Enabling Structured Queries over Unstructured Documents

Fisnik Kastrati, Xiang Li, Christoph Quix and Mohammadreza Khelghati
Informatik 5 (Databases and Information Systems)
RWTH Aachen University, Germany
Email: LastName@dbis.rwth-aachen.de

***Abstract*—With the information explosion on the internet, finding precise answers efficiently is a prevalent requirement by many users. Today, search engines answer keyword queries with a ranked list of documents. Users might not be always willing to read the top ranked documents in order to satisfy their information need. It would save lots of time and efforts if the the answer to a query can be provided directly, instead of a link to a document which might contain the answer. To realize this functionality, users must be able to define their information needs precisely, e.g., by using structured queries, and, on the other hand, the system must be able to extract information from unstructured text documents to answer these queries.**

To this end, we introduce a system which supports structured queries over unstructured text documents, aiming at finding structured answers to the users' information need. Our goal is to extract answers from unstructured natural text, by applying various efficient techniques that allow fast query processing over text documents from the web or other heterogeneous sources. A key feature of our approach is that it does not require any upfront *integration efforts* such as the definition of a common data model or ontology.

I. Introduction

The sheer amount of data residing in heterogeneous sources (unstructured, structured, and partially structured) poses difficult challenges for applications accessing such sources.

This problem has divided the research community into two main research streams. On the one hand, the database (DB) community tries to solve this heterogeneity problem by providing data integration solutions, e.g., a unified query service over heterogeneous sources. As these solutions require the definition of an integrated schema, such approaches are called "schema first" approaches. They require semantic integration of all sources which in turn requires deep knowledge of schemas. Users may have neither the required knowledge to perform this kind of integration, nor the required technical expertise. Thus, such data integration approaches are not feasible for personal information systems, which are created and maintained by a single user (or a small group of users) to fulfill her information requirements.

On the other hand, the information retrieval (IR) community has been building systems that offer query services by indexing sources, and answering queries by supporting full-text search. The idea here is that a complete integration of all data might not always be required. Thus, queries can be answered without much up-front efforts as in data integration. This solution however does not offer precise query infrastructure as in database systems, i.e., it does not support SQL-like query language or data mining tasks.

Franklin et. al, envisioned a data co-existence approach for heterogeneous sources, and coined the term *dataspaces* [1]. Dataspaces do not require a complete integration, but offer search and query services right from the start. The integration tasks are left to the user, and can be done in *pay-as-you-go* fashion, i.e., incrementally during the lifetime of the system. Further integration will be performed as needed while the system is growing and extended. One good example of such a system is the iMeMeX system [2].

The existing solutions for querying dataspaces center mainly around the following query paradigms:

- keyword queries search over unstructured, semi-structured or structured sources
- structured queries over structured sources

Answering structured queries over unstructured text is an area which has not been deeply explored by the research community. This line of research was mainly touched by the Information Extraction (IE) community. IE systems such as TextRunner [3], KnowItAll [4], or Snowball [5] work by *a-priori* extracting all the relations from a corpus, this way creating tables containing all the relations that could be extracted. Finally, these tables can be queried.

By combining the benefits from data integration and information retrieval, structured queries over unstructured sources would enable precise queries without much integration efforts. One good application of such an approach would be the World Wide Web containing enormous load of information which is not structured, but is often embedded in plain text. Enabling such a system would ease the user's task in finding information without any integration efforts, or manually inspecting high ranked documents returned by keyword search engines.

Our work focuses on supporting structured queries over unstructured data, by trying to identify relationships in the plain text, without a "global schema" or requiring up-front integration of the sources. We present a system which returns ranked results to a given structured query, where answers are tuples satisfying the conditions present in a given query, and not solely documents containing "possible" answers as keyword-based approaches do it today.

As query processing at web-scale is a challenging issue, our approach combines techniques from information retrieval, natural language processing (NLP), and databases to achieve

an effective and efficient system. We show three different techniques in order to achieve this goal:

- Keyword Neighboring
- Named Entity Recognition
- Dependency Analysis

For the first approach, we utilize positional inverted lists, and inspect relative distance of keywords in the query occurring in the web pages returned by a search engine. We extract top ranked pages from search engine's API, by applying query transformation techniques, judiciously picking only the most selective keywords in order to minimize the search scope, thus avoid topic drift (i.e., avoid search engine distraction by unnecessary keywords), this is done in similar fashion as in [6]. In contrast to IE systems, such as [3]–[5], we do not require indexing a large corpus beforehand, but only the pages that are probable to contain the answers. Furthermore, indexing of documents is done afresh for each query, thus, only one small index at a time is kept. This offers huge advantages in terms of time and space, as indexing a large corpus results in large indexes which increase the query processing time and memory requirements. Furthermore, our system requires no setup cost before querying, as it does not rely on training a model for text extraction, nor on precomputed inverted lists. The documents collected from a search engine are processed only once, and final answers are presented to the user.

The second and third approach rely on IE techniques, in that we identify entities, and their relationships using NLP tools. As these approaches are costly, they are judiciously applied only over the top ranked sentences. For each matched query constraint, entities, and their surrounding keywords are kept in a local index, and their corresponding scores are aggregated. Only the top-k tuples are returned to the user.

In section II, we present related work and compare it with our system. Section III contains the main contributions of our work, it describes in details the query processing infrastructure and the answer extraction from plain text. Furthermore, we describe key components of our system and describe in detail the three approaches introduced above. Section IV compares and discusses the results of the three approaches, before we conclude in section V.

II. Related Work

Besides instance level heterogeneity such as data inconsistency and entity resolution, query level heterogeneity is also inevitable in dataspaces. That is, there are multiple possibilities of query languages and various types of data sources, structured, semi-structured or unstructured. Depending on the formats of the input and output, query engines in dataspaces can be categorized into:

- keyword queries over structured databases,
- keyword queries over text,
- structured queries on text, and
- structured queries over structured databases.

Among the above four categories, keyword queries over structured databases and structured queries over text are the key problems in order to provide a unified query interface across both structured and unstructured data sources. The remaining two are handled natively by IR engines and database engines respectively.

DBXplorer [7], DISCOVER [8] and BANKS [9] are good examples of keyword search on top of relational databases, i.e., keyword-in-tuple-out paradigm. Their work focuses on a minimal join tree around the user given keywords. The main challenge is to efficiently explore potentially exponential space of candidate tuple join trees. A recent survey of this topic can be found in [10].

In this paper, we focus on the other direction, i.e., answering structured queries over text. There are two categories of approaches: query translation and information extraction.

The former [6], [11], following a paradigm of keyword-in-document-out, focuses on transforming a structured query into a keyword query and then feeds the keyword query to an information retrieval engine to obtain a collection of documents. The main benefit is that the query rewriting procedure is only a light-weighted layer beyond the IR engine and therefore it makes use of existing technology and demands less efforts. However, as the results are in the form of documents, structured answers still need further extraction when needed.

The latter line of work, including [3], [5], [12]–[14], produce structured answers over text, i.e., query-in-tuple-out paradigm. They first extract triples from a given collection of documents into a triple store, and then answer the structured query against the triple store. [15] is a good survey on the topic of information extraction. Advantages of this paradigm of work include saving user's efforts on exploring text documents to identify answers, and possibility of machine processing of the structured answers in a pipelined fashion. However, there two problems with the IE based approaches. First such approaches are costly, as they require preprocessing of very large collections of documents, training, and indexing. Second, as web data are fast evolving, IE based approaches have the problem of stale data, i.e., an out-of-date version. In this paper, we present a novel system, which tries to avoid the problems of the aforementioned approaches while still maintaining a query-in-tuple-out paradigm. We adopt a just-in-time query processing over a fixed size collection of documents, which are results of a corpus selection procedure. The query processing occurs only at runtime and hence incurs no preprocessing overhead. Furthermore, as documents are retrieved just-in-time, the stale data problem is avoided.

WHIRL [16] is a system using textual similarity to extend the normal "join" in databases. Similar to our approach, they also handle combining query processing in databases and the non-uniform text data. However, a main distinction is that structure of data is not an issue in WHIRL, but data representations are heterogeneous. In our case, we focus more on recognizing possible structures in text.

Question answering systems (e.g., [17] and [18]) are another line of work dealing with extracting answers from unstructured text. The focus is on analyzing the question sentence and modeling the similarity or probabilistic dependencies between

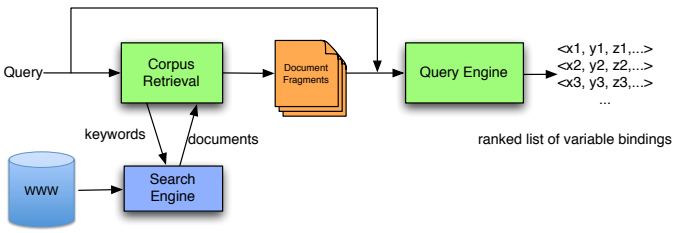

Fig. 1. Overview of System Architecture

the question sentence and a candidate sentence/document. In our approach, we do not need to perform heavy query analysis as the structure is explicitly given. We also need to address the challenge to extract tuples out of the text, which is not the focus of question answering systems.

III. Processing Structured Query Over Text

In this section, we detail our approach, which provides a novel way of answering structured queries over unstructured data. We first give an overview of our system architecture and the query process. Then in Section III-B, we describe the syntax of our *Conjunctive Triple Query (CTQ)*, which is a limited sublanguage of the Select-Project-Join queries in relational databases tailored to the world of web entities. CTQs are evaluated at runtime, with the help of a query engine and a search engine. Section III-C describes the retrieval of the text corpus which is used to answer the query. The main part of the query process is described in Section III-D.

A. System Overview

An overview of the system architecture is given in Figure 1. Given a declarative query by the user, a collection of potentially relevant documents are first selected with the help of a search engine. Keywords will be extracted from the given structured query and the used to query the web search engine. As explained in more detail in Section III-C, the returned documents will be preprocessed before fed into the query engine. An important step is the extraction of document fragments as not all parts of a document are relevant to answer a query. This also reduces the complexity of the natural language processing steps during query processing.

The selected document fragments, together with the structured query of the user, are then passed to the query engine. The output is a (ranked) list of tuples, each of the arity the same as the number of distinct variables in the input query. It is important to note, that although we use triple queries as described in the next subsection, we do not actually generate triples and evaluate the query over the generated set of triples. Besides performance reasons, a reason for this approach is that the context of the predicates can only be understood if we consider several predicates at the same time, and do not evaluate each predicate separately and then join the results.

B. Query Language

We first provide a formal definition of the input queries. The *domain*, denoted by $\mathbf{D}$, is composed of two disjoint set **TEXT** and **VAR**, where **TEXT** is a set of natural language texts while **VAR** is a set of variables.

A *triple* is in the form of $(Subject, Predicate, Object)$ or simply (S, P, O) with S, P and O all from $\mathbf{D}$. A statement like "*Bono is member of U2*", could be written in a triple form as *('Bono', 'is a member of', 'U2')*. The predicate here is the attribute "is a member of", whereas "Bono" is the subject and "U2" is the object. In RDF all data items are represented in such triples. A query for all members of "U2" is specified by *(X, 'is a member of', 'U2')* where $X \in \mathbf{VAR}$ is a variable.

We also use a binary predicate $typeOf(X, Y)$ to denote a special type of triple $(X,'has\ a\ type\ of', Y)$, which provides type information about variables. This class of "type of" predicates are also processed differently from normal triples, which state relationships between entities. A *conjunctive triple query* (CTQ) is a conjunction of triples. Please note that in this form, CTQ is strictly less expressive than SPARQL [19] for RDF data. As an example, suppose we would like to find all the information of a person studying in some university:

$$q(X, Y) \ :- \ (X,'studies\ at', Y), typeOf(Y,'university').$$

In order to simplify the procedure of query processing, we impose a restriction that the predicates cannot take the form of a variable. In our current implementation, we do not yet allow projections of variables, i.e., all variables appearing in the body are returned to the user.

C. Corpus Retrieval

As we have already pointed out, the existing approaches on answering structured query over unstructured data need to put a lot of efforts on extracting information from the web documents before even being able to process a single query. There are several undesired consequences from these approaches. First, the large upfront setup cost could be prohibitively high. In contrast, our just-in-time paradigm follows a pay-as-you-go style [1], and allow no setup efforts at all. Second, they do not respond well to the evolving web. The web is constantly evolving; new information and updated versions cannot be easily reflected in an extracted database which just reflects a snapshot of the web.

The corpus retrieval component in Figure 1 is implemented on the top of a search engine (e.g., Google). This component performs two tasks, retrieving a collection of potentially relevant documents to the user query, and retrieving fragments of the retrieved documents to feed the query engine.

The first task requires a procedure to derive a keyword query from a structured query. An approach using domain based weighting of keywords is presented in [6], in which it is also shown that admitting more keywords from the structured query into the keyword query may degrade the precision of the top documents. In our prototype, we do not use the native ranking of documents as returned by the search engine, but instead retrieve several pages of documents. Therefore, keyword selection is less important in our case, though it would be an interesting research topic.

The second task tries to locate the relevant parts of the retrieved documents that are potentially interesting to the user. The motivation behind this task is to reduce the costs of later computation intensive natural language processing steps (NLP). We retrieve the fragments using a technique called *keyword neighboring*. To put it simply, we retrieve text around the keywords of the query. This is based on the assumption that the information is local, and hence useful fragments are near the keywords. Of course, there are cases when use of subordinate clauses, pronouns and similar language constructs may result in a long distance between the keywords and the real entity's name in the documents. However, as dereferencing of pronouns and analyzing non-local structures is itself a research topic in natural language processing [20], we do not focus on this issue. As the results in section IV will show, we can find enough evidence using this assumption.

D. Query Processing

The query engine takes as input a collection of corpus fragments (sentences) and a CTQ, to produce a ranked list of tuples, each representing a binding to the variables in the query. The query processing procedure is based on two assumptions. First, information is local, that is, answers are near the keywords. Second, repeating indicates higher confidence of an answer. The former assumption decides the way we search for candidate bindings of variables, while the latter influences how we rank the candidates.

Since we confine the input CTQ to have no variables for predicates, we can focus on retrieving entities. In general, in order to extract the answers from unstructured text, we have evaluated three separate different techniques:

1) Keyword Neighborhood,
2) Named Entity Recognition, and
3) Dependency Analysis

The **Keyword Neighborhood** approach is based on selecting words and noun phrases surrounding the keywords present in the query. We use this technique in order to find words and noun phrases which are most related to the query terms. The method uses TreeBank Chunker [21] for parsing the text and Part-Of-Speech-Tagging.

The answers are typically noun phrases, listed in decreasing order of their scores. Their respective scores are calculated by the frequency of the query keywords co-occurring with such noun phrases. Since we evaluate queries non-conjunctively (i.e., andish mode), it is not required that all the keywords must occur in close proximity to a particular term, in order for that term to be considered an answer. The more query keywords surround a particular term, the higher is the likelihood for that term to appear in the top-k answer list (the higher is the score).

A term is considered as co-occurring with one or more query keywords, if it occurs within a system predefined distance. The distance is simply measured by counting the number of terms between any two phrases, e.g.,: "***Bill Clinton*** *studied in England,* ***Oxford***", in this example, the distance between the noun phrase "Bill Clinton" and "Oxford" is three.

Another factor influencing the scoring is subsumptions between entities. For example, "Bush" and "George Bush" may both appear in the documents. Here, we adopt the methodology in [22], i.e., preferring a longer and more specific entity name. Therefore, during scoring, subsumed entity's score is accumulated to that of a more specific entity names.

The **Named Entity Recognition** approach relies on Named Entity Recognition (NER) in order to extract entities with the same type as of the expected answer. NER suffers from two drawbacks: 1) the computation efforts are high, and 2) the available entity types are usually limited while domain specific modeling requires a lot of training. Since we use the OpenNLP [21] tool for entity recognition, this approach is limited only to the types that this tool currently supports: *Date, Person, Percentage, Location, Organization, Money and Time*. However, when applicable, the precision is high. In order to alleviate the limited availability of categories, we try to find out in the query whether the specified entity type is a subtype of one of the available type. For example, consider the triple query: ("Bill Clinton", "studied at", X), type(X, "University"). In order to evaluate this query, the type of the variable in the query must be in one of the supported types by the NER method, therefore we first apply NER on the keywords in the query, and find out that "University" is of type "Organization". A matching type increase ranking of a candidate noun phrase.

The **Dependency Analysis** approach relies on dependency analysis over the retrieved fragments of text, in particular, the sentences containing one or more query terms. This approach works by analyzing sentences and identifying relationships among subjects and objects. The keyword neighborhood approach fails to find answers when answer terms are not located within a predefined distance to the query terms. Dependency analysis provides linguistic distance between entities, instead of term distance in the text. Therefore, the limitation on a predefined term distance threshold is avoided. Such analysis is also quite useful in the cases when there are more than one subject and object in a sentence, as it helps figure out which noun phrase is most related to a predicate.

IV. RESULTS

For evaluating the system, we have pre-compiled a set of 32 triple queries, which are shown in the appendix A. The queries are partly taken from the QALD-1 Open Challenge[1] and the "Who wants to be a millionaire" show[2]. Precision, recall and F-measure (the harmonic mean of precision and recall) are then measured to compare different strategies. As is common for evaluating effectiveness of search engines, where the collection of documents is not predefined, we measure the metrics at top-k results in the answer set.

We have evaluated our system using various settings and strategies. In particular, we perform experiments using keyword neighborhood, named entity recognition, and shallow syntactic analysis. We further categorize these approaches into

[1]http://www.sc.cit-ec.uni-bielefeld.de/qald-1
[2]http://www.dadt.com/millionaire/

a number of sub-approaches. *Approach 1* is based on keyword neighborhood, its sub-variant 1-a computes the answer set from collecting all the surrounding words of the query terms in a set and ranking according to their frequencies. Approach 1-b is a refined version of 1-a, in which some decorating terms such as numbers, adjectives, verbs and adverbs are removed.

Approaches 1-a, and 1-b would not perform well for queries that should return more than one word as an answer. In approach 1-c, noun phrases in high-ranked sentences would be detected and ranked based on their frequency in those sentences. Approach 1-c does not increase the precision, mainly because the bounded variables (in the form of noun phrases in the query) are not distinguished from others and the retrieval of such phrases degrades the answer quality. To address the issue, approach 1-d rules out the noun phrases in the query from the answer set, unless the same noun phrases appear more than once in one sentence (this is to avoid missing answer to a self-referring query). With the distinction step, approach 1-d performs very well on average.

Approach 2 is designed to employ named entity recognition. In this approach, we identify entities using the OpenNLP tool, we check the type of identified entities, and we return to the user only those answers which are of the same type as expected in the query. This approach does not obtain a best precision, because some queries in the workload does not have a type that is supported by the out-of-box categories of OpenNLP.

We have performed another set of experiments, denoted emphapproach 3 in the graph. This approach makes use of dependency analysis provided by TreeBank Parser. Approach 3-a and 3-b are two variants, in which the former is a simple predicate-centered approach by searching for dependents of verb phrases matching the query predicate, and the latter restricts the answer set of 3-a to those that containing bounded subjects/objects. Though requiring a lot of computation, dependency analysis alone does not perform well in terms of precision. This is mainly due to the presence of predicate synonyms in the text corpora, which may hint at the missing answers. An improvement can be to make use of dependency analysis together with our approach 1 to enhance ranking instead of restricting the answer set. In Figure 2 we have shown the average precision of all approaches over the query set. The average precision has been calculated at four different k levels, namely top-1, top-3, top-5, and top-10 levels. As it can be seen in the figure, the refined approaches perform significantly better, with the approach 1-d performing the best.

The recall measures in Figure 3 are particularly close in approaches 1-b and 1-d. Approach 1-b gives as answers only words (not necessarily complete answers), while approach 1-d gives as answers noun phrases. For example, for the query (X, recorded, Thriller Album), approach 1-b had "Michael", or "Jackson" as result, whereas the approach 1-d produced "Michael Jackson" as an answer. This happens because approach 1-d identifies "Michael Jackson" as a noun phrase.

Finally, Figure 4 shows the f-measure as a single measure of the system's overall performance. Approach 1-d performed best in this work.

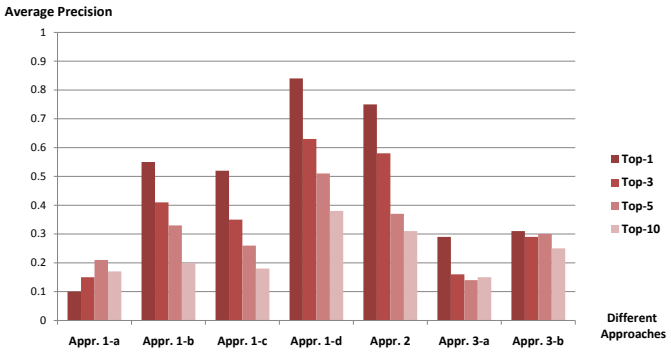

Fig. 2. Average precision for all approaches over the whole query set

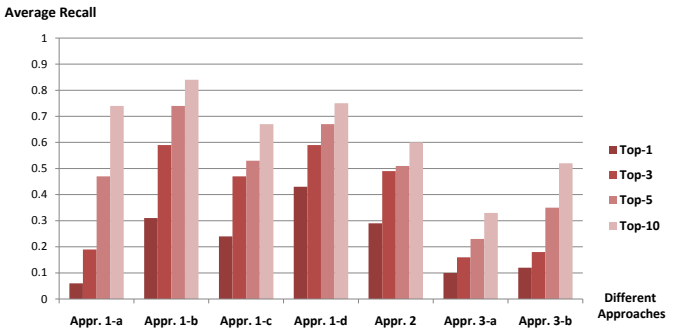

Fig. 3. Average recall for all approaches over the whole query set

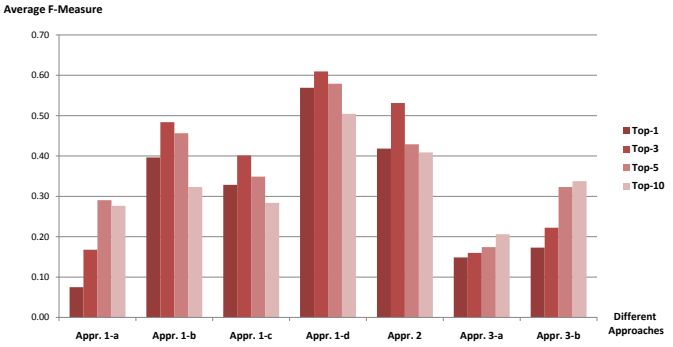

Fig. 4. Average F-measure Graph

All experiments were run on a PC equipped with an Intel Quad-Core CPU running at 2,83 GHz and 8 GB RAM. However, we do not make use of multiple threads in this first prototypical implementation. Google was used as search engine. In approach 1, each query took in average about 10 seconds, about 20 seconds for approach 2, and about 40-60 seconds for approach 3. Please note that we focussed so far on the quality of the results rather than on the performance. Thus, there is still a lot of room for improving the efficiency of our system.

V. Conclusions and Future Work

In this paper, we have presented a novel prototype representing a first step towards answering structured queries over unstructured data in a query-in-tuple-out paradigm. In comparison to existing work, our approach requires no set-up efforts, processes at runtime the most up-to-date version of text documents (i.e., no stale snapshot), and extracts ef-

ficiently the relevant portion of a whole corpus. Since only light-weighted linguistic analysis is adopted, it is possible that some potential answers, which can be obtained through heavy computational linguistic procedure, will be missed. However, as a runtime-only approach, our approach is a trade-off between computation overhead and query answer quality. In addition, the evaluation has shown that the light-weight linguistic techniques used in approach 1, combined with the frequency analysis delivers already very satisfying results.

There are several directions for future work. First, we can extend to allow projections of variables in CTQ, which requires aggregation of scores of tuples. Second, we assume in our current approach independence among triples within a CTQ. Further investigation can be carried along to employ a more sophisticated model. Last but not least, we see a lot of optimization opportunities in our current prototype.

ACKNOWLEDGMENT

This work is supported by the DFG Research Cluster on Ultra High-Speed Mobile Information and Communication (UMIC, http://www.umic.rwth-aachen.de).

REFERENCES

[1] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *SIGMOD Record*, vol. 34, no. 4, pp. 27–33, 2005.

[2] M. A. V. Salles, J.-P. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi, "itrails: Pay-as-you-go information integration in dataspaces," in *Proc. VLDB*, 2007, pp. 663–674.

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web," in *Proc. IJCAI*, 2007, pp. 2670–2676.

[4] O. Etzioni, M. J. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in knowitall: (preliminary results)," in *Proc. WWW*, 2004, pp. 100–110.

[5] E. Agichtein and L. Gravano, "Snowball: Extracting relations from large plain-text collections," in *Proc. 5th ACM Intl. Conf. on Digital Libraries*, 2000, pp. 85–94.

[6] J. Liu, X. Dong, and A. Y. Halevy, "Answering structured queries on unstructured data," in *Proc. WebDB*, 2006.

[7] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: a system for keyword-based search over relational databases," in *Proc. ICDE*, 2002, pp. 5–16.

[8] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword Search in Relational Databases," in *Proc. VLDB*, 2002, pp. 670–681.

[9] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *Proc. ICDE*, 2002, pp. 431–440.

[10] J. X. Yu, L. Qin, and L. Chang, "Keyword search in relational databases: A survey," *IEEE Data Eng. Bull.*, vol. 33, no. 1, pp. 67–78, 2010.

[11] P. Roy, M. K. Mohania, B. Bamba, and S. Raman, "Towards automatic association of relevant unstructured content with structured query results," in *Proc. CIKM*, 2005, pp. 405–412.

[12] M. J. Cafarella, C. Ré, D. Suciu, O. Etzioni, and M. Banko, "Structured querying of web text: A technical challenge," in *Proc. CIDR*, 2007.

[13] M. N. Gubanov and P. A. Bernstein, "Structural text search and comparison using automatically extracted schema," in *Proc. WebDB*, 2006.

[14] A. Jain, A. Doan, and L. Gravano, "Sql queries over unstructured text databases," in *Proc. ICDE*, 2007, pp. 1255–1257.

[15] S. Sarawagi, "Information extraction," *Foundations and Trends in Databases*, vol. 1, no. 3, pp. 261–377, 2008.

[16] W. W. Cohen, "Integration of heterogeneous databases without common domains using queries based on textual similarity," in *Proc. SIGMOD*, 1998, pp. 201–212.

[17] Z. Zheng, "Answerbus question answering system," in *Proc. HLT Human Language Technology Conference*, 2002.

[18] C. C. T. Kwok, O. Etzioni, and D. S. Weld, "Scaling question answering to the web," *ACM Trans. Inf. Syst.*, vol. 19, no. 3, pp. 242–262, 2001.

[19] E. Prud'hommeaux and A. Seaborne, "Sparql query language for rdf," http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, 2008, W3C Recommendation.

[20] J. R. Finkel, T. Grenager, and C. D. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proc. ACL*, 2005.

[21] openNLP, "Natural language processing tools," http://opennlp.sourceforge.net/projects.html.

[22] A. G. Parameswaran, H. Garcia-Molina, and A. Rajaraman, "Towards the web of concepts: Extracting concepts from large datasets," *PVLDB*, vol. 3, no. 1, pp. 566–577, 2010.

APPENDIX

Set of the test queries used for evaluating the system:

1) (X, "recorded", "Thriller Album"), type(X, "Person")
2) (X, "hosts", "jeopardy show"), type(X, "Person")
3) ("Kurt Cobain", "founded", X), type(X, "Band")
4) (X, "founded", "McDonald's"), type(X, "Person")
5) ("la via lactea", "means", X)
6) ("barrack Obama", "has", X), type(X, "Nickname")
7) ("Courtenay Cox", "starring", X), type(X, "Comedy on ABC")
8) ("http", "stands for", X)
9) ("baklava", "made of", X)
10) ("bill Clinton", "studied at", X), type(X, "University")
11) (X, "was", "third president of USA"), type(X, "Person")
12) (X, "won", "physics Nobel prize"), type(X, "Person") date(X, 2010)
13) (X, "appeared on", "laugh in"), type(X, "President of USA")
14) (X, "called", "FreshMaker"), type(X, "Candy")
15) ("cup of Joe", "is", X), type(X, "Beverage")
16) (X, "won", "peace Nobe prize"), type(X, "President of USA")
17) (X, "directed", "Fahrenheit 9 11"), type(X, "Person")
18) ("Kobe Bryant", "plays", X), type(X, "Sports")
19) (X, "refers to", "rear of ship"), type(X, "Term")
20) (X, "ordered", "war"), type(X, "President of USA")
21) (X, "ended", "war"), type(X, "President of USA ")
22) (X, "is", "national bird of USA")
23) (X, "is", "name for a male goat")
24) ("empire state building", "located in", X) type(X, "City")
25) ("john Lennon and Paul McCartney", "founded", X), type(X, "Band")
26) (X, "founded", "Yahoo Company"), type(X, "Person"
27) (X, "was", "female German chancellor")
28) (X, "starring in", "batman begins"), type(X, "Actor")
29) (X, "recorded", "Unison Album"), type(X, "Person")
30) (X, "hosts", "daily show"), type(X, "Person")
31) (X, "born in", "1945"), type(X, "President of Israel")
32) (X, "won", "peace Nobel prize"), type(X, "President USA"), (X, "ended", "war"), type(X, "President USA")