Data Extraction from the Web Using Wild Card Queries

Davood Rafiei
Computing Science Department
University of Alberta
drafiei@cs.ualberta.ca

Haobin Li
Computing Science Department
University of Alberta
haobin@cs.ualberta.ca

ABSTRACT

This paper presents an overview of our work for searching and retrieving facts and relationships within natural language text sources. In this work, an extraction task over a text collection is expressed as a query that combines text fragments with wild cards, and the query result is a set of facts in the form of unary, binary and general n-ary tuples. Despite being both simple and declarative, the framework can be applied to a wide range of extraction tasks. This paper presents an overview of the work and its various components. We also report some of our experiments and an evaluation of the proposed querying framework in extracting relevant information to a task.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval; H.5.2 [**Information Systems**]: User Interfaces

General Terms

Algorithms,Experimentation,Measurement

Keywords

DeWild, Data Extraction, Web Search, Ranking

1. INTRODUCTION

The World Wide Web contains a vast amount of information and is a rich source for data extraction, but manually extracting data from the Web is a tedious and time consuming process, especially when a large amount of data matches the extraction criteria. Example extraction tasks include compiling a list of Canadian writers, finding a list of medications for a disease, etc. Unless such lists have already been compiled and made available on the Web, one has to query a search engine, examine the pages returned, and extract a handful of instances from each page (if there is any at all). The problem is further complicated by the flexibility of natural languages. Consider the example of extracting *Canadian writers*; many bona fide writers are not referred to as writers. Instead, they are often coined as *authors*, *novelists*, *journalists*, etc. If only the phrase "Canadian writers" is used in the query, many qualified instances will not be extracted, thus the extraction quality is compromised. Many previous data extraction systems either focus on a more specific task by imposing tight restrictions on the type of data that can be extracted (e.g. finding course offerings and job postings) or are only applicable to documents that follow a specific formatting (e.g. wrappers). For example, the KnowItAll [6] system can extract hyponyms of a user-specified class. The online prototype of the system is further extended to support a few more specific binary relations (e.g. *X "ceo of" Y*). A challenge facing many data extraction systems in general is that the extraction task often is not easy to define or a definition may not be accurate, leading to low precision and recall. Limiting the extraction task to a few predefined classes is one way to reduce the complexity of the problem.

This paper presents an overview of a framework that allows an extraction task to be encoded as a simple query. A query in this framework is a natural language sentence or phrase with some wild cards, and the result of a query is a ranked list of matching tuples. For instance, given the query "% is a car manufacturer", the output is expected to be a ranked list of car manufacturers, preferably the real car manufacturers ranked the highest. This query only uses one wild card, here denoted with %. In general, a query can use more than one % wild card, and the result of the query in this case is a table with one column for each occurrence of the wild card.

One of our contributions is a declarative querying framework for data extraction. Integration of wild cards in our queries can generally reduce the number of queries that must be issued, hence simplifying the extraction task. Another contribution is the idea of query expansion through a set of declarative rewriting rules between paraphrases. Since a given user query may not retrieve an adequate number of facts, rewriting rules are generally expected to improve the coverage of the queries and the quality of the results. Finally, as our last contribution, we report an experimental evaluation of our work in the setting of the Web.

The rest of the paper is organized as follows. Section 2 describes both the syntax and the semantics of wild cards, as well as the queries in our framework. An overview of our query evaluation in the context of the Web is given in Section 3. An experimental evaluation of the work is presented

CIKM'09, November 2–6, 2009, Hong Kong, China.

in Section 4 and the related work is reviewed in Section 5. We end the paper with conclusions and future work in Section 6. See the extended version of this paper [11] for more details on our query expansion, ranking algorithms and a more extensive evaluation of the work.

2. WILD CARD QUERIES

The use of wild cards is prevalent in many areas of computer science, with examples found in SQL, operating system shells and scripting languages such as Perl, Awk and Python. Unlike many of these systems, our introduced wild cards iterate over the domains of parts of speech or other meaningful natural language word groupings. In particular, we introduce two types of wild cards, namely * and %.

% wild card: The % wild card represents one or more noun phrases. A noun phrase may consist of one or multiple words; for instance, "movie" and "action movie" are both noun phrases. This wild card, when used in a query, indicates the location of a noun phrase or noun phrases that should be extracted. For example, the query "summer movies such as %" will extract noun phrases *Harry Potter*, *Shrek*, and *Spiderman* from the following sentence: "Popular summer movies such as Harry Potter, Shrek and Spiderman appeal to audience of all ages."

*** wild card**: The * wild card represents a set of phrases with the same or similar meanings to a given phrase. Consider the task of finding a listing of summer movies; we may type the query "% is a summer movie". However, some bona fide movies are often referred to as "films", "blockbusters", and so on. In a naive approach, one may have to try other terms similar to "movie" manually, save the results each time, and put the results together at the end. The naive method is tedious and inefficient. In our queries, a term may be enclosed within a pair of * to instruct that the search should be extended to include terms and phrases similar to the given one. For example, the user can re-formulate the query as "% is a summer *movie*", and the query will be automatically expanded to include related queries such as "% is a summer film", "% is a summer blockbuster", etc. The * wild card should always be used in pairs.

It is feasible to consider other wild cards. For instance, we could have wild cards that only match verbs, adjectives, or a union of nouns, verbs and adjectives. It is also possible to have a wild card that matches a prefix or suffix of a term or a fixed-length sequence of terms. In an attempt to keep the syntax of our queries simple, this paper only considers the two wild cards % and *, as discussed above. The following is a list of example queries:

- % is a *country*
- % is a summer *blockbuster*
- % invented the light bulb
- Google *acquired* %

A query may use any number of wild cards. Given a query with k % wild cards, the result of the query is a table with k columns, one for each % wild card. A query can also have any number of * wild cards. Given a query q with some * wild cards, let $q_1, \ldots, q_s$ be the set of queries that are obtained by replacing each * wild card with *similar terms*. A row matches q if it matches at least one of $q_1, \ldots, q_s$. For our purpose, two terms are considered *similar* if they have the same meanings (e.g. synonyms), one is a generalization of the other, or the two terms can be used interchangeably in the same context.

3. EVALUATING WILD CARD QUERIES – AN OVERVIEW

This section provides an overview of evaluating wild card queries over a text corpus. Without loss of generality, our discussion in this section is centered around the Web and uses the query "% is a *blockbuster*" as an example. In particular, we consider the scenario where the extraction engine is built on top of a search engine. Naturally the same steps can be taken when the source data is stored locally, with a difference that a local collection may be better indexed and the queries can be better optimized. To limit the scope of the paper, we do not address the issues related to indexing and query optimization.

Step 1 - query flattening As the first step, the query is analyzed and expanded by replacing the words enclosed by pairs of * wild cards (if any) with their similar terms. In the given query, the word "blockbuster" is enclosed by *'s; similar words to blockbuster, based on an online system [9], are "movie" and "film". Two new queries, "% is a movie" and "% is a film" are formed and added to the expanded query set. In general, more queries may be added if multiple synonyms are found.

Step 2 - query rewriting In the next step, each query in the query set is passed to a Part-Of-Speech (POS) tagger. Each tagged query is compared with a set of precompiled patterns for possible rewritings. The result of query tagging is not always reliable, in particular for short queries. To account for those cases, queries are also rewritten using rules that do not require tagging. Let's consider the query "% is a movie" first; after tagging, the query conforms to the pattern "NP1 is a(n) NP2" where NP stands for a noun phrase; note that the wild card % matches a noun phrase, as we defined earlier. A pattern may be found relevant to a pre-determined class, based on a specific relationship it describes, and may be rewritten by other patterns in the same class. The pattern "NP1 is a NP2", in particular, belongs to the *hyponym* class, since the template indicates that NP1 is a (hyponym of) NP2. Other patterns in the hyponym class include "NP2 such as NP1List", "NP2 including NP1List", etc. All patterns in the matching class (i.e. the hyponym class) are instantiated according to the matched query. Thus, the query set is expanded with extra queries like "movies such as %" and "movies including %". More detailed discussion of our query rewritings can be found elsewhere [11]. Similarly, the query "% is a blockbuster" also matches a pattern in the hyponym class, and the query set is further expanded. If the query cannot match any pattern, no query expansion will occur at this step.

Step 3 - information retrieval engine As the third step, all queries in the query set are sent to a search engine. For each query, the matching snippets are downloaded for further processing. When there is a large number of matches, only a fixed number of them are selected. HTML tags are stripped from downloaded snippets for each query and the remaining text is analyzed to identify the pieces that match the query. Noun phrases that appear in the positions of % wild cards of a query are extracted from the text and

are saved in the result set. Words other than noun phrases should not be extracted even if they appear in target locations. Suppose the query "% invented the light bulb" is sent to a search engine and the following two snippets are among those returned.

- Thomas Edison is often said to have *invented the light bulb*.
- We all learned in our history classes that Thomas Edison *invented the light bulb* in 1879.

The POS tagger identifies that the word "have" in the first snippet is not a noun phrase, while "Thomas Edison" from the second snippet is. Therefore, the phrase "Thomas Edison" is extracted but "have" is not.

Step 4 - relevance ranking The result of extraction in the previous step is a set of rows; for the given example, each row is a noun phrase. A ranking algorithm is applied to the extracted set. More details on our ranking algorithms can also be found elsewhere [11]. Finally, a sorted list of rows is returned. The rest of this paper will focus on *query expansion* and *relevance ranking*.

4. THE TOOL AND OUR EXPERIMENTS

To evaluate our querying framework and ranking algorithms, we built a search tool, called *DeWild*, which relies on the Web as its source of data [1].

Using the Web compared to a closed collection has both benefits and drawbacks. A benefit is that its information redundancy can compensate for the relatively small size and coverage of our rewriting rule set and the lightweight NLP techniques used. A drawback is that the text collection often is not clean and there are many bogus tuples that need to be filtered.

DeWild takes advantage of existing commercial search engines and currently queries Google via its search API. In our experiments, 200 snippets are downloaded for each extraction pattern (our online system only downloads 30 snippets per pattern and also lists the set of extraction patterns that are tried). If there are fewer than 200 snippets found for a pattern, then all available snippets are downloaded[2]. The snippets returned by a search engine typically consists of the search query and its surrounding text. Since the target data appears immediately before or after the user query, they can be often extracted using the snippets only (without downloading the actual pages), hence network and processing costs are significantly reduced. Though it should be noted that a snippet may lack sentence boundaries, and this can reduce POS tagging accuracy. For ranking the results, DeWild natively uses an algorithm referred to as PT-hits (see [11] for more details). Our reported experiment in this paper is also based on the same ranking.

A publicly available POS tagger called NLProcessor[3] is used to identify the part of speech from retrieved text, so that only noun phrases are extracted for % wild cards. For * wilds cards, our system uses a collection of related words automatically compiled [9] from Wall Street Journal corpus, but it can equally use other collections as well. Next we report our experiments with DeWild.

[1] *DeWild* stands for Data Extraction using Wild cards. The system is available online at `dewild.cs.ualberta.ca`.

[2] Our online demo downloads at most 30 snippets for each query and each rewriting to keep the response time short.

[3] `www.infogistics.com/textanalysis.html`

	Question length		
	< 6 terms	≥ 6 terms	all
DeWild	0.88	0.41	0.68
OpenEphyra	0.41	0.22	0.32

Table 1: Precision of the results for natural language questions

To measure the effectiveness of the proposed querying framework in expressing different extraction tasks, we used the tool to express and answer natural language questions seeking short answers. We randomly selected 100 natural language questions from the AOL query log [10], with the selection criteria that the queries were all started with one of the wh-words *what, which, who* and *where* and were followed by *is, was, was* or *were*. The break-down of the queries were as follows: what-84, who-6, where-7 and which-3. The *why* queries were excluded since these queries often seek long answers and cannot be expressed in our framework. Even among the selected queries, not all queries were expected to have short noun phrases as answers (e.g. What is a hedge fund? Or, what are the causes of poverty?); in fact, 58% of the queries were in this category, and they could not be properly expressed as queries. Another 7% of the queries were ill-formed search queries in the form of questions which we didn't expect them to have answers (e.g. what is my account balance?). We removed both classes of queries before further considerations. We further removed another 2% of the queries where we couldn't find an answer after an extensive search on the Web, and we were not sure if the questions had an answer at all. The remaining 33% of the queries were all expressed as queries in our framwork and were evaluated using our online tool.

As a baseline for comparison, we also passed the queries as they were expressed to OpenEphyra [4], a full-fledged open-source question answering system. Similar to ours, OpenEphyra uses the Web, and in particular the Google search engine, to evaluate the questions, hence both DeWild and OpenEphyra had access to the same data collection. However, OpenEphyra does much more extensive parsing of the questions and the answers, and has components for detecting answer types (of questions) and named entity classes in text.

Top 5 results of each query, as returned by each comparison system, were given to two annotators who were asked to check of the set included a correct answer to the original question. Table 1 shows the precision, averaged over two annotations per question, for both systems. DeWild achieves a much better precision, with a good annotator agreement (kappa) of 0.66, for both short and a bit longer questions, retrieving a correct answer for twice as many queries as OpenEphyra.

5. RELATED WORK

There is a large body of work on question answering. Many systems use a combination of NLP techniques (deep or shallow), learning algorithms and hand-crafted rules to classify the questions and to establish relationships between

[4] http://www.ephyra.info

terms of a question and a possible answer sentence (e.g. [8, 5]). Despite some overlap, there are fundamental differences between our work and the work on question answering. Our work is not a replacement for QA systems that can often handle complex questions; it is rather a natural way of expressing short questions or extraction targets. It is possible to integrate our work within a question answering system if natural language questions can be mapped to DeWild queries.

Large-scale data extraction from the Web has been the subject of various recent work [1]. In particular, Brin [4] suggests an algorithm which takes a small number of examples of a class as a seed set and extracts more examples of the same class. His algorithm learns a set of extraction patterns for each page (or pages with the same URL prefix) that contains some of the examples and use those patterns to extract more tuples from the same page(s). This algorithm does a good job when data is structured in a tabular format but is not expected to work on free text. This is because it is generally unlikely to find more than one example (of the seed set) in a text document such that their surrounding texts are the same.

KnowItAll [6] takes the description of a concept or class (e.g. cities) as input and extracts instances (e.g. Paris, New York, ...) of the class. The system maintains a set of rules which can be instantiated with an input class to produce keywords that must be used to extract the instances of the class. KnowItAll uses co-occurrence statistics, specifically mutual information, to assess the relatedness of each instance. Our approach differs from KnowItAll in several important aspects: First, the query-based interface and the support of wild cards make DeWild more adaptive to different extraction tasks. Second, unlike KnowItAll where a concise description of a class must be given, a DeWild query may specify only the context in which the instances may appear, and this is useful when a concise class description is not available (this problem is somewhat addressed in TextRunner [2]). Last but not least, our approach of porting link-based ranking algorithms for assessing extraction results from text is novel and performs better than the one used in KnowItAll.

Other related work includes the work on query expansion [3] and the work on Web query languages and wrappers (see [7] for a survey of this area before 1998). Finally, Google's *fill-in-the-blank* is related to our wild cards but is different. Google returns a ranked list of pages for a fill-in-the-blank search but the ranking is different (and the detail is not published). An evaluation of Google's fill-in-the-blank compared to DeWild can be found in the extended version of this paper [11].

6. CONCLUSIONS

We presented a framework for querying and large-scale data extraction from natural language text, and evaluated the effectiveness of our framework within a few data extraction tasks on the Web. We analyzed our rankings of the results in terms of the stability and the locality of the scoring functions and conducted experiments comparing their effectiveness in terms of precision and recall. Our querying interface is intuitive for writing queries and scalable to large number of rewritings and with more wild cards.

Our work opens a few interesting directions for future work. First, it would be interesting to study other wild cards, querying schemes and formalisms that are simple for writing queries and at the same time have a well-defined syntax and semantics for query evaluations and optimizations. We consider our wild card queries as a first step toward more formal querying of natural language text. Second area for possible future work is data storage and indexing. Despite the extensive work on indexing free text, there is not much work on indexing natural language text in particular. Given that natural language text can be parsed, there is much room for research on building indexes that are aware of sentence structures and queries. Third, a more formal query syntax and semantics opens the door for more study on mapping queries to evaluation plans and access path selection and optimization. These are important issues when querying large repositories and/or posing complex queries. Another area for further study is on extracting *n-ary* relations for $n > 3$; the problem in general is difficult since the columns of target rows can be scattered in multiple sentences. Yet one more area is on mapping natural language questions to more formal queries that can be efficiently evaluated.

Acknowledgments

This work is supported by Natural Sciences and Engineering Research Council of Canada.

7. REFERENCES

[1] E. Agichtein. *Extracting relations from large text collections*. PhD thesis, Columbia University, 2005.

[2] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *Proc. of ACL*, 2008.

[3] M. W. Billoti. Query expansion techniques for question answering. MSc thesis, MIT, 2004.

[4] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at the EDBT Conf.*, pages 172–183, 1998.

[5] S. Dumais, M. Banko, E. Brill, J. Lin, and A. Ng. Web question answering: Is more always better? In *Proc. of the SIGIR Conf.*, pages 291–298, 2002.

[6] O. Etzioni and et al. Web-scale information extraction in knowitall: (preliminary results). In *Proc. of the WWW Conf.*, pages 100–110, 2004.

[7] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the World Wide Web : a survey. *ACM SIGMOD Record*, 27(3):59–74, September 1998.

[8] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *Proc. of the WWW Conf.*, pages 150–161, 2001.

[9] D. Lin. Using syntactic dependency as local context to resolve word sense ambiguity. In *Proc. of the ACL Conf.*, pages 64–71, 1997. (online demo at www.cs.ualberta.ca/∼lindek/demos/depsim.htm).

[10] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *The 1st Intl. Conf. on Scalable Information Systems*, 2006.

[11] D. Rafiei and H. Li. Wild card queries for searching resources on the web. *CoRR*, arXiv:0908.2588v1, 2009.