Structured Queries Over Web Text

Michael J. Cafarella, Oren Etzioni, Dan Suciu
University of Washington
Seattle, WA 98195
{mjc, etzioni, suciu}@cs.washington.edu

Abstract

The Web contains a vast amount of text that can only be queried using simple keywords-in, documents-out *search queries. But Web text often contains structured elements, such as hotel location and price pairs embedded in a set of hotel reviews. Queries that process these structural text elements would be much more powerful than our current document-centric queries. Of course, text does not contain metadata or a schema, making it unclear what a structured text query means precisely. In this paper we describe three possible models for structured queries over text, each of which implies different query semantics and user interaction.*

1 Introduction

The Web contains a vast amount of data, most of it in the form of unstructured text. Search engines are the standard tools for querying this text, and generally perform just one type of query. In response to a few keywords, a search engine will return a relevance-ranked list of documents. Unfortunately, treating Web text as nothing but a collection of standalone documents ignores a substantial amount of embedded structure that is obvious to every human reader, even if current search engines are blind to it. Web text often has a rich, though implicit, structure that deserves a correspondingly-rich set of query tools.

For example, consider a website that contains hotel reviews. Although each review is a self-contained text that is authored by a single person, the set of all such reviews shows remarkable regularity in the type of information that is contained. Most reviews will contain standard values such as the hotel's price and general quality. Reviews of urban hotels might discuss how central the hotel's location is, and reviews of resorts will mention the quality of the beach and pool. In other words, the hotel reviews have a messy and implicit "schema" that is not designed by any database administrator but is nonetheless present.

If a query system made this structure explicit, users could navigate the hotel reviews using, say, a fragment of SQL. A user could easily list all hotels that are both very quiet and in central Manhattan, even though doing so would be impossible with a standard search engine.

Note that we are *not* suggesting that the hotel site publish any sort of structural metadata. We do not ask Web publishers to add any additional information about their text. Our proposed query systems preserve the near-universal applicability of search engines, using only structural elements that are *already present* in the text

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

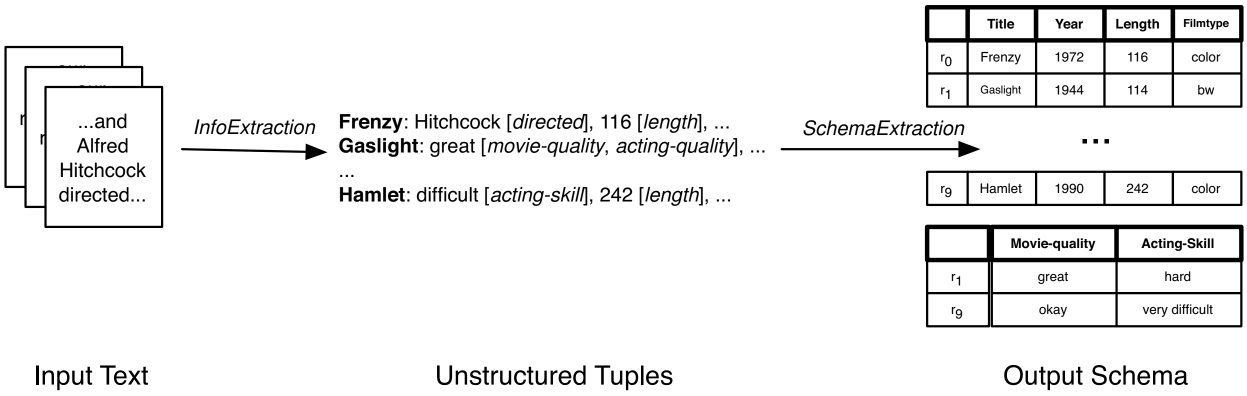

Figure 1: In the Schema Extraction Model, the query system receives some **Input Text** and runs it through an information extraction system. The result is a set of **Unstructured Tuples**, consisting of a set of unique keys plus affiliated values and possible attribute labels. We then use these tuples to choose an **Output Schema**, and thus create a domain-appropriate structured database. The **InfoExtraction** stage is provided by an external system, but the **SchemaExtraction** stage is novel.

or in a user's query. Of course, some text (*e.g.*, poetry) might not contain any reasonable structure, but this text is also unlikely to be the subject of a structured query.

This paper discusses an ongoing research program at the University of Washington to explore structured queries over the Web text corpus. Combining techniques from databases, information retrieval, and artificial intelligence, we are investigating three different models for processing structured text queries. They are quite different, but all involve some amount of information extraction and a novel query language. The models differ in how much structure they attempt to extract from the text, and how much they gather from the query.

Although we believe some of our query models present use scenarios that are quite compelling, it is still too early to determine a single best method for querying Web text. Indeed, it may be that different applications will require different query models.

2 The Schema Extraction Model

The **Schema Extraction Model** is the one described in our example in the introduction. The goal is to examine the corpus of Web text and create a "structured overlay" that describes data contained in the text. This structured overlay is essentially a relational database consisting of values found using an information extraction (IE) system. Users can then query this structured overlay using a fragment of SQL.

Figure 1 shows the execution pipeline. We start with a series of input texts; in the example here, some movie reviews. We then run an IE system, such as TextRunner, KnowItAll, or Snowball, over the text to obtain a series of fact-like assertions [2, 7, 1]. For a given identified object (*e.g.*, `Frenzy`) we have a series of extracted values, each of which has at least one accompanying attribute name (for `Frenzy`, perhaps values `1972` and `Alfred Hitchcock`, with attributes `year` and `director`). We call the extracted sets of values for one object an **unstructured tuple**.

These IE systems have shown to be quite effective at extracting small pieces of information embedded in text. For example, the KnowItAll system used the Web to compile a list of US states with precision 1.0 at recall 0.98 (that is, it made no incorrect guesses and missed only one of 50) and a list of countries with precision 0.79 at recall 0.87 [6].

We would like to assemble these unstructured tuples into a relational database that will form the structured overlay. However, the unstructured tuples do not have a uniform set of attributes, so the correct schema is unclear. Although most pages in a domain will contain similar information (here, `title`, `year`, etc), many will have missing values or off-topic extraneous ones. These missing and spurious values may be due to variance in the original text, or due to the inevitably-flawed IE mechanism. Some attributes will appear in nearly every unstructured tuple, whereas others will appear very rarely. The set of unstructured tuples may even describe a subset relationship (*e.g.*, horror movies may share only some attributes with action movies).

How can we generate a high-quality schema for these tuples? One approach is to create a relational table for each unique attribute signature in the unstructured tuples. If the unstructured tuples had no variance in their attributes (*i.e.*, they are "perfect"), this naïve algorithm would work well. However, in practice the result is a baroque and unusable schema, containing many similar but non-identical tables. In a small experiment, we mutated 20% of the cells in a set of perfect unstructured tuples. The result was a single-row table for every input tuple. Clearly, such a schema would be unsatisfactory to any query-writer.

For real-world data, especially data derived from many distinct texts without any explicit structure requirements, there will be no single indisputably "best" schema. Even if all the input data had been generated with a single schema in mind, some fields will be absent and some eccentric ones will appear. We must be willing to accept a "lossy" database that drops some extraneous extracted values found in the Unstructured Tuples. We must also accept NULLs where the appropriate value simply could not be recovered.

Of course, there is no genuine data loss here, since the original Web text still survives. But the lost data is not present in the structured overlay, so it will be unavailable to structured queries. An imperfect structured overlay is regrettable, but the only alternative is a "lossless" one that is so complicated as to be useless for querying. Choosing the lossy output that is least offensive for query-writers is a major challenge of this approach.

The Schema Extraction Model places the entire structural burden on the Web text itself. All of its technical challenges arise from the automated schema design. The structured query language itself is no more interesting than SQL.

Of course, if we can somehow find an extremely large corpus of schemas, then a slight variant of the Schema Extraction Model is possible. Instead of computing a schema directly from the extracted values, we can use these extractions to rank all schemas in the corpus, and then choose the best one. This schema corpus would have to be much larger than any corpus we have seen; it might be possible to compile it by crawling structured HTML tables, deep web services, etc.

3 The Text Query Model

For the **Text Query Model**, consider a slightly different scenario. Imagine a student who wants to know when a favorite band will be playing nearby. The band's itinerary is available on its website. In a single query, the user would like to a) extract the city/date tuples from the band's website, b) indicate the city where she lives, c) compute the dates when the band's city and her own city are within 100 miles of each other. Of course, the user only wants to know about appearances in the future, even if old data is still on the website.

We might write that query as follows:

```
SELECT bandCity, bandDate
FROM ("http://thebandilike.com/**",
      ["to appear in <string> on <date>", bandCity, bandDate])
WHERE
bandDate > 2006 AND
geographicdist(bandCity, "Seattle") =< 100
```

In the Text Query Model, the user must indicate every relevant part of the query's structure. The `FROM` clause indicates a relevant set of web pages, an extraction expression that can be applied to the text to generate

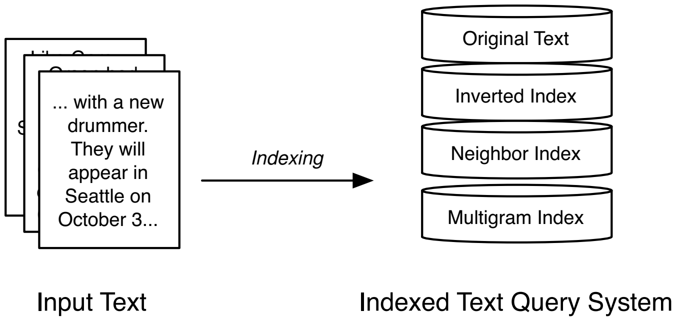

Figure 2: In the Text Query Model, we do not do any information extraction ahead of time. Instead, we extensively index the Web text so that we can efficiently run users' extraction-driven queries.

a table with two columns, and labels for those columns. The `WHERE` clause tests the date to see if it is valid, declares the user's current city, and uses a built-in function to measure the distance between two cities. The user is completely explicit about which structural elements are useful for answering the query.

A naïve execution model requires downloading pages at all possible URLs that match the `FROM` clause, parsing them using the extraction phrase in the second part of the `FROM` clause, and finally testing the selection predicates on the resulting set of tuples. Previous Web query systems, such as Squeal, WebSQL, and W3QL, have not followed this algorithm exactly, but have fetched remote pages in response to a user's query (rather than, say, using preindexed and prefetched pages) [12, 10, 8]. The result is that this simple query would take an extremely long time to run over the entire Web, in stark contrast to search engine queries that routinely execute in under a second.

The standard indexing tools used by search engines and traditional relational databases cannot help us much. Consider a search engine's inverted index, which efficiently maps words to documents. We could start to execute the above query by using the inverted index to find documents that contain strings `to appear in` and `on` within the `thebandilike.com` domain. This inverted-lookup step, the basic component of all search engine queries, is very efficient. However, we still need to run selection predicates on the strings that match the `<string>` and `<date>` variables. These strings can only be found by examining and parsing the original document text. The inverted-lookup step gave us the document IDs, but fetching each such document from the disk will require a disk seek.

The resulting inverted-index query plan requires a very large number of disk seeks. Even if the selection predicates are so aggressive that they disqualify every possible result from being returned, the query needs to examine each document in the relevant domain. Query time will scale directly with the size of the document set being considered. The resulting query time should be much faster than the naïve approach, but still seems needlessly slow.

It is not as clear how to model this problem using a relational query engine, but it is still possible. One approach is to automatically create a relational table of extracted values using the `FROM` clause, and then execute the selection clauses against the resulting database. Even with a technique to efficiently find documents in the indicated domain, this technique is very expensive, requiring that we parse and process all the relevant text before we even begin running selection predicates. If our workload features repeated common `FROM` clauses, it might be profitable to index the extracted table, but such a workload seems unlikely.

We believe that a system that is built to process these mixed *extractive* and *structural* queries can do substantially better than any of the approaches listed above. For example, we could construct a *neighbor index*, which folds small amounts of the document text into an inverted index [3]. In the above example, such an index allows us to extract values for `<string>` and `<date>` immediately from the index itself, without loading the original

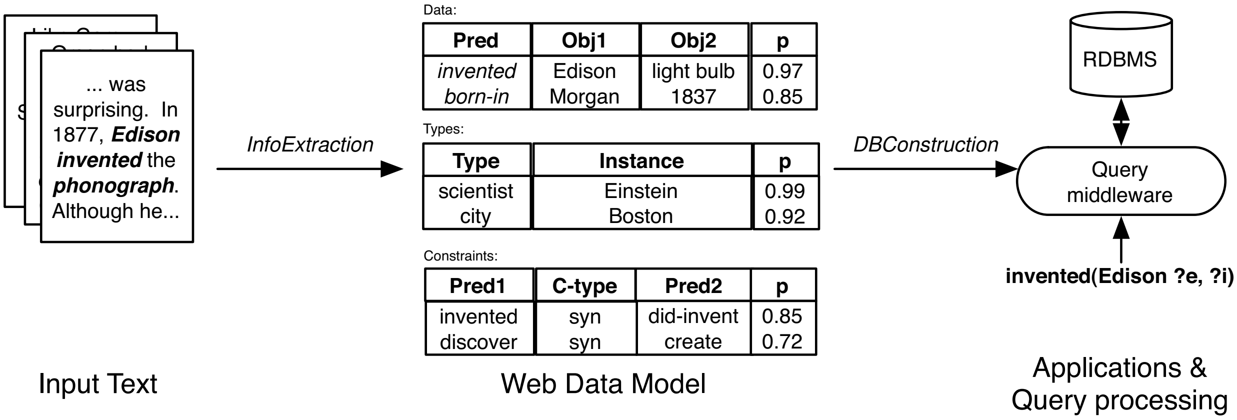

Figure 3: In the **Extraction Database Model**, we use the output of several IE systems to create a series of tables for the **Web Data Model**. These tables describe very primitive data relationships, such as *facts*, *is-a* relationships, *synonym* constraints, and others. Each tuple retains a probability of being true. We place the tuples into a probabilistic database and run user queries over that database.

documents and incurring disk seek costs. We can push selection predicates down into the inverted index lookup, saving time when the predicates eliminate possible results.

The neighbor index is not appropriate for all queries, as it can only efficiently handle queries that return strings up to a fixed length. For example, we could pose a query whose execution requires a neighbor index much larger than the original document set, losing the efficiency of an inverted index. However, the neighbor index is an example of the kind of novel optimization technique we believe the Text Query Model both demands and makes possible. (Another possibility is the *multigram index*, which enables fast regular expression processing over a large corpus [4].) This is an active area for our research into structured query techniques for text.

4 The Extraction Database Model

The **Extraction Database Model** takes a middle approach. As with the **Schema Extraction Model**, we use information extraction techniques to discern useful tuples from Web text. However, we do not attempt to assemble these extractions into a full and coherent schema. Rather, we store the tuples until query time. The user's query contains additional structural information that describes the desired output schema. Unlike the previous two models, we obtain structural hints from both the text and the user's query.

Figure 3 shows the series of steps needed to create an Extraction Database query system. Note that our model of IE output is somewhat different. We assume that extractions come in one of a handful of forms. Most come in the form of fact triples, which consist of two *objects* and a linking *predicate*, and are similar to the unstructured tuples of the Schema Extraction Model. We also obtain types, which describe an *is-a* relationship between two objects. Also, we extract constraints, which describe various relationships between predicates. There are well-known mechanisms to generate each of these extraction types, such as TextRunner for trinary facts, KnowItAll for *is-a* relationships, and DIRT for synonyms [2, 7, 9]. Together, we call these extracted values the Web Data Model.

Finally, we also retain probabilities for all extractions. Because we can only determine a tuple's importance in light of a relevant query, we cannot threshold away low-likelihood extractions (as we could in the Schema Extraction case). We place all of these tuples into a probabilistic database, such as the MYSTIQ system [5, 11].

Users pose queries against the stored tuples using a SQL-like notation. It is easy to translate queries in the Extraction Database query language into a probabilistic SQL query over the Web Data Model tables. For example, in order to search for all scientists born before 1880:

```
SELECT s FROM scientist
WHERE
born-in(s, y) AND
y IN year AND
y < 1880 AND
```

This query returns just the value bound to variable `s`, which is constrained to be of type `scientist`, to appear as the first object in a fact with predicate `born-in`. The second object in that fact must be of type `year`. We also require that the value bound to variable `y` be less than `1880`. All Extraction Database queries elicit probabilistic tuples, which will be returned in descending order of probability.

Note that unlike the Schema Extraction Model, we do not try to discern before query-time whether, say, `born-in` is a salient attribute of an instance of `scientist`. The probabilistic store contains good and spurious extractions alike. We rely on the user's query to tell us that the `born-in` attribute is actually a good one to test. However, unlike the Text Query Model, the query does not do all the hard work; for example, the query-writer does not have to know about the actual text layout of real web pages. The Extraction Database Model tries to strike a balance between the two, by extracting facts that might be useful in the future, but not making firm decisions about the schema until the user's query provides more information.

Efficient probabilistic query processing is a difficult problem, described elsewhere at length [5, 11]. We exacerbate it here by the sheer scale of our extracted tables. A single high-quality page of text could generate several extractions for *each sentence*, depending on the actual IE systems and parameters being used. Query processing poses a substantial problem with the Schema Extraction Model, and is an area of current research.

5 Conclusions

Web text contains huge volumes of useful data, most of it now out of reach. Current keyword-driven search engines give access to single Web pages, ignoring the implicit structure in Web text that is apparent to every user. By recognizing this structure and making it available to query-writers, we can offer a query system that is hugely more powerful than current systems, all without asking publishers for any additional effort.

We have described three possible structured query models for Web text: Schema Extraction, Text Query, and Extraction Database. These systems offer different advantages regarding the information extraction technology required and the expressiveness of the queries. Each promises, in a different way, to take better advantage of the vast amounts of text available to us.

References

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Procs. of the Fifth ACM International Conference on Digital Libraries*, 2000.

[2] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[3] M. Cafarella and O. Etzioni. A Search Engine for Natural Language Applications. In *Procs. of the 14th International World Wide Web Conference (WWW 2005)*, Tokyo, Japan, 2005.

[4] J. Cho and S. Rajagopalan. A fast regular expression indexing engine. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[5] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Thirtieth International Conference on Very Large Data Bases(VLDB)*, 2004.

[6] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Web-Scale Information Extraction in KnowItAll. In *WWW*, pages 100–110, New York City, New York, 2004.

[7] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[8] D. Konopnicki and O. Shmueli. W3QS - A System for WWW Querying. In *13th International Conference on Data Engineering (ICDE'97)*, 1997.

[9] D. Lin and P. Pantel. Discovery of inference rules from text. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 323–328, 2001.

[10] A. O. Mendelzon, G. A. Mihalia, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1996.

[11] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007. to appear.

[12] E. Spertus and L. A. Stein. Squeal: A Structured Query Language for the Web. In *WWW*, pages 95–103, 2000.