Structured Annotations of Web Queries

Nikos Sarkas*
University of Toronto
Toronto, ON, Canada
nsarkas@cs.toronto.edu

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

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

ABSTRACT

Queries asked on web search engines often target structured data, such as commercial products, movie showtimes, or airline schedules. However, surfacing relevant results from such data is a highly challenging problem, due to the unstructured language of the web queries, and the imposing scalability and speed requirements of web search. In this paper, we discover latent structured semantics in web queries and produce *Structured Annotations* for them. We consider an annotation as a mapping of a query to a table of structured data and attributes of this table. Given a collection of structured tables, we present a fast and scalable tagging mechanism for obtaining all possible annotations of a query over these tables. However, we observe that for a given query only few are sensible for the user needs. We thus propose a principled probabilistic scoring mechanism, using a generative model, for assessing the likelihood of a structured annotation, and we define a dynamic threshold for filtering out misinterpreted query annotations. Our techniques are completely unsupervised, obviating the need for costly manual labeling effort. We evaluated our techniques using real world queries and data and present promising experimental results.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval
General Terms: Algorithms, Performance, Experimentation
Keywords: keyword search, structured data, web

1. INTRODUCTION

Search engines are evolving from textual information retrieval systems to highly sophisticated answering ecosystems utilizing information from multiple diverse sources. One such valuable source of information is structured data, abstracted as relational tables or XML files, and readily available in publicly accessible data repositories or proprietary databases. Driving the web search evolution are the user needs. With increasing frequency users issue queries that target information that does not reside in web pages, but can be found in structured data sources. Queries about products (e.g., "50 inch LG lcd tv", "orange fendi handbag", "white tiger book"), movie showtime listings (e.g., "indiana jones 4 near boston"), airlines schedules (e.g., "flights from boston to new york"), are only a few examples of queries that are better served using information from structured data, rather than textual content. User scenarios like the ones above are forcing major search engines like Google, Yahoo, Bing and Amazon to look more seriously into web scale search over structured data. However, enabling such functionality poses the following important challenges:

Web speed: Web users have become accustomed to lightning fast responses. Studies have shown that even sub-second delays in returning search results cause dissatisfaction to web users, resulting in query abandonment and loss of revenue for search engines.

Web scale: Users issue over 100 million web queries per day. Additionally, there is an abundance of structured data [2] already available within search engines' ecosystems from sources like crawling, data feeds, business deals or proprietary information. The combination of the two makes an efficient end-to-end solution non trivial.

Free-text queries: Web users targeting structured data express queries in unstructured free-form text without knowledge of schema or available databases. To produce meaningful results, query keywords should be mapped to structure.

For example, consider the query "50 inch LG lcd tv" and assume that there exists a table with information on TVs. One way to handle such a query would be to treat each product as a bag of words and apply standard information retrieval techniques. However, assume that LG does *not* make 50 inch lcd tvs – there is a 46 inch and a 55 inch lcd tv model. Simple keyword search would retrieve nothing. On the other hand, consider a structured query that targets the table "TVs" and specifies the attributes *Diagonal* = "50 inch", *Brand* = "LG", *TV Type* = "lcd tv". Now, the retrieval and ranking system can handle this query with a range predicate on *Diagonal* and a fast selection on the other attributes. This is not an extreme example; most web queries targeting structured data have similar characteristics, incorporating latent structured information. Their evaluation would greatly benefit from structured mappings that expose these latent semantics.

Intent disambiguation: Web queries targeting structured data use the same language as all web queries. This fact violates the underlying closed world assumption of systems that handle keyword queries over structured data, rendering our problem significantly harder. Web users seek information in the open world and issue queries oblivious to the existence of structured data sources, let alone their schema and their arrangement. A mechanism that directly maps keywords to structure can lead to misinterpretations of the user's intent for a large class of queries. There are two possible types of misinterpretations: between web versus structured data, and between individual structured tables.

For example, consider the query "white tiger" and assume there

*Work done while at Microsoft Research.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

is a table available containing Shoes and one containing Books. For "white tiger", a potential mapping can be *Table* = "Shoes" and attributes *Color* = "white" and *Shoe Line* = "tiger", after the popular Asics Tiger line. A different potential mapping can be *Table* = "Books" and *Title* = "white tiger", after the popular book. Although both mappings are possible, it seems that the book is more applicable in this scenario. On the flip side, it is also quite possible the user was asking information that is not contained in our collection of available structured data, for example about "white tiger", the animal. Hence, although multiple structured mappings can be feasible, it is important to determine which one is more plausible among them and which ones are at all meaningful. Such information can greatly benefit overall result quality.

A possible way of addressing all the above challenges would be to send every query to every database and use known techniques from the domain of keyword search over databases or graphs, e.g., [12, 18, 15, 10, 19, 14, 11], to retrieve relevant information. However, it is not clear that such approaches are designed to handle the web speed and scale requirements of this problem space. Web queries are in the order of hundreds of millions per day with only a small fraction really applicable to each particular table. Routing every query to every database can be grossly inefficient. More importantly, the final results surfaced to the web user would still need to be processed via a meta-rank-aggregation phase that combines the retrieved information from the multiple databases and only returns the single or few most relevant. The design of such arbitration phase is not obvious and almost certainly would require some analysis of the query and its mappings to the structured data. In conclusion, we cannot simply apply existing techniques to this problem and address the aforementioned challenges.

Having said that, previous work in this area is not without merit. To address the scenario of web queries targeting structured data, a carefully thought-out end-to-end system has to be considered. Many of the components for such system can be reused from what already exists. For example, once the problem is decomposed into isolated databases, work on structured ranking can be reused. We take advantage of such observations in proposing a solution.

1.1 Our Approach

In this paper, we exploit latent structured semantics in web queries to create mappings to structured data tables and attributes. We call such mappings *Structured Annotations*. For example an annotation for the query "50 inch LG lcd tv" specifies the *Table* = "TVs" and the attributes *Diagonal* = "50 inch", *Brand* = "LG", *TV Type* = "lcd tv". In producing annotations, we assume that all the structured data are given to us in the form of tables. We exploit that to construct a *Closed Structured Model* that summarizes all the table and attributes values and utilize it to deterministically produce all possible annotations efficiently.

However, as we have already demonstrated with query "white tiger", generating all possible annotations is not sufficient. We need to estimate the plausibility of each annotation and determine the one that most likely captures the intent of the user. Furthermore, we need to account for the fact that the users do not adhere to the closed world assumption of the structured data: they use keywords that may not be in the closed structured model, and their queries are likely to target information in the open world.

To handle such problems we designed a principled probabilistic model that scores each possible structured annotation. In addition, it also computes a score for the possibility of the query targeting information outside the structured data collection. The latter score acts as a dynamic threshold mechanism used to expose annotations that correspond to misinterpretations of the user intent.

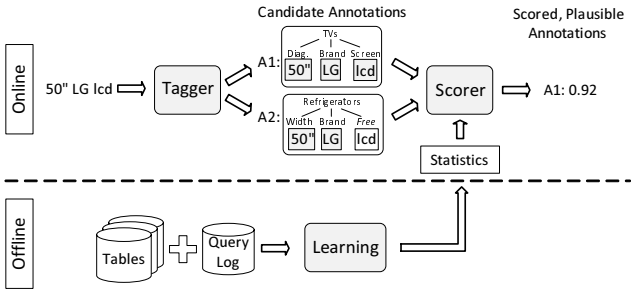

Figure 1: Query Annotator Overview

Model probabilities are learned in an unsupervised fashion on the combination of structured data and query logs. Such data are easily accessible within a search engine ecosystem.

The result is a *Query Annotator* component, shown in Figure 1. It is worth clarifying that we are not solving the end to end problem for serving structured data to web queries. That would include other components such as indexing, data retrieval and ranking. Our *Query Annotator* component sits on the frond end of such end-to-end system. Its output can be utilized to route queries to appropriate tables and feed annotation scores to a structured data ranker.

Our contributions with respect to the challenges of web search over structured data are as follows.

1. **Web speed:** We design an efficient tokenizer and tagger mechanism producing annotations in milliseconds.
2. **Web scale:** We map the problem to a decomposable closed world summary of the structured data that can be done in parallel for each structured table.
3. **Free-text queries:** We define the novel notion of a Structured Annotation capturing structure from free text. We show how to implement a process producing all annotations given a closed structured data world.
4. **Intent disambiguation:** We describe a scoring mechanism that sorts annotations based on plausibility. Furthermore, we extend the scoring with a dynamic threshold, derived from the probability a query was not described by our closed world.

The rest of the paper is organized in the following way. We describe the closed structured world and *Structured Annotations* in Section 2. We discuss the efficient tokenizer and tagger process that deterministically produces all annotations in Section 3. We define a principled probabilistic generative model used for scoring the annotations in Section 4 and we discuss unsupervised model parameter learning in Section 5. We performed a thorough experimental evaluation with very promising results, presented in Section 6. We conclude the paper with a discussion of related work in Section 7 and some closing comments in Section 8.

2. STRUCTURED ANNOTATIONS

We start our discussion by defining some basic concepts. A *token* is defined as a sequence of characters including space, i.e., one or more words. For example, the bigram "digital camera" may be a single token. We define the *Open Language Model* (OLM) as the infinite set of all possible tokens. All keyword web queries can be expressed using tokens from OLM.

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 *en-*

[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.

tities sharing a set of *attributes*. We denote the attributes of table T as $T.\mathcal{A} = \{T.A_1, T.A_2, \ldots, T.A_\alpha\}$. Attributes can be either *categorical* or *numerical*. The *domain* of a categorical attribute $T.A_c \in T.\mathcal{A}_c$, i.e., the set of possible values that $T.A_c$ can take, is denoted with $T.A_c.\mathcal{V}$. We assume that each numerical attribute $T.A_n \in T.\mathcal{A}_n$ is associated with a single *unit* U of measurement. Given a set of units $\mathcal{U}$ we define $\mathrm{Num}(\mathcal{U})$ to be the set of all tokens that consist of a numerical value followed by a unit in $\mathcal{U}$. Hence, the *domain* of a numerical attribute $T.A_n$ is $\mathrm{Num}(T.A_n.U)$ and the *domain* of all numerical attributes $T.\mathcal{A}_n$ in a table is $\mathrm{Num}(T.\mathcal{A}_n.\mathcal{U})$.

An example of two tables is shown in Figure 2. The first table contains TVs and the second Monitors. They both have three attributes: Type, Brand and Diagonal. Type and Brand are categorical, whereas Diagonal is numerical. The domain of values for all categorical attributes for both tables is $\mathcal{T}.\mathcal{A}_c.\mathcal{V}$ = {TV, Samsung, Sony, LG, Monitor, Dell, HP}. The domain for the numerical attributes for both tables is $\mathrm{Num}(\mathcal{T}.\mathcal{A}_n.\mathcal{U})$ = $\mathrm{Num}(\{\mathrm{inch}\})$. Note that $\mathrm{Num}(\{\mathrm{inch}\})$ does not include only the values that appear in the tables of the example, but rather all possible numbers followed by the unit "inch". Additionally, note that it is possible to extend the domains with synonyms, e.g., by using "in" for "inches" and "Hewlett Packard" for "HP". Discovery of synonyms is beyond the scope of this paper, but existing techniques [21] can be leveraged.

We now give the following definitions.

DEFINITION 1 (TYPED TOKEN). *A* typed token t *for table* T *is any value from the* domain *of* $\{T.\mathcal{A}_c.\mathcal{V} \cup \mathrm{Num}(T.\mathcal{A}_n.\mathcal{U})\}$.

DEFINITION 2 (CLOSED LANGUAGE MODEL). *The* Closed Language Model *CLM of table* T *is the set of all duplicate-free typed tokens for table* T.

For the rest of the paper, for simplicity, we often refer to *typed tokens* as just *tokens*. The closed language model $\mathrm{CLM}(T)$ contains the duplicate-free set of all tokens associated with table T. Since for numerical attributes we only store the "units" associated with $\mathrm{Num}(\mathcal{U})$ the representation of $\mathrm{CLM}(T)$ very compact.

The closed language model $\mathrm{CLM}(\mathcal{T})$ for all our structured data $\mathcal{T}$ is defined as the union of the closed language models of all tables. Furthermore, by definition, if we break a collection of tables $\mathcal{T}$ into k sub-collections $\{\mathcal{T}_1, ..., \mathcal{T}_k\}$, then $\mathrm{CLM}(\mathcal{T})$ can be decomposed into $\{\mathrm{CLM}(\mathcal{T}_1), ..., \mathrm{CLM}(\mathcal{T}_k)\}$. In practice, $\mathrm{CLM}(\mathcal{T})$ is used to identify tokens in a query that appear in the tables of our collection. So compactness and decomposability are very important features that address the web speed and web scale challenges.

The closed language model defines the set of tokens that are associated with a collection of tables, but it does not assign any *semantics* to these tokens. To this end, we define the notion of an *annotated token* and *closed structured model*.

DEFINITION 3 (ANNOTATED TOKEN). *An* annotated token *for a table* T *is a pair* $AT = (t, T.A)$ *of a token* $t \in \mathrm{CLM}(T)$ *and an attribute* A *in table* T*, such that* $t \in T.A.\mathcal{V}$.

For an annotated token $AT = (t, T.A)$, we use $AT.t$ to refer to underlying token t. Similarly, we use $AT.T$ and $AT.A$ to refer to the underlying table T and attribute A. Intuitively, the *annotated token* AT assigns structured semantics to a token. In the example of Figure 2, the annotated token (LG, TVs.Brand) denotes that the token "LG" is a possible value for the attribute TVs.Brand.

DEFINITION 4 (CLOSED STRUCTURED MODEL). *The* Closed Structured Model *of table* T, $\mathrm{CSM}(T) \subseteq \mathrm{CLM}(T) \times T.\mathcal{A}$, *is the set of all annotated tokens for table* T.

TVs		
Type	Brand	Diagonal
TV	Samsung	46 inch
TV	Sony	60 inch
TV	LG	26 inch

Monitors		
Type	Brand	Diagonal
Monitor	Samsung	24 inch
Monitor	Dell	12 inch
Monitor	HP	32 inch

Figure 2: A two-table example

Note that in the example of Figure 2, the annotated token (LG, TVs.Brand) for $\mathrm{CSM}(\mathrm{TVs})$ is different from the annotated token (LG, Monitors.Brand) for $\mathrm{CSM}(\mathrm{Monitors})$, despite the fact that in both cases the name of the attribute is the same, and the token "LG" appears in the closed language model of both TVs and Monitors table. Furthermore, the annotated tokens (50 inch, TVs.Diagonal) and (15 inch, TVs.Diagonal) are part of for $\mathrm{CSM}(\mathrm{TVs})$, despite the fact that table TVs does not contain entries with those values.

The closed structured model for the collection $\mathcal{T}$ is defined as the union of the structured models for the tables in $\mathcal{T}$. In practice, $\mathrm{CSM}(\mathcal{T})$ is used to map all recognized tokens $\{t_1, ..., t_n\}$ from a query q to tables and attributes $\{T_1.A_1, ..., T_n.A_n\}$. This is a fast lookup process as annotated tokens can be kept in a hash table. To keep a small memory footprint, $\mathrm{CSM}(\mathcal{T})$ can be implemented using token pointers to $\mathrm{CLM}(\mathcal{T})$, so the actual values are not replicated. As before with CLM, $\mathrm{CSM}(\mathcal{T})$ is decomposable to smaller collections of tables. Fast lookup, small memory footprint and decomposability help with web speed and web scale requirements of our approach.

We are now ready to proceed with the definition of a *Structured Annotation*. But first, we introduce an auxiliary notion that simplifies the definition. For a query q, we define a *segmentation* of q, as the set of tokens $G = \{t_1, ..., t_k\}$ for which there is a permutation π, such that $q = t_{\pi(1)}, ..., t_{\pi(k)}$, i.e., the query q is the sequence of the tokens in G. Intuitively, a segmentation of a query is a sequence of non-overlapping tokens that cover the entire query.

DEFINITION 5 (STRUCTURED ANNOTATION). *A structured annotation* S_q *of query* q *over a table collection* $\mathcal{T}$*, is a triple* $\langle T, \mathcal{AT}, \mathcal{FT}\rangle$*, where* T *denotes a table in* $\mathcal{T}$*,* $\mathcal{AT} \subseteq \mathrm{CSM}(T)$ *is a set of annotated tokens, and* $\mathcal{FT} \subseteq \mathrm{OLM}$ *is a set of words such that* $\{\mathcal{AT}.t, \mathcal{FT}\}$ *is a segmentation of* q.

A *structured annotation*[2] $S_q = \langle T, \mathcal{AT}, \mathcal{FT}\rangle$ of query q is a mapping of the user-issued keyword query to a structured data table T, a subset of its attributes $\mathcal{AT}.A$, and a set of *free tokens* $\mathcal{FT}$ of words from the open language model. Intuitively, it corresponds to an interpretation of the query as a request for some entities from table T. The set of annotated tokens $\mathcal{AT}$ expresses the characteristics of T's entities requested, as pairs $(t_i, T.A_i)$ of a table attribute $T.A_i$ and a specific attribute value t_i. The set of free tokens $\mathcal{FT}$ is the portion of the query that cannot be associated with an attribute of table T. Annotated and free tokens together cover all the words in the query, defining complete segmentation of q.

One could argue that it is possible for a query to target more than one table and the definition of a structured annotation does not cover this case. For example, query "chinese restaurants in san francisco" could refer to a table of Restaurants and one of Locations. We could extend our model and annotation definitions to support multiple tables, but for simplicity we choose to not to, since the single-table problem is already a complex one. Instead, we assume that such tables have been joined into one materialized view.

Now, consider the keyword query q ="50 inch LG lcd". Assume that we have a collection $\mathcal{T}$ of three tables over TVs, Monitors,

[2]For convenience we will often use the terms *annotation*, *annotated query* and *structured query* to refer to a structured annotation. The terms are synonymous and used interchangeably throughout the paper.

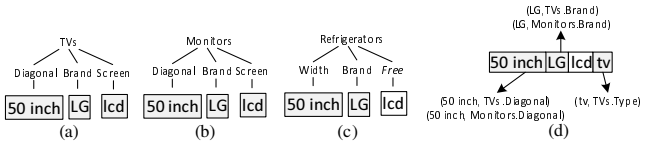

Figure 3: Examples of annotations and annotation generation.

and Refrigerators, and there are three possible possible annotations $\langle T, \mathcal{AT}, \mathcal{FT} \rangle$ of q (shown in Figure 3(a-c)):

(a) $S_1 = \langle \text{TVs}, \{(\text{50 inch, TVs.Diagonal}), (\text{LG, TVs.Brand}), (\text{lcd, TVs.Screen})\}, \{\}\rangle$
(b) $S_2 = \langle \text{Monitors}, \{(\text{50 inch, Monitors.Diagonal}), (\text{LG, Monitors.Brand})\}, (\text{lcd, Monitors.Screen}), \{\}\rangle$
(c) $S_3 = \langle \text{Refrigerators}, \{(\text{50 inch, Refrigerators.Width}), (\text{LG, Refrigerators.Brand})\}, \{\text{lcd}\}\rangle$

The example above highlights the challenges discussed in Section 1. The first challenge is how to efficiently derive all possible annotations. As the size and heterogeneity of the underlying structured data collection increases, so does the number of possible structured annotations per query. For instance, there can be multiple product categories manufactured by "LG" or have an attribute measured in "inches". This would result in an even higher number of structured annotations for the example query q ="50 inch LG lcd". Hence, efficient generation of all structured annotations of a query is a highly challenging problem.

PROBLEM 1 (ANNOTATION GENERATION). *Given a keyword query q, generate the set of all* structured annotations $\mathcal{S}_q = S_1, \ldots, S_k$ *of query q.*

Second, it should be clear from our previous example that although many structured annotations are possible, only a handful, if any, are *plausible* interpretations of the keyword query. For instance, annotation S_1 (Figure 3(a)) is a perfectly sensible interpretation of q. This is not true for annotations S_2 and S_3. S_2 maps the *entire* keyword query to table Monitors, but it is highly unlikely that a user would request Monitors with such characteristics, i.e., (50 inch, Monitors.Diagonal), as users are aware that no such large monitors exist (yet?). Annotation S_3 maps the query to table Refrigerators. A request for Refrigerators made by LG and a Width of 50 inches is sensible, but it is extremely unlikely that a keyword query expressing this request would include free token "lcd", which is irrelevant to Refrigerators. Note that the existence of free tokens does not necessarily make an annotation implausible. For example, for the query "50 inch lcd *screen* LG", the free token "screen" increases the plausibility of the annotation that maps the query to the table TVs. Such subtleties demand a robust scoring mechanism, capable of eliminating implausible annotations and distinguishing between the (potentially many) plausible ones.

PROBLEM 2 (ANNOTATION SCORING). *Given a set of candidate annotations $\mathcal{S}_q = S_1, \ldots, S_k$ for a query q, define a score $f(S_i)$ for each annotation S_i, and determine the* plausible *ones satisfying $f(S_i) > \theta_q$, where θ_q is a query-specific threshold.*

We address the Annotation Generation problem in Section 3, and the Annotation Scoring problem in Sections 4 and 5.

3. PRODUCING ANNOTATIONS

The process by which we map a web query q to *Structured Annotations* involves two functions: a tokenizer fTOK and an tagger fTAG. The tokenizer maps query q to a set of annotated tokens $\mathcal{AT}_q \subseteq CSM(\mathcal{T})$ from the set of all possible annotated tokens in the closed structured model of the dataset. The tagger consumes the query q and the set of annotated tokens AT_q and produces a set of structured annotations $\mathcal{S}_q$.

Algorithm 1 Tokenizer

Input: A query q represented as an array of words $q[1, \ldots, \text{length}(q)]$
Output: An array $\mathcal{AT}$, such that for each position i of q, $\mathcal{AT}[i]$ is the list of annotated tokens beginning at i; A list of free tokens $\mathcal{FT}$.

for $i = 1 \ldots \text{length}(q)$ **do**
  Compute the set of annotated tokens $\mathcal{AT}[i]$ starting at position i of the query.
  Add word $q[i]$ to the list of free tokens $\mathcal{FT}$.
return Array of annotated tokens $\mathcal{AT}$ and free tokens $\mathcal{FT}$.

Tokenizer: The tokenizer procedure is shown in Algorithm 1. The tokenizer consumes one query and produces all possible annotated tokens. For example, consider the query "50 inch LG lcd tv" and suppose we use the tokenizer over the dataset in Figure 3. Then the output of the tokenizer will be $\text{fTOK}(q)$ ={(50 inch, TVs.Diagonal), (50 inch, Monitors.Diagonal), (LG, Monitors. Brand), (LG, TVs.Brand), (tv, TVs.Type)} (Figure 3(d)). The token "lcd" will be left unmapped, since it does not belong to the language model $\text{CLM}(\mathcal{T})$.

In order to impose minimal computational overhead when parsing queries, the tokenizer utilizes a highly efficient and compact string dictionary, implemented as a Ternary Search Tree (TST) [1]. The main-memory TST is a specialized key-value dictionary with well understood performance benefits. For a collection of tables $\mathcal{T}$, the Ternary Search Tree is loaded with the duplicate free values of categorical attributes and list of units of numerical attributes. So semantically TST stores $\mathcal{T}.\mathcal{A}_c.\mathcal{V} \cup \mathcal{T}.\mathcal{A}_n.\mathcal{U}$.

For numbers, a regular expressions matching algorithm is used to scan the keyword query and make a note of all potential numeric expressions. Subsequently, terms adjacent to a number are looked-up in the ternary search tree in order to determine whether they correspond to a relevant *unit* of measurement, e.g., "inch", "GB", etc. If that is the case, the number along with the unit-term are grouped together to form a typed token.

For every parsed typed token t, the TST stores pointers to all the attributes, over all tables and attributes in the collection that contain this token as a value. We thus obtain the set of all annotated tokens $\mathcal{AT}$ that involve token t. The tokenizer maps the query q to the closed structured model $\text{CSM}(\mathcal{T})$ of the collection. Furthermore, it also outputs a free token for every word in the query. Therefore, we have that $\text{fTOK}(q) = \{\mathcal{AT}_q, \mathcal{FT}_q\}$, where $\mathcal{AT}_q$ is the set of all possible annotated tokens in q over all tables, and $\mathcal{FT}_q$ is the set of words in q, as free tokens.

Tagger: We will now describe how the tagger works. For that we need to first define the notion of *maximal annotation*.

DEFINITION 6. *Given a query q, and the set of all possible annotations $\mathcal{S}_q$ of query q, annotation $S_q = \langle T, \mathcal{AT}, \mathcal{FT} \rangle \in \mathcal{S}_q$ is* maximal, *if there exists no annotation $S'_q = \langle T', \mathcal{AT}', \mathcal{FT}' \rangle \in \mathcal{S}_q$ such that $T = T'$ and $\mathcal{AT} \subset \mathcal{AT}'$ and $\mathcal{FT} \supset \mathcal{FT}'$.*

The *tagger* fTAG is a function that takes as input the set of annotated and free tokens $\{\mathcal{AT}_q, \mathcal{FT}_q\}$ of query q and outputs the set of all maximal annotations $\text{fTOK}(\{\mathcal{AT}_q, \mathcal{FT}_q\}) = \mathcal{S}^*_q$. The procedure of the tagger is shown in Algorithms 2 and 3. The algorithm first partitions the annotated tokens per table, decomposing the problem to smaller subproblems. Then, for each table it constructs the candidate annotations by scanning the query from left to right, each time appending an annotated or free token to the end on an existing annotation, and then recursing on the remaining uncovered query. This process produces all valid annotations. We

Algorithm 2 Tagger

Input: An array $\mathcal{AT}$, such that for each position i of q, $\mathcal{AT}[i]$ is the list of annotated tokens beginning at i; A list of free tokens $\mathcal{FT}$.
Output: A set of structured annotations $\mathcal{S}$

Partition the lists of annotated tokens per table.
for each table T **do**
  $\mathcal{L} = \mathbf{ComputeAnnotations}(\mathcal{AT}_T, \mathcal{FT}, 0)$
  Eliminate non-maximal annotations from $\mathcal{L}$
  $\mathcal{S} = \mathcal{S} \cup \mathcal{L}$
return $\mathcal{S}$

Algorithm 3 ComputeAnnotation

Input: An array $\mathcal{AT}$, such that $\mathcal{AT}[i]$ is the list of annotated tokens; A list of free tokens $\mathcal{FT}$; A position k in the array $\mathcal{AT}$.
Output: A set of structured annotations $\mathcal{S}$ using annotated and free tokens from $\mathcal{AT}[j]$, $\mathcal{FT}[j]$ for $j \geq k$.

if $k > \text{length}(\mathcal{AT})$ **then**
  return $\emptyset$
Initialize $\mathcal{S} = \emptyset$
for each annotated or free token $AFT \in (\mathcal{AT}[k] \cup \mathcal{FT}[k])$ **do**
  $k' = k + \text{length}(AFT.t)$
  $\mathcal{L} = \mathbf{ComputeAnnotation}(\mathcal{AT}, \mathcal{FT}, k')$
  for each annotation S in $\mathcal{L}$ **do**
    $S = \{AFT, S\}$
    $\mathcal{S} = \mathcal{S} \cup S$
return $\mathcal{S}$

perform a final step to remove the non-maximal annotations. This can be done efficiently in a single pass: each annotation needs to be checked against the "current" set of maximal annotations, as in skyline computations. It is not hard to show that this process will produce all possible maximal annotations.

LEMMA 1. *The tagger produces all possible maximal annotations $\mathcal{S}_q^*$ of a query q over a closed structured model* CSM($\mathcal{T}$).

As a walk through example consider the query "50 inch LG lcd tv", over the data in Figure 2. The input to the tagger is the set of all annotated tokens $\mathcal{AT}_q$ computed by the tokenizer (together with the words of the query as free tokens). This set is depicted in Figure 3(d). A subset of possible annotations for q is:

S_1 = ⟨TVs,{(50 inch,TVs.Diagonal)},{LG, lcd, tv}⟩
S_2 = ⟨TVs,{(50 inch,TVs.Diagonal),(LG, TVs.Brand)}, {lcd, tv}⟩
S_3 = ⟨TVs,{(50 inch,TVs.Diagonal),(LG, TVs.Brand),(tv, TVs.Type)}, {lcd}⟩
S_4 = ⟨Monitors,{(50 inch,Monitors.Diagonal)}, {LG, lcd, tv}⟩
S_5 = ⟨Monitors,{(50 inch,Monitors.Diagonal),(LG, Monitors.Brand)}, {lcd, tv}⟩

Out of these annotations, S_3 and S_5 are maximal, and they are returned by the tagger function. Note that the token "lcd" is always in the free token set, while "tv" is a free token only for Monitors.

4. SCORING ANNOTATIONS

For each keyword query q, the tagger produces the list of all possible structured annotations $\mathcal{S}_q = \{S_1, ..., S_k\}$ of query q. This set can be large, since query tokens can match the attribute domains of multiple tables. However, it is usually quite unlikely that the query was actually intended for all these tables. For example, consider the query "LG 30 inch screen". Intuitively, the query most likely targets TVs or Monitors, however a structured annotation will be generated for all tables that contain any product of LG (DVD players, cell phones, cameras, etc.), as well as all tables with attributes measured in inches.

It is thus clear that there is a need for computing a *score* for the annotations generated by the tagger that captures how "likely" an annotation is. This is the responsibility of the scorer function, which given the set of all annotations $\mathcal{S}_q$, it outputs for each annotation $S_i \in \mathcal{S}_q$ the probability $P(S_i)$ of a user requesting the information captured by the annotation. For example, it is unlikely that query "LG 30 inch screen", targets a DVD player, since most of the times people do not query for the dimensions of a DVD player and DVD players do not have a screen. It is also highly unlikely that the query refers to a camera or a cell phone, since although these devices have a screen, its size is significantly smaller.

We model this intuition using a *generative probabilistic model*. Our model assumes that users "generate" an annotation S_i (and the resulting keyword query) as a two step process. First, with probability $P(T.\mathcal{A}_i)$, they decide on the table T and the subset of its attributes $T.\mathcal{A}_i$ that they want to query, e.g., the product type and the attributes of the product. Since the user may also include free tokens in the query, we extend the set of attributes of each table T with an additional attribute $T.f$ that emits free tokens, and which may be included in the set of attributes $T.\mathcal{A}_i$. For clarity, we use $T.\tilde{\mathcal{A}}_i$ to denote a subset of attributes taken over this extended set of attributes, while $T.\mathcal{A}_i$ to denote the subset of attributes from the table T. Note that similar to every other attribute of table T, the free-token attribute $T.f$ can be repeated multiple times, depending on the number of free tokens added to the query.

In the second step, given their previous choice of attributes $T.\tilde{\mathcal{A}}_i$, users select specific annotated and free tokens with probability $P(\{\mathcal{AT}_i, \mathcal{FT}_i\}|T.\tilde{\mathcal{A}}_i)$. Combining the two steps, we have:

$$P(S_i) = P(\{\mathcal{AT}_i, \mathcal{FT}_i\}|T.\tilde{\mathcal{A}}_i)P(T.\tilde{\mathcal{A}}_i) \quad (1)$$

For the "LG 30 inch screen" example, let $S_i = \langle$ TVs, {(LG, TVs.Brand), (30 inch, TVs.Diagonal)},{screen}$\rangle$ be an annotation over the table TVs. Here the set of selected attributes is {TVs.Brand, TVs.Diagonal, TVs.f}. We thus have:

$$\begin{aligned} P(S_i) &= P(\{\text{LG, 30 inch}\},\{\text{screen}\}|(\text{Brand, Diagonal}, f)) \\ &\quad \cdot P(\text{TVs.Brand, TVs.Diagonal, TVs.}f) \end{aligned}$$

In order to facilitate the evaluation of Equation 1 we make some simplifying but reasonable assumptions. First, that the sets of annotated $\mathcal{AT}_i$ and free $\mathcal{FT}_i$ tokens are independent, conditional on the set of attributes $T.\tilde{\mathcal{A}}_i$ selected by the user, that is:

$$P(\{\mathcal{AT}_i, \mathcal{FT}_i\}|T.\tilde{\mathcal{A}}_i) = P(\mathcal{AT}_i|T.\tilde{\mathcal{A}}_i)P(\mathcal{FT}_i|T.\tilde{\mathcal{A}}_i)$$

Second, we assume that the free tokens $\mathcal{FT}_i$ do not depend on the exact attributes $T.\tilde{\mathcal{A}}_i$ selected by the user, but only on the table T that the user decided to query. That is, $P(\mathcal{FT}_i|T.\tilde{\mathcal{A}}_i) = P(\mathcal{FT}_i|T)$. For example, the fact that the user decided to add the free token "screen" to the query depends only on the fact that she decided to query the table TVs, and not on the specific attributes of the TVs table that she decided to query.

Lastly, we also assume that the annotated tokens $\mathcal{AT}_i$ selected by a user do not depend on her decision to add a free token to the query, but instead only on the attributes $T.\mathcal{A}_i$ of the table that she queried. That is, $P(\mathcal{AT}_i|T.\tilde{\mathcal{A}}_i) = P(\mathcal{AT}_i|T.\mathcal{A}_i)$. In our running example, this means that the fact that the user queried for the brand "LG", and the diagonal value "30 inches", does not depend on the decision to add a free token to the query.

Putting everything together, we can rewrite Equation 1 as follows:

$$P(S_i) = P(\mathcal{AT}_i|T.\mathcal{A}_i)P(\mathcal{FT}_i|T)P(T.\tilde{\mathcal{A}}_i) \quad (2)$$

Given the annotation set $\mathcal{S}_q = \{S_1, ..., S_k\}$ of query q, the scorer function uses Equation 2 to compute the probability of each annotation. In Section 5 we describe how given an annotation S_i we obtain estimates for the probabilities involved in Equation 2.

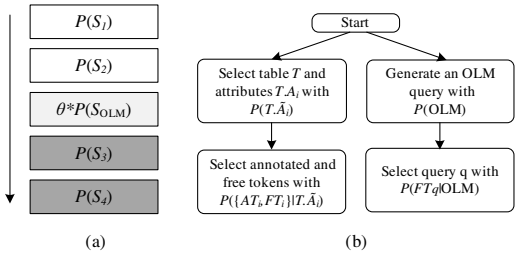

Figure 4: The scorer component.

The probabilities allow us to discriminate between less and more likely annotations. However, this implicitly assumes that we operate under a closed world hypothesis, where all of our queries are targeting some table in the structured data collection $\mathcal{T}$. This assumption is incompatible with our problem setting where users issue queries through a web search engine text-box and are thus likely to compose web queries using an open language model targeting information outside $\mathcal{T}$. For example, the query "green apple" is a fully annotated query, where token "green" corresponds to a Color, and "apple" to a Brand. However, it seems more likely that this query refers to the fruit, than any of the products of Apple. We thus need to account for the case that the query we are annotating is a regular web query not targeting the structured data collection.

Our generative model can easily incorporate this possibility in a consistent manner. We define the open-language "table" OLM which is meant to capture open-world queries. The OLM table has only the free-token attribute $\mathrm{OLM}.f$ and generates all possible free-text queries. We populate the table using a generic web query log. Let $\mathcal{FT}_q$ denote the free-token representation of a query q. We generate an additional annotation $S_{\mathrm{OLM}} = \langle \mathrm{OLM}, \{\mathcal{FT}_q\}\rangle$, and we evaluate it together with all the other annotations in $\mathcal{S}_q$. Thus the set of annotations becomes $\mathcal{S}_q = \{S_1, ..., S_k, S_{k+1}\}$, where $S_{k+1} = S_{\mathrm{OLM}}$, and we have:

$$P(S_{\mathrm{OLM}}) = P(\mathcal{FT}_q|\mathrm{OLM})P(\mathrm{OLM}) \quad (3)$$

The S_{OLM} annotation serves as a "control" against which all candidate structure annotations need to measured. The probability $P(S_{\mathrm{OLM}})$ acts as an adaptive threshold which can be used to filter out *implausible* annotations, whose probability is not high enough compared to $P(S_{\mathrm{OLM}})$. More specifically, for some $\theta > 0$, we say that a structured annotation S_i is *plausible* if $\frac{P(S_i)}{P(S_{\mathrm{OLM}})} > \theta$. In other words, an annotation, which corresponds to an interpretation of the query as a request which can be satisfied using structured data, is considered plausible if it is *more probable* than the open-language annotation, which captures the absence of demand for structured data. On the other hand, implausible annotations are *less probable* than the open-language annotation, which suggests that they correspond to misinterpretations of the keyword query.

The value of θ is used to control the strictness of the plausibility condition. The scorer outputs only the set of plausible structured annotations (Figure 4(a)). Notice that multiple plausible annotations are both possible and desirable. Certain queries are naturally ambiguous, in which case it is sensible to output more than one plausible annotations. For example, the query "LG 30 inch screen" can be targeting either TVs or Monitors.

5. LEARNING THE GENERATIVE MODEL

In order to fully specify the generative model described in Section 4 and summarized in Figure 4(b), we need to describe how to obtain estimates for the probabilities $P(\mathcal{AT}_i|T.\mathcal{A}_i)$, $P(\mathcal{FT}_i|T)$, and $P(T.\tilde{\mathcal{A}}_i)$ in Equation 2 for every annotation S_i in $\mathcal{S}_q$, as well as $P(\mathcal{FT}_q|\mathrm{OLM})$ and $P(\mathrm{OLM})$ in Equation 3 for the open language annotation S_{OLM}. In order to guarantee highly efficient annotation scoring, these estimates need to be pre-computed off-line, while to guarantee scoring precision, the estimates need also be accurate.

5.1 Estimating token-generation probabilities

Generating Annotated Tokens.

We need to compute the conditional probability $P(\mathcal{AT}_i|T.\mathcal{A}_i)$, that is, the probability that the query q on table T and attributes $T.\mathcal{A}_i$ contains a specific combination of values for the attributes. A reasonable estimate of the conditional probability is offered by the fraction of table entries that actually contain the values that appear in the annotated query. Let $\mathcal{AT}_i.\mathcal{V}$ denote the set of attribute values associated with annotated tokens $\mathcal{AT}_i$. Also, let $T(\mathcal{AT}_i.\mathcal{V})$ denote the set of entries in T where the attributes in $T.\mathcal{A}_i$ take the combination of values $\mathcal{AT}_i.\mathcal{V}$. We have:

$$P(\mathcal{AT}_i|T.\mathcal{A}_i) = \frac{|T(\mathcal{AT}_i.\mathcal{V})|}{|T|}$$

For example, consider the query "50 inch LG lcd", and the annotation $S = \langle$ TVs, {(LG,TVs.Brand),(50 inch, TVs.Diagonal)},{lcd}$\rangle$. We have $T.\mathcal{A} = \{\mathrm{Brand}, \mathrm{Diagonal}\}$ and $\mathcal{AT}.\mathcal{V} = \{\mathrm{LG}, 50\mathrm{inch}\}$. The set $T(\mathcal{AT}.\mathcal{V})$ is the set of all televisions in the TVs table of brand LG with diagonal size 50 inch, and $P(\mathcal{AT}|T.\mathcal{A})$ is the fraction of the entries in the TVs table that take these values.

Essentially, our implicit assumption behind this estimate is that *attribute values appearing in annotated queries and attribute values in tables follow the same distribution*. For example, if a significant number of entries in the TVs table contains brand LG, this is due to the fact that LG is popular among customers. On the other hand, only a tiny fraction of products are of the relatively obscure and, hence, infrequently queried brand "August".

Similarly, we can expect few queries for "100 inch" TVs and more for "50 inch" TVs. That is, large TVs represent a niche, and this is also reflected in the composition of table TVs. Additionally, we can expect practically no queries for "200 inch" TVs, as people are aware that no such large screens exist (yet?). On the other hand, even if there are no TVs of size 33 inches in the database, but TVs of size 32 inches and 34 inches do exist, this is an indication that 33 may be a reasonable size to appear in a query.

Of course, there is no need to actually issue the query over our data tables and retrieve its results in order to determine conditional probability $P(\mathcal{AT}|T.\mathcal{A})$. Appropriate, lightweight statistics can be maintained and used, and the vast literature on *histogram construction* [13] and *selectivity estimation* [20] can be leveraged for this purpose. In this work, we assume by default independence between the different attributes. If $T.\mathcal{A} = \{T.A_1, ..., T.A_a\}$ are the attributes that appear in the annotation of the query, and $\mathcal{AT} = \{(T.A_1.v, T.A_1), ..., (T.A_a.v, T.A_a)\}$ are the annotated tokens, then we have:

$$P(\mathcal{AT}|T.\mathcal{A}) = \prod_{j=1}^{a} P(T.A_j.v|T.A_j)$$

For the estimation of $P(T.A_j.v|T.A_j)$, for categorical attributes, we maintain the fraction of table entries matching each domain value. For numerical attributes, a histogram is built instead, which is used as an estimate of the probability density function of the values for this attribute. In that case, the probability of a numerical attribute value v is computed as the fraction of entities with values in range $[(1-\epsilon)v, (1+\epsilon)v]$ (we set $\epsilon = 0.05$ in our implementation). The resulting data structures storing these statistics are extremely compact and amenable to efficient querying.

In the computation of $P(\mathcal{AT}|T.\mathcal{A})$, we can leverage information we have about synonyms or common misspellings of attribute values. Computation of the fraction of entries in table T that contain a specific value v for attribute A, is done by counting how many times v appears in the table T for attribute A. Suppose that our query contains value v', which we know to be a synonym of value v, with some confidence p. The closed world language model for T will be extended to include v' with the added information that this maps to value v with confidence p. Then, estimating the probability of value v' can be done by counting the number of times value v appears, and weight this count by the value of p. The full discussion on finding, modeling and implementing synonym handling is beyond the scope of our paper.

Finally, we note that although in general we assume independence between attributes, multi-attribute statistics are used whenever their absence could severely distort the selectivity estimates derived. Such an example are attributes Brand and Model-Line. A Model-Line value is completely dependent on the corresponding Brand value. Assuming independence between these two attributes would greatly underestimate the probability of relevant value pairs.

Generating Free Tokens.

We distinguish between two types of free tokens: the free tokens in $\mathcal{FT}_q$ that are generated as part of the open language model annotation S_{OLM} that generates free-text web queries, and free tokens in $\mathcal{FT}_i$ that are generated as part of an annotation S_i for a table T in the collection $\mathcal{T}$.

For the first type of free tokens, we compute the conditional probability $P(\mathcal{FT}_q|\text{OLM})$ using a simple unigram model constructed from a collection of generic web queries. The assumption is that that each free token (word in this case) is drawn independently. Therefore, we have that:

$$P(\mathcal{FT}_q|\text{OLM}) = \prod_{w \in \mathcal{FT}_q} P(w|\text{OLM})$$

Obviously, the unigram model is not very sophisticated and is bound to offer less than perfect estimates. However, recall that the OLM table is introduced to act as a "control" against which all candidate structured annotations need to "compete", in addition to each other, to determine which ones are plausible annotations of the query under consideration. An annotation S_i is plausible if $P(S_i) > \theta P(S_{\text{OLM}})$; the remaining annotations are rejected. A rejected annotation S_i is less likely to have generated the query q, than a process that generates queries by drawing words independently at random, according to their relative frequency. It is reasonable to argue that such an interpretation of the query q is implausible and should be rejected.

For the second type of free tokens, we compute the conditional probability $P(\mathcal{FT}_i|T)$, for some annotation S_i over table T, using again a unigram model UM_T that is specific to the table T, and contains all unigrams that can be associated with table T. For construction of UM_T, we utilize the names and values of *all* attributes of table T. Such words are highly relevant to table T and therefore have a higher chance of being included as free tokens in an annotated query targeted at table T. Further extensions of the unigram model are possible, by including other information related to table T, e.g., crawling related information from the web, or adding related queries via toolbar or query log analysis. This discussion is beyond the scope of this paper.

Using the unigram model UM_T we now have:

$$P(\mathcal{FT}_i|T) = \prod_{w \in \mathcal{FT}_i} P(w|T) = \prod_{w \in \mathcal{FT}_i} P(w|\text{UM}_T)$$

Note that free tokens are important for disambiguating the intent of the user. For example, for the query "LG 30 inch computer screen" there are two possible annotations, one for the Monitors table, and one for the TV table, each one selecting the attributes Brand and Diagonal. The terms "computer" and "screen" are free tokens. In this case the selected attributes should not give a clear preference of one table over the other, but the free term "computer" should assign more probability to the Monitors table, over the TVs table, since it is related to Monitors, and not to TVs.

Given that we are dealing with web queries, it is likely that users may also use as free tokens words that are generic to web queries, even for queries that target a very specific table in the structured data. Therefore, when computing the probability that a word appears as a free token in an annotation we should also take into account the likelihood of a word to appear in a generic web query. For this purpose, we use the unigram open language model OLM described in Section 4 as the *background* probability of a free token w in $\mathcal{FT}_i$, and we interpolate the conditional probabilities $P(w|\text{UM}_T)$ and $P(w|\text{OLM})$. Putting everything together:

$$P(w|T) = \lambda P(w|\text{UM}_T) + \mu P(w|\text{OLM}) \text{ , } \lambda + \mu = 1 \qquad (4)$$

The ratio between λ/μ controls the confidence we place to the unigram model, versus the possibility that the free tokens come from the background distribution. Given the importance and potentially deleterious effect of free tokens on the probability and plausibility of an annotation, we would like to exert additional control on how free tokens affect the overall probability of an annotation. In order to do so, we introduce a tuning parameter $0 < \phi \leq 1$, which can be used to additionally "penalize" the presence of free tokens in an annotation. To this end, we compute:

$$P(w|T) = \phi(\lambda P(w|\text{UM}_T) + \mu P(w|\text{OLM}))$$

Intuitively, we can view ϕ as the effect of a process that outputs free tokens with probability zero (or asymptotically close to zero), which is activated with probability $1 - \phi$. We set the ratio λ/μ and penalty parameter ϕ in our experimental evaluation in Section 6.

5.2 Estimating Template Probabilities

We now focus on estimating the probability of a query targeting particular tables and attributes, i.e., estimate $P(T.\tilde{\mathcal{A}}_i)$ for an annotation S_i. A parallel challenge is the estimation of $P(\text{OLM})$, i.e., the probability of a query being generated by the open language model, since this is considered as an additional type of "table" with a single attribute that generates free tokens. We will refer to table and attribute combinations as *attribute templates*.

The most reasonable source of information for estimating these probabilities is web query log data, i.e., user-issued web queries that have been already witnessed. Let $\mathcal{Q}$ be a such collection of witnessed web queries. Based on our assumptions, these queries are the output of $|\mathcal{Q}|$ "runs" of the generative process depicted in Figure 4(b). The unknown parameters of a probabilistic generative process are typically computed using *maximum likelihood estimation*, that is, estimating attribute template probability values $P(T.\tilde{\mathcal{A}}_i)$ and $P(\text{OLM})$ that maximize the likelihood of generative process giving birth to query collection $\mathcal{Q}$.

Consider a keyword query $q \in \mathcal{Q}$ and its annotations $\mathcal{S}_q$. The query can either be the formulation of a request for structured data captured by an annotation $S_i \in \mathcal{S}_q$, or free-text query described by the S_{OLM} annotation. Since these possibilities are disjoint, the probability of the generative processes outputting query q is:

$$P(q) = \sum_{S_i \in \mathcal{S}_q} P(S_i) + P(S_{\text{OLM}}) =$$

$$= \sum_{S_i \in \mathcal{S}_q} P(\{\mathcal{AT}_i, \mathcal{FT}_i\}|T.\tilde{\mathcal{A}}_i)P(T.\tilde{\mathcal{A}}_i) + P(\mathcal{FT}_q|\text{OLM})P(\text{OLM})$$

A more general way of expressing $P(q)$ is by assuming that all tables in the database and all possible combinations of attributes from these tables could give birth to query q and, hence, contribute to probability $P(q)$. The combinations that do not appear in annotation set $\mathcal{S}_q$ will have zero contribution. Formally, let T_i be a table, and let $\mathcal{P}_i$ denote the set of all all possible combinations of attributes of T_i, including the free token emitting attribute $T_i.f$. Then, for a table collection $\mathcal{T}$ of size $|\mathcal{T}|$, we can write:

$$P(q) = \sum_{i=1}^{|\mathcal{T}|} \sum_{\mathcal{A}_j \in \mathcal{P}_i} \alpha_{qij}\pi_{ij} + \beta_q \pi_o$$

where $\alpha_{qij} = P(\{\mathcal{AT}_{ij}, \mathcal{FT}_{ij}\}|T_i.\tilde{\mathcal{A}}_j)$, $\beta_q = P(\mathcal{FT}_q|\text{OLM})$, $\pi_{ij} = P(T_i.\tilde{\mathcal{A}}_j)$ and $\pi_o = P(\text{OLM})$. Note that for annotations $S_{ij} \notin \mathcal{S}_q$, we have $a_{qij} = 0$. For a given query q, the parameters α_{qij} and β_q can be computed as described in Section 5.1. The parameters π_{ij} and π_o correspond to the unknown attribute template probabilities we need to estimate.

Therefore, the log-likelihood of the entire query log can be expressed as follows:

$$\mathcal{L}(\mathcal{Q}) = \sum_{q \in \mathcal{Q}} \log P(q) = \sum_{q \in \mathcal{Q}} \log \left(\sum_{i=1}^{|\mathcal{T}|} \sum_{\mathcal{A}_j \in \mathcal{P}_i} \alpha_{qij}\pi_{ij} + \beta_q \pi_o \right)$$

Maximization of $\mathcal{L}(\mathcal{Q})$ results in the following problem:

$$\max_{\pi_{ij}, \pi_o} \mathcal{L}(\mathcal{Q}), \text{ subject to } \sum_{ij} \pi_{ij} + \pi_o = 1 \qquad (5)$$

Condition $\sum_{ij} \pi_{ij} + \pi_o = 1$ follows from the fact that based on our generative model all queries can be explained either by an annotation over the structured data tables, or as free-text queries generated by the open-wold language model.

This is a large optimization problem with millions of variables. Fortunately, objective function $\mathcal{L}(\pi_{ij}, \pi_o|\mathcal{Q})$ is concave. This follows from the fact that the logarithms of linear functions are concave, and the composition of concave functions remains concave. Therefore, any optimization algorithm will converge to a global maximum. A simple, efficient optimization algorithm is the Expectation-Maximization (EM) algorithm [3].

LEMMA 2. *The constrained optimization problem described by equations 5 can be solved using the Expectation-Maximization algorithm. For every query keyword query q and variable π_{ij}, we introduce auxiliary variables γ_{qij} and δ_q. The algorithm's iterations are provided by the following formulas:*

- *E-Step:* $\gamma_{qij}^{t+1} = \alpha_{qij}\pi_{ij}^t / \left(\sum_{km} \alpha_{qkm}\pi_{km}^t + \beta_q \pi_o^t\right)$
  $\delta_q^{t+1} = \beta_q \pi_o^t / \left(\sum_{km} \alpha_{qkm}\pi_{km}^t + \beta_q \pi_o^t\right)$
- *M-Step:* $\pi_{ij}^{t+1} = \sum_q \gamma_{qij}^{t+1} / |\mathcal{Q}|$
  $\pi_o^{t+1} = \sum_q \delta_q^{t+1} / |\mathcal{Q}|$

The proof is omitted due to space constraints. For a related proof, see [3]. The EM algorithm's iterations are extremely lightweight and progressively improve the estimates for variables π_{ij}, π_o.

More intuitively, the algorithm works as follows. The E-step, uses the current estimates of π_{ij}, π_o to compute for each query q probabilities $P(S_{ij})$, $S_{ij} \in \mathcal{S}_q$ and $P(S_{\text{OLM}})$. Note that for a given query we only consider annotations in set $\mathcal{S}_q$. The appearance of each query q is "attributed" among annotations $S_{ij} \in \mathcal{S}_q$ and S_{OLM} proportionally to their probabilities, i.e., γ_{qij} stands for the "fraction" of query q resulting from annotation S_{ij} involving table T_i and attributes $T_i.\tilde{\mathcal{A}}_j$. The M-step then estimates $\pi_{ij} = P(T_i.\tilde{\mathcal{A}}_j)$ as the sum of query "fractions" associated with table T_i and attribute set $T_i.\tilde{\mathcal{A}}_j$, over the total number of queries in $\mathcal{Q}$.

6. EXPERIMENTAL EVALUATION

We implemented our proposed Query Annotator solution using C# as a component of Helix [22]. We performed a large-scale experimental evaluation utilizing real data to validate our ability to successfully address the challenges discussed in Section 1.

The structured data collection $\mathcal{T}$ used was comprised of 1176 structured tables available to us from the Bing search engine. In total, there were around 30 million structured data tuples occupying approximately 400GB on disk when stored in a database. The same structured data are publicly available via an XML API.[3]

The tables used represent a wide spectrum of entities, such as Shoes, Video Games, Home Appliances, Televisions, and Digital Cameras. We also used tables with "secondary" complementary entities, such as Camera Lenses or Camera Accessories that have high vocabulary overlap with "primary" entities in table Digital Cameras. This way we stress-test result quality on annotations that are semantically different but have very high token overlap.

Besides the structured data collection, we also used logs of web queries posed on the Bing search engine. For our detailed quality experiments we used a log comprised of 38M distinct queries, aggregated over a period of 5 months.

6.1 Algorithms

The annotation generation component presented in Section 3 is guaranteed to produce all maximal annotations. Therefore, we only test its performance as part of our scalability tests presented in Section 6.5. We compare the annotation scoring mechanism against a greedy alternative. Both algorithms score the same set of annotations, output by the annotation generation component (Section 3).

Annotator SAQ: The SAQ annotator (Structured Annotator of Queries) stands for the full solution introduced in this work. Two sets of parameters affecting SAQ's behavior were identified. The first, is the *threshold* parameter θ used to determine the set of plausible structured annotations, satisfying $\frac{P(S_i)}{P(S_{\text{OLM}})} > \theta$ (Section 4). Higher threshold values render the scorer more conservative in outputting annotations, hence, usually resulting in higher precision. The second are the language model parameters: the ratio λ/μ that balances our confidence to the unigram table language model, versus the background open language model, and the penalty parameter ϕ. We fix λ/μ = 10 which we found to be a ratio that works well in practice, and captures our intuition for the confidence we have to the table language model. We consider two variations of SAQ based on the value of ϕ: SAQ-MED (medium-tolerance) using $\phi = 0.1$, and SAQ-LOW (low-tolerance) using $\phi = 0.01$.

Annotator IG-X: The *Intelligent Greedy* (IG-X) scores annotations S_i based on the number of annotated tokens $|\mathcal{AT}_i|$ that they contain, i.e., Score$(S_i) = |\mathcal{AT}_i|$. The Intelligent Greedy annotator captures the intuition that higher scores should be assigned to annotations that interpret structurally a larger part of the query. Besides scoring, the annotator needs to deploy a threshold, i.e., a cri-

[3] See http://shopping.msn.com/xml/v1/getresults.aspx?text=televisions for for a table of TVs and http://shopping.msn.com/xml/v1/getspecs.aspx?itemid=1202956773 for an example of TV attributes.

terion for eliminating meaningless annotations and identifying the plausible ones. The set of plausible annotations determined by the Intelligent Greedy annotator are those satisfying (i) $|\mathcal{FT}_i| \leq X$, (ii) $|\mathcal{AT}_i| \geq 2$ and (iii) $P(\mathcal{AT}_i|T.\mathcal{A}_i) > 0$. Condition (i) puts an upper bound X on the number of free tokens a plausible annotation should contain: an annotation with more than X free tokens cannot be plausible. Note that the annotator completely ignores the affinity of the free tokens to the annotated tokens and only reasons based on their number. Condition (ii) demands a minimum of two annotated tokens, in order to eliminate spurious annotations. Finally, condition (iii) requires that the attribute-value combination identified by an annotation has a non-zero probability of occurring. This eliminates combinations of attribute values that have zero probability according to the multi-attribute statistics we maintain (Section 5.1).

6.2 Scoring Quality

We quantify annotation scoring quality using precision and recall. This requires obtaining labels for a set of queries and their corresponding annotations. Since manual labeling could not be realistically done on the entire structure data and query collections, we focused on 7 tables: Digital Cameras, Camcorders, Hard Drives, Digital Camera Lenses, Digital Camera Accessories, Monitors and TVs. The particular tables were selected because of their high popularity, and also the challenge that they pose to the annotators due to the high overlap of their corresponding closed language models (CLM). For example, tables TVs and Monitors or Digital Cameras and Digital Camera Lenses have very similar attributes and values.

The ground truth query set, denoted Q, consists of 50K queries explicitly targeting the 7 tables. The queries were identified using relevant click log information over the structured data and the query-table pair validity was manually verified. We then used our tagging process to produce all possible maximal annotations and labeled manually the correct ones, if any.

We now discuss the metrics used for measuring the effectiveness of our algorithms. An annotator can output multiple plausible structured annotations per keyword query. We define $0 \leq TP(q) \leq 1$ as the fraction of correct plausible structured annotations over the total number of plausible structured annotations identified by an annotator. We also define a keyword query as *covered* by an annotator, if the annotator outputs at least one plausible annotation. Let also $\text{Cov}(Q)$ denote the set of queries covered by an annotator. Then, we define:

$$\text{Precision} = \frac{\sum_{q \in Q} TP(q)}{|\text{Cov}(Q)|}, \text{Recall} = \frac{\sum_{q \in Q} TP(q)}{|Q|}$$

Figure 5 presents the Precision vs Recall plot for SAQ-MED, SAQ-LOW and the IG-X algorithms. Threshold θ values for SAQ were in the range of $0.001 \leq \theta \leq 1000$. Each point in the plot corresponds to a different θ value. The SAQ-based annotators and IG-0 achieve very high precision, with SAQ being a little better. To some extent this is to be expected, given that these are "cleaner" queries, with every single query pre-classified to target the structured data collection. Therefore, an annotator is less likely to misinterpret open-world queries as a request for structured data. Notice, however, that the recall of the SAQ-based annotators is significantly higher than that of IG-0. The IG-X annotators achieve similar recall for $X > 0$, but the precision degrades significantly. Note also, that increasing the allowable free tokens from 1 to 5 does not give gains in recall, but causes a large drop in precision. This is expected since targeted queries are unlikely to contain many free tokens.

Since the query data set is focused only on the tables we consider, we decided to stress-test our approach even further: we set threshold $\theta = 0$, effectively removing the adaptable threshold separating plausible and implausible annotations, and considered only the most probable annotation. SAQ-MED precision was measured at 78% and recall at 69% for $\theta = 0$, versus precision 95% and recall 40% for $\theta = 1$. This highlights the following points. First, even queries targeting the structured data collection can have errors and the adaptive threshold based on the open-language model can help precision dramatically. Note that errors in this case happen by misinterpreting queries amongst tables or the attributes within a table, as there are no generic web queries in this labeled data set. Second, there is room for improving recall significantly. A query is often not annotated due to issues with stemming, spell-checking or missing synonyms. For example, we do not annotate token "cannon" when it is used instead of "canon", or "hp" when used instead of "hewlett-packard". An extended structured data collection using techniques as in [6, 8] can result in significantly improved recall, but the study of such techniques is out of scope for this paper. Finally, we measured that in approximately 19% of the labeled queries, not a single token relevant to the considered table attributes was used in the query. This means there was no possible mapping from the open language used in web queries to the closed world described by the available structured data.

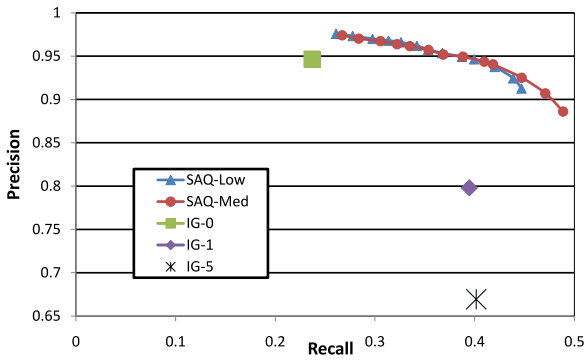

Figure 5: Precision and Recall using Targeted Queries

6.3 Handling General Web Queries

Having established that the proposed solution performs well in a controlled environment where queries are known to target the structured data collection, we now investigate its quality on general web queries. We use the full log of 38M queries, representative of an everyday web search engine workload. These queries vary a lot in context and are easy to misinterpret, essentially stress-testing the annotator's ability to supress false positives.

We consider the same annotator variants: SAQ-MED, SAQ-LOW and IG-X. For each query, the algorithms output a set of plausible annotations. For each alternative, a uniform random sample of covered queries was retrieved and the annotations were manually labeled by 3 judges. A different sample for each alternative was used; 450 queries for each of the SAQ variations and 150 queries for each of the IG variations. In total, 1350 queries were thoroughly hand-labeled. Again, to minimize the labeling effort, we only consider structured data from the same 7 tables mentioned earlier.

The plausible structured annotations associated with each query were labeled as *Correct* or *Incorrect* based on whether an annotation was judged to represent a highly likely interpretation of the query over our collection of tables $\mathcal{T}$. We measure precision as:

$$\text{Precision} = \frac{\text{\# of correct plausible annotations in the sample}}{\text{\# of plausible annotations in the sample}}$$

It is not meaningful to compute recall on the entire query set of 38 million. The vast majority of the web queries are general purpose queries and do not target the structured data collection.

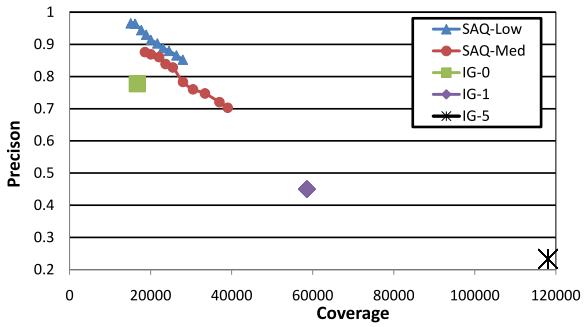

Figure 6: Precision and Coverage using General Web Queries

To compensate, we measured *coverage*, defined as the number of covered queries, as a proxy of *relative recall*.

Figure 6 presents the annotation precision-coverage plot, for different threshold values. SAQ uses threshold values ranging in $1 \leq \theta \leq 1000$. Many interesting trends emerge from Figure 6. With respect to SAQ-MED and SAQ-LOW, the annotation precision achieved is extremely high, ranging from 0.73 to 0.89 for SAQ-MED and 0.86 to 0.97 for SAQ-LOW. Expectedly, SAQ-LOW's precision is higher than SAQ-MED, as SAQ-MED is more tolerant towards the presence of free tokens in a structured annotation. As discussed, free tokens have the potential to completely distort the interpretation of the remainder of the query. Hence, by being more tolerant, SAQ-MED misinterprets queries that contain free tokens more frequently than SAQ-LOW. Additionally, the effect of the threshold on precision is pronounced for both variations: a higher threshold results value results in higher precision.

The annotation precision of IG-1 and IG-5 is extremely low, demonstrating the challenge that free tokens introduce and the value of treating them appropriately. Even a single free token (IG-1) can have a deleterious effect on precision. However, even IG-0, which only outputs annotations with *zero* free tokens, offers lower precision than the SAQ variations. The IG-0 algorithm, by not reasoning in a probabilistic manner, makes a variety of mistakes, the most important of which to erroneously identify latent structured semantics in open-world queries. The "white tiger" example mention in Section 1 falls in this category. To verify this claim, we collected and labeled a sample of 150 additional structured annotations that were output by IG-0, but rejected by SAQ-MED with $\theta = 1$. SAQ's decision was correct approximately 90% of the time.

With respect to coverage, as expected, the more conservative variations of SAQ, which demonstrated higher precision, have lower coverage values. SAQ-MED offers higher coverage than SAQ-LOW, while increased threshold values result in reduced coverage. Note also the very poor coverage of IG-0. SAQ, by allowing *and* properly handling free tokens, increases substantially the coverage, without sacrificing precision.

6.4 Understanding Annotation Pitfalls

We performed micro benchmarks using the hand-labeled data described in Section 6.3 to better understand why the annotator works well and why not. We looked at the effect of annotation length, free tokens and structured data overlap.

Number of Free Tokens: Figures 7(a) and 8(a) depict the fraction of correct and incorrect plausible structured annotations with respect to the number of free tokens, for configurations SAQ-LOW (with $\theta = 1$) and IG-5 respectively. For instance, the second bar of 7(a) shows that 35% of *all* plausible annotations contain 1 free token: 24% were correct, and 11% were incorrect. Figures 7(b) and 8(b) normalize these fractions for each number of free tokens. For instance, the second bar of Figure 7(b) signifies that of the structured annotations with 1 free token output by SAQ-LOW, approximately 69% were correct and 31% were incorrect.

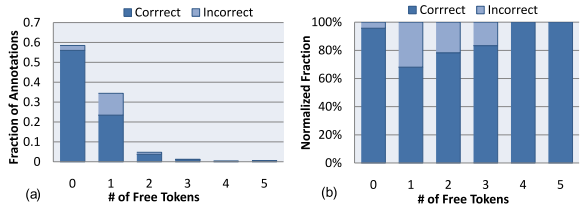

Figure 7: SAQ-LOW: Free tokens and precision.

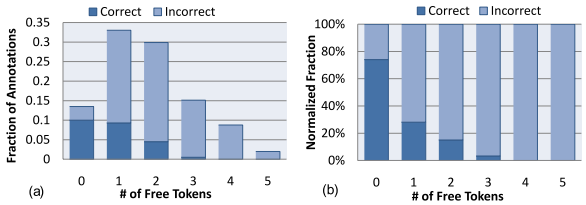

Figure 8: IG-5: Free tokens and precision.

The bulk of the structured annotations output by SAQ-LOW (Figure 7) contain either none or one free token. As the number of free tokens increases, it becomes less likely that a candidate structured annotation is correct. SAQ-LOW penalizes large number of free tokens and only outputs structured annotations if it is confident of their correctness. On the other hand, for IG-5 (Figure 8), more than 50% of structured annotations contain at least 2 free tokens. By using the appropriate probabilistic reasoning and dynamic threshold, SAQ-LOW achieves higher precision even against IG-0 (zero free tokens) or IG-1 (zero or one free tokens). As we can see SAQ handles the entire gamut of free-token presence gracefully.

Overall Annotation Length: Figures 9 and 10 present the fraction and normalized fraction of correct and incorrect structured annotations outputted, with respect to annotation *length*. The length of an annotation is defined as number of the annotated and free tokens. Note that Figure 10 presents results for IG-0 rather than IG-5. Having established the effect of free tokens with IG-5, we wanted a comparison that focuses more on annotated tokens, so we chose IG-0 that outputs zero free tokens.

An interesting observation in Figure 9(a) is that although SAQ-LOW has not been constrained like IG-0 to output structured annotations containing at least 2 annotated tokens, only a tiny fraction of its output annotations contain a single annotated token. Intuitively, it is extremely hard to *confidently* interpret a token, corresponding to a single attribute value, as a structured query. Most likely the keyword query is an open-world query that was misinterpreted.

The bulk of mistakes by IG-0 happen for two-token annotations. As the number of tokens increases, it becomes increasingly unlikely that all 3 or 4 annotated tokens from the same table appeared in the same query by chance. Finally, note how different the distribution of structured annotations is with respect to the length of SAQ-LOW (Figure 9(a)) and IG-0 (Figure 10(a)). By allowing free tokens in a structured annotation, SAQ can successfully and correctly annotate longer queries, hence achieving much better recall without sacrificing precision.

Types of Free Tokens in Incorrect Annotations: Free tokens can completely invalidate the interpretation of a keyword query captured by the corresponding structured annotation. Figure 11 depicts a categorization of the free tokens present in plausible annotations output by SAQ and labeled as *incorrect*. The goal of the experiment is to understand the source of the errors in our approach.

We distinguish four categories of free tokens: **(i)** *Open-world altering tokens*: This includes free tokens such as "review", "drivers"

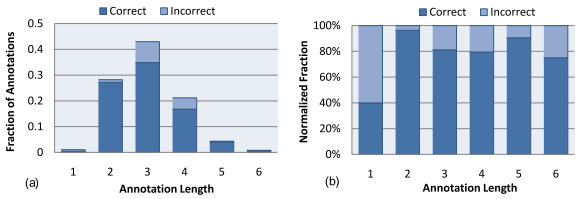

Figure 9: SAQ-LOW: Annotation length and precision.

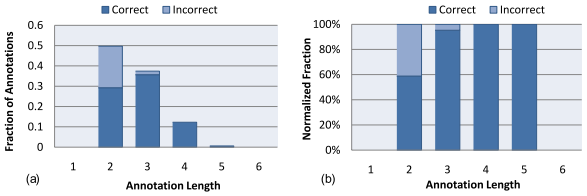

Figure 10: IG-0: Annotation length and precision.

that invalidate the intent behind a structured annotation and take us outside the closed world. **(ii)** *Closed-world altering tokens*: This includes relevant tokens that are not annotated due to incomplete structured data and eventually lead to misinterpretations. For example, token "slr" is not annotated in the query "nikon 35 mm slr" and as a result the annotation for Camera Lenses receives a high score. **(iii)** *Incomplete closed-world*: This includes tokens that would have been annotated if synonyms and spell checking were enabled. For example, query "panasonic video camera" gets misinterpreted if "video" is a free token. If "video camera" was given as a synonym of "camcorder" this would not be the case. **(iv)** *Open-world tokens*: This contains mostly stop-words like "with", "for", etc.

The majority of errors are in category (i). We note that a large fraction of these errors could be corrected by a small amount of supervised effort, to identify common open-world altering tokens. We observe also that the number of errors in categories (ii) and (iii) is lower for SAQ-LOW than SAQ-MED, since (a) SAQ-LOW is more stringent in filtering annotations and (b) it down-weights the effect of free tokens and is thus hurt less by not detecting synonyms.

Overlap on Structured Data: High vocabulary overlap between tables introduces a potential source of error. Table 1 presents a "confusion matrix" for SAQ-LOW. Every plausible annotation in the sample is associated with two tables: the actual table targeted by the corresponding keyword query ("row" table) and the table that the structured annotation suggests as targeted ("column" table). Table 1 displays the row-normalized fraction of plausible annotations output for each actual-predicted table pair. For instance, for 4% of the queries relevant to table Camcorders, the plausible structured annotation identified table Digital Cameras instead. We note that most of the mass is on the diagonal, indicating that SAQ correctly determines the table and avoids class confusion. The biggest error occurs on camera accessories, where failure to understand free tokens (e.g., "batteries" in query "nikon d40 camera batteries") can result in producing high score annotations for the Cameras table.

6.5 Efficiency of Annotation Process

We performed an experiment to measure the total time required by SAQ to generate and score annotations for the queries of our full web log. The number of tables was varied in order to quantify the effect of increasing table collection size on annotation efficiency. The experimental results are depicted in Figure 12. The figure presents the mean time required to annotate a query: approximately *1 millisecond* is needed to annotate a keyword query in the presence of 1176 structured data tables. Evidently, the additional overhead to general search-engine query processing is minuscule, even in the presence of a large structured data collection. We also

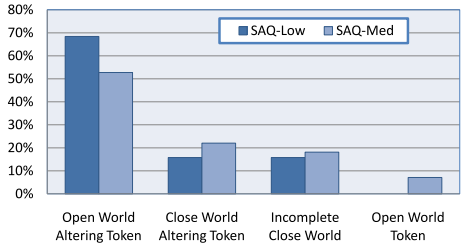

Figure 11: Free tokens in incorrect annotations.

Predicted→ Actual↓	Cameras	Camcorders	Lenses	Accessories	OLM
Cameras	92%	2%	4%	2%	0%
Camcorders	4%	96%	0%	0%	0%
Lenses	2%	0%	94%	4%	0%%
Accessories	13%	3%	3%	81%	0%
OLM	7%	2%	0%	1%	90%

Table 1: Confusion matrix for SAQ-LOW.

observe a linear increase of annotation latency with respect to the number of tables. This can be attributed to the number of structured annotations generated and considered by SAQ increasing at worst case linearly with the number of tables.

The experiment was executed on a single server and the closed structured model for all 1176 tables required 10GB of memory. It is worth noting that our solution is decomposable, ensuring high parallelism. Therefore, besides low latency that is crucial for web search, a production system can afford to use multiple machines to achieve high query throughput. For example, based on a latency of 1ms per query, 3 machines would suffice for handling a hypothetical web search-engine workload of 250M queries per day.

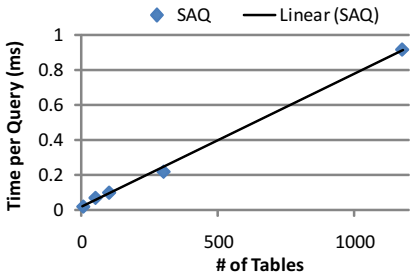

Figure 12: SAQ: On-line efficiency.

7. RELATED WORK

A problem related to generating plausible structured annotations, referred to as *web query tagging*, was introduced in [17]. Its goal is to assign each query term to a specified category, roughly corresponding to a table attribute. A Conditional Random Field (CRF) is used to capture dependencies between query words and identify the most likely joint assignment of words to "categories". Query tagging can be viewed as a simplification of the query annotation problem considered in this work. One major difference is that in [17] structured data are not organized into tables.This assumption severely restricts the allowed applicability of the solution to multiple domains, as there is no mechanism to disambiguate between arbitrary combinations of attributes. Second, the possibility of not attributing a word to any specific category is not considered. This assumption is incompatible with the general web setting. Finally, training of the CRF is performed in a *semi-supervised* fashion and hence the focus of [17] is on automatically generating and utilizing training data for learning the CRF parameters. Having said that, the scale of the web demands an unsupervised solution; anything less will encounter issues when applied to diverse structured domains.

Keyword search on relational [12, 18, 15], semi-structured [10, 19] and graph data [14, 11] (Keyword Search Over Structured Data, abbreviated as KSOSD) has been an extremely active research topic. Its goal is the efficient retrieval of relevant database tuples, XML sub-trees or subgraphs in response to keyword queries. The problem is challenging since the relevant pieces of information needed to assemble answers are assumed to be scattered across relational tables, graph nodes, etc. Essentially, KSOSD techniques allow users to formulate complicated join queries against a database using keywords. The tuples returned are ranked based on the "distance" in the database of the fragments joined to produce a tuple, and the textual similarity of the fragments to query terms.

The assumptions, requirements and end-goal of KSOSD are radically different from the web query annotation problem that we consider. Most importantly, KSOSD solutions implicitly assume that users are aware of the presence and nature of the underlying data collection, although perhaps not its exact schema, and that they explicitly intent to query it. Hence, the focus is on the assembly, retrieval and ranking of relevant results (tuples). On the contrary, web users are oblivious to the existence of the underlying data collection and their queries might even be irrelevant to it. Therefore, the focus of the query annotation process is on discovering latent structure in web queries and identifying plausible user intent. This information can subsequently be utilized for the benefit of structured data retrieval and KSOSD techniques. For a thorough survey of the KSOSD literature and additional references see [7].

Some additional work in the context KSOSD, that is close to our work appears in [5, 9]. This work identifies that while a keyword query can be translated into multiple SQL queries, not all structured queries are equally likely. A Bayesian network is used to score and rank the queries, based on the data populating the database. Similar ideas for XML databases are presented in [16]. This information is subsequently used in ranking query results. All three techniques consider the *relative* likelihood of each alternative structured query, without considering their plausibility. In other words, the intent of the user to query the underlying data is taken for granted. Explicit treatment of free tokens in a keyword query and the successful use of query log data further distinguishes our approach from the aforementioned line of work.

The focus of [23] is on pre-processing a keyword query in order to derive "high scoring" segmentations of it. A segmentation is a grouping of nearby semantically related words. However, a high-scoring query segmentation is a poorer construct than a structured annotation. Finally, [4] study the problem of querying *for* tables present in a corpus of relational tables, extracted from the HTML representation of web pages. The precise problem addressed is the retrieval of the top-k tables present in the corpus, which is different from the more elaborate one considered in this work.

8. CONCLUSIONS

Fetching and utilizing results from structured data sources in response to web queries presents unique and formidable challenges, with respect to both result quality and efficiency. Towards addressing such problems we defined the novel notion of *Structured Annotations* as a mapping of a query to a table and its attributes. We showed an efficient process that creates all such annotations and presented a probabilistic scorer that has the ability to sort and filter annotations based on the likelihood they represent meaningful interpretations of the user query. The end to end solution is highly efficient, demonstrates attractive precision/recall characteristics and is capable of adapting to diverse structured data collections and query workloads in a completely unsupervised fashion.

9. REFERENCES

[1] J. L. Bentley and R. Sedgewick. Fast Algorithms for Sorting and Searching Strings. In *SODA*, 1997.

[2] M. Bergman. The Deep Web: Surfacing Hidden Value. *Journal of Electronic Publishing*, 7(1), 2001.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 1st edition, 2006.

[4] 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.

[5] P. Calado, A. S. da Silva, A. H. F. Laender, B. A. Ribeiro-Neto, and R. C. Vieira. A Bayesian Network Approach to Searching Web Databases through Keyword-based Queries. *Inf. Process. Man.*, 40(5), 2004.

[6] S. Chaudhuri, V. Ganti, and D. Xin. Exploiting Web Search to Generate Synonyms for Entities. In *WWW*, 2009.

[7] Y. Chen, W. Wang, Z. Liu, and X. Lin. Keyword Search on Structured and Semi-structured Data. In *SIGMOD*, 2009.

[8] T. Cheng, H. Lauw, and S. Paparizos. Fuzzy Matching of Web Queries to Structured Data. In *ICDE*, 2010.

[9] F. de Sá Mesquita, A. S. da Silva, E. S. de Moura, P. Calado, and A. H. F. Laender. LABRADOR: Efficiently Publishing Relational Databases on the Web by Using Keyword-based Query Interfaces. *Inf. Process. Manage.*, 43(4), 2007.

[10] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD*, 2003.

[11] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD*, 2007.

[12] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In *VLDB*, 2003.

[13] Y. E. Ioannidis. The History of Histograms. In *VLDB*, 2003.

[14] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB*, 2005.

[15] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar Semantic Search: A Database Approach to Information Retrieval. In *SIGMOD06*.

[16] J. Kim, X. Xue, and W. B. Croft. A Probabilistic Retrieval Model for Semistructured Data. In *ECIR*, 2009.

[17] X. Li, Y.-Y. Wang, and A. Acero. Extracting Structured Information from User Queries with Semi-supervised Conditional Random Fields. In *SIGIR*, 2009.

[18] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective Keyword Search in Relational Databases. In *SIGMOD*, 2006.

[19] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. *PVLDB*, 1(1), 2008.

[20] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, U. Srivastava, and T. M. Tran. Consistent selectivity estimation via maximum entropy. *VLDB J.*, 16(1), 2007.

[21] G. A. Miller. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11):39–41, 1995.

[22] S. Paparizos, A. Ntoulas, J. C. Shafer, and R. Agrawal. Answering Web Queries Using Structured Data Sources. In *SIGMOD*, 2009.

[23] K. Q. Pu and X. Yu. Keyword Query Cleaning. *PVLDB*, 1(1):909–920, 2008.