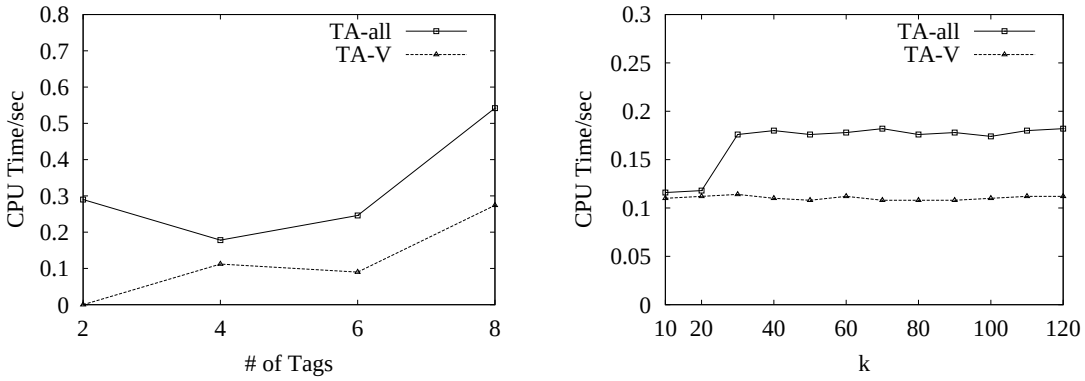


(c) Recall when $\delta = 1$

Figure 9: Precision and recall by varying k

ues with semantics `author`, `title` and `publisher` are close to one another. The tag `Year-of-publication` is however nearer to numerical types, with the minimum distance to `Publication_Date`.

We then selected 10 queries in SQL, first process them on the original BX data, then translated them into TagQL, and queried in the TagTable database constructed by BX and FB together. As FB dataset is really heterogeneous, there are some invalid result returned by TagQL, we thus invited some of colleague to help identify which are the valid result. The result is shown in Figure 13, the average number of valid answers has increased from about 8 to about 12, which is significant. We believe building TagTable under our probabilistic tagging framework does help people integrate their own data with community web services to perform query more effectively.



(a) CPU time by varying T

(b) CPU time by varying k

Figure 10: CPU time by varying k or t

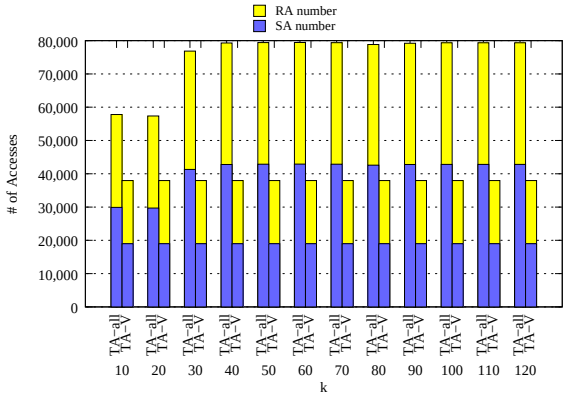


Figure 11: Number of access by varying k

6 Related Work

Query Relaxation In this recent decade, there are several efforts of driving database technologies from a strict and well-defined manner to a relaxed and user-oriented manner. The first category of such efforts is to relax the way of querying. Inspired by the *Information Retrieval (IR)* community, a series of approaches that support keyword search in database have been studied extensively [3, 22, 25, 28, 31, 40]. while IR community focuses on processing unstructured queries, i.e. keyword-based queries, on data with little structure, e.g. text documents, keyword-based search in database processes unstructured queries on structured data. With keyword search, users are freed from the necessity of forming queries in a specific database language, but writing the queries in the same fashion of submitting a search over Internet. So then database can be utilized even for users without expertise.

The limitation of keyword search is that it does not supply a function to the users about what they are looking for, as it is a value-based approach. In order for users to specify what they query about, a way of describing the semantics of the values, including the keywords and expected query results, must be provided. [10, 20, 29] have illustrated the usage of tags, and tags has been proven to be an effective way on taxonomy building, and thus to deduce the semantics of the objects being tagged. Tagging on multimedia objects has been widely applied on web 2.0 application [1, 2], as a way to describe the content of the semantics of such object.

Database Relaxation The other category to relax database queries is to make database itself less constrained by schemas. As databases evolved over time introduce significant amount of noise, or become heterogeneous by integrating with other databases,

BX Tags	FB Tags	Distance
Book-Author	Book	0.622
	Name	0.957
	Publisher	3.129
Book-Title	Book	0.169
	Name	0.357
	Publisher	1.369
Publisher	Book	0.038
	Name	0.052
	Publisher	0.129
	Place_of_Publication	6.217
Year-of-publication	Publication_Date	0.390
	ISBN	1.635
	LCCN	2.702
	OCLC_Number	3.494

Figure 12: Tag distance between BX tags and FB tags

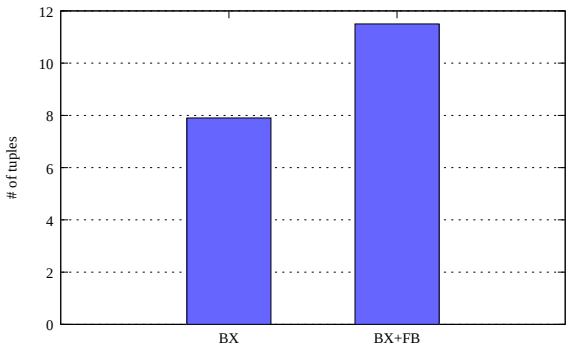


Figure 13: BX dataset v.s. BX+FB dataset on the average number of valid result returned

schemas are hard to obey at all times. This drives researchers to study on malleable schemas [15, 41] or even database without specific schemas like Bigtable [7, 8], or wide-table[9, 24]. Conceptually, Bigtable maps a unique tuple ID and a unique column ID to a data value. Each row of data only relates a small fraction of the total number of columns, which is in the magnitude of hundreds or thousands. Bigtable is totally schema free, user can just insert his own data into appropriate column without worrying about the schema of the database. User may even create a new column if there is no existing column that suits the data to be entered. Another column-wise storage, wide-table, stores column names and values in pairs. However, under both data model, one value is assigned to a unique column, which make it impossible to process a column whose name does not match with the query.

Our work demonstrates a way of relaxing databases and queries at the same time. Firstly, we completely free the data from schema like what wide-table does. Besides that, we associate each value to a set of columns so that values can be accessed from different columns.

Probabilistic Database Once the database constraints are relaxed, the uncertainty is involved. Instead of that a tuple is either an answer to a given query or not in RDBMS, each tuple in probabilistic databases is associated with a probability to match the query. Studies on probabilistic databases have been ongoing for many years. [14, 32, 33, 13]. Both our work and current researches on probabilistic databases are focusing on finding an effective method for probabilistic data management and query processing. However, there are three essential differences, namely, uncertainty types, data modeling and query processing. More researches on probabilistic databases has been done to deal with the uncertainty about row existing[19, 36], value existing or both of them [39, 21].

In our work, both rows and values exist, but the uncertainty comes from the flexibility of assigning values to differ-

ent columns. *Possible world* is a popular model used for probabilistic database to resolve semantic ambiguity[36, 26]. But possible world semantic does not fit to our framework, as we do not consider tags being mutual exclusive, there are correlations between tags, which also allow us to adopt statistical inference to explore a more generic relationship between values and tags. Secondly, query ranking criteria are different from ours. Current query processing in probabilistic databases try to find an appropriate aggregation that accounts for both scores and existence probabilities of tuples[19, 27, 26, 35] However, these two measures are not consistent with each other. In our work, we make it a trade-off between scores and existence probabilities so that users can customize it on their own preference.

Information Integration The majority of researches in data integration or information integration maps heterogeneous schema to one centralized mediated schema. But it is not easy to figure out a mediated schema that every other schema can be squeezed through without losing any information. Prior work which is close to our concept is on probabilistic mediate schema and probabilistic schema mapping proposed by X. L. Dong and A. Y. Halevy [16, 34]. A mediated schema based on the source attribute clustering is constructed, and the probabilistic schema mapping is produced at the same time. To assign the probability of the possible mediated schema, it is assumed that the attributes in source data are mutually exclusive with each other. In addition, only the more frequent attributes are considered, the less frequent ones are just omitted. Under our probabilistic tagging framework, all attributes or tags can coexist in our system, and values associated with such tags can be inferred from other tags.

7 Conclusions

There can be many extension based on this framework, one of which is to reduce the storage space, as each value has been stored multiple times under multiple tag tables in our current model. We now conclude our paper as follows.

In order to grant more privilege to database users who do not know much about the underlying schema while maintaining the database quality, we propose probabilistic tagging, which associates data values to tags with probabilities. Under this framework, we have introduced our data storage, TagTable, our query language, TagQL, the method of doing tag inference for all values based on clustering, and the fast and effective online matching for query processing.

References

[1] Flickr - photo sharing, http://www.flickr.com.

[2] Youtube - broadcast yourself, http://www.youtube.com.

[3] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16. IEEE Computer Society, 2002.

[4] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[5] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. W. 0002, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[6] C. Y. Chan, B. C. Ooi, and A. Zhou, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*. ACM, 2007.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data (awarded best paper!). In *OSDI*, pages 205–218. USENIX Association, 2006.

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[9] E. Chu, J. L. Beckmann, and J. F. Naughton. The case for a wide-table approach to manage sparse relational data sets. In Chan et al. [6], pages 821–832.

[10] M. Cole and J. Gwizdka. Tagging semantics: investigations with wordnet. In R. L. Larsen, A. Paepcke, J. L. Borbinha, and M. Naaman, editors, *JCDL*, page 446. ACM, 2008.

[11] B. T. Dai, N. Koudas, B. C. Ooi, D. Srivastava, and S. Venkatasubramanian. Rapid identification of column heterogeneity. In *ICDM*, pages 159–170. IEEE Computer Society, 2006.

[12] B. T. Dai, N. Koudas, D. Srivastava, A. K. H. Tung, and S. Venkatasubramanian. Validating multi-column schema matchings by type. In *ICDE*, pages 120–129. IEEE, 2008.

[13] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[14] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

[15] X. Dong and A. Y. Halevy. Malleable schemas: A preliminary report. In A. Doan, F. Neven, R. McCann, and G. J. Bex, editors, *WebDB*, pages 139–144, 2005.

[16] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, editors, *VLDB*, pages 687–698. ACM, 2007.

[17] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2):109–118, 2002.

[18] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*. ACM, 2001.

[19] T. Ge, S. B. Zdonik, and S. Madden. Top-*k* queries on uncertain data: on score distribution and typical answers. In *SIGMOD Conference*, pages 375–388, 2009.

[20] S. A. Golder and B. A. Huberman. The structure of collaborative tagging systems. *CoRR*, abs/cs/0508082, 2005.

[21] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[22] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[23] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, 2001.

[24] B. Li, M. Hui, J. Li, and H. Gao. iva-file: Efficiently indexing sparse wide tables in community systems. In *ICDE*, pages 210–221. IEEE, 2009.

[25] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In Wang [38], pages 903–914.

[26] J. Li and A. Deshpande. Consensus answers for queries over probabilistic databases. In *PODS*, pages 259–268, 2009.

[27] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[28] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In S. Chaudhuri, V. Hristidis, and N. Polyzotis, editors, *SIGMOD Conference*, pages 563–574. ACM, 2006.

[29] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *WWW*, pages 641–650, 2009.

[30] T. Neumann and G. Weikum. Rdf-3x: a risc-style engine for rdf. *PVLDB*, 1(1):647–659, 2008.

[31] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of rdbms. In U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, editors, *SIGMOD Conference*, pages 681–694. ACM, 2009.

[32] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895. IEEE, 2007.

[33] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.

[34] A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In Wang [38], pages 861–874.

[35] M. A. Soliman and I. F. Ilyas. Ranking with uncertain scores. In *ICDE*, pages 317–328, 2009.

[36] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[37] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *CoRR*, physics/0004057, 2000.

[38] J. T.-L. Wang, editor. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. ACM, 2008.

[39] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[40] B. Yu, G. Li, K. R. Sollins, and A. K. H. Tung. Effective keyword-based selection of relational databases. In Chan et al. [6], pages 139–150.

[41] X. Zhou, J. Gaugaz, W.-T. Balke, and W. Nejdl. Query relaxation using malleable schemas. In Chan et al. [6], pages 545–556.