Facet Discovery for Structured Web Search: A Query-log Mining Approach

Jeffrey Pound*
University of Waterloo
Waterloo, Canada
jpound@cs.uwaterloo.ca

Stelios Paparizos
Microsoft Research
Mountain View, CA, USA
steliosp@microsoft.com

Panayiotis Tsaparas
Microsoft Research
Mountain View, CA, USA
panats@microsoft.com

ABSTRACT

In recent years, there has been a strong trend of incorporating results from structured data sources into keyword-based web search systems such as Bing or Amazon. When presenting structured data, facets are a powerful tool for navigating, refining, and grouping the results. For a given structured data source, a fundamental problem in supporting faceted search is finding an ordered selection of attributes and values that will populate the facets. This creates two sets of challenges. First, because of the limited screen real-estate, it is important that the top facets best match the anticipated user intent. Second, the huge scale of available data to such engines demands an automated unsupervised solution.

In this paper, we model the user faceted-search behavior using the intersection of web query-logs with existing structured data. Since web queries are formulated as free-text queries, a challenge in our approach is the inherent ambiguity in mapping keywords to the different possible attributes of a given entity type. We present an automated solution that elicits user preferences on attributes and values, employing different disambiguation techniques ranging from simple keyword matching, to more sophisticated probabilistic models. We demonstrate experimentally the scalability of our solution by running it on over a thousand categories of diverse entity types and measure the facet quality with a real-user study.

Categories and Subject Descriptors: H.2.8 [Database Management]: Data Mining

General Terms: Algorithms, Experimentation, Human Factors.

Keywords: Facets, Faceted search, Structured web-search.

1. INTRODUCTION

In recent years, search engines like Google or Bing have evolved to include in their results information from structured data sources along with text documents. Furthermore, there are specialized search verticals like Amazon that rely on an even greater variety of structured data sources for their results. Structured data have the advantage of having rich meta-data, allowing for an improved search experience compared to just keyword search.

Faceted search is a common interaction paradigm that utilizes structured information to provide a user-friendly alternative to keyword search. In this paradigm, facets are used as summarized data axes that allow the users to pivot on the information shown and create a different or more selective result set. If we abstract the structured data as typed entities grouped together into logical tables, one per entity type. A table *facet* corresponds to a table attribute and underlying attribute values. Faceted search is enabled via a *facet system* which exposes an ordered list of attributes and an ordered list of attribute values.

For example, for a table of digital cameras, a facet could be a popular attribute such as *brand* and prominent values such as '*Canon*', '*Nikon*', and '*Sony*'. Other facets could be *megapixel resolution*, *color*, *size* and corresponding values per attribute. Exposing them into a facet system, allows the user to quickly zoom into the appropriate camera subset by selecting characteristics of interest. Similar examples exist in other domains with structured data, such as automobiles or movies.

It is clear that the user experience for structured web-search can benefit from facets. However for an engine to support facets effectively it needs to address two groups of challenges: (a) Given the limited screen real estate and the large number of possible facets to consider, we need to select the top-k most important facets, where k is usually a small number. Facet importance in this case can be measured by the *utility* of a facet towards a user's anticipated action like a pivot or refinement. Since an entity-type can have over one hundred attributes, the challenge becomes finding the few most important attributes, and attribute values with the maximum anticipated utility. (b) There is a huge number of structured data sources currently available to engines like Google or Amazon. If we abstract the data as de-normalized entity-type tables, there are thousands of such tables to consider. So a solution that finds facets from these tables must be fully automated to scale appropriately.

The best method to learn the utility of attributes for users would be to consume user-produced signals. Such an ideal signal would be produced if all possible attributes for all possible entity-type tables are shown and then the system observes and records which ones are being selected by an extensive user study. Unfortunately, this is not feasible due to the scale of data and the number of required users for statistical significance. Instead, common practice [8, 10] has been to show attributes selected manually by experts with values populated from sampling the results. It is debatable how well a domain expert can match the average user intent, but regardless, the scale of structured data available to web engines makes it impossible to go this route with consistently good quality across all tables. As a result, existing approaches [3, 22] rely on automatically learning

*Work done while at Microsoft Research.

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

facets from the structured data. However, such approaches do not capture the true user intent since they focus on the structured data without incorporating user signals.

In this work we use web query logs intersected with the structured data to model the facet utility. Our intuition is that query logs act as a proxy for the exploration process of the users when navigating through a structured domain. We assume that users issue queries to refine or alter their results into an appropriate consideration set in a similar manner they would have used facets if there were available. The attributes that appear frequently in the query logs are the ones that users would use to narrow down their search, and thus they are appropriate for facets. The values of these attributes that appear frequently in the queries are good values to populate the facets. This formulation has the advantage that it can lead to a scalable automated solution that captures the real user needs better than the opinion of an expert, or the just the statistics of the structured data. Additionally, it adapts to the needs and preferences of the users as they change over time. It is worth mentioning that trying to match the user intent for attribute importance has been studied in the past, see [7, 21], with the application domain being either attribute ranking or product visibility. Although the high-level principle is similar, the underlying models and corresponding solutions are different from those for discovering the facet utility.

Using the web query logs to find frequently queried values and attributes would be easy if the attribute-value pairs were clearly specified in the queries as in SQL statements. However, although the data is structured, web queries are free-text keywords. Finding the frequent attributes poses two hard challenges due to the inherent ambiguity of translating the keywords to attribute-value pairs. (a) Mapping the query to the appropriate structured source – e.g., map the keyword '*apple*' to either mp3 players, computers, or fruit; and (b) identifying and disambiguating the occurrence of attributes in the query amongst attributes of the same entity-type – e.g., for the table televisions, disambiguate the keyword '*30inch*' between the *diagonal screen size*, *height* or *width* of a television; or for the table watches, map the keyword '*gold*' to the *color* or *material* of a watch.

In this paper, we assume that the first problem, understanding if a query targets a source of structured data and which source in particular, has been addressed from existing literature, such as [13, 19, 24]. We use the existing work to find the appropriate target data source. We focus on the second problem, disambiguating between attributes within an entity-type domain. Using appropriate disambiguation, we learn the most suitable facet attributes and values. Furthermore, we also discuss how data statistics can be used to help with the disambiguation process by being tolerant to data noise and imperfections, and also which data statistics indicate good facets.

Our contributions are summarized as follows: (i) We formulate the problem of query-log based facet attribute and value selection, presented in Section 2. (ii) We introduce different attribute-value identification and disambiguation techniques in Section 3, which we extend for computing facet values, including context-dependent ones, in Section 4. (iii) We explore the effect of appropriate data statistics in facet selection, presented in Section 5. And (iv), we perform a detailed experimental study on a large scale data set with a thorough evaluation on the quality of the results from real users, presented in Section 6, where we discuss the qualitative difference between our techniques as well as show our best approach outperforming state-of-the-industry systems. We conclude with related work in Section 7 and final thoughts in Section 8.

2. FACET SELECTION

In this section we formulate our problem, and provide the necessary definitions for the remainder of the paper.

2.1 Problem Formulation

We assume that structured data are organized as a collection of *tables* $\mathcal{T} = \{T_1, T_2, \ldots, T_\tau\}$[1]. A table T is a set of related *entities* defined over a set of *attributes*, $T.\mathcal{A} = \{T.A_1, T.A_2, \ldots, T.A_\alpha\}$. We use $T.A.\mathcal{V}$ to denote the domain of the attribute $T.A$. We omit the prefix T and use A, whenever the table is unambiguous.

Given a table T we define a *facet system* $\mathcal{F} = \langle F_1, ..., F_k \rangle$ for table T as an ordered list of k *facets*, where k is is the facet budget. A facet $F_i = \{A_i, \langle A_i.v_1, ..., A_i.v_{m_i} \rangle\}$ consists of an attribute A_i and an ordered list of m_i values, where m_i is the value budget for attribute A_i. For the following we use $\mathcal{A}_\mathcal{F}$ (or $\mathcal{F}.\mathcal{A}$) to denote the set of attributes of $\mathcal{F}$, and $A.\mathcal{V}_\mathcal{F}$ to denote the set of values for attribute $A \in \mathcal{A}_\mathcal{F}$ in facet system $\mathcal{F}$. We say that a facet system *supports* an attribute A if $A \in \mathcal{A}_\mathcal{F}$, and we define the attribute-support function as follows:

$$\text{ASUP}(\mathcal{F}, A) = \begin{cases} 1, & \text{if } \mathcal{F} \text{ supports } A \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

We also say that a facet attribute $A_f \in \mathcal{A}_\mathcal{F}$ supports value v if $v \in A_f.\mathcal{V}_\mathcal{F}$, and we define the attribute-value support function as follows:

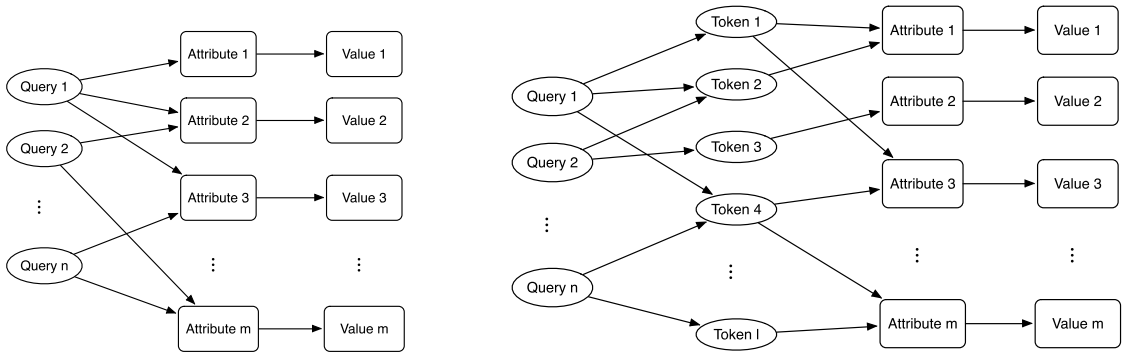
$$\text{VSUP}(A_f, v) = \begin{cases} 1, & \text{if } A_f \text{ supports } v \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

A facet system is exposed through the user interface, and it enables the user to navigate through the data in the table T by selecting values for the different attributes. The design and analysis of faceted user interfaces is an active research area in the HCI community [10]. In any faceted search system, the facet selection is paramount to the success of the system. The order of facets is also important since it determines how the facets will be displayed, with more important facets being displayed more prominently. The goal is to maximize the user engagement and satisfaction, and the i-th facet corresponds to the i-th best attribute for this task. A similar reasoning applies also to the ordering of the facet values within a facet. However, for a selected facet attribute there is usually a greater flexibility in the presentation of values (drop-down menus, slider bars, text boxes) making the value selection and ordering less critical. We thus consider the *facet attribute ordering* problem and the *facet attribute value ordering* problem separately, with the former being more important than the latter.

In this work we are interested in constructing a facet system using web search queries. We assume that we have a query log $\mathcal{Q} = \{q_1, ..., q_n\}$, consisting of queries posed to a generic web search engine. For each query q_i we also have a weight w_i denoting the importance of the query, e.g., the number of times that the query appears in the query log.

Although queries are posed to the search engine, it is common that users make queries that are targeted to structured data. For example, the query "*canon powershot*" can be interpreted as targeting a table with structured data about digital cameras. For a given table T let $\mathcal{Q}_T \subseteq \mathcal{Q}$ denote the subset of queries that target table T. We will drop the subscript T whenever the table is unambiguous. We assume that each query $q \in \mathcal{Q}_T$ specifies a set of attribute-value pairs $\mathcal{AV}_q = \{AV_i\}$ of the form $AV_i = (A_i, A_i.v_j)$. For example, the query "*canon powershot*" specifies the value '*canon*' for attribute *brand*, and the value '*powershot*' for attribute *product line*. We can thus map the query $q =$"*canon powershot*" to the set $\mathcal{A}_q =$ {(*brand*,'*canon*'),(*product line*,'*powershot*')}. We use $\mathcal{A}_q$

[1]The organization of data into tables is purely conceptual and orthogonal to the underlying storage layer: the data can be physically stored in XML files, relational tables, retrieved from remote web services, etc. Our assumption is that a mapping between the storage layer and the "schema" of table collection $\mathcal{T}$ has been defined.



(a) Queries to discrete attribute-value pairs (b) Queries to attribute-value pairs through tagged tokens

Figure 1: Query to Attribute-Value Pairs Mappings

to denote the set of attributes specified in the query. Figure 1(a) depicts graphically an example. Note that we only assume that such a mapping exists; obtaining it from the web log is a challenging problem as we elaborate below.

We now define the *attribute utility* of a facet system $\mathcal{F}$, making use of the attribute support function defined in Equation 1.

DEFINITION 1 (ATTRIBUTE UTILITY). *Given a facet system $\mathcal{F}$, and a query q, we define the* attribute utility *of $\mathcal{F}$ for q as*

$$U_a(\mathcal{F}, q) = \sum_{A \in \mathcal{A}_q} \text{ASUP}(\mathcal{F}, A)$$

the number of attributes in $\mathcal{A}_q$ supported by $\mathcal{F}$. The attribute utility of $\mathcal{F}$ for query log $\mathcal{Q}$ is defined as

$$U_a(\mathcal{F}, \mathcal{Q}) = \sum_{q \in \mathcal{Q}} w_q U_a(\mathcal{F}, q)$$

where w_q denotes the weight of query q.

We can similarly define the *attribute-value utility* of a facet attribute A_f, making use of the attribute-value support function defined in Equation 2.

DEFINITION 2 (ATTRIBUTE-VALUE UTILITY). *Given a facet attribute A_f and a query q, we define the* attribute-value utility *of A_f for q as*

$$U_v(A_f, q) = \sum_{(A,v) \in \mathcal{AV}_q : A = A_f} \text{VSUP}(A_f, v)$$

the number of values in $\mathcal{AV}_q$ supported by A_f. The attribute-value utility of A_f for query log $\mathcal{Q}$ is defined as

$$U_v(A_f, \mathcal{Q}) = \sum_{q \in \mathcal{Q}} w_q U_v(A_f, q)$$

where w_q denotes the weight of query q.

As we have already argued, queries act as a proxy for the exploration process of the users over the structured data. Users include in their queries attributes over which they want to dissect the table. A good faceted system should anticipate the information need of the user and select attributes (and values) that maximize the utility to the users. We thus have the following definition of the *query-based* facet selection problems.

PROBLEM 1 (QUERY-BASED FACET ATTRIBUTE SELECTION). *Given a table T, and a query log $\mathcal{Q}_T$ over table T, create an ordering $\mathcal{L}(T.\mathcal{A})$ of the attributes $T.\mathcal{A}$ such that for any $k \geq 1$, a facet system $\mathcal{F}$ with attribute set $\mathcal{F}.\mathcal{A} = \mathcal{L}_k(T.\mathcal{A})$ — the top-k attributes according to ordering $\mathcal{L}$ — has maximum attribute utility $U_a(\mathcal{F}, \mathcal{Q}_T)$ over all facet systems with attribute set of size k.*

PROBLEM 2 (QUERY-BASED FACET VALUE SELECTION). *Given a table T, an attribute $A \in T.\mathcal{A}$, and a query log $\mathcal{Q}_T$ over table T, create an ordering $\mathcal{L}(A.\mathcal{V})$ of the values in $A.\mathcal{V}$ such that for any $m \geq 1$, a facet system $\mathcal{F}$ that contains A with attribute value set $A.\mathcal{V}_\mathcal{F} = \mathcal{L}_m(A.\mathcal{V})$ — the top-m attributes according to ordering $\mathcal{L}$ — has maximum attribute-value utility $U_v(A, \mathcal{Q}_T)$ over all facet systems that contain A with attribute-value set of size m.*

If we are given the mapping $\mathcal{AV}_q$ from queries to attribute value pairs, as in Figure 1(a) then the facet selection problems we defined above are easy to solve. This is the case for example with structured query logs (e.g. SQL), where the user specifies explicitly the attributes and values they are interested in. Then, it is straightforward to see that the optimal solution for Problem 1 is to order the attributes according to their *popularity* in the query log. The popularity $F_\mathcal{Q}(A_i)$ of attribute A_i over query log $\mathcal{Q}$ is computed as

$$F_\mathcal{Q}(A_i) = \sum_{q \in \mathcal{Q}} \sum_{(A_i, v) \in \mathcal{AV}_q} w_q \tag{3}$$

The popularity $F_\mathcal{Q}(A_i)$ is thus the number of attribute-value pairs in $\mathcal{Q}$ that contain attribute A_i weighted by the weight of the query in which they appear.

Similarly, the optimal solution for Problem 2 is to order the values within attribute A_i according to their popularity in the query log. The popularity of a value v_j of attribute A_i is computed as

$$F_\mathcal{Q}(A_i.v_j) = \sum_{q \in \mathcal{Q}} \sum_{(A_i, v_j) \in \mathcal{AV}_q} w_q \tag{4}$$

the number of attribute value pairs in the query log, where A_i is the attribute and v_j is the value.

Unfortunately, in the case of web query logs this mapping is not given to us and we need to discover it from the data. We elaborate on this challenge in the following section.

2.2 Mapping Web Queries to Structured Data

In order to extract attribute value pairs from the queries in the unstructured web query log, we need to address two issues: (a) for every table T, identify the queries in $\mathcal{Q}$ that target table T, i.e., determine the set $\mathcal{Q}_T$, and (b) identify and disambiguate the attribute occurrences in the queries. As previously discussed, we use existing work [13, 24] to solve (a) and instead focus on the latter problem.

Given a query $q \in \mathcal{Q}_T$ we now need to identify the occurrences of the attributes in $T.\mathcal{A}$ in the query q. We assume a *tagging* process TAGGER that performs this task. Define a *token* to be a string consisting of one or more contiguous words, and let L_q denote the set of all tokens in q. The TAGGER produces all tokens $t \in L_q$, and matches them with attributes in $T.\mathcal{A}$, producing attribute-token

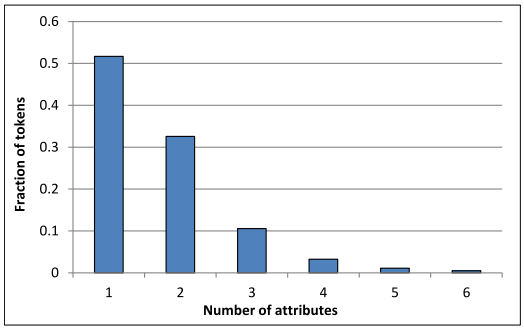


Figure 2: Fraction of tokens mapped to varying num of attributes.

pairs, $AT = (\mathcal{A}, t)$. For a categorical attribute $A_c \in T.\mathcal{A}$, the TAGGER outputs an attribute-token pair $AT = (A_c, t)$, if the token t appears in $A_c.\mathcal{V}$. Approximate, or synonymous matching is also possible as an extension of the same process, although discussing it in detail is outside the scope of this paper. For a numerical attribute A_n with unit u (e.g., inches, Kg, or x zoom), the TAGGER produces an attribute-token pair $AT = (A_n, t)$ for every token t that consists of a number followed by a unit u. For example, the tagger output for the query "*5x nikon*" over the digital cameras table is {(*digital zoom*, '*5x*'), (*optical zoom*, '*5x*'), (*brand*, '*nikon*')}.

For a query q we use $\mathcal{AT}_q = \{(A, t)\} \subseteq T.\mathcal{A} \times L_q$ to denote the set of attribute-token pairs output by the TAGGER . Note that attribute-token pairs are different from attribute-value pairs, since a token may not be a value in the table T. This is the case with synonyms and approximate matches, as well as with numeric attributes. For example, in the query "*30 inch lcd TV*", the token '*TV*' can match the data value '*Television,*' and the token '*30 inch*' can be mapped to *diagonal screen size* even though we may only have televisions with 27 and 32 inch diagonals.

Given the query log $\mathcal{Q}$ the TAGGER outputs a set of attribute-token pairs $\mathcal{AT}_\mathcal{Q}$. It would appear that our job is now done, since we can compute the popularity of attributes and values as before, simply by interchanging tokens with values, and attribute-value pairs with attribute-token pairs. However, this is not the case due to *ambiguity* in the token interpretation. The domains of attributes in a table often overlap, and as a result a token t can map to more than one attribute. For example, in the query "*30 inch lcd TV*", the token '*30 inch*' is potentially mapped to the *diagonal*, *width*, *height*, and *depth* attributes, since all these attributes are numeric with the same unit; similarly in the query "*camera with 3x zoom*", '*3x*' can be either *optical zoom* or *digital zoom*. Such confusion is not limited to just numeric attributes. For example in the query "*gold watch*", the token '*gold*' can specify either the *material*, or the *color* attribute; similarly for "*leather dress shoes*" the token '*leather*' can map to either the *top shoe material* or the *shoe sole*. There are many such examples that appear in web queries. We computed the number of attributes of the same table that a token matches on average, over all queries. As we can see from Figure 2, approximately half the tokens are confused over at least two or more attributes. So ambiguity is a real problem that needs to be addressed.

Estimating attribute popularity in the presence of ambiguity becomes problematic. Our data no longer looks like Figure 1(a), but instead like Figure 1(b). Computing the attribute popularity over attribute-token pairs using Equation 3 will lead to misleading results: TV *height* will incorrectly be deemed equally important to TV *diagonal*, and camera *digital zoom* the same as *optical zoom*. In order to obtain correct estimates for the true popularity of an attribute, we need to *disambiguate* the ambiguous tokens.

Let $\tilde{t}$ denote a token in a query q. We say that token $\tilde{t}$ is ambiguous for a table T, if there are at least two attribute-token pairs $(A_i, \tilde{t})$ involving $\tilde{t}$. We use $\mathcal{A}_{\tilde{t}}$ to denote the set of candidate attributes to which the token $\tilde{t}$ is mapped. Our goal is to disambiguate between the attributes in $\mathcal{A}_{\tilde{t}}$. We do this by estimating the probability of an attribute given the ambiguous token. We thus have the following problem definition.

PROBLEM 3 (ATTRIBUTE DISAMBIGUATION). *Given a table T, a query log $\mathcal{Q}$ over T, and an ambiguous token $\tilde{t} \in L_\mathcal{Q}$ that maps to the set of attributes $\mathcal{A}_{\tilde{t}} \subseteq T.\mathcal{A}$, compute the* disambiguation probability distribution*, denoted $P(A|\tilde{t})$, over the set of candidate attributes $A \in \mathcal{A}_{\tilde{t}}$.*

Given $P(A|\tilde{t})$ we now have a measure of likelihood for the mapping of token $\tilde{t}$ to attribute A. We can use this probability to perform either hard or soft disambiguation. In hard disambiguation, we map token $\tilde{t}$ to the most likely attribute $A_{\tilde{t}} = \arg\max_{A \in \mathcal{A}_{\tilde{t}}} P(A|\tilde{t})$. In hard disambiguation our data will take the form of Figure 1(a), and we can apply directly Equation 3 to estimate the attribute popularity. In soft disambiguation, we modify Equation 3 such that each occurrence of attribute A with the ambiguous token $\tilde{t}$ is weighted by the probability $P(A|\tilde{t})$. We thus have

$$\tilde{F}_\mathcal{Q}(A_i) = \sum_{q \in \mathcal{Q}} \sum_{(A_i, t) \in \mathcal{AT}_q} w_q P(A_i|t) \quad (5)$$

We have $P(A|t) = 1$ if token t maps unambiguously to attribute A.

3. ATTRIBUTE DISAMBIGUATION

In this section we describe four different approaches for attribute disambiguation. The first three rely on the principle that the best solution is the one that better *explains* the attribute-token pairs we observe. If we assume that the attributes in the table are generating queries and tokens, we identify the attribute that is most likely to have generated the tokens we are observing. The three different algorithms differ in the granularity at which they try to explain the data, becoming progressively more complex. The last approach relies on user feedback from a click-log to disambiguate between different interpretations of a token.

3.1 Token-level Disambiguation

The first algorithm we consider treats each token independently and tries to find the attribute that is most likely to have generated this specific token. Let $\tilde{t}$ denote an ambiguous token and let $\mathcal{A}_{\tilde{t}}$ the set of candidate attributes for $\tilde{t}$. We want to estimate $P(A|\tilde{t})$ for all $A \in \mathcal{A}_{\tilde{t}}$. Using Bayes rule we have

$$P(A|\tilde{t}) = \frac{P(\tilde{t}|A)P(A)}{\sum_{A_i \in \mathcal{A}_{\tilde{t}}} P(\tilde{t}|A_i)P(A_i)}$$

We will estimate the right-hand side using the data in T. Assuming that all attributes are equally likely in the table (i.e., $P(A)$ is the same for all attributes in the table), then $P(A|\tilde{t})$ is proportional to $P_T(\tilde{t}|A)$, the probability that the token $\tilde{t}$ is generated from the distribution of A in the table. We have that

$$P_T(\tilde{t}|A_i) = \frac{|T(A, \tilde{t})|}{|A|}$$

where $T(A, \tilde{t})$ denotes the set of entries in the table where the attribute A takes the value $\tilde{t}$, and $|A|$ is the number of table entries with some value for attribute A. Numeric attributes are similarly described using histograms to deal with the continuity of numbers.

Therefore, according to this algorithm, the most likely attribute is the one that is most likely to have generated token $\tilde{t}$. From an information theoretic point-of-view, this is the attribute who's value distribution can encode token $\tilde{t}$ with the fewest number of bits, i.e., the one that reduces the uncertainty the most for $\tilde{t}$.

3.2 Cluster-level Disambiguation

The token-level disambiguation approach considers the probability of each token independently mapping to a candidate attribute. A natural extension to this model is to consider the ambiguity among a cluster of tokens confused with the same set of attributes. In this cluster-based approach, we aggregate over all tokens in the query-log to find clusters of ambiguous attribute-token pairs. We then resolve which attributes in the cluster are better candidates to model the distribution of confused tokens over the *full* cluster.

Consider the bipartite graph formed by the set of attribute-token pairs in $\mathcal{AT}_\mathcal{Q}$ (e.g., see Figure 3 (a)). Each time a token is mapped to a set of attributes, it *supports* the ambiguity of these attributes. For example, if the token '*8x*' is mapped to both *digital zoom* and *optical zoom*, it supports the ambiguity of these two attributes. If the token '*2x*' is mapped to *digital zoom*, *optical zoom*, and *model* it supports the ambiguity of all three of these attributes. By aggregating over all tokens in the query log, we can find the groups of attributes most commonly confused. In particular, if we consider the set of all tokens that form a bi-clique with a set of attributes, then these attributes are all pairwise ambiguous over the same set of tokens, and the support for the ambiguity of this cluster is proportional to the number of tokens in the bi-clique. It is these strongly supported bi-cliques that we aim to find.

To find clusters of ambiguous attributes, we first construct an *attribute ambiguity graph* per table. For a table T, consider a graph $G = (V, E, w)$ where there exists a vertex $v_a \in V$ for every attribute $a \in \mathcal{T}.\mathcal{A}$. We create an edge $e = \langle v_{A_1}, v_{A_2} \rangle$ between two vertices v_{A_1} and v_{A_2} if some token in the query log is mapped to both A_1 and A_2 over all interpretations of queries. The weight function $w(e)$ assigns a weight to the edge e equal to the number of tokens confused between (the attributes denoted by) v_{A_1} and v_{A_2}. We then enumerate all cliques in the attribute ambiguity graph, with the support being the minimum edge weight in the clique. For example, in Figure 3 (b), the support of the 2-clique containing *digital zoom* and *optical zoom* is ten, while the support of the 3-clique containing *digital zoom*, *optical zoom*, and *model* is two.

After identifying the clusters of ambiguous attributes, we want to compute how likely each attribute is to model the set of confused tokens. We do this by computing the KL-divergence (Kullback-Leibler divergence) between the value distribution of the attribute and the set of confused tokens. Let $C = (\mathcal{A}_C, L_C)$ denote an attribute-token cluster. We compute the disambiguation probability of an attribute $A \in \mathcal{A}_C$ given the cluster C as follows.

$$P(A|C) = \frac{KL(A||C)}{\sum_{Ai \in C} KL(Ai||C)}$$

where the KL-divergence between two distributions P and Q is defined as follows.

$$KL(P||Q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

For every ambiguous attribute-token pair $(A, \tilde{t}) \in L_C$, we set the disambiguation probability $P(A|\tilde{t}) = P(A|C)$.

The intuition in this kind of disambiguation is that the attribute which forms a better model for the query token distribution is more likely to be the correct attribute users had in mind when formulating these queries. For example, if a collection of tokens around measurements are confused among television *height* and *diagonal*, we would expect to find the distribution of values in the *diagonal* attribute to be a better model for the set of confused query tokens.

As an example, consider Figure 3. There are eight tokens with the same connectivity of the token labeled '*8x*', two tokens with the same connectivity as '*2x*', and five tokens with the same connectivity as '*Pix.*' The ambiguous attribute clusters correspond to the 2-clique {*Digital Zoom*, *Optical Zoom*} with support 10, the 2-clique {*Product Name*, *Model*} with support five, and lastly the 3-clique {*Digital Zoom*, *Optical Zoom*, *Model*} with a support of two. The 3-clique will be used to disambiguate among tokens confused with all three attributes, while the sub-clique will be used to disambiguate between tokens confused among only those two attributes. We may also find the remaining two 2-cliques {*Digital Zoom*, *Model*} and {*Optical Zoom*, *Model*} however in our implementation we are not concerned with these as there are no tokens that confuse *Model* with one of the other attributes and not the third (i.e., all tokens confused with *Model* and *Optical Zoom* are also confused with *Digital Zoom*, and all tokens confused with *Model* and *Digital Zoom* are also confused with *Optical Zoom*).

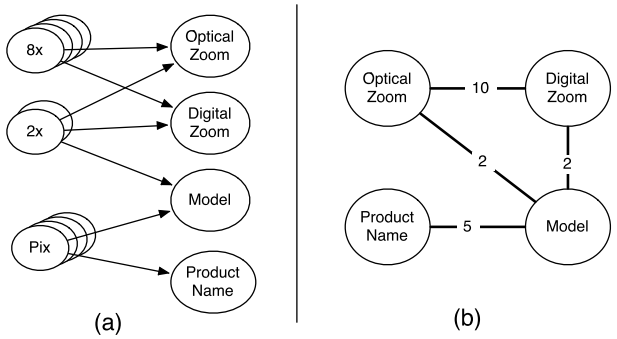


Figure 3: (a) a bipartite token-attribute graph and (b) the resulting attribute ambiguity graph.

3.3 Query-log-level Disambiguation

We now present an approach that considers the tokens in the full query log, and estimates the probability $P(A|\tilde{t})$ for ambiguous tokens $\tilde{t}$, such that the likelihood of the full observed query log is maximized. Let $\mathcal{Q}$ be a query log over table T. Let $L_\mathcal{Q}$ denote the set of all tokens generated by the TAGGER that appear in the attribute-token pairs $\mathcal{AT}_\mathcal{Q}$. Each token t is assigned a weight

$$w_t = \sum_{q \in \mathcal{Q}} \sum_{(A_i, t) \in \mathcal{AT}_q} w_q,$$

the total weight of queries in which token t appears. For simplicity we can consider w_t to be the frequency of the token t in $\mathcal{AT}_\mathcal{Q}$.

We assume that the tokens in $L_\mathcal{Q}$ were generated according to a generative model, which generates tokens as follows: an attribute A_i is selected with probability $P(A_i)$, and then a token t is drawn from the data distribution $P_T(t|A_i)$. Thus token t is generated with a probability given by the following.

$$P(t) = \sum_{A_i \in T.\mathcal{A}} P_T(t|A_i) P(A_i)$$

The probability of observing the full set of tokens $L_\mathcal{Q}$ is then

$$P(L_\mathcal{Q}) = \prod_{t \in \mathcal{Q}} P(t)^{w_t}$$

The generative model we have described has α parameters $\pi_i = P(A_i)$, one for each attribute A_i in the table T. Parameter π_i is the probability that attribute A_i is activated for the generation of a given token, and we have that $\sum_{i=1}^{\alpha} \pi_i = 1$. Let π denote the vector of parameters $\pi_1, ..., \pi_\alpha$. We perform Maximum Likelihood

Estimation of π, and we look for the value of π that maximizes the probability that we observe the tokens in $L_{\mathcal{Q}}$. Using log-likelihood notation we want to minimize the following.

$$\mathcal{L}(L_{\mathcal{Q}}) = -\log P(L_{\mathcal{Q}}) = -\sum_{t \in_{\mathcal{Q}}} w_t \log P(t)$$

We use an iterative EM algorithm for finding the Maximum Likelihood Estimate for π. The algorithm initializes the values of the parameters π_i to an arbitrary distribution, in our implementation uniform. Then it alternates between the following two steps.

In the E-step given an estimation for the parameters π, for each attribute-token pair (A_i, t_j) in $L_{\mathcal{Q}}$ we compute the posterior probability of the attribute A_i given token t_j

$$P(A_i|t_j) = \frac{P(t_j|A_i)\pi_i}{\sum_{A_\ell \in \mathcal{AT}.\mathcal{A}} P(t_j|A_\ell)\pi_\ell} \quad (6)$$

In the M-step, we can now estimate new values for the parameters in π as follows,

$$\pi_i = \sum_{(A_i,t) \in \mathcal{AT}_{\mathcal{Q}}} P(A_i|t)P(t) = \frac{w_t}{W} \sum_{(A_i,t) \in \mathcal{AT}_{\mathcal{Q}}} P(A_i|t) \quad (7)$$

where $W = \sum_{t \in L_{\mathcal{Q}}} w_t$ is the total weight of all tokens.

We repeat these two steps until the algorithm converges. Given that the optimization function is convex we know that it will reach a global maximum. After the algorithm has converged, for an ambiguous token $\tilde{t}$ we can now compute $P(A|\tilde{t})$ using Equation 6.

The Maximum Likelihood Estimation of the probabilities $P(A_i)$ finds a set of values that best explains the full query log we observe. Intuitively what this means is that when observing a token t, even though attribute A_1 may have the highest probability $P(t|A_1)$, the token t may be attributed to another attribute A_2 since this is overall more likely to occur in the log. For example, consider a query-log with queries over the televisions table, and let L_{inch} denote the set of all tokens of the form number followed by the unit "inch". If the overall distribution of these tokens agrees better with the attribute *diagonal* than *height* then for a token like '*30 inch*' in L_{inch} the attribute *diagonal* will have higher probability $P(A|t)$ even though the value is more likely in the *height* attribute. This agrees well with our intuition that '*30 inch*' is more likely to refer to the *diagonal* rather than the *height*, since it is typical for users to query for *diagonal* when querying for televisions. The iterative algorithm captures nicely this intuition.

3.4 Clicks-based Disambiguation

The ambiguity problem is essentially an intent issue. Given an ambiguous token, and multiple interpretations of the token in the data table we have no indication which interpretation was intended by the user. In an online environment, a strong indication of intent is the user click behavior.

A query-click log $\mathcal{QC} = \{(q_1, e_1), ..., (q_n, e_n)\}$ over a table T is a set of query-entity pairs, where q is a query posed in a commercial search engine, and e is an entity in table T that was shown to the user as a result of the query and the user chose to click on it to obtain further information about it. An entity e is an entry in table T. We represent it as a set of attribute-value pairs $e = \{(A_i, A_i.v_j)\}$ for every attribute A_i in table T. Given a query-click pair (q, e), we make the assumption that the user who posed the query intended to access the attributes of clicked entity e.

Since we now have an indication for the intent of the user, the disambiguation problem becomes significantly easier to tackle. Given an ambiguous token $\tilde{t}$ that maps to attributes $\mathcal{A}_{\tilde{t}} = \{A_{i_1}, ..., A_{i_k}\}$ we disambiguate by selecting from entity e the attribute $A^* \in \mathcal{A}_{\tilde{t}}$ that has value $A^*.v$ that best *matches* token $\tilde{t}$.

The notion of a match depends on the type of token that we consider. In the case of categorical tokens we require that $A^*.v = \tilde{t}$. In such cases we have that $P(A^*|\tilde{t}) = 1$, since we assume complete certainty in our mapping. If there is more than one attribute that match token $\tilde{t}$ then we assume that $\tilde{t}$ maps to all of them, and we assign equal probability to all. It is worth noting that, although possible, this event is not very probable. In our experiments, we did not observe any case where multiple categorical attributes match on the same token for the same clicked entity. In the case of numerical tokens, we map token $\tilde{t}$ to the attribute A^* that has the closest value, i.e., it minimizes the difference $|\tilde{t} - A^*.v|$. we compute the disambiguation probability as

$$P(A^*|\tilde{t}) = \frac{\exp(-|\tilde{t} - A^*.v|)}{\sum_{A \in \mathcal{A}_{\tilde{t}}} \exp(-|\tilde{t} - A^*.v|)}$$

favoring the attributes with smallest distance.

Clicks have been used extensively in web search for eliciting the preferences of users [14, 19]. It is well known that they contain a very strong signal for the intent of the user, but they also come with different caveats, such as presentation bias (users click only on what they are shown), ranking bias (users tend to click on the top positions), and noise. We discuss some of these issues in the experimental evaluation (see Section 6).

4. ATTRIBUTE VALUE SELECTION

Equally important to finding important attributes, is the problem of finding important values to populate the facets. We apply the same philosophy for values as we do for attributes; popular values in a query log will be strong facet values that are important to users. In the following section, we discuss two types of facet value selection: category-dependent facet value selection, and dynamic context-dependent facet value selection.

4.1 Table Dependent Value Selection

To compute the value score, we consider Equation 4 and modify it such that each occurrence of attribute A and value v with the ambiguous token $\tilde{t}$ is weighted by the probability $P(A.v|\tilde{t})$. We then have the popularity of a value as the following.

$$\tilde{F}_{\mathcal{Q}}(A_i.v_j) = \sum_{q \in \mathcal{Q}} \sum_{(A_i,\tilde{t}) \in \mathcal{AT}_q} w_q P(A_i.v_j|\tilde{t}) \quad (8)$$

In the case that value v_j does not map to the token $\tilde{t}$, the probability will be zero. Also, for all attributes A_i of the same table we have $P(A_i.v_j|t) = P(A_i|t)$ since the token to value mapping is part of the attribute based disambiguation. Hence each of the attribute disambiguation methods from Section 3 are directly applicable. For numeric attributes, we can compute weights for buckets of values via the usage of a histogram. This method allows for each table attribute to display a set of values based on user popularity. For example, for the golf clubs table and *brand* attribute, the top 3 facet values would be '*Nike*', '*Ping*', and '*TaylorMade*'.

4.2 Context Dependent Value Selection

We described how to use query log popularity to find good facet values given the table and an attribute. However, we still have the problem that certain combination of values across attributes will return empty results. For example, consider a faceted search engine which includes *brand* and *golf club type* among its facets for the golf clubs table. There exist some specialized brands that only produce very few types of golf clubs, i.e. '*Odyssey*' or '*Scotty Cameron*' only produce '*putters*'. So now when a user selects '*Odyssey*', then *golf club type* values such as '*drivers*' or '*wedges*' are not good facet

choices since selecting them would produce empty results. Showing '*putters*' would be a much better choice.

Also consider the flip side of this example that exposes another interesting problem. When the first user action is to select '*putters*' as *golf club type*, then popular choices such as '*Nike*' or '*TaylorMade*' for *brand* will produce valid results, albeit not necessarily the most desirable ones. Golf aficionados may prefer '*Odyssey*' or '*Scotty Cameron*' putters. Although it is not that dramatic to put the generally popular '*Nike*' on top, it is arguably a better user experience to be able to show '*Odyssey*' given the '*putters*' selection. The example output from our implementation can be seen in Figure 4. In summary, it is preferable for a facet system to take into consideration the existing user context, in the form of pre-selected attribute-value pairs from a facet selection or a user query (via [20, 24]), and dynamically adjust the shown facet values.

One way to achieve this task would be to consider the data and pre-compute the correlation between all possible value combinations across attributes. This information is then stored for online processing and use the context to filter the possible choices. However, such computation is very expensive for the scale of data sets available in a web search engine. Even after this expensive computation, supporting the retrieval of such information for real-time search is a non-negligible investment that will have an overall impact in resource usage and result response time. Besides the computational cost, it is also not clear whether data-correlation frequency corresponds well with the user intent.

To limit the computational cost while still trying to best satisfy the user, we turn again to query logs. We compute co-occurrence for value pairs across different attributes that appear frequently together in the same query. We then validate the co-occurring values against the database to ensure a non-zero entity count in the results. By going to the query logs we significantly reduce the space of possible data value pairs to consider when compared to all attributes and all values of a table. As a result, we effectively address the issue of empty results without significantly affecting the response time, resource usage and off-line computation cost. More importantly, we further optimize the user experience by dynamically placing the most relevant facet values at the top of the list for a given selection.

To compute the co-occurrence, we define the conditional score of a value for a given attribute by the probability that the value represents the attribute in a query that co-occurs with a given attribute value pair. Let $A_c.v_c$ denote conditional value v_c bound to attribute A_c, $A.v$ denote value v bound to attribute A, and $\tilde{t}$ be a token. We want to estimate $P(A.v|\tilde{t}, A_c.v_c)$, the probability of value v bound to attribute A given the observed token and conditional attribute-value binding. Since tokens are tagged independent of one another, this simplifies to $P(A.v|\tilde{t})$ which we can compute using Equation 8. The difference is in which queries are used from the query-log when computing the popularity. Given a query log $\mathcal{Q}$ and attribute-value pair $AV = (A_c, A_c.v_c)$, let $\mathcal{Q}_{AV}$ denote the subset of queries $\{q \mid q \in \mathcal{Q} \wedge (A_c, A_c.v_c) \in \mathcal{AV}_q\}$. We then estimate the popularity of a value over all queries satisfying the given attribute-value condition.

$$\tilde{F}_{\mathcal{Q}_{AV}}(A.v) = \sum_{q \in \mathcal{Q}_{\mathcal{AV}}} \sum_{(A,v) \in \mathcal{AV}_q} w_q P(A.v|\tilde{t})$$

The last consideration for context-dependent facet values is the case where there are multiple selection conditions specified in the query. This may occur either as multiple recognized attribute values in a keyword query, or as multiple facet selections. Supporting n-way conditionals with exact information is impractical. The data-driven approach suffers the combinatorial explosion of an already large data set, and the query log driven approach suffers from the sparseness of queries. After filtering by multiple selection conditions, the number of queries from which to compute popularity frequencies becomes too small. For this scenario, we adopt a simple approach of intersecting value lists and aggregating popularity counts by addition. This will find values relevant to multiple selections, and score them proportionally to how relevant they are to their respective conditional selections. Handling an empty intersection is more of a user interface design decision. One could eliminate the facet entirely, or default to the category-dependent values.

Input	Table: Golf Clubs	Query: "golf putters"
Facet Values	Nike	Odyssey
	Ping	Ping
	TaylorMade	Scotty Cameron

Figure 4: Example top-3 facet values for golf club brand.

5. FACETS AND DATA STATISTICS

The query logs can reveal which attributes in the table are popular among the users. However, just because an attribute is popular does not necessarily mean that it is good to be used as a facet. In this section we discuss how to use some signals from the data to discover which attributes make better facets.

Information Content: Recall that the goal of facets is to enable the users to quickly zoom into a smaller subset of products that are of interest to them. Therefore, in order for an attribute A to be a good candidate for a facet, it should have high information content, that is, the value selection v for the attribute A should give information about the tuples of interest to the user. This property is naturally captured in the *entropy* of the attribute A. For an attribute A defined over the domain $A.\mathcal{V}$, the entropy of A is defined as

$$H(A) = -\sum_{v \in \mathcal{V}} P_A(v) \log P_A(v)$$

where $P_A(v)$ is the fraction of entries in the table where attribute A takes value v, over the number of entries in the table where the attribute A takes any value (i.e., is non-null).

Attributes with low entropy do not make good facets. For such attributes the distribution of values is highly skewed, and the knowledge of the value gives very little information for the subset of entities of interest to the user. This is clear in the extreme example when the entropy is zero, meaning that all entities take the same value. In this case, the attribute is useless for navigation, since it gives no information about which entities the user is interested in. For example, the attribute *color* in the televisions table conveys no useful information as a facet if all televisions are '*black*'.

An alternative interpretation of $H(A)$ is that it captures the decrease in the uncertainty for the preference of the user for the entities (rows) in table T. When no facet attribute has been selected, any entity $e \in E$ is equally likely to be of interest to the user, therefore, if N is the number of entities in the table, each entity has probability $P(e) = 1/N$. The uncertainty is captured in the entropy of the random variable E, which in this case is maximum, $H(E) = \log N$. Now assume that the user is allowed to use attribute A to zoom in on a smaller subset of entries. The uncertainty for E decreased given the knowledge of the value in A is measured as $H(E) - H(E|A)$, where $H(E|A)$ is the conditional entropy of E given A. It is well known that $H(E|A) = H(E, A) - H(A)$. Since each entity is distinct from the rest, and assuming that A takes a value for all entities, we have that $H(E, A) = H(E) = \log N$. Therefore, $H(E|A) = H(E) - H(A)$. Thus the entropy $H(A)$ captures how much our uncertainty for the entities that the user is interested in will

decrease when we have the knowledge for attribute A. If $H(A) = 0$, then the knowledge of A gives no extra information for E and thus it is redundant. It follows naturally, that attributes with low entropy should not be used as facets.

Sparsity: Real-world data is often noisy with missing or incomplete information. As a result there are often cases where some attributes are only sparsely populated. If such an attribute is selected for faceted navigation, then selecting any value will immediately discard most of the entities since they do not have a values for the sparse attribute. This could be ok if this corresponded to a rare feature, in which case missing information corresponds to negative information for the existence of the feature. However, it is often the case that sparsity is due to noise in the data collection, in which case, the entities that are eliminated are valid for the selected facet, yet will never be visible to users. Furthermore, noisy attributes are often likely to contain incorrect values, confusing the tagging process of the queries and the corresponding probabilities.

Therefore, we exclude from the candidate attributes the ones that are very sparse. The sparsity of attribute A is defined as

$$R(A) = \frac{|T(A)|}{|T|}$$

where $T(A)$ is the set of entries in which the attribute has a non-null value, and $|T|$ is the total number of entries in the table. Similarly to entropy, sparse attributes are usually not good facet candidates.

6. EXPERIMENTAL EVALUATION

We abstract the structured data as tables, each containing entities of the same type. For our experiments we have 1164 such tables that we crawled using the MSN shopping XML API[2]. The tables correspond to products used to answer shopping queries and are similar to the data used by sites like Amazon or Google Product Search. We consider each category of products to be a table of entities of the same type. The available range covers entities from electronics like digital cameras or televisions to golf clubs and soft goods like shoes and shirts. In total, there were around 10 million structured distinct product entities occupying approximately 300GB on disk when stored in our database. Besides the structured data, we also use a web query log and a vertical click log.

The *web query log* contains queries posed on a major search engine on a period of five months from July to November 2009. The web queries are approximately 31 million distinct queries all in the English language. As query weight we use the aggregated impressions count – each time a query is asked it increments its impression count. We limit ourselves to queries with at least 10 impressions to reduce noise in the queries. The total aggregate impression weight of the log is approximately 4.2 billion impressions. The average query length is 3.42 words and 22.04 characters.

The *click log* is available via toolbar usage logs collected from users that have explicitly opted-in to install the toolbar browser add-on of a major search engine. It contains queries and clicks on Amazon products over a period of one year from March 2009 to February 2010. Since our structured data set is in the shopping domain, we were able to map the Amazon click log to our product entities using unique identifier references like UPC and EIN codes. The format of the log is query, entity-id and number of clicks as the query weight. The total number of distinct queries is in the order of 2.2 million with average length of 3.67 and 24.1 characters. It is worth noting that the click log is smaller although the period used is longer. This is attributed to two reasons: there are more queries on a search engine than on Amazon, and the toolbar, as an add-on, means that it can only capture a fraction of the click activity since it is opt-in only and many users do not install such add-ons. However both logs provide very valuable information.

For our experiments, we implemented our work as part of [23] and exploit the query annotator described in [24] to map queries to tables in our collection and provide the token to attribute-value mapping. We run the annotator on our web query log and kept only the interpretation to tables that are above the threshold of $\theta = 1$. Since a query can map to more than one table, we took all the plausible tables and normalized their probabilities to fractions summing to 1. Then we used each fraction to map the query to the table while appropriately adjusting the query weight. Thus we produced a subset of the query log for each table, with each query having a weight that is the corresponding fraction of its impressions multiplied by the normalized table probability.

We tested the various techniques we proposed in the paper and we present the results below. The naming scheme is as follows: *DataProb* is the probabilistic disambiguation from Section 3.1, *ClusterProb* the cluster-based approach from Section 3.2, *Iterative* is the approach described in Section 3.3, and *Clicks* utilizes the click log as described in Section 3.4.

In addition to these techniques, we use a simpler one, shown as *NoProb*, that computes the score using for each query mapped to a table T, all the plausible token-attribute pairs from the table T, while assigning to all of them equally the query weight multiplied by the normalized table weight produced by the annotator. *NoProb* is not blind counting because it does consider the query weight and annotation probability, from [24], that we used in assigning the queries to tables. Thus we consider *NoProb* as a state-of-art baseline. The main difference with our proposed disambiguation techniques is that it does not use an explicit probabilistic disambiguation on the possible token-attribute mappings within each table.

6.1 Relevance Judgments

To measure the effectiveness of our results we conducted an extensive study using Amazon Mechanical Turk. Assessing the entire dataset of all possible tables and all possible attributes would have been prohibitively expensive. Instead we chose a diverse subset of our tables covering a variety of different areas representing as much as possible the entire data spectrum. The tables used in our evaluation were televisions, microwave ovens, cell phones, golf clubs, mp3 players, shoes, laptop computers, printers, watches, video games, engagement rings, digital cameras, skin cleansers, and camera bags and cases.

We ran all our approaches on the full set of tables and produced a set of facet attributes for each table and each algorithm. Then we took the results for the evaluation tables and created a single pool of attributes for judging by merging the results of all algorithms. We posted Mechanical Turk HITs (Human Intelligence Tasks) to obtain user judged relevance for the quality of attributes for search. Users were asked to judge how important a given attribute was for searching within a given category (with the example of faceted search explained) on a three-point scale. The scale items were labeled "highly important", "somewhat important", and "not important."

Producing high quality judgments in a crowd-sourced environment like Mechanical Turk is a challenging problem on its own. We employed a number of methods to ensure quality judgments. First, we created HITs of ten judgments from our result set and we added an extra honey-pot judgment used for spam detection. The additional honey-pot judgment was drawn from a pool of judgments we performed manually, and for which we felt there was a clear "highly

[2]See http://shopping.msn.com/xml/v1/getresults.aspx?text=digital+cameras for for a table of digital cameras and http://shopping.msn.com/xml/v1/getspecs.aspx?-itemid=25236859 for an example of a camera entity with its attributes.

judgements		mturk		
		1	2	3
manual judgments	1	6	15	3
	2	6	34	4
	3	2	17	50

Figure 5: Confusion matrix between mturk and manual judgments.

important" or "not important" answer. The deviation of workers from our gold standard tasks was often a clear indication of spam. However, to ensure fair judgment, we manually inspected the results of any candidate spammer for consistency. The average completion time of a worker was also used as an aid in detecting unreliable workers. Lastly, each task was repeated by nine unique workers. This allowed us to find majority decisions for roughly two-thirds of all attribute judgments. For the remaining third without a majority, we took the average score. The result is a single valuation of the importance of each attribute.

Using these graded judgments we computed the normalized discounted cumulative gain (NDCG) for the ranked output of each approach. Given a list of results L, DCG was computed using linear gain and a logarithmic rank discount as follows.

$$\mathrm{DCG}(L) = \sum_{i=1}^{k} \frac{rel(L_i)}{log_2(i+1)}$$

Where $rel(L_i)$ denotes the judged relevance of result L_i. Relevance scores where assigned from zero to two for the three judgment levels. Let I denote the ideal result ordering (the list of all possible results ordered by judged relevance), then NDCG is defined as follows.

$$\mathrm{NDCG}(L) = \frac{\mathrm{DCG}(L)}{\mathrm{DCG}(I)}$$

We report on various values of k, the number of attributes returned. If a system returned less than k attributes it was penalized with a relevance score of 0 for each missing value.

While we were confident in our quality assurance techniques, we further validated the collection for significant outliers, by manually judging five categories for which we have personal knowledge of the domain through shopping experience. The table in Figure 5 shows the confusion matrix for the judgments. Across the top are the Mechanical Turk judgments, and on the left are the manual judgments. Entry (i, j) is the number of times the manual judgment was i, and the Mechanical Turk was j. We round averaged Mechanical Turk valuations to the nearest integer for this computation. We can see from this that the Mechanical Turk workers often favor the "safe" middle valuation. The effect of averaging judgements that do not reach a majority may also push valuations to the middle. The manual judgments tend to distribute more judgments to the "highly important" and "not important" valuations. Overall, we see exact agreement on 66% of judgments, with opposing 1 vs 3 valuations on only 4%. The Mechanical Turk valuations are therefore somewhat more coarse grained than our careful manual evaluations, but are still similar in overall trend. We have also run all of our experiments over the manual judgments and found the results to be equivalent in terms of trend and rank order of the systems. We omit these graphs for brevity of presentation.

6.2 Attribute Pre-selection with Data Filters

We start with an experiment that measures the effect of data statistics on the produced facets such as selectivity and entropy as described in Section 5. The results are summarized in Figure 6.

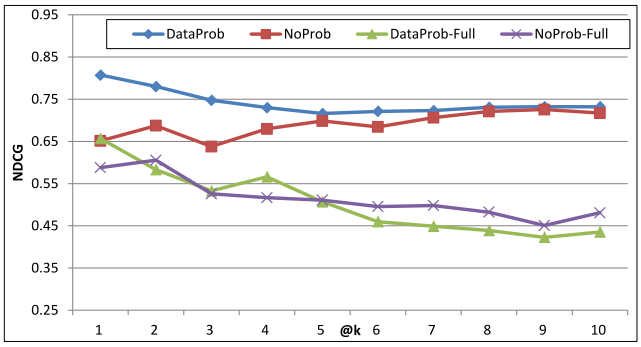


Figure 6: Attribute pre-selection using data filters.

The first step was to take all attributes for each table[3] and perform a run on our two simpler algorithms, shown as *NoProb-Full* and *DataProb-Full* in Figure 6. Surprisingly, *DataProb-Full* does not improve much over *NoProb-Full*. In fact it actually performs worse at higher values of k. Upon further investigation, this is understandable due to how certain attributes affect the computed probabilities. For example, there is an attribute called *video input type* for digital cameras where the value is almost always '*digital camera*'. The probability $P(t|A)$ for the token '*digital camera*' was very high, affecting the disambiguation process whenever it was recognized in a query. As a result the *DataProb-Full* got skewed in counting the proper attribute importance incorrectly. There were other such attributes with similar characteristics that produced a skewed behavior. We observed that such attributes had in common certain data statistics such as low value entropy and low attribute sparsity.

We did a second run where we removed attributes that appear in less than 10% ($R(A) < 0.1$) of the entities and have entropy less than 0.1 ($H(A) < 0.1$). Using this data set we run again the same two algorithms, shown as *NoProb* and *DataProb* in Figure 6. The pre-selection step reduced noise in the data significantly and also removed attributes that have little information content (low entropy) and are not appropriate for facets. As a result the same algorithm with the pre-selected data set perform much better. Now the actual data probabilities used for intra-table attribute disambiguation are more meaningful and *DataProb* performs better than *NoProb*.

Entropy and sparsity can be seen as continuous discount values. We saw in practice that using them as parameterized filters is easier and produces good results. The specific values we used capture the problems with our particular data set and might not be optimal for all data sets. But the point we wish to make is that one has to consider such data filters to pre-select good attribute candidates in the mining algorithms. The full attribute set triggers bad results even on the more complex algorithms. It is not shown here for presentation simplicity. The remainder of the experimental section uses the same pre-selected set of attributes for all our techniques.

6.3 Explicit Disambiguation

Using the above mentioned preselected attribute set we ran all of our disambiguation algorithms. The results are shown in Figure 7. As we discussed in Section 6.2, *DataProb* has clear advantages over *NoProb*. However *DataProb* has the problem of treating attributes independently. For example, in the query "*30 inch television,*" *DataProb* will incorrectly determine the *height* attribute to be most probable over *diagonal screen size*. The confusion happens because televisions in that range generally have diagonals of 32 or 27 inches, whereas large 50 inch televisions tend to have a *height* around 30

[3] We exclude metadata and boolean attributes with values like 'yes/no', 'true/false'

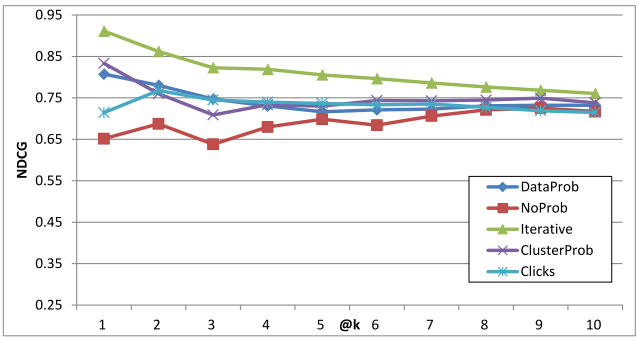


Figure 7: Token-attribute disambiguation comparison

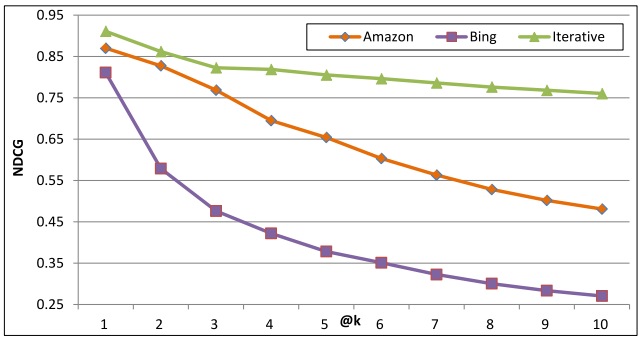


Figure 8: Comparing with Amazon and Bing.

inches. Since there are many 50 inch televisions in the data, *height* is assigned a higher probability for the token '*30 inch.*'

While *ClusterProb* appeared to be an intuitive solution to this issue, we see in practice that it performs no better than *DataProb*. *ClusterProb* works well for categorical attributes, but it actually does even worse on numerical ones for a couple of reasons. First, since user queries do not always reflect the exact values of numeric attributes it tries to map the continuous domain into discrete buckets. That discretization process makes the issue of the query token '*30 inch*' matching the *diagonal* data values 27 or 32 inch even worse than *DataProb*. Because the actual data values vary from the query tokens, the values of the user-intended attribute do not form a good model for the query tokens. Furthermore, KL-divergence is only well defined for distributions that have non-zero probability over the entire value domain, meaning even depth with actual small numeric values is assigned a small probability for large values like 50 inches, thus further enhancing the confusion of numeric attributes.

Iterative tries to detect correlations between data and queries and it exploits the overall bias of users towards the correct attributes. Intuitively, the reason it disambiguates the best is due to how users enter queries. Although the system can be confused with multiple interpretations, users actually know which attributes they are looking for when they enter their values. When examined across the full query log, the user behavior tends to match the intended attribute. For example, although token '*30 inch*' can be consider closer to a TV *height* than *diagonal screen size* for many modern televisions, users query much more frequently for *diagonal screen size* including other values like '*50inch*' or '*55inch*'. This creates a bias for *inch* tokens towards the attribute diagonal, which *Iterative* applies to the '*30 inch*' token. This causes diagonal size to be preferred over height even for that particular value.

One surprise in the disambiguation methods is that *Clicks* do more or less similarly to *DataProb*. At first, one would think that clicks offer the perfect disambiguation method. A user click on a particular entity id allows us to select only the attributes that closely match that entity id disambiguating attributes almost perfectly. However, clicks have problems in the way they were produced that make them less than optimal for learning good facets. First, a user can only click on results shown to them, so clicks incorporate the engine bias and do not represent the true popularity of how many times all possible queries are asked. Hence some attributes are under-represented in a click log due to the engine bias of what results were shown. Second, there are far fewer queries with clicked results than generally asked, meaning many categories have very few queries. This means the mined facets do not have much support and are somewhat erratic. Finally, the queries that tend to trigger clicks are very specific queries looking for one or few products, e.g. "*canon eos 50d digital camera.*" More general category based queries tend to have few clicks and commonly followed by a refinement since they return a large result set that users cannot easily consider, e.g. "*12 megapixel digital camera.*" This creates a skew of the importance learned towards attributes that act as unique keys. However these attributes are not well-suited for facets. If a user already knows the unique key they are looking for (like camera model), they can simply select it from the results and do not need a facet for it. Instead such attributes are better suited for entity snippets, that is summarized views that help users differentiate amongst entities.

6.4 Commercial Faceted Search

Commercial web engines are already supporting faceted search. Although we do not know the technical details of industrial implementations, we felt the best way to test the real world effectiveness of our solution is by comparing against state of the industry engines. Since our data is shopping based, we considered Amazon as the world's most popular shopping engine. Furthermore, since our data comes from the public MSN Shopping API, we considered Bing Shopping (redirects from MSN Shopping) as a shopping engine that is using the same data set as our experiments. Both Amazon and Bing Shopping show facets. To test them, we crawled the attributes shown on Amazon and Bing for our test categories and submitted them in the same pool of attributes that we labeled with mturk judges. To remove any bias, we dropped any affiliation information on the web site. Furthermore, we did not consider the handful of generic shopping attributes shown on both Amazon and Bing, like *price* or *shipping options*. Although very valuable facets there is no need for an automated technique to discover them, as such attributes can easily be added to all categories. Instead we limited our comparison to only *category specific* attributes from each solution. The results are summarized in Figure 8.

Our best solution performs better than Amazon and significantly better than Bing Shopping. For many categories Amazon shows very few attributes and Bing shows even less. As a result, both Amazon and Bing drop dramatically as the size k increases in our experiments. It is important to note that Amazon does show up to 12 category-dependent facets in some cases (e.g., watches) so screen real-estate is not a limiting factor. For small values of k ($k \leq 3$) we still do better but the difference, particularly with Amazon, is smaller. We are not familiar with the details of their facet selection approach and so their techniques cannot be contrasted to ours. However, since they are a popular site with lots of query traffic and domain knowledge, we expect their facet selection to be of the highest caliber. Having our best solution perform even better than Amazon, demonstrates the true value of a pure query-based automated solution and how well it approximates the desired facet

Judgment	Context-depndt	Table-depndt	Equivalent
Preferred	46.3%	23.96%	29.74%

Figure 9: Comparison of facet values for context-dependent vs. category-dependent approach.

utility. Furthermore, it shows that our best method deals more than adequately with ambiguity amongst attributes within each table.

6.5 Values

Our approach populates facets with the most popular values found in the query-log. This is done in a query-independent way, by looking at all values for all attributes as described in Section 4. We report on results using token-level disambiguation, though we find the resulting value orders to be similar with the other techniques as well. So for every category, for the top facets we have computed the top values in a category-dependent list. It would be very expensive to perform an NDCG experiment on values, as the potential judgements needed to capture the value relevancy, in a statistically significant way, is very large. Furthermore, given how we find the popular values, we believe that the quality of those values will correlate with the results of our disambiguation techniques and such NDCG experiments would be moot. However, we wanted to validate the benefit of showing the context-dependent conditional values, which were computed off-line as described in Section 4.2.

We created a synthetic online experiment by generating queries using the top values for the top facets for our test categories. For example, using the *brand* for golf clubs, we generated "*odyssey golf clubs*" or "*callaway golf clubs*" as test queries. For each query, we judged the quality of the context-dependent conditional value list vs. the table-dependent value list for the top facets. The judgement favored one or the other list for each facet, based on the relevance of the values shown given the query. If the values seemed equally relevant, they were judged to be equal. In many cases, the query itself may dictate whether the values are correct or not based on what is present in the data. For example, if the condition specifies the brand '*taylormade*', then a value specific to a '*nike*' product is simply incorrect. Other times, results require specialized knowledge of the domain, for example, that the brand '*odyssey*' is well known for putters or that '*callaway*' is famous for drivers. The results are summarized in Figure 9.

As expected, context-dependent values are preferred more than table-dependent ones. However, at first look, it is surprising to see the table-dependent values gaining so much preference. This happens because context-independent attributes are quite often a super set of the context-dependent ones. The nature of conditional computation has smaller support thus eliminating some values. For example, the conditional list may show values {'*mens*', '*womens*'} while the table-dependent list shows {'*mens*', '*womens*', '*juniors*'} which is perceived as better or more complete. A solution to this is to take the cases where the context-dependent is a subset of the table-dependent list, show the superset but force the conditional ones to the top. We performed the same experiment after we applied this simple heuristic and found that only 1.98% of the queries went to table-dependent whereas the context-dependent jumped to 65.58%. There was no significant change in the equivalent cases, as they are generally similar value sets with small order differences.

7. RELATED WORK

Our problem of discovering important facets in a query-log is related to a number of research areas, including faceted search, attribute ordering, query understanding, and search over structured data. We survey a sample of the relevant work in each of these areas.

Faceted Search The problem of automatically discovering facets (or facet hierarchies) from text has been studied in the past in [5, 6, 18, 25]. The principle challenges in this domain are in finding the terms which can be used to describe (sub)categories, and in building the hierarchy of categories. This is in contrast to our setting, where the actual facets have a direct correspondence to the structured data attributes. The process of extracting facets from text often makes use of structured resources like WordNet [5, 25] or semi-structured sources like Wikipedia [5], but only for categorization and hierarchy extraction, rather than the extraction of attributes. In these works, the facet selection is achieved by ranking based on a navigation cost [6, 18], or by frequency-based metrics [5].

Recently, there is also work on facet selection for structured data [3, 16, 17, 22], which looks for structured data attributes as potential facets. Facet ranking is generally done by estimating the user cost to identify a result when searching with the facets [3], or by ordering the facets according to some property in the data, such as the partitioning balance, cardinality, or attribute frequency [22]. This does not have the benefit of approximating the true user intent as in our query-log based approach. The use of data properties however, is similar to our application of data statistics. In [17] facets are ranked based on a history of user preference for documents. However the facet-value pairs are given explicitly for each document so there are no challenges in resolving ambiguity. In [16], a cost model driven by user actions is approximated using user relevance judgements over attributes. This is in contrast to our scenario, where user relevance is not explicit and must be extracted from query-logs.

Attribute Ordering The problem of ordering or ranking attributes of structured data repositories is a related problem to facet discovery. The goals of attribute ordering problems however, often differ to that of ranking facets. In [7, 15], the aim was to choose a subset attributes that are most "useful" to display to a user when rendering structured search results with limited screen space. Usefulness was formalized as the attributes most influential in the search systems ranking function. A related problem is product visibility, in which attributes are chosen that will distinguish one's product from others in the database. In [21] the product visibility problem was approached by finding the k attributes that cause a product to appear in the maximum number of results over a given query log.

The idea of using a query-log as a source of important attributes has also be studied in previous work. Miah et al. proposed to compute the frequency of the the attribute names in a query log as a signal of importance for the product visibility problem [21]. However, they assume binary attributes, such that each attribute is essentially an attribute/value pair in our data model (e.g., the attribute *four door* describes the boolean property of a car having four doors or not, as opposed to an attribute like *doors* having values '*two*', '*three*', '*four*', or '*five*'). This is different from our approach in that we mine attribute instances from the query-log, and thus have to deal with the ambiguity in mapping instances to their correct attributes. Chaudhuri et al. have also used query-log mining to rank attributes based on their probability of being correlated with a query, with the intention of ranking structured results [4]. However, their work considered structured queries, and thus a structured query-log which does not suffer from ambiguity in query tokens.

Query Understanding A number of recent works have looked at query understanding using structured data [9, 20, 24]. Because our approach rests on having query interpretations, this field is particularly relevant. As part of our experiments we implemented the work in [24] and benefit from it in finding plausible query interpretations and assigning queries to tables. However, the facet discovery problem poses more strenuous requirements than previous query understanding work in disambiguating amongst multiple query to

attribute mappings within the same table. In other words, it is not enough to just find the plausible interpretations of a query on a table. For facet discovery we have to actually be able to distinguish amongst them with high confidence. An approximation of this effect is the results of *NoProb* vs. all the other disambiguation techniques we discussed – *NoProb* can be viewed as using the annotation score by [24] to disambiguate. As we can see, such deep query understanding provides a very good baseline, however our work is different and can be considered an extension to previous work. In fact, we have observed that the attribute importance score learned by our intra-table disambiguation can be used as a signal to further improve the quality of structured annotations described in [24], however further discussion on this is outside the scope of this paper.

A related problem to query understanding is search over structured data, a field that has received a lot of attention in recent years. Search over structured data is also the core motivation for our work of supporting faceted search over structured collections. In this context, each unique mapping of a keyword query into a structured database represents an understanding of the query. Early work focused on how to assemble search results, with ranking based on candidate network size [2, 12], graph-based weight propagation [1], or integrating IR style ranking functions [11]. However, no particular effort has been made in this field to disambiguate among possible mappings of query tokens to attributes.

8. CONCLUSIONS

Structured web search in the form of a unified ecosystem like Google or Bing, or in the form of a specialized vertical like Amazon can take advantage of rich meta-data to produce a better user experience with the appropriate facets. An important problem in this setting is that of selecting the attributes and values for the facet system. Discovering facets must also deal with the scale of data and the restricted screen real estate that only allows for the ones best matching the anticipated user intent.

In this paper we proposed a model that exploits the intersection of web query-logs and structured data to find the facet utility. We showed a variety of techniques that dealt with disambiguating amongst different overlapping attribute-value pairs per query. We extended them to capture values and conditional values that give a dynamic context-dependent facet experience, and we also discussed how data statistics play a role in producing better facets. Finally, we showed experimentally that our approach is scalable, it works well in practice, and even outperforms popular commercial systems that offer facet functionality.

9. REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, P. Parag, and S. Sudarshan. Banks: browsing and keyword searching in relational databases. In *Proc. VLDB Conf.*, pages 1083–1086. VLDB Endowment, 2002.

[2] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proc. ICDE Conf.*, page 5, Washington, DC, USA, 2002. IEEE Computer Society.

[3] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania. Minimum-effort driven dynamic faceted search in structured databases. In *Proc. CIKM*, pages 13–22, 2008.

[4] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.*, 31(3):1134–1168, 2006.

[5] W. Dakka and P. G. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. In *Proc. ICDE Conf.*, pages 466–475, Washington, DC, USA, 2008. IEEE Computer Society.

[6] W. Dakka, P. G. Ipeirotis, and K. R. Wood. Automatic construction of multifaceted browsing interfaces. In *Proc. CIKM*, pages 768–775, New York, NY, USA, 2005. ACM.

[7] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *Proc. SIGMOD Conf.*, pages 395–406, New York, NY, USA, 2006. ACM.

[8] P. Dmitriev, P. Serdyukov, and S. Chernov. Enterprise and desktop search. In *Proc. WWW Conf.*, pages 1345–1346, April 2010.

[9] R. Fagin, B. Kimelfeld, Y. Li, S. Raghavan, and S. Vaithyanathan. Understanding queries in a search database system. In *Proc. PODS Conf.*, pages 273–284, 2010.

[10] M. A. Hearst. *Search User Interfaces*. Cambridge University Press, 2009.

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

[12] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *Proc. VLDB Conf.*, pages 670–681. VLDB Endowment, 2002.

[13] P. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proc. VLDB Conf.*, August 2002.

[14] T. Joachims and F. Radlinski. Search engines that learn from implicit feedback. *IEEE Computer*, 40(8):34–40, 2007.

[15] N. Kapoor, G. Das, V. Hristidis, S. Sudarshan, and G. Weikum. Star: A system for tuple and attribute ranking of query answers. In *Proc. ICDE Conf.*, 2007.

[16] A. Kashyap, V. Hristidis, and M. Petropoulos. Facetor: cost-driven exploration of faceted query results. In *Proc. CIKM Conf.*, 2010.

[17] J. Koren, Y. Zhang, and X. Liu. Personalized interactive faceted search. In *Proc. WWW Conf.*, pages 477–486, 2008.

[18] C. Li, N. Yan, S. B. Roy, L. Lisham, and G. Das. Facetedpedia: dynamic generation of query-dependent faceted interfaces for wikipedia. In *Proc. WWW Conf.*, pages 651–660, 2010.

[19] X. Li, Y.-Y. Wang, and A. Acero. Learning query intent from regularized click graphs. In *Proc. SIGIR Conf.*, pages 339–346, 2008.

[20] X. Li, Y.-Y. Wang, and A. Acero. Extracting Structured Information from User Queries with Semi-supervised Conditional Random Fields. In *Proc. SIGIR Conf.*, pages 572–579, 2009.

[21] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *Proc. ICDE Conf.*, pages 356–365, 2008.

[22] E. Oren, R. Delbru, and S. Decker. Extending faceted navigation for RDF data. *The Semantic Web-ISWC 2006*, pages 559–572, 2006.

[23] S. Paparizos, A. Ntoulas, J. Shafer, and R. Agrawal. Answering web queries using structured data sources. In *Proc. SIGMOD Conf.*, 2009.

[24] N. Sarkas, S. Paparizos, and P. Tsaparas. Structured annotations of web queries. In *Proc. SIGMOD Conf.*, June 2010.

[25] E. Stoica, M. A. Hearst, and M. Richardson. Automating creation of hierarchical faceted metadata structures. *In Proc. Human Language Technology Conf. (NAACL HLT*, 2007.