Schema-As-You-Go: On Probabilistic Tagging and Querying of Wide Tables

Meiyu Lu[‡] Divyakant Agrawal[§] Bing Tian Dai[‡] Anthony K.H. Tung[‡]
[‡]School of Computing
National University of Singapore
{lumeiyu, dbt, atung}@comp.nus.edu.sg
[§]Department of Computer Science
University of California, Santa Barbara
agrawal@cs.ucsb.edu

ABSTRACT

The emergence of Web 2.0 has resulted in a huge amount of heterogeneous data that are contributed by a large number of users, engendering new challenges for data management and query processing. Given that the data are unified from various sources and accessed by numerous users, providing users with a unified mediated schema as data integration is insufficient. On one hand, a deterministic mediated schema restricts users' freedom to express queries in their preferred vocabulary; on the other hand, it is not realistic for users to remember the numerous attribute names that arise from integrating various data sources. As such, a user-oriented data management and query interface is required.

In this paper, we propose an out-of-the-box approach that separates users' actions from database operations. This separating layer deals with the challenges from a semantic perspective. It interprets the semantics of each data value through tags that are provided by users, and then inserts the value into the database together with these tags. When querying the database, this layer also serves as a platform for retrieving data by interpreting the semantics of the queried tags from the users. Experiments are conducted to illustrate both the effectiveness and efficiency of our approach.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications

General Terms

Management, Performance

Keywords

Probabilistic Tagging, Wide Table, Top-k Query Processing

1. INTRODUCTION

The rapid growth of Web 2.0 technologies and social networks has provided us with new opportunities for sharing data and collaborating with others. Within a community, users may like to contribute their own data, share the data with friends, and meanwhile explore the shared data at the community scale. Effectively and efficiently managing and exploring these data at a web-scale level is an interesting and important problem.

In this paper, we focus on the management of shared structured tables (e.g. Google fusion table [16]) which is an important type of data in Web 2.0. Such data have some inherent properties that make their management challenging.

Heterogeneity. Data provided by different users may be widely heterogeneous, containing different attribute names that describe semantically similar values or same attribute name with different semantics. For example, in Figure 1(a), user A chooses attribute `make` to describe the car manufacturer in his/her private data, while user B uses `manufacturer` to represent the same meaning. Similarly, the semantics of `car` in (b) is model but in (c) it contains both make and model information.

Numerosity. As all users can publish their data into the community as shown in Figure 1(a), (b) and (c), the number of distinct attributes will grow drastically as more users participate. This results in a *wide table* [7] as Figure 1(d) which is essentially a structured table with a large number of attributes and many empty cells. Given such a wide table, most existing data integration approaches [23] create a global mediated schema that usually contains a huge number of attributes. Such huge mediated schema is difficult to query as users cannot remember all the attributes. Therefore, a more flexible and user-oriented query interface is required. This interface should allow users to choose different attribute names to represent the same semantics in different queries, and have the power to express their confidence over the attribute semantic heterogeneity in posed queries.

Personalized Attributes. Since attributes are contributed by users, there could be many personalized attributes, which cannot be captured by existing ontology or taxonomy, such as DBpedia[1] and WordNet[2]. Examples of such personalized attributes include `myear` in Figure 1(c), which represents the production year of a car, and `eadd`, the abbreviation for email address. Due to this, existing techniques that rely on ontology for schema mapping [26, 30] will not be suitable any more. A better system should be able to infer more popular

[1]DBpedia, http://dbpedia.org/About
[2]WordNet, http://wordnet.princeton.edu

SIGMOD'11, June 12–16, 2011, Athens, Greece.

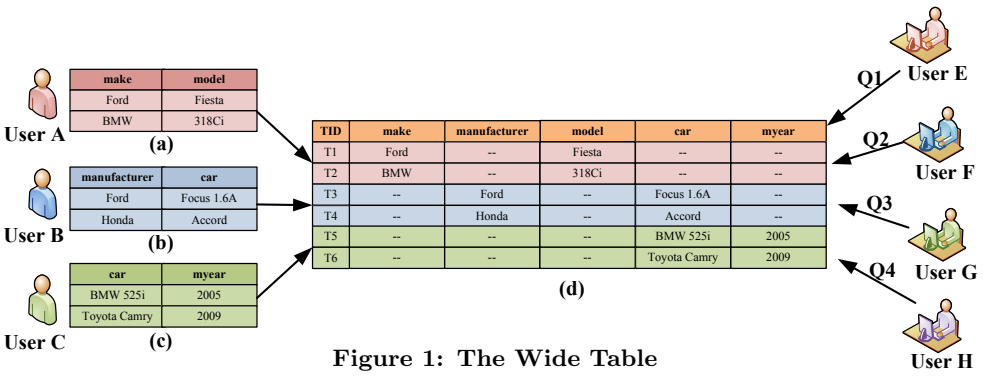


Figure 1: The Wide Table

Tag1	Tag2	Similarity
make	manufacturer	0.95
make	car	0.5
manufacturer	car	0.52
model	car	0.65

Figure 2: Tag Similarity

attributes for such personalized ones, so that these data can be explored by the other users asides from its owner.

Due to the inherent semantic heterogeneity of the attributes in user contributed data, we will no longer refer to them as "attribute". Instead, we borrow the term *tag* from the multimedia domain [1, 2], and view these attribute names as just a way to annotate the values that are provided by users[3]. In this paper, we describe a query system which enables users to effectively query a large quantity of user contributed data. Our system provides the following functionalities.

- Users can contribute and share data by inserting them into a single wide table together with their tags, as shown in Figure 1(d).
- Our system is able to automatically discover the semantic relationships between users' tags, which is handled by our tag inference approach in Section 3. Figure 2 illustrates the semantic similarities between each pair of tags produced by our approach for the example data in Figure 1[4].
- Users can use their own preferred tags to query the database, and the system then dynamically determines the semantics of these tags at query time.

Combining the data from Figure 1(d) with the tag similarities in Figure 2, we can easily obtain the virtual probabilistic tagged data, as shown in Figure 3, where each value is associated with a list of $\langle tag, probability \rangle$ pairs. The probability indicates the semantic association strength between the value and tag pair. The generation of such semantic association probabilities is called *probabilistic tagging*, and its formal definition will be given in Section 2.

We next present an example to illustrate several characteristics of our query interface via a set of queries.

[3]In this case, users might even choose to annotate the same value with multiple tags.

[4]As our similarity measure is symmetric, we only show unidirectional tag similarity in Figure 2.

EXAMPLE 1. *Figure 1 shows a scenario where four users, A, B, C and D contribute and share their datasets. The wide table is used to store the shared data. User E, F, G and H then submit four different queries which are Q1, Q2, Q3 and Q4, respectively. The queries are shown in Table 1 in an extended SQL format. The delimiter '@' in the query is for users to explicitly specify their confidence over tag semantics. Its formal definition is presented in Section 2. Here we can interpret* `make@0.9` *as a set of tags in Figure 2 which are similar to* `make` *with confidence no less than 90%. The table name* `CAR` *refers to the probabilistic tagging table in Figure 3. Values in the result tuples have the same order as the tags in the* `SELECT` *clause.*

From the query result in Table 1, we observe that ***i)*** *different users can choose different tags to express the same semantics, such as Q1 and Q2;* ***ii)*** *explicit confidence specification gives users control over the search scope. For instance, with lower confidence Q3 retrieves two more tuples than Q1;* ***iii)*** *schema for users' query is dynamically determined at query time, which is user-oriented rather than predefined. Take Q4 as an example, where a user would like to retrieve values that are associated with* `car` *and* `model` *from* `CAR`*. From the data perspective, for T3 and T4, tag* `car` *in the query should be aligned with* `car` *in the source table. But from the users' perspective, intuitively the best schema alignment for T3 and T4 should be (car, model). This is what our system retrieves given Q4.*

To further see the benefit of our approach, Figure 4 shows the result of Q4 over tuples T3 and T4 using our approach and the one proposed by Dong and Halevy in [12, 25][5]. Both records returned by our approach are correct and ranked in the right order. However, the top two ranked tuples retrieved by probabilistic integration are not correct although their score is much higher than the correct ones. The result differs because we are dynamically determining the semantics for user posed tags at query time. The reasons for such differences will be explained in Section 2.2.

Besides the functionalities discussed above, our work also makes the following contributions:

- We proposed the concept of *probabilistic tagging* to represent the associations between values and tags.
- We presented an effective distance function to measure the semantic distance between a pair of tags. Based

[5]Please refer to Appendix A for the detailed score computation for probabilistic integration.

Table 1: Example Queries

QID	Query	Result
Q1	`SELECT make@0.9 FROM CAR;`	T1(Ford),T2(BMW),T3(Ford),T4(Honda)
Q2	`SELECT manufacturer@0.85 FROM CAR;`	T1(Ford),T2(BMW),T3(Ford),T4(Honda)
Q3	`SELECT make@0.5 FROM CAR;`	T1(Ford),T2(BMW),T3(Ford),T4(Honda), T5(BMW 525i),T6(Toyota Camry)
Q4	`SELECT car, model FROM CAR;`	T1(Ford,Fiesta),T2(BMW,318Ci), T3(Ford,Focus 1.6A),T4(Honda, Accord)

TID	Value	Probabilistic Tagging
T1	Ford	<make,1.0>,<manufacturer,0.95>,<car,0.5>
T1	Fiesta	<model,1.0>,<car,0.65>
T2	BMW	<make,1.0>,<manufacturer,0.95>,<car,0.5>
T2	318Ci	<model,1.0>,<car,0.65>
T3	Ford	<manufacturer,1.0>,<make,0.95>,<car,0.52>
T3	Focus 1.6A	<car,1.0>,<model,0.65>,<manufacturer,0.52>,<make,0.5>
T4	Honda	<manufacturer,1.0>,<make,0.95>,<car,0.52>
T4	Accord	<car,1.0>,<model,0.65>,<manufacturer,0.52>,<make,0.5>
T5	BMW	<car,1.0>,<model,0.65>,<manufacturer,0.52>,<make,0.5>
T5	2005	<myear,1.0>
T6	Toyota	<car,1.0>,<model,0.65>,<manufacturer,0.52>,<make,0.5>
T6	2009	<myear,1.0>

Figure 3: Probabilistic Tagging

QID | **Q4**

Query: SELECT car, model FROM CAR

Our Result

TID	car	model	rank
T3	Ford	Focus 1.6A	1
T4	Honda	Accord	2

Data Integration Result

TID	car	model	rank
T3	Focus 1.6A	Focus 1.6A	1
T4	Accord	Accord	2
T3	Ford	Focus 1.6A	3
T4	Honda	Accord	4

Figure 4: Query Result Comparison

on this, we are able to automatically infer semantically similar tags for each value.

- We proposed an efficient dynamic instantiation approach to associate the queried tags and data values during query processing.
- A complete and extensive experimental study is conducted to validate our approach.

2. PROBLEM DEFINITION

2.1 Probabilistic Tagging

As discussed earlier, one of our main objectives is to provide a flexible interface, which allows users to query the underlying database with any tag they like. The very first challenge is to discover which tags are similar in semantics. However, this is insufficient since some tags are more similar to each other in semantics while others are not. Taking T1 in Figure 3 as an example, it is intuitive that the semantics of 'Ford' is better reflected by `manufacturer` than `car` since the semantics of `manufacturer` is quite specific, while `car` is a relatively more general term. Therefore, the second issue is that the semantic association strength between a value and tag should be well quantified, as shown in Figure 3.

In this paper, for a given value v in the wide table, we interpret its semantic association with a tag t as the likelihood/belief that v is tagged by t.

DEFINITION 1 (PROBABILISTIC TAGGING). *Let T be a set of tags, for a value v in Wide Table and a tag $t \in T$, probabilistic tagging refers to the process of associating a probability $\zeta_{v,t}$ for v and t, such that value v is tagged by tag t with probability $\zeta_{v,t}$.*

The probability $\zeta_{v,t}$ is called the associated probability or association probability. If the semantic association between value v and tag t is stronger, then their associated probability $\zeta_{v,t}$ will be higher and vice versa. As shown by T1 in Figure 3, for the value 'Ford', its associated probability with its original tag `make` is 1.0 while its associated probability with inferred tags `manufacturer` and `car` are 0.95 and 0.5, showing difference in the strength of semantic association.

We note here that unlike previous works [12, 25, 17] in data integration, the probabilities for all tags being associated with a value v do not sum up to 1. This is because there is no assumption of mutual exclusion between tags at this stage of data processing. We avoid doing so to prevent a label bias problem in which a value v will have much lower associated probability with each tag if it can be semantically described by many different tags. For example, a value associated with 5 or more tags will have much lower probability than value with 2 or less tags. Instead, we choose to introduce the assumption of mutual exclusion during query time where the semantic meaning of the tags are clearer.

2.2 Query Semantics

After discovering and quantifying the tag semantic relationships during probabilistic tagging, the resulting probabilistic table is shown as in Figure 3. In this table, each value is associated with a $\langle tag, probability \rangle$ pair list. Query over such probabilistic table is much more challenging than with a predefined schema. We will now look into the query relevant issues over a probabilistically tagged table.

2.2.1 Extended SQL Query

In this paper we adopted a SQL like query syntax to support our tag based queries. Specifically, we extended SQL query to allow users to express their semantic confidence for each tag using the delimiter '@'. We show some examples in Table 1. Below is a more complex extended SQL query.

```
SELECT    car, year@0.9, price@0.3
FROM      CAR
WHERE     make@0.5 = 'Ford'
      AND price ≥ 10,000
      AND price ≤ 20,000
```

When a user provides a tag with high confidence, it means that he is very sure about the semantics of the tag, and does not want our system to do much tag inference. In contrast, if the user is unsure of the semantics and he likes to use more tags that are similar to the queried tag, then he can provide a lower confidence threshold.

DEFINITION 2 (TAG EXPANSION). *Given a tag t_q and its semantic confidence δ in query q, $t_q@\delta$ refers to the expanded tag set $T_e = \{t_{e_i} | i = 1, \ldots, n\}$, where for each tag $t_{e_i} \in T_e$, $Semsim(t_q, t_{e_i}) \geq \delta$. We call T_e as the expanded tag set for t_q.*

In the above definition, $Semsim(t_q, t_{e_i})$ is the semantic similarity between tag t_q and t_{e_i}, which is formally defined by Equation 6 in Section 3. When processing query q, t_q will be expanded to a tag set T_e, a set of tags that are semantically similar to t_q based on a threshold δ. All values that are originally associated with any of the tags in T_e are valid for this tag expansion condition, and should be taken into consideration when answering the query. If no semantic confidence is explicitly specified for t_q, we will expand it to all our inferred tags.

Take the queries in Table 1 as example, `make@0.5` will be expanded to all the tags that are similar to `make` in semantics inferred by our system, that is {`manufacturer`,`car`}, but `make@0.9` will only be expanded to {`manufacturer`}.

2.2.2 Query Answering

Before explaining the semantics of query answering, we first look at the intuitions behind our query answering techniques. Intuitively, syntactically equivalent tags in user posed queries should be instantiated with the same value, no matter how many places it appears in the query. For example, although `price` occurs three times in the example query in Section 2.2.1, once in the `SELECT` clause and twice in the `WHERE` clause, it is obvious that all the occurrences of `price` refer to exactly the same semantics, i.e. price of a car. Thus, in the resulting record, all these three instances of `price` should be instantiated with the same value. Based on this, we proposed the following **Consolidation Rule** for our query answering.

RULE 1 (CONSOLIDATION RULE). *Let set $T_q = \{t_i | i = 1, \ldots, n\}$ be the set of tags in query q, for two tags $t_i, t_j \in T_q$, if $t_i = t_j$, then t_i and t_j should be instantiated with the same value from the underlying record.*

Consider the query in Section 2.2.1, although we know tag `car` and `make` are similar in semantics from Figure 2, in this query they actually represent different attributes because the user specifies them as different. In other words, `car` and `make` are mutually exclusive from each other in this query, and should be instantiated with different values in the resultant record. We proposed our **Mutual Exclusion Rule** as below.

RULE 2 (MUTUAL EXCLUSION RULE). *Let $T_q = \{t_i | i = 1, \ldots, n\}$ be the set of tags in query q, for two tags $t_i, t_j \in T_q$, if $t_i \neq t_j$, then t_i and t_j should be instantiated with different values from the underlying record.*

We have discussed the impact of our mutual exclusion rule using Q4 earlier (in Example 1). There for T3, we argued that `car` in the query should be aligned with `manufacturer` in the source table. Here we look at the query in Section 2.2.1, where a user would like to retrieve the value associated with `car` from a record whose value 'Ford' is associated with `make` (for simplicity, we skipped the other constraints). Now for T3, it is more reasonable for `car` in the query to be aligned with `car` in the source table. Comparing these two queries, we can see that with our mutual exclusion rule, the same tag (`car`) can be aligned with different semantics in different queries even for the same record (T3). As explained earlier, applying the mutual exclusion rule during query answering time allows us to avoid the label bias problem. Moreover, our mutual exclusion rule cannot be trivially introduced in [12, 25, 17] since they enforced the fact that two attributes/tags are either semantically equivalent (associated with the same value) or not in each possible world. As such, the problem illustrated by Figure 4 in Example 1 is inherent for [12, 25, 17].

For a given tuple/record, we aim to find its possible best alignment to the queried tags. As the semantics of queried tags are determined dynamically on the fly, our query answering is referred to as **Dynamic Instantiation**.

DEFINITION 3 (DYNAMIC INSTANTIATION). *Let set $T_q = \{t_i | i = 1, \ldots, n\}$ be the set of tags in query q, and $V_r = \{v_j | j = 1, \ldots, m\}$ be the set of values from a record r, we refer to a mapping $f : T_q \rightarrow V_r$ as the dynamic instantiation of V_r with respect to T_q. Based on f, the instantiated score $Score(r)$ for r with respect to q is defined as the multiplication of associated probabilities for edges in f, i.e. $Score(r) = \prod_{i=1}^{n} \zeta_{f(t_i),t_i}$. In addition, all the following conditions must be satisfied in f:*

1. *f satisfies all the value constraints in the `WHERE` clause.*
2. *f satisfies both Consolidation Rule and Mutual Exclusion Rule;*
3. *The associated probability for each edge in f, i.e. $\zeta_{f(t_i),t_i}$, must satisfy its corresponding tag expansion;*
4. *Among the mappings in which all the above three conditions hold, dynamic instantiation refers to f whose score is maximized, i.e. $f = \arg\max_f Score(r)$.*

The first condition states that if tag t has value constraints in the `WHERE` clause, then its mapped value $f(t)$ should satisfy all the value constraints over t in query q. The second condition is for us to better capture the users' intention as discussed earlier. The third condition is used to meet the tag semantic specifications. Finally, the last condition is to find the best possible instantiation between tags set T_q and values set V_r, i.e. the one with maximum instantiation score $Score(r)$. Our instantiated score is defined as the multiplication of associated probabilities in the instantiation f, that is because in our case the associations between tag and value pairs are independent, and based on our mutual exclusion rule, all the edges falling outside of f are invalid.

Due to the semantic uncertainty of tags, answering a query q can result in a large number of records with score larger than 0. In view of this, we mainly focus on top-k query processing to find the best k answers based on our query answering semantics.

DEFINITION 4 (TOP-k QUERY ANSWERING). *Given a query q, a user specified positive integer k, and a set of*

records $R = \{r_i | i = 1, \ldots, |R|\}$, the problem of top-k query answering is to retrieve a record set RS, such that $RS \subseteq R$, $|RS|$ is maximized with the condition $|RS| \leq k$, $\forall r \in RS$ and $\forall r' \in \overline{RS}$, which is the complementary set of RS, $Score(r) \geq Score(r')$.

From the above definition, it is possible that less than k results are retrieved. We however feel that it is a worthy cost to pay for freeing users from a fixed mediated schema and providing relaxation over the tag specifications.

3. TAG INFERENCE

We next describe our tag inference process which is essential for discovering the semantic relationships between tags, and computing the associated probabilities between tag and value pairs. We use $T = \{t_i | i = 1, \ldots, n\}$ to denote all the tags in our database, and $V = \{v_j | j = 1, \ldots, m\}$ to denote all the values. For a tag $t \in T$, we use $V_t \subseteq V$ to represent the set of values that fall under the tag t in the wide table.

Effectively discovering the semantic similarities between user contributed tags however is challenging. Although several approaches have been proposed in the literature [23, 26, 30], none of them is suitable for our case. First, string similarity over tag names could not convey all cases [26, 23]. With such approaches, `model` and `model_year` will be treated as similar but `make` and `manufacturer` will be treated as dissimilar. Second, ontology and taxonomy based methods [26, 30] are not applicable, as there are many personalized tags like `myear` and `eadd`, which cannot be captured by any existing ontology and taxonomy. Finally, similarity function based on values overlap [23] works only for categorical values, not for string and numerical values. For example, given two set of emails that are contributed by different users using two different tags, it is unlikely that there is much overlap between the two set of emails.

Our approach to discovering the semantic similarities in this paper can generally be described in two steps. First, soft clustering [28] is applied to the values in the wide table to separate them probabilistically into K clusters, i.e. each value can belong to each of the K clusters with different probabilities. Since each tag is associated with a set of values with different probabilities, we can thus indirectly compute the probability of these tags being associated with the K clusters. In the second step, we measure the distance between tags by comparing the probability distribution of their associated membership to each of the K clusters using KL-divergence [8, 9]. KL-divergence is a function for measuring the relative entropy between two distributions. Specifically, if the probabilistic cluster membership distributions for two tags are p and q respectively, their relative entropy is $KL(p\|q) = \sum_{i=1..K} p_i \log \frac{p_i}{q_i}$, where p_i is the probability of the first tag being associated to cluster i and q_i is the probability of the second tag being associated to cluster i.

3.1 Tag Semantic Distance Discovery

To perform the soft clustering, our approach first extracts a set of features $\mathcal{F}$ from the underlying data V. Depending on the type of values in t, different approaches are adopted to generate the features for t. If values in t are string values, we extract a set of frequent q-grams called *sigrams* as its features. If values in t are numerical values, bins of the distribution histogram over V_t are extracted as its features. After extracting the features $\mathcal{F}$ for all values V, soft clustering [28, 8] is performed over $\mathcal{F}$. In soft clustering, the membership of a value in a cluster is probabilistic. Suppose there are K clusters $\mathcal{C} = \{c_1, \ldots, c_K\}$, for a value v, its membership probabilities in each cluster form a vector $\tau(v) = \langle p_1, \ldots, p_K \rangle$, where $\sum_{i=1}^{K} p_i = 1$, and p_i is the probability that v belongs to cluster c_i. We call $\tau(v)$ the cluster probability distribution for value v over $\mathcal{C}$. Based on [8], our soft clustering based tag inference is robust when K>100. We set K to be 200 in this paper.

Since a tag t can be associated with different values which have different cluster probability distributions on the K clusters, we can indirectly compute a cluster probability distribution for a tag t (denoted as $\tau(t)$) by taking the average cluster probability distributions of all its associated values V_t:

$$\tau(t) = \sum_{v_t \in V_t} \frac{1}{|V_t|} \cdot \tau(v_t)$$

To measure the semantic relationship between two tags, t_1 and t_2, one important factor that needs to be considered first is the semantic scope of each of these tags. Intuitively, if the semantics of a tag t is more general, the values associated with it will be more diverse. For instance, both make and model information of a car can be tagged by `car`, while only the make information can be tagged by `make`. Thus, the semantic scope $Scope(t)$ for tag t can be captured by the average KL-divergence between $\tau(v_t)$ and $\tau(t)$:

$$Scope(t) = \sum_{v_t \in V_t} \frac{1}{|V_t|} \cdot KL(\tau(v_t)||\tau(t)) \quad (1)$$

Another important factor we need to consider is the distance between tag t_1 and the *values* of tag t_2. If t_1 is similar to t_2, then $\tau(v_{t_1})$ should be close to $\tau(t_2)$. For clarity, we define the distance between the value set V_{t_1} of tag t_1 and another tag t_2 as $D_V(V_{t_1}, t_2)$:

$$D_V(V_{t_1}, t_2) = \sum_{v_{t_1} \in V_{t_1}} \frac{1}{|V_{t_1}|} \cdot KL(\tau(v_{t_1})||\tau(t_2)) \quad (2)$$

Based on the above discussion, if two tags t_1 and t_2 are similar in semantics, they should *i)* have similar semantic scopes; and *ii)* close to the value set of each other. Now we consider one extreme case: the distance between two semantically equivalent tags t_1 and t_1. We can easily find that $D_V(V_{t_1}, t_1) = Scope(t_1)$. From this we can see that if tag t_1 is more similar to t_2, the ratio of $D_V(V_{t_1}, t_2)$ over $Scope(t_1)$ will be closer to 1, and vise versa. Thus, the dissimilarity between two tags t_1 and t_2 is defined by such ratios:

$$Dis(t_1, t_2) = \frac{D_V(V_{t_1}, t_2)}{Scope(t_1)} + \frac{D_V(V_{t_2}, t_1)}{Scope(t_2)} \quad (3)$$

However, it is costly to compute such tag dissimilarity for each tag pairs based on Equation 3 since we need to enumerate all values in V_{t_1} and compute their KL-divergence to t_2 to obtain $D_V(V_{t_1}, t_2)$ and vice versa. Assuming there are n tags in T, and each tag t being associated with m values, the complexity for computing all tag-pair dissimilarity using Equation 3 is $O(C_n^2 \cdot m + nm)$.

Fortunately, we are able to use the following proposition to reduce the computational complexity.

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

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

As $\sum_{i=1}^n \pi(x_i)(x_i - \mu) = 0$, and 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. □

Intuitively, our proposition simply states that the expected KL-divergence between a random variable $\mathcal{X}$ following the probability distribution π and a point y is equivalent to the sum of the KL-divergence between $\mathcal{X}$ and its means μ plus the KL-divergence between μ and y. By this proposition and [3], we can show that the distance between the set of values associated with tag t_1 (i.e. V_{t_1}) and the tag t_2 can in fact be computed simply by using $Scope(t_1)$ and the KL-divergence between $\tau(t_1)$ and $\tau(t_2)$.

$$\begin{aligned}
D_V(V_{t_1}, t_2) &= \sum_{v_{t_1} \in V_{t_1}} \frac{1}{|V_{t_1}|} \cdot KL(\tau(v_{t_1}) || \tau(t_2)) \\
&= \sum_{v_{t_1} \in V_{t_1}} \frac{1}{|V_{t_1}|} \cdot KL(\tau(v_{t_1}) || \tau(t_1)) \\
&\quad + KL(\tau(t_1) \| \tau(t_2)) \\
&= Scope(t_1) + KL(\tau(t_1) \| \tau(t_2)) \qquad (4)
\end{aligned}$$

By applying Equation 4 over Equation 3, we can now define our semantic distance between t_1 and t_2 as:

$$D(t_1, t_2) = \frac{KL(\tau(t_1) \| \tau(t_2))}{Scope(t_1)} + \frac{KL(\tau(t_2) \| \tau(t_1))}{Scope(t_2)} \qquad (5)$$

$D(t_1, t_2)$ is symmetric, however, it does not satisfy triangle inequality. This coincides with the intuition that two tags cannot be inferred as similar via a third tag. For example, we cannot say `model` and `make` are similar although both are semantically similar to `car`.

Note that unlike Equation 3, the computation of the new distance function only involves the semantic scope, and pairwise KL-divergence between tags. The computational complexity for all tag pair distances is thus reduced from $O(C_n^2 \cdot m + nm)$ to $O(C_n^2 + nm)$.

3.2 Tag Inference

With the tag distance defined above, now we give our semantic similarity between two tags by taking the power of the negative distance, i.e.

$$SemSim(t_1, t_2) = e^{-\alpha \cdot D(t_1, t_2)} \qquad (6)$$

[6]Note that KL-divergence is a special instance of Bregman divergence [32] and thus the derivation here immediately applies to KL-divergence.

Table 2: TagTable for `make`

TID	LID	Value	Probability
1	1	Ford	1.0
2	1	BMW	1.0
3	1	Ford	0.95
4	1	Honda	0.95
5	1	BMW 525i	0.5
6	1	Toyota Camry	0.5

where α is a parameter which controls the power of inference. When α is set to be 0, every value will be associated with every tag; when α is set to be infinity, there will be no inference among tags. In the experiments, we set α at 0.5. An example tag similarity table is shown in Figure 2.

Suppose value v is originally associated with tag t in user contributed data, then the associated probability of v with respect to other tags $t_i \in T$ is defined as follows.

$$\zeta_{v,t_i} = \zeta_{v,t} \cdot Semsim(t, t_i) = Semsim(t, t_i) \qquad (7)$$

associated probability $\zeta_{v,t} = 1$ as v is originally associated with t. After applying the above tag inference equation, each value in the database will be associated with a set of semantically similar tags with proper probabilities. One example probabilistic tagged data is shown in Figure 3.

4. TOP-k QUERY PROCESSING

Next we will look at our top-k query processing approaches.

4.1 Data Organization

To facilitate top-k query processing, we partition the probabilistic tagged table according to tags. For a specific tag t, all its associated values are stored into one relational table, and this table is named by the tag name, i.e. t. We call such a table **TagTable**. Table 2 shows the TagTable for `make`.

Each TagTable has four attributes: `TID` (tuple id), `LID` (location offset in the source tuple), `Value` (data value) and `Probability` (associated probability). Note that the majority of ζ is approximately 0 and only values with associated probability above a cutoff threshold are stored. On the other hand, value v may be associated with multiple tags, so there is a copy of v in each of its associated TagTable. For each TagTable, a B^+-tree index is built over the attributes $\langle Value, Probability\rangle$, which will be used for the sorted list retrieval in top-k query processing.

4.2 Dynamic Instantiation

Given a query q and record r, let T_q be the set of tags in q, and V_r be the set of values from r. We can build a weighted bipartite graph $G = (U, V, E)$ as follows. Let $U = V_r$, $V = T_q$, and for each $v_r \in V_r$, each $t_q \in T_q$, if v_r satisfies the value constraints over t_q and ζ_{v_r,t_q} meets the tag expansion condition, we add an edge $e\langle v_r, t_q\rangle$ to E, with weight $\ln \zeta_{v_r,t_q}$. For simplicity, we denote this bipartite graph as $G = (V_r, T_q, \ln \zeta_{V_r,T_q})$.

After taking logarithm over the associated probabilities ζ_{V_r,T_q}, it is easy for us to apply Hungarian algorithm [18] on $G = (V_r, T_q, \ln \zeta_{V_r,T_q})$ to find the best one to one matching between V_r and T_q, where the total sum of cost/weight is maximized. We denote the maximum score found by Hungarian algorithm for record r as $Score_H(r)$. Based on Definition 3, we can see that the mapping found by Hungarian

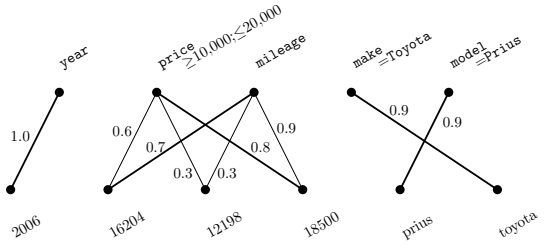


Figure 5: Dynamic Instantiation Illustration

algorithm is exactly the dynamic instantiation. The first and third statements in Definition 3 are met by our edge insertion constraints discussed above; one to one mapping guarantees the second statement; and the last statement is fulfilled as the solution found by Hungarian algorithm is optimal. From the above discussion, it can be easily proven that

$$Score(r) = e^{Score_H(r)}$$

Take the following extended SQL query as an example. The values associated with tags `year`, `price` and `mileage` are to be retrieved, with constraints issued over tags `make`, `model` and `price`.

```
SELECT    year@0.9, price@0.3, mileage
FROM      CAR
WHERE     make@0.8 = 'Toyota'
      AND model = 'Prius'
      AND price ≥ 10,000
      AND price ≤ 20,000
```

The corresponding bipartite graph is shown in Figure 5. For clarity, edge weights in the figure are original associated probabilities without logarithm. As illustrated by Figure 5, if `mileage` is instantiated with `16204` and `price` is instantiated with `18500` as the darker lines indicate, it gives the best instantiation as the multiplication of the associated probabilities is maximized.

4.3 Top-k Query Answering

Next we will discuss the retrieval of top-k answers. For ease of illustration, we refer to tags in the `SELECT` clause as **QTags**, and tags in the `WHERE` clause as **VTags**. Tag set in query q is denoted as T_q.

4.3.1 Sorted List Retrieval

For each queried tag $t_q \in T_q$, we aim to retrieve a list of valid values from its TagTable, sorted over associated probability from high to low. For a tag that appears in both VTags and QTags, we merge its constraints from `SELECT` and `WHERE` clauses, and retrieve one sorted list.

We first study the situation where a user-specified semantic confidence δ is issued over t_q, i.e. $t_q@\delta$. Based on Definition 2, t_q will be expanded to a tag set $T_e = \{t_{e_i} | Semsim(t_q, t_{e_i}) \geq \delta\}$. Next we will look at the associated probabilities with t_q for valid and invalid values, respectively. **Valid case**: Given a valid value v, suppose it is originally associated with tag $t_{e_i} \in T_e$. The associated probability between v and t_q is $\zeta_{v,t_q} = Semsim(t_q, t_{e_i}) \geq \delta$. This means that for each valid value, its associated probability in TagTable t_q is not less than δ. **Invalid case**: Given an invalid value v', we know that its original tag falls outside of T_e. Assume its original tag is $t_{v'}$. From Definition 2, we know that $Semsim(t_q, t_{v'}) < \delta$. Therefore, the associated probability between v' and t_q, i.e. $\zeta_{v',t_q} = Semsim(t_q, t_{v'}) < \delta$. In other words, for each invalid value, its associated probability in TagTable t_q is less than δ. In conclusion, associated probability with valid value is not less than δ, while with invalid value is. Hence, the sorted list for $t_q@\delta$ can be retrieved by the following SQL statement issued over the TagTable for t_q:

```
SELECT * FROM t_q WHERE Probability ≥ δ
                  ORDER BY Probability DESC;
```

We then look at the complex situation where there is value constraint over t_q besides the tag expansion, i.e. $t_q@\delta$ `op` v_q. Here `op` is a comparison operator and v_q is a specific value. Its sorted list can be resolved by adding one `WHERE` condition "`Value op` v_q" into the above SQL statement. For the simple cases where there is no tag expansion or value constraint over t_q, their sorted lists can be easily retrieved by removing the corresponding `WHERE` condition from the SQL statement. Benefiting from the B^+-tree index built over attributes $\langle Value, Probability \rangle$ for each TagTable, most sorted lists can be done by an index-only scan.

After retrieving the sorted list for each tag t_q, Threshold Algorithm (TA) [13] can be applied to fetch the top-k results, using Hungarian algorithm to find the best instantiation. We proposed two TA-based approaches for top-k query processing: **Eager** and **Lazy**.

4.3.2 Eager Top-k Query Answering

In eager query answering, for each tag t_q in query q, its sorted list is retrieved and maintained. TA [13] is then performed over all the retrieved sorted lists. The main idea is shown in Algorithm 1.

A priority queue RS is maintained to store the best k records that have been processed so far. All the scanned records are inserted into S_r to avoid accessing the same record more times. Each time, we scan the top unseen records from all the sorted lists, and pop the record r with the highest logarithmic probability (MP) from its sorted list (line 14). If there are more than one sorted lists whose top probability is equal to MP, our algorithm randomly chooses one from them. The popped record r is then retrieved and Hungarian algorithm is applied to get its score $Score_H(r)$ (line 15). Finally, priority queue RS is updated accordingly (line 16-19), and r is inserted into the scanned record set S_r (line 20).

For all the unseen records, their scores are upper bounded by the summation of the top probabilities from each sorted list, i.e. UBS. This is because our sorted list is ranked over probability, so the probability for each unseen record in SL_{t_q} is not greater than $SL_{t_q}.top.prob$. Algorithm 1 can be terminated when the k^{th} score in the result set RS is not less than the upper bound score UBS (line 23), or any sorted list is exhausted (line 21).

Eager query processing searches through the sorted lists for all queried tags, including all QTags and VTags. However, this is not efficient enough. Generally, the sorted lists for QTags are much longer than those for VTags, because VTags have value constraints in the `WHERE` clause, while most QTags do not, especially for those that only exist in QTags. Due to the low selectivity of QTags, their sorted lists usually

Algorithm 1: Eager top-k Query Processing

Input : A query q in extended SQL
A positive integer k
Output: Result set RS

```
1  T_q ← tags in query q;
2  RS ← ∅; // top-k result set
3  S_r ← ∅; // scanned records
4  foreach tag t_q ∈ T_q do
5      SL_{t_q} ← sorted list of t_q;
6  Let SL_{t_q}.top be the most top unseen item in SL_{t_q};
7  Let SL_{t_q}.top.prob be the probability of item SL_{t_q}.top;
8  while true do
9      UBS ← 0; // upper bound score for unseen records
10     MP ← −∞; // maximum in all SL_{t_q}.top.prob
11     foreach tag t_q ∈ T_q do
12         UBS ← UBS + SL_{t_q}.top.prob;
13         MP ← max(MP, SL_{t_q}.top.prob);
14     Pop record r from SL_t where SL_t.top.prob = MP;
15     Let score = Score_H(r); // computed by Hungarian
16     if |RS| < k or score > the k^th score in RS then
17         if |RS| = k then
18             pop one item from RS;
19         insert (score, r) into RS;
20     insert record r into S_r;
21     if SL_t has reached its end then
22         break; // while loop is terminated
23     if |RS| = k and UBS ≤ the k^th score in RS then
24         break; // while loop is terminated
25 return RS;
```

contain lots of invalid items. Based on this, we proposed another approach, namely lazy top-k query answering, which only focuses on the sorted lists of VTags.

4.3.3 Lazy Top-k Query Answering

Unlike the eager approach, our lazy approach only retrieves and maintains sorted lists for VTags in query q. For a record, any values that are not related to Vtags are only accessed when computing its Hungarian score (line 15). For queried tags falling outside of Vtags, we assume their associated probabilities to be 1 (the largest associated probability) when computing the upper bound score (UBS). Compared to the eager approach, our lazy approach does not need to probe the long sorted lists which include many invalid records, making it more I/O efficient.

In both eager and lazy approaches, the score for each unseen record is upper bounded by the summation of all the top item scores. Hence, the correctness of our algorithms can be guaranteed.

5. EXPERIMENTAL STUDIES

We now present the experiments to validate the performance of our system. We have two main goals in this section: 1) to examine the effectiveness of our tag inference approach, 2) to show that our top-k query processing is capable of retrieving top-k results with high precision and low cost.

5.1 Experimental Setup

We implement our system on top of MySQL. The tag inference and query answering algorithms are written in C++. The system is set up on a PC with Intel(R) Core(TM)2 Duo CPU at speed 2.33GHz and 3.5GB memory. Through all the experiments, we randomly select 2% samples for soft clustering.

1. Car Datasets (CAR Domain). The first datasets we deal with are datasets in the car domain. We insert 101 different tables from different sources into a wide table to simulate the scenario where different users share their data. Among the 101 tables, one is obtained from Google Base, and has 24 thousand tuples and 15 columns, while the others are downloaded from *Many Eyes*[7]. These tables have different sizes, with number of tuples ranging from dozens to thousands, and the number of columns ranging from 5 to 31. In total, there are 94,854 rows, 284 distinct tags and 850,215 distinct values after combining the data into one wide table.

2. Directory Datasets (DIR Domain). We extract three directory datasets from three websites that provide directory services[8]. The directory data sets consist of information about companies, such as company name, address, telephone, fax, email and so on. In total, there are 139,022 tuples, 31 distinct tags and 495,908 distinct values.

5.2 Tag Inference Validation

We verify the effectiveness of our tag inference approach first on real datasets, and compare it with the matching results from Open II. We then further validate it over the semi-real Google Base car datasets.

5.2.1 Validation over Real Datasets

To evaluate our tag inference approach, we first compare it with Open II [26] based on the *golden standard*. Open II is an open source data integration toolkit that has implemented several schema matching methods[9]. In this experiment, for Open II, we choose all the matching methods it implemented to produce its matching result. These matchers include "Name Similarity Matcher", "Documentation Matcher", "Mapping Matcher", "Exact Matcher", "Quick Matcher" and "WordNet Matcher". The *golden standard* is identified by manually scanning through all the columns in each domain.

In car domain, totally there are 160 semantically equivalent tag pairs in the golden standard. Both the output format of Open II and our system are a list of tag pairs, each associated with a similarity score. For car domain, Figure 6 shows the precision for the top-k similar tag pairs in Open II and our approach, with k varying from 20 to 200. The result clearly demonstrates that our approach achieves better precision than Open II when we are looking at the same number of top-k similar tag pairs. The reason is that we define the tag semantic similarity based on soft clustering and

[7]http://manyeyes.alphaworks.ibm.com/manyeyes/

[8]They are thegreenbook.com, streetdirectory.com and yellowpages.com.sg respectively.

[9]Note that Open II is a schema level mapper while our approach is an instance-based approach [23]. As such, Open II takes only one second for all its attribute mappings while our approach takes on average 2.5 hours to perform high dimensional clustering. These operations are however preprocessing which do not affect the query answering efficiency of our approach.

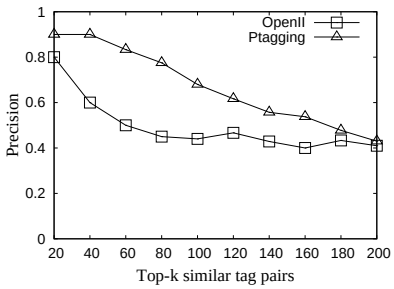


Figure 6: Comparison with Open II

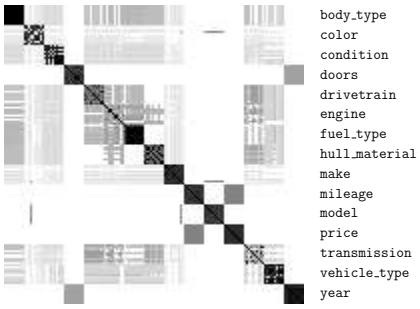


Figure 7: Tag Similarity Bitmap

KL-divergence, rather than directly over attributes. Hence, we are able to discover similar tags that have different syntaxes, such as `make` and `manufacturer`, while they cannot be found by Open II. At the same time, Open II makes many wrong predictions on tag pairs which are similar in names but different in semantics. Examples include `fuel_type` and `body_type`, `model` and `model_year` etc.

Figure 6 also indicates that our precision decreases when k increases. This is because there are some semantically different tags whose value distributions are so similar that our approach cannot tell them apart, such tags include `price` and `mileage` in car domain, `telephone` and `fax` in directory domain. For such tag pairs, it might not even be possible for human being to distinguish them if no extra information is provided. The result in directory domain is similar, due to limited space, we do not list it here.

5.2.2 Validation over Semi-real Datasets

We further validate our tag inference by artificially introducing semantically similar tags. The large Google Base, which contains 15 different columns and about 2 million records, is pre-processed as follows[10]. For each attribute, we create 9 different attributes from the original one, with shared prefix. For example, we create another 9 attributes `make_a`, `make_b`, ..., `make_i` from attribute `make`. These attribute names are then used to simulate the scenario where different tags with same semantics come from heterogeneous data sources.

In addition, with the observation that there are many small datasets but few large ones, we take this into account in our data generation. The 10 semantically similar tags are divided into four groups with different probabilities. The first group contains 4 attributes, `make`, `make_a`, `make_b` and `make_c`, each with probability 5%; the second group contains 3 attributes, `make_d`, `make_e` and `make_f`, each with probability 10%; the third group has only two attributes, `make_g` and `make_h`, each having probability 15%; the last group, with the sole attribute `make_i`, is assigned with probability 20%. We then assign every value to a tag according to this probability distribution and with an associated probability of 1. In the experiment, we consider each attribute as a tag, and run our soft clustering 3 times with different initializations on a 2% sample (∼40 thousand data values).

Figure 7 shows the pairwise similarity between tags. Each row and column represents a tag. The similarity between the row and column tag at a given pixel is related to the darkness of the pixel. The darker the square is, more similar the tag pairs are. Note that the tags are arranged in the same order in both row and column orientations, and tags generated from the same original tags are grouped together. From Figure 7 we can see that the squares on the diagonal are much darker than the other squares. This coincides with the fact that tags generated from the same attribute are much more similar with each other than the other tags. However, the only outlier is the fourth last attribute `price` and the sixth last attribute `mileage`. Their diagonal intersections only give a moderate grey scale. This is because `mileage` and `price` share very similar value distributions, as explained earlier.

[10]We used the entire Google Base data in data generation, but only 10% samples in tag inference to avoid dominance.

5.3 Top-k Query Processing Validation

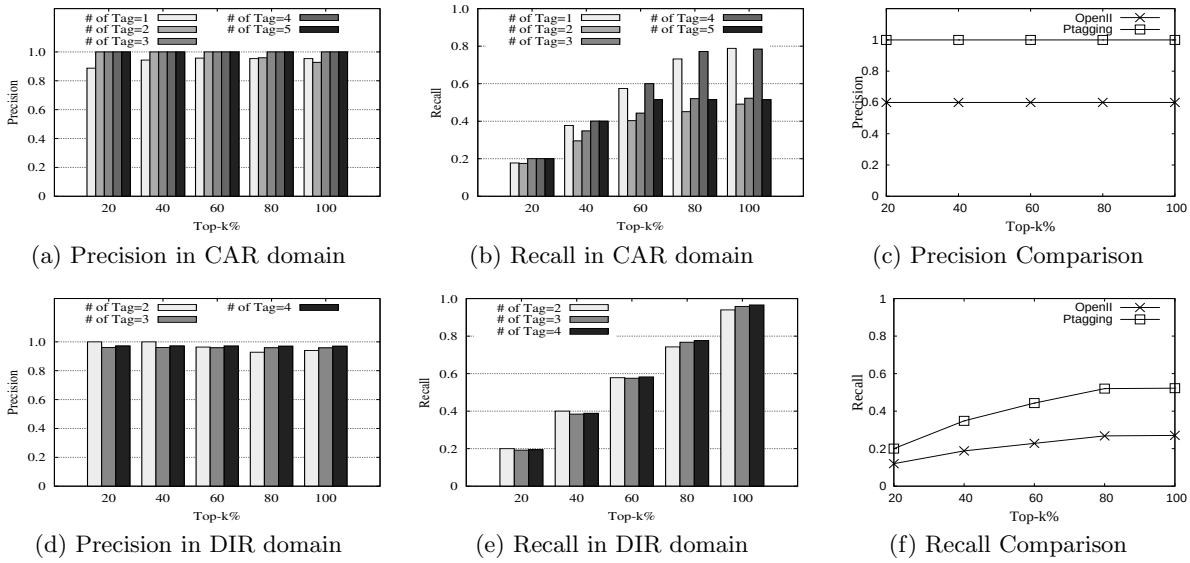
We evaluate the performance of our top-k query processing over both car domain datasets and directory domain datasets. For car domain data, we vary the number of distinct tags in a query from 1 to 5, and denote them as $T1$ to $T5$. For each $Ti, i \in \{1, \ldots, 5\}$, we choose 5 different queries with approximately equal number of tags in `SELECT` clause and `WHERE` clause. When selecting queries, we randomly vary the selectivity for predicates in the `WHERE` clause. The process is similar for directory domain data, except that we vary the distinct number of tags from 2 to 4, as directory domain has less number of tags.

To get the precision and recall for query result, we build the *golden standard* as follows. We manually check the extended SQL query to determine its semantics, then translate it to a set of traditional SQL queries over each source data set, and finally merge the results retrieved by SQL queries. For instance, if tags in {`make`, `manufacturer`, `g_make`} are semantically equivalent to each other, then an extended SQL query containing `make` will be broadcasted to each data source that includes any equivalent tag of `make`. On average, a tag-based query can be translated to 4 to 6 SQL queries.

Based on golden standard, we define *precision* and *recall* as follows: let P be the set of answers retrieved by the query processor, and GS be the set of answers in *golden standard*, then $Precision = \frac{|P \cap GS|}{|P|}$, and $Recall = \frac{|P \cap GS|}{|GS|}$.

5.3.1 Effectiveness

Figure 8(a) shows the precision over car datasets, with top-k% varying from 20% to 100%. For each k%, it also illustrates the precision for each Ti, with i (i.e., the number of distinct tags) ranging from 1 to 5. Note that our top-k% query is a little different from the traditional top-k query. In top-k query, the number of returned answers is k. How-



(a) Precision in CAR domain (b) Recall in CAR domain (c) Precision Comparison

(d) Precision in DIR domain (e) Recall in DIR domain (f) Recall Comparison

Figure 8: Precision and Recall by Varying k%

ever, in top-k% query it is $|GS| * k$%. The reason we choose top-k% query is that we would like to see how recall changes with different number of distinct tags. As golden standard size for different queries may differ a lot, setting the same absolute k for all queries will make recall bias the performance towards queries with small answer set. Top-k% query avoids this problem since no matter how much the golden standard differs, recall for top-k% query is upper bounded by k%.

Figure 8(a) shows that our approach achieves high precision in all the cases, and is not sensitive to k%. It also indicates that for $T1$, precision increases slightly with larger k% while $T2$ has a slight decrement. The reason is that our tag inference approach is unable to distinguish tag pairs with similar value distributions (e.g. `price` and `mileage`). Such false positives are ranked higher in $T1$ than in $T2$. However, this problem can be eliminated by our dynamic instantiation when `price` and `mileage` are queried together, as our query processor always attempts to find the best instantiation.

Figure 8(b) shows the corresponding recall for car domain datasets. We can see that when k% increases, there is a steady increase for recall. As mentioned before, recall in k% case is upper bounded by k%. However, since we missed some true similar tag pairs in tag inference, we cannot grantee 100% recall. Such missed tags are those which are semantically similar but have heterogeneous values. For example, it is difficult to associate `year` which has format "yyyy" with `model_year` which has format "yy". This problem will become severer if there are more tags in query, because the possibility of including missed tags also increases, as shown in case $T5$. However, for queries containing less distinct tags, our approach is still able to achieve good recall. Consider the first 4 cases $T1$ to $T4$, at the point where k% equals to 100%, our approach can achieve 0.7 recall in average. Together with the high precision we provide, our proposed approach is quite effective.

We also get precision and recall on the directory data, shown in Figure 8(d) and Figure 8(e) respectively. As there are less number of tags than the car dataset, we vary its distinct number of tags from 2 to 4. We observe similar behavior as over car datasets. Since the number of tags in the directory datasets is less, and most are well-formatted, its recall is much better than the car datasets.

Finally, we compare our work with Open II for the query processing. Comparisons over precision and recall are shown in Figure 8(c) and Figure 8(f), respectively. In particular, we only list the results for $T3$ case over car datasets. The reason is that Open II makes many wrong inferences, as such, precision in most cases is really low. Here we choose its best case, i.e. $T3$, for comparison. It is indicated that our approach provides better precision and recall than Open II. Specifically, among the 5 queries in $T3$, Open II produces 0% precision for two of them because they contain wrong matched tags, and 100% precision for the rest three. In average, Open II has 60% precision across all k% cases. With less correct records returned, Open II has lower recall than ours.

5.3.2 *Efficiency*

We validate the efficiency of our top-k query processing over $T3$ on car datasets. For the directory data sets and the other queries, the overall trend is similar.

We first compare the running time of our approaches with the translated SQL queries, as shown in Figure 9. Note that when k% is set to 100%, our approaches retrieve almost the same number of tuples as the translated SQLs. As illustrated, the running time of the translated SQL queries is quite stable with respect to different top-k%, because they always retrieve the golden standard (i.e. all the true answers), no matter how k% changes. In contrast, the running time of our approaches, i.e., lazy retrieval and eager retrieval, increases linearly when top-k% increases. However, eager retrieval takes longer time than lazy retrieval by almost 1.5 times since eager retrieval maintains and probes the sorted lists for QTags, but most tuples in them are invalid. The running time that lazy retrieval takes is almost k% of translated SQLs, which clearly shows the efficiency of our lazy retrieval approach.

We next look at how the number of random and sequential

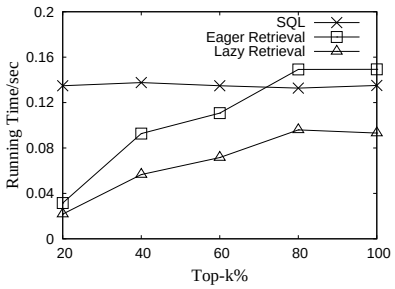
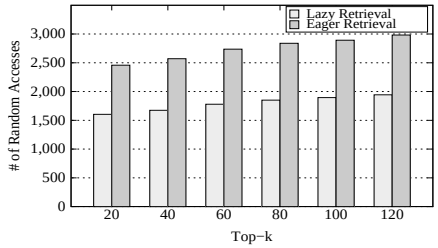
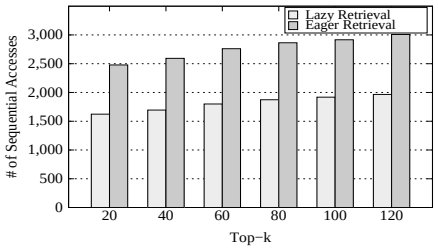


Figure 9: Running Time by Varying k%

accesses vary as we vary k. Figure 10(a) shows the number of random accesses for eager and lazy retrieval when varying k from 20 to 120. The corresponding graphs for sequential I/Os are in Figure 10(b). Note that in both figures, we adopt top-k rather than top-k%, as the number of tuples returned by top-k% varies proportionally to query's answer set. Consistent with earlier observation that eager retrieval takes almost 1.5 times longer than lazy retrieval, the same conclusion holds for both the number of sequential accesses and random accesses.



(a) Random I/O Comparison



(b) Sequential I/O Comparison

Figure 10: Random I/O and Sequential I/O

6. RELATED WORK

Query Relaxation In the recent decade, there are several efforts to provide users a more friendly query interface that requires minimal or no user knowledge of the database schema. Such effort includes the extensive studies about keyword searches over RDBMS [19, 31]. However, keyword query confines users from issuing structural constraints, and it is not applicable for range queries. Moreover, it does not allow users to express semantic confidence over tags. In [15, 21], it was illustrated that tags on multimedia objects are prevalent over Web 2.0 application [1, 2], to facilitate searching over multimedia objects. Our use of tags here serves the same purpose.

Database Relaxation. There is also some effort to relax the strict schema defined in RDBMS, including researches on malleable schemas [11, 33], Bigtable [6], Wide-table [7], WebTable [5], and RDF tables [22]. These approaches provide some flexibility for users to interact with the schema without the strict constraints of a RDBMS. However, most of them only provide a flexible interface to store data, but the columns or attributes in their system play the same role as in RDBMS. While [33] did look at issues involving query relaxation, they only do so for string attributes and have to generate many alternate relaxed queries in order to retrieve relevant data. Users are also not given the control over which attributes to relax.

Probabilistic Database. There also exist a large body of work over probabilistic databases, dealing with row existence and value existence uncertainty [14, 27], or lineages [24, 29]. However, the uncertainties they aim to handle are different with ours. Our uncertainties on one hand, come from the semantic heterogeneities of the attributes and on the other hand, also come from the semantic uncertainties of user's queries. In other words, our uncertainties are in schemas. As far as we know, none of the existing approaches over probabilistic databases have considered schema uncertainties.

Data Integration and Schema Matching. Data integration aims to create a unified mediated schema. One main drawback of most data integration approaches [23] is that when the number of data sources becomes large, the global mediated schema will be too huge for users to query. Our work discovers semantic relationships among different tags so that users can choose preferred tag/attribute to query the database, rather than over the mediated schema. Among them, probabilistic data integration proposed by Dong and Halevy [12, 25] is closest to our work. However, their mutual exclusion rule is issued prior to the query answering, and our query-time mutual exclusion rule cannot be trivially applied to them. Consequently, they are not able to dynamically determine the tag semantics as ours. Moreover, we do not need to enumerate two levels of probability (i.e. mediated schema level and schema mapping level) in query answering as they do, so our work is more efficient.

Our tag inference approach also differs from the existing instance-based schema matching approaches [20, 10, 4]. They assume the existence of a database with a "correct" schema which data from other sources will be mapped to. Data from this database is used as training data in supervised learning (i.e. classification) to map attributes from other sources to the "correct" schema. Our problem in this paper however assumes no existence of a "correct" schema. Instead we need to derive the semantic similarity for many different sets of tags provided by different users. As such, we adopt unsupervised learning (i.e. clustering) as a way to find the similarity between tags. Our emphasis is on not confining the users to a single mediated schema and allowing them to query the data freely with different tags. The clustering-based tag inference method is essential towards this aim.

7. CONCLUSION

We proposed a framework of tagging data values with probabilities on top of the wide table. Under this framework, we introduced our model of probabilistic tagging, ap-

proach of performing tag inference for all values based on their relative entropy, and the efficient and effective dynamic instantiation for query processing. We believe our approach is able to grant more privileges to non-expert database users and to better maintain the database quality.

8. REFERENCES

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

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

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

[4] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, pages 452–466, 2002.

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

[6] 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. In *OSDI*, pages 205–218, 2006.

[7] E. Chu, J. L. Beckmann, and J. F. Naughton. The case for a wide-table approach to manage sparse relational data sets. In *SIGMOD Conference*, pages 821–832, 2007.

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

[9] 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, 2008.

[10] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, pages 509–520, 2001.

[11] X. Dong and A. Y. Halevy. Malleable schemas: A preliminary report. In *WebDB*, pages 139–144, 2005.

[12] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

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

[14] 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.

[15] S. A. Golder and B. A. Huberman. The structure of collaborative tagging systems. *CoRR*, 2005.

[16] H. Gonzalez, A. Halevy, C. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen. Google fusion tables: data management, integration and collaboration in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 175–180. ACM, 2010.

[17] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1):429–438, 2010.

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

[19] 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 *SIGMOD Conference*, pages 903–914, 2008.

[20] W.-S. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In *VLDB*, pages 1–12, 1994.

[21] 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.

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

[23] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

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

[25] A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD Conference*, pages 861–874, 2008.

[26] L. Seligman, P. Mork, A. Halevy, K. Smith, M. J. Carey, K. Chen, C. Wolf, J. Madhavan, A. Kannan, and D. Burdick. Openii: an open source information integration toolkit. In *SIGMOD '10*, pages 1057–1060, 2010.

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

[28] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *CoRR*, 2000.

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

[30] D. Xin and H. Alon. Indexing dataspaces. In *ACM SIGMOD*, pages 43–54, 2007.

[31] B. Yu, G. Li, K. R. Sollins, and A. K. H. Tung. Effective keyword-based selection of relational databases. In *SIGMOD Conference*, pages 139–150, 2007.

[32] Z. Zhang, B. Ooi, S. Parthasarathy, and A. Tung. Similarity search on bregman divergence: Towards non-metric indexing. *Proceedings of the VLDB Endowment*, 2(1):13–24, 2009.

[33] X. Zhou, J. Gaugaz, W.-T. Balke, and W. Nejdl. Query relaxation using malleable schemas. In *SIGMOD Conference*, pages 545–556, 2007.

APPENDIX

A. PROBABILISTIC DATA INTEGRATION

Score computation for tuple T3 and T4 over query Q4 using approaches from [25] is discussed below. As assumed in [25] that different data sources are independent, now we only consider the data source that T3 and T4 come from (i.e. Figure 1(b)) and denote it as S. For clarity, We denote `manufacturer` in S as B_1, `car` as B_2, and $p(M)$ is used to represent the probability of M. The attribute set in mediated schema is {`manufacturer`, `car`, `model`}. Due to the semantic heterogeneity of `car` (semantics of `manufacturer` and `model` are specific), probabilistic mediated schema $\mathbf{M}$ has two possible schemas, i.e. $\mathbf{M} = \{M_1, M_2\}$. Specifically, attributes in M_1 are $A_1 = \{manufacturer, car\}$ and $A_2 = \{model\}$, and attributes in M_2 are $A_3 = \{manufacturer\}$ and $A_4 = \{car, model\}$. According to [25], $p(M_1) + p(M_2) = 1$, and $p(M_1) < p(M_2)$ because `manufacturer` and `car` co-occur in S. Thus, $p(M_1) < 0.5$ and $p(M_2) > 0.5$.

Next we consider probabilistic schema mappings. As the actual semantics of B_1 is manufacturer, and B_2 is model, we firstly drop the wrong correspondences such as (B_1, A_2), (B_2, A_1), (B_1, A_4) and (B_2, A_3). Based on the tag similarities from Figure 2, the attribute correspondences between S and M_1 are $p(B_1, A_1) = (1 + 0.52)/(1 + 0.52) = 1$ and $p(B_2, A_2) = 0.65/(1 + 0.52) = 0.43$; and correspondences between S and M_2 are $p(B_1, A_3) = 1/(1 + 0.65) = 0.61$ and $p(B_2, A_4) = (1+0.65)/(1+0.65) = 1$. According to [25], we list all the possible schema mappings as follows, each followed by the probability.

Schema mappings between S and M_1 are:
`m1:` (B_1, A_1), (B_2, A_2) : `0.43`
`m2:` (B_1, A_1) : `0.57`

Schema mappings between S and M_2 are:
`m3:` (B_1, A_3), (B_2, A_4) : `0.61`
`m4:` (B_2, A_4) : `0.39`

Scores for the result tuples produced by [25] are as follows.
$p(\text{Focus 1.6A}, \text{Focus 1.6A}) = (0.39 + 0.61) * p(M_2) = p(M_2) > 0.5$
$p(\text{Accord}, \text{Accord}) = (0.39 + 0.61) * p(M_2) = p(M_2) > 0.5$
$p(\text{Ford}, \text{Focus 1.6A}) = 0.43 * p(M_1) < 0.215$
$p(\text{Honda}, \text{Accord}) = 0.43 * p(M_1) < 0.215$