Data Extraction from the Web Using Wild Card Queries

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

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

ABSTRACT

We propose a domain-independent framework for querying and extracting large volumes of facts stored in natural language text sources. In this framework, an extraction task is expressed as a declarative query, combining text fragments with wild cards, and the result of the query over a natural language text collection is a set of facts in the form of unary, binary and general n-ary tuples. A significance of our querying mechanism is that, despite being both simple and declarative, it can be applied to a wide range of extraction tasks. However, a user-specified query over natural language text can be too restrictive and may not return enough matches. Unlike term queries which can be relaxed by removing some of the terms (as is done in search engines), removing terms from a wild card query is more challenging and can ruin its meaning. Also, any query expansion has the potential to introduce false positives. In this paper, we address the problem of query expansion, and also analyze a few ranking alternatives to score the results and to remove false positives. We conduct experiments evaluating our scoring functions and comparing the results of our framework with alternative approaches. The experiments show that our approach outperforms an alternative less general system and a well-known statistics-based method in terms of both precision and recall.

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 that can treat 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 [12] 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.

In this paper, we address the problem by introducing 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[1] 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.

Our first contribution 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. In our earlier example about *Canadian writers*, for instance, a user can use one type of wild card to indicate that terms similar to *writers* should also be considered. Another type of wild card may be used to indicate a probable position of the desired data, from which values can be extracted. Combining such wild cards with natural language phrases can provide a simple but powerful interface, which can handle much more extraction tasks than previous systems. There is a close correspondence between our queries and star-free regular expressions; our queries make use of certain abstractions geared toward natural languages which make it simpler

[1]We assume query phrases are in English, but our framework should be applicable to other languages as well.

2008

to write queries. Since the result of a query is a relation, our queries with some syntactic sugar can be integrated in the *from* clause of SQL queries.

Our second 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. Our experiments, as reported in Section 6, show that increasing the number of rewritings can improve both recall and precision.

As our third contribution, we address the problem of ranking in the context of rewriting rules. We propose a class of algorithms for ranking the extracted results where each algorithm exploits some of the relationships that exist between the set of matching tuples and also between the set of query patterns. We use the general term *pattern* to refer to both query and query rewriting. When the results are ranked, it is possible to set a cutoff threshold to filter invalid rows from the result, making the final results more accessible to the user.

Finally, as our last contribution, we theoretically analyze our algorithms and experimentally evaluate them in the setting of the Web. Our experimental results include comparisons with previously-proposed approaches to data extraction.

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. Section 4 discusses the details of our rewriting rules and patterns. Our ranking algorithms are discussed in Section 5. Experimental results are presented in Section 6 and the related work is reviewed in Section 7. We end the paper with conclusions and future work in Section 8.

2. WILD CARD QUERIES

To express extraction tasks concisely and in a flexible manner, we make use of wild cards in our queries. The use of wild cards is prevalent in many areas of computer science. Examples are 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 groupings of natural language words. In particular, we introduce two types of wild cards, namely * and %.

2.1 Query syntax and semantics

The syntax and semantics of our queries are as follows.

% 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.

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, our queries extend phrase queries of a typical search engine with 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. We assume that the result of a query is ranked and each row is assigned a score as an indication of the level of support the row receives. This score may depend on the size and the coverage of the text collection that is being queried and the set of query rewritings that are being used. Section 5 discusses a few measures to rank the matching tuples of a query.

A query can 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 $q_1, \ldots, q_s$; the score of the row for q is an aggregation of the scores of the row for $q_1, \ldots, q_s$. 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. The similar terms can be often obtained from dictionaries, thesaurus, online corpus [18], etc.

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 *country*" 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 can be better indexed (e.g. [7, 25, 9]) 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 "country" is enclosed by *'s; a similar word to country, based on an online system [18], is "nation". A new query, "% is a nation", is formed and added to the expanded query set. More than one query can 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 country" 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 "countries such as %" and "countries including %". Section 4 discusses our query rewritings in more detail. Similarly, the query "% is a nation" 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. Section 5 gives the details of our ranking algorithm. Finally, a sorted list of rows is returned. The rest of this paper will focus on *query expansion* and *relevance ranking*.

4. REWRITING QUERIES

A challenge in querying and information retrieval from natural language text is the possibility of a mismatch between the expressions of queries and texts that have the relevant information. A fact can potentially be expressed in many different contexts and a query that gives one context can miss many qualified candidates. We propose rewriting rules to express the set of transformations that can be applied to a wild card query leading to alternative query expressions that are expected to return the same or semantically related results but are syntactically different. Query rewriting is expected to increase recall, as it can be seen from our example in Section 3. Our experiments in Section 6 confirm that there is also a correlation between the number of rewritings and the precision of the retrieved results. There are more evidence in favor of rewriting natural language questions. For example, at TREC-10 QA evaluation, the winning system used an extensive set of rewriting rules as its only resource [23].

4.1 Rewriting Rule Language

The rewriting rule language lists different ways of rewriting a query. Each rule here is of the form *rule-head* $\rightarrow$ *rule-body*. A rule head consists of one or more regular expressions, and a rule body consists of one or more rewritings with place holders. Multiple regular expressions in the head or rewritings in the body can be listed each in a new line. A rule matches a query if any one of the regular expressions in the head matches the query. When a rule matches a query, the query is expanded with all rewritings in the rule body. For each match, keywords from the query may be remembered using *capturing groups* (e.g. parentheses) and the remembered values may be recalled using back references in the rule body. The remembered values can be transformed (e.g. from a singular noun to a plural noun) before being used in the rewritings. Each transformation is usually done by looking up a table, from a set of tables compiled in advance from a dictionary, thesaurus, and other offline sources. This allows us to write generic rewritings that can match a large number of queries. Our rewriting rule language is extendable and one can add more rules as they become available. Here is an example of a rule. Given the query "countries such as %", the rule generates "%, and other countries" and "% is a country" as possible rewritings.

```
(.+),?  such as (.+)
(.+),?  including (.+)
→
$2, and other $1 && plural($1)
$2 is a $1       && singular($1)
```

Let n denote the average number of rewritings produced for each query. If a query uses k star wild cards and each of these wild cards is replaced with $m > 0$ similar terms on average, the query expansion would produce $m^k(n+1)$ queries. To appreciate the power of the rewriting rule language, consider a manual data extraction where one has to enumerate and try many of those queries manually.

4.2 Compiling Rewriting Rules

Preparing an exhaustive set of rewritings for a small set of queries is not hard and may be done manually. Compiling a comprehensive set of rewritings for a large set of possible queries in advance is more challenging. Given two text fragments, deciding if one entails another is in general

computationally intractable, so is the problem of deciding query rewritings. A more pragmatic approach is to specialize the rules to more specific domains; this is also the approach adopted by the Pascal Recognizing Textual Entailment (RTE) challenge benchmark [22].

On the other hand, the effectiveness of a rule depends on the precision and the recall for its rewritings and the fraction of queries the rule matches. We can group the rewriting rules into two categories: generic and specific. The generic rules potentially match many queries and can be compiled independent of a particular domain, whereas the specific rules are domain-dependent. Two classes of generic rules that we can easily identify are *hyponyms* and *morphological variants*. A hyponym pattern describes lexico-syntactic relations that can be used to infer one element is a hyponym of another within a sentence. Hearst gives a list of hyponym patterns [14]. A sample of hyponym patterns (either from Hearst's or hand-crafted by us) can be found in Table 1.

NP1 {,} "such as" NP2List
"such" NP1 "as" NP2List
NP1 {,} "especially" NP2List
NP1 {,} "including" NP2List
NP2List "and other" NP1
NP2List "or other" NP1
NP2, "a(n)" NP1
NP2 "is a(n)" NP1
NP1 NP2

Table 1: Hyponym patterns

The morphological variants of verbs are useful for rewriting many queries that contain verbs. A given query may be rewritten by simply changing its verb tense and without much affecting its meaning. Many extraction tasks are expressed in the form of "Subject transitive-Verb Object" which can be rewritten in a passive form and vice versa. For example, if a user wants to find out who invented the light bulb, she can express the extraction as "% invented the light bulb"; a rewriting of the query is "the light bulb was invented by %". Our morphological patterns enumerate different verb tenses (e.g. present tense, past tense, ...) and use both active and passive forms. We were able to express all the relationships described by patterns in the hyponym and morphological classes as rules in our rule language.

Although generic patterns and rewritings can be applied to a wide range of queries, it is not hard to find queries where no generic pattern is applicable or sufficient. In both cases, rewriting rules need to be customized to a particular domain or extraction task. Specialized rules are likely to match a larger number of high quality tuples, which will lead to improvements in both recall and precision. Manually compiling specific rewriting rules for potentially large number of different queries can be expensive. There is an active research on automatically generating paraphrases in more specific settings [19, 21], and the results are encouraging. A notable work is by Lin and Pantel [19] on gathering over 182,000 classes of similar paths in the dependency trees of a parsed newspaper text corpus. The collection should be used with care though since two patterns in the same class may not be paraphrases but only related. The same or a similar algorithm may be used to find closely related relationships from a given text corpus, and these relationships can be translated into rewriting rules after further inspections and verifications.

4.3 Rewriting Quality

Given a query, all rewritings are not expected to be equally *strong* in terms of the quality of the results they may retrieve. For example, "NP1 such as NP2List" is considered a strong pattern in our hyponym class because a noun phrase that appears at NP2List is very likely to be a hyponym of the one that appears at NP1. On the other hand, "NP2, a(n) NP1" may be considered a *weak* pattern because sometimes the hyponym relation inferred by this pattern is incorrect (e.g. "select a city, a country, and a region from the list."). For the rest of the time, the pattern can be used to extract hyponyms from sentences like "...New York, a city of neighborhoods ...". Similarly, "NP1 NP2" may also be seen as a weak pattern, but we find it very effective in certain cases, such as the names of people. For example, the template can be used to infer from the sentence "Prime Minister Paul Martin attends a Canada Day ceremony ..." that "Paul Martin" is a "Prime Minister". The effects of using weak patterns are two folds. First, weak patterns can become strong ones in some cases, and under those circumstances they can improve both recall and precision. Second, weak patterns often introduce more false positives than other patterns. We believe that the negative effects of weak patterns are alleviated since the final results are ranked and the false positives are likely to be removed or assigned very low ranks.

5. BRINGING ORDER TO RESULTS

The data extraction process discussed in this paper can accumulate a large set of candidates. Some of the candidates are correct, meaning that a user would expect to see them, while the rest are errors.

5.1 Sources of Errors

A query typed by user can match many non-relevant rows, in the sense that they may not be anticipated by the user. For instance, the query "% is a country" matches the statement "Joe is a country singer," thus Joe will be added to the list of country names. Certain Natural Language Processing (NLP) techniques may be employed to reduce the number of those false positives [12]. Using these techniques can be expensive, and based on our experiments, the techniques do not scale up well to large volumes of data and queries (e.g. on the Web). In general, broad queries are likely to match more false positives. Rewriting queries can also broaden the queries and introduce additional false positives.

False positives also arise when queries are posed to uncontrolled collections such as the Web which contains many incorrect statements. Since the published content may not be verified for correctness, statements, such as "New York is the capital of the United States" are not rare.

One more source of error is due to using a POS tagger. Although POS taggers produce good results most of the time, the accuracy usually depends on factors such as the number of words in the lexicon, etc. Sometimes verbs are misclassified as noun phrases, or vice versa.

Since correct extractions are inter-mixed with errors, it is important to rank all candidates in terms of their relevance to the user query. A good ranking algorithm should consistently rank correct matches higher than errors, so that errors are pushed down to the bottom of the sorted list. A good ranking would make it easier to draw a cutoff line somewhere in the sorted list, so that we can trade recall for higher precision.

5.2 Ranking Heuristics

A problem related to ours is ranking the result of a natural language question over a text corpora, for instance in a question answering system. The precision of an answer usually depend on factors such as the quality and the size of the corpora and the relationships between the text of a question and possible answers. In our case, given a query and a set of rewritings, we want to find out meaningful ways of ordering the results. In this section, we present a few heuristics before we analyze them within a more general ranking scheme in the next section.

Number of Matched Pages or Documents (NPages): Relevant tuples are likely to appear frequently within a query pattern[2] or one of its rewritings. One heuristic is to rank a tuple based on the number of pages or documents in which it matches the query or one of its rewritings. On large collections and on the Web, it is either costly or impossible to access all pages, and the numbers could be approximated using a sample.

Mutual Information (MI) Another ranking scheme which has been used previously to quantify the relationship between two random variables is the *Mutual Information* (MI). If we denote the probability that a document contains the text of query q (ignoring wild cards) with $P(q)$, the probability that a document contains a row r with $P(r)$, and the probability that a document contains a proper encoding of r in q with $P(q,r)$, then the mutual information between q and r is defined as

$$MI(q,r) = log\frac{P(q,r)}{P(q)P(r)}.$$

In some formulations of the mutual information, the above formula is multiplied by $P(q,r)$ [8]. This measure is used in the past, for instance to evaluate the association between words [10], and also between the instances of a class and a discriminative phrase [12]. In our case, since $P(q)$ is fixed for a given query, the score of a tuple can be estimated as the ratio $P(q,r)/P(r)$. For a given query and tuple, the MI measures the conditional probability that the tuple appears within the query template given that the tuple appears in a document.

Number of Matched Patterns (NPatterns) Another simple ranking is to count for each tuple, the number of different patterns (including the query and its rewritings) that would extract the tuple. Because of the semantic relationship between a query and its rewritings, if a tuple is retrieved by multiple patterns, then there is probably a good indication that the tuple is indeed a good match.

Discussion of Ranking Heuristics Our experiments comparing these heuristics (as reported in Sec. 6) show mixed results for different queries. A general drawback for NPages and NPatterns is that all query patterns are treated equally important. With NPages, some correct instances that are not frequent cannot be well-separated from false positives. Also under NPages, the scores would be inflated when there are duplicates (such as those on the Web). A drawback for MI is that a tuple may not appear with the query but it may appear with one of its rewritings. Selecting a single pattern is not guaranteed to achieve a high recall. Also it is not clear how MI can be extended to account for multiple rewritings of a query and also queries with multiple extractors. Furthermore, obtaining hit counts can be costly and may not be reliable (e.g. see [3]).

[2]The query would not have been issued in the first place if it is assumed otherwise.

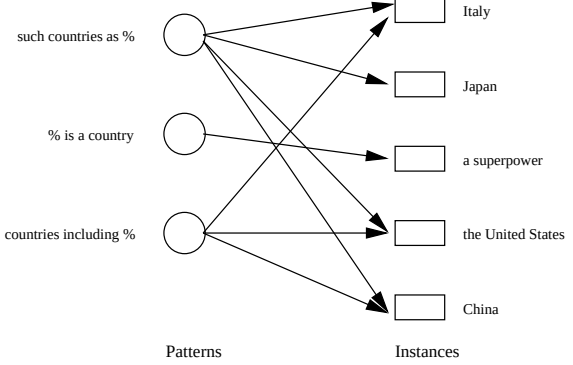

Figure 1: Mutually reinforcing relationship between patterns and tuples

5.3 Relationship Graph between Patterns and Tuples

Let S_P denote the set of query patterns (i.e. the user-specified query and all of its rewritings) and S_T denote the set of matching tuples. Consider the bipartite graph G formed between S_P and S_T with an edge from $p \in S_P$ to $t \in S_T$ if t matches p in some text. In some settings, the edges of the graph may be assigned weights to indicate the degree of a match. We define a ranking F as a function that maps S_T to an n-dimensional vector where n is the size of S_T. Some of our earlier heuristics can be seen as special cases of this ranking function. In particular, $NPatterns(t) = indegree(t)$ and

$$NPages(t) = \sum_{p \in S_P} w(p \rightarrow t)$$

where $w(p \rightarrow t)$ is the number of pages that give rise to a match between t and p.

A limitation of NPatterns and NPages is that all patterns have the same influence on the scores. Our observations, however, indicate that often there are a few patterns that retrieve many good quality rows while the rest retrieve many false positives. To remedy the problem, we propose weighting the patterns and propagating the weights to the matching tuples. The weights may be assigned at the same time rewritings are compiled if they are available and are not expected to change much. For generic rewritings, however, the weights can change from one query to next and a more dynamic weighting scheme may be preferred.

A hypothesis is that *good tuples* and *good patterns* exhibit a mutually reinforcing relationship: a good tuple is extracted by many good patterns; a good pattern extracts many good tuples. For example, if *Canada* is indeed a good match for the query "% is a country", it should be extracted by many good related patterns, such as "countries including %", "such countries as %", and so on. Similarly, if "countries such as %" is indeed a good pattern for extracting country names, it should extract many good instances, like *the U.S.*, *Canada*, *China*, etc. This mutually reinforcing relationship between patterns and tuples is illustrated in Figure 1.

The same hypothesis is made in a hyperlinked environment where several ranking algorithms are proposed for finding authoritative Web pages. Many of the observations made

for hyperlinked pages are consistent with those we have seen between patterns and tuples, hence the ranking algorithms developed there may be applied here. Our experiments use an adaptation of Kleineberg's HITS algorithm [15], but there are indications that other two-level influence propagation algorithms can equally be used.

Algorithm PT-hits Let's associate weight $w_T(t)$ to each tuple t, and weight $w_P(p)$ to each pattern p. In an iterative and alternating fashion, the weights can be updated as follows:

$$w_T(t) = \sum_{\{p|p \text{ extracts } t\}} w_P(p) \quad (1)$$

$$w_P(p) = \sum_{\{t|t \text{ is extracted by } p\}} w_T(t) \quad (2)$$

The weights can initially be all set to 1, then propagated from patterns to tuples and vice versa. After each iteration, the weights are normalized to keep them within a bound and also to check the convergence. It is easy to show that this iteration converges (see [17] for details).

5.4 Analyses of the Rankings

A property that is probably desirable for a scoring function is *monotonicity*.

DEFINITION 1. *A scoring function F is monotone if the following property holds for the result set: for every pair of tuples t_1 and t_2, if every pattern p that extracts t_1 also extracts t_2 and $w(p \rightarrow t_1) \geq w(p \rightarrow t_2)$, then $w_T(t_1) \geq w_T(t2)$.*

THEOREM 1. *The scoring functions NPages, NPatterns and PT-hits are all monotone.*

Another property is how the scores change if the input data (text collection, in our case) or the set of patterns slightly change. We would prefer small changes in the input or the set of patterns to have small effects on the scores of the tuples. Let $\mathcal{G}_{m,n}$ be a collection of bipartite graphs over a set of m patterns and n tuples, and $F(G)$ denotes the weight vector of tuples after a scoring function F is applied to a bipartite graph G.

DEFINITION 2. *The normalized Kendall tau distance between two n-dimensional real vectors w_1 and w_2 is defined as*

$$d_{kt}(w_1, w_2) = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} I_{w_1,w_2}(i,j)$$

where

$$I_{w_1,w_2}(i,j) = \begin{cases} 1 & if\ w_1(i) < w_1(j)\ AND\ w_2(i) > w_2(j) \\ 0 & otherwise. \end{cases}$$

The Kendall tau distance measures the number of pairwise disagreements between two (ranked) lists. The Kendall tau distance changes if there is a change in ordering, but any change in the actual scores would not affect the Kendall tau distance if the ordering remains the same.

DEFINITION 3. *The normalized Manhattan distance between two n-dimensional real vectors w_1 and w_2 is defined as*

$$d_1(w_1, w_2) = \frac{1}{n} \sum_{i=1}^{n} |w_1(i) - w_2(i)|.$$

DEFINITION 4. *A scoring function F is stable on $\mathcal{G}_{m,n}$ under distance function d if for every fixed k, we have*

$$\lim_{m \text{ or } n \rightarrow \infty} \max_{G \in \mathcal{G}_{m,n}, e_1, \ldots e_k \in E(G)} d(F(G), F(G \backslash \{e_1, \ldots, e_k\})) = 0$$

where $E(G)$ denotes the edges of G and $G\backslash\{e_1, \ldots, e_k\}$ is the graph that is obtained after removing edges $e_1, \ldots, e_k$ of G.

DEFINITION 5. *A scoring function F is local if for every graph $G \in \mathcal{G}_{m,n}$ and every edge $e \in E(G)$, if $w_1 = F(G)$ and $w_2 = F(G \backslash e)$, then $I_{w_1,w_2}(i,j) = 0$ for all tuples i and j which are not adjacent to e.*

Based on the results reported for HITS [5], PT-hits is neither stable nor local. We can derive the following results for NPatterns and NPages.

THEOREM 2. *The scoring function NPatterns is stable under both d_{kt} and d_1.*

PROOF. Let $G' = G\backslash\{e_1, \ldots, e_k\}$ and $\{1, \ldots, l\}$ be the set of tuples in G' which are affected by the change. Clearly $l \leq k$.
(Case for d_{kt}) Consider the ordering of the tuples under NPatterns(G). With every edge e_i removed from the graph, the ordering can be disturbed by n substitutions at most. This extreme case arises when all tuples have the same scores and removing e_i moves a tuple to the bottom of the list. In total, there are at most k tuples affected. Therefore the maximum for $d_kt(w_1, w_2) \leq 2(kn)/n(n-1) = 2k/(n-1)$, and $\lim_{n\rightarrow\infty} d_{kt}(w_1, w_2) = 0$.
(Case for d_1) The sum of the changes in scores for affected nodes cannot exceed k whereas this sum for non-affected nodes is 0. Thus $\lim_{n\rightarrow\infty} d_1(w_1, w_2) = \lim_{n\rightarrow\infty} k/n = 0$. □

THEOREM 3. *NPages is stable under d_{kt}. NPages is also stable under d_1 if $w(p \rightarrow t)$ bound to a fixed constant.*
PROOF. *(Case for d_{kt})* The stability proof is similar to the one given for NPatterns. With every edge e_i removed from the graph, the ordering can be disturbed by n substitutions at most. With at most kn substitutions in total, $\lim_{n\rightarrow\infty} d_{kt}(w_1, w_2) = 0$.
(Case for d_{kt}) Suppose $w(p \rightarrow t)$ is bound to a constant c. The sum of the changes in scores for affected nodes cannot exceed ck, and the rest follows. □

THEOREM 4. *The scoring functions NPatterns and NPages are both local.*

PROOF. The locality follows from the fact that removing an edge e only affects the score of a tuple that is linked by e. □

6. EXPERIMENTS

To experiment with our querying interface and to evaluate our algorithms, we built a system called *DeWild* which relies on the Web as its source of data[3]. 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 queries Google and Yahoo (when Google does not

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

respond, due to either its workload or the number of our queries exceeding the engine limit) via their APIs. DeWild lists the set of rewritings used for each query. In our experiments, 200 snippets are downloaded for each extraction pattern (our online system lists the set of extraction patterns that are tried for each query). If there are fewer than 200 snippets found for a pattern, then all available snippets are downloaded[4]. The snippets returned by a search engine typically consist of the search query and its surrounding text. Since the target data appear 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. If it is not explicitly stated otherwise, DeWild uses PT-hits as its native ranking. For our experimental comparisons, we also implement the other heuristics discussed in Section 5.

A publicly available POS tagger called NLProcessor[5] 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 [18] from Wall Street Journal corpus, but it can equally use other collections as well.

Next we report our experiments with DeWild.

6.1 Recall and Precision

In general, it is difficult to measure recall on the Web since we often do not know the full answer set. The answer set may not be all on the Web, or it can be scattered in many pages of which some may not be crawled or indexed by a search engine. To measure both recall and precision under these constraints, we decided to extract instances of some known classes. To make a comparison with an alternative system, the class names were chosen from those reported for KnowItAll [12].

Pattern	Weight
US states, including %	0.739794
US states such as %	0.526682
% and other US states	0.320306
such US states as %	0.227648
US states, especially %	0.113638
% or other US states	0.074993
% is a US state	0.046522
US states %	0.013729
%, a US state	0

Table 2: Patterns that are used to extract country and the US state names, with the weights computed by PT-hits in each case

In one experiment, we used the names of 50 US states as the ground truth and tried to retrieve and rank the same data using PT-hits and our other heuristics. The query was formulated as "US states such as %"; Table 2 shows the extraction patterns which matched our intial query, after instantiating "US states" in our generic patterns, as well as their weights computed by PT-hits. Clearly, any of the patterns in the table could have been used as a query and the result returned by DeWild would have been the same. In our evaluation, a retrieved state name was treated "correct" if it was either a full state name or an abbreviation. Figure 2(a) and (b) shows precision and recall for PT-hits, NPages, NPatterns, MI and KnowItAll. Like PT-hits, KnowItAll has a precision of 1 when recall is less than 0.35, meaning that the top 35% of the answer is correct. For higher recalls, the precision for KnowItAll drops sharply whereas PT-hits has a precision of 1 for all recall values less than 0.75. Even for higher recall values, the precision for PT-hits does not drop sharply. NPatterns also performs reasonably well but the precision of NPages for recalls over 0.25 drops under 0.8. To do a ranking using mutual information (MI), we could use either the query or one of its rewritings. Since it is not clear which rewriting performs the best, we ran the algorithm three times with the discriminative phrases "US state of X", "US states such as X", and "X is a US state". These variations of MI are respectively referred to as MI-1, MI-2 and MI-3. Figure 2-b compares PT-hits to MI and to the online system KnowItAll[6]. We have to point out that there are differences between DeWild and KnowItAll. DeWild uses search engines as its data source; even though the result of a search engine is ranked, we are not making use of this ranking. KnowItAll was originally using Google but it had switched to its own local collection when we tested it. The lack of sufficient details in the KnowItAll paper prevented us from directly implementing it. Since we are comparing precision at each recall, the size of the collection should not have much impact on the comparison. The precision of MI is very poor at low recall rates, which means that the highly ranked instances by MI are mostly errors. Both MI-2 and MI-3 perform poorly in terms of precision for all recall values. MI-1 performs good for higher recall values but not so good for smaller recalls, meaning that many incorrect instances show up at the top of the list.

In another experiment, we used a list of 192 country names, compiled by the US State Department[7], as the ground truth. The query "countries such as %" was used for the extraction in DeWild. As shown in Figure 2-c, not only PT-hits but also NPatterns and NPages suprisingly perform very well. Further investigation showed that this was due to the high quality of search engine results for this particular query. On the other hand, the MI heuristics didn't perform that well (as shown in Figure 2-d) because the search engine results to our count queries were less reliable and this pushed some invalid tuples up in the ranked list. PT-hits is less-affected by the precision of the input (i.e. search engine results) and often pulls out valid tuples that may be scattered among a large number of incorrect ones.

6.2 Number of Rewritings

Adding each query rewriting introduces some cost at the query processing time, and a question is if this additional cost is justified. To evaluate the effect of the number of rewritings on the precision and recall, we used PT-hits to compile a list of "US states" but varied the number of rewritings that were used. We chose the best sets of 2, 3, and 5 patterns (i.e. those with the highest weights) from Table 2-b and ran PT-hits each time with only one of these sets. The precision-recall curve in each case is shown in Figure 3. At the same recall rate, the precision improves significantly when the number of patterns increases from 2 to 3. The pre-

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

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

[6]`www.cs.washington.edu/research/knowitall`

[7]`www.state.gov/www/regions/independent_states.html`

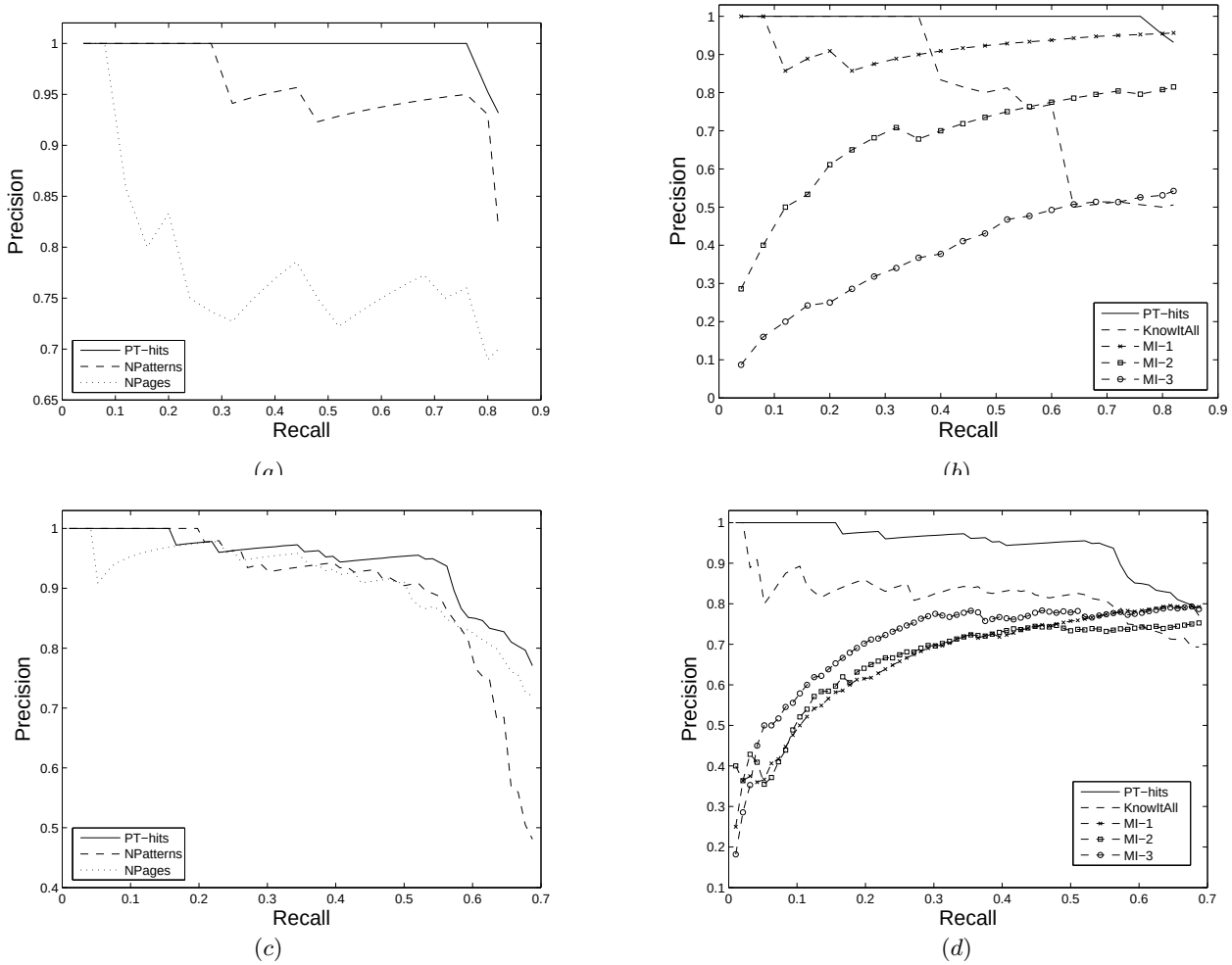

Figure 2: Precision and recall with the extraction target set to the US states in (a) and (b) and country names in (c) and (d)

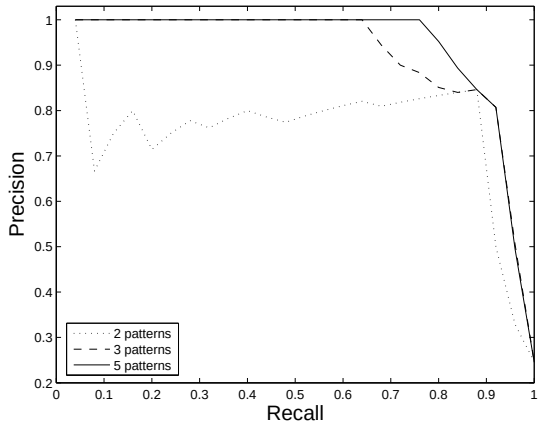

Figure 3: PT-hits precision and recall varying the number of patterns for target US states

cision at higher recalls is further improved when the number of patterns is increased from 3 to 5. We did the same experiment with the country names and the results were the same, hence they are not reported.

6.3 Handling Question-Answering Tasks

To do a further evaluation, we tried to use DeWild for question-answering where one of the goals is to return the actual answer to a question, rather than an entire paragraph or a sentence. If a question is formulated as a DeWild query, we can use our approach to locate the answer from the Web. For our evaluation, we took the first five QA targets from the TREC 2004 dataset [24]; since a QA target consisted of multiple questions, we ended up with a total of 22 questions in the experiment. For each question, we report the number of correct answers given by TREC, the number of answers from DeWild, the number of overlaps between the two, and the number of rewritings used in DeWild. The result of the evaluation is presented in Table 3.

For 55% of the questions (i.e. 12 questions), all answers returned by TREC were also returned by DeWild. For 18% of the questions, we couldn't find a pattern between the question and possible answers; hence we couldn't form a query. These are marked with "na" in the table. We found out that the TREC answers for question 1.3 were not the ground truth on the Web; therefore there was small overlap between TREC and DeWild. For questions 2.3 and 4.4, there were more than one formulation of the query but these different formulations were not in our rewriting set; this explains the small overlap between TREC and DeWild. For questions 4.5, the TREC answer was not supported on the Web and

we could only find it in NIST's TREC pages. For question 5.4, which asked for the CEO of AARP, TREC had "Horace Deets" or "Tess Canja" as the correct answer; this was based on the information in year 2004. At the time of running our experiments, the correct answer was "Bill Novelli" or "Marie Smith". DeWild extracted the more up-to-date correct answer.

question id	ans. in TREC	ans. in DeWild	overlaps	rewritings
1.1	1	2	1	3
1.2	1	na	na	na
1.3	14	5	2	1
1.4	1	na	na	na
1.5	1	4	1	1
2.1	1	2	1	1
2.2	1	4	1	1
2.3	5	7	3	1
2.4	1	7	1	11
3.1	1	3	1	1
3.2	1	na	na	na
3.3	1	na	na	na
4.1	1	1	1	1
4.2	1	1	1	1
4.3	1	15	1	1
4.4	4	7	3	1
4.5	1	2	0	1
5.1	1	6	1	1
5.2	1	2	1	1
5.3	1	20	1	1
5.4	1	12	0	11
5.5	6	17	2	13

Table 3: DeWild's handling of the first 22 questions from TREC 2004

DeWild sometimes returned additional instances of which some were correct and others were incorrect but appeared with the query and gave additional information. For instance, consider the question "Who discovered prions?" from TREC which has only one correct answer. We transformed the question to "prions are discovered by %" and passed it as a query to DeWild. The highest ranked instance, "Stanley Prusiner", was the correct answer to the question, and it also received a substantially larger weight than the second best instance. Our system returns other acceptable answers, including the 8th-ranked "Dr. Stanley Prusiner", the 9th-ranked "researcher Stanley Prusiner", and the 12-th ranked "Nobel Prize winner Stanley Prusiner". These other answers show that Stanley Prusiner was a doctor, a researcher, a Nobel prize winner and he was from the University of San Francisco.

6.4 Ad Hoc Data Extractions

As our last experiment, we tried to compile useful resource lists which we could not find in a list format anywhere on the Web. In one case, we tried to find the names of summer movies. Although some online resources maintain a quite complete list of movies, they don't classify movies as summer movies or otherwise. The pattern "% is a summer *blockbuster*" is used as the query for the task. The term *blockbuster*, which is enclosed by * wild cards in the query, is augmented by two extra related terms: *movie* and *film*. We manually evaluated the extracted results using the Internet Movie Database (IMDB) and concluded that all the results in the top 10 were indeed correct movie names, and their release dates were in the summer.

In one more experiment, we used the query "% is a Canadian writer" to compile a list of Canadian writers. This time, we put together a set of rewritings that were specific to the query. The query returned over 1300 names. We could verify that 91 of the first 100 rows were real Canadian writers. Of the first 200 rows that we verified, 156 were real Canadian writers. We also compared the first 200 tuples to two of the most comprehensive online lists of Canadian writers that we could find. DeWild retrieved 86 real author names which could not be found in one list[8] and 70 names which were not in the other list[9]. After combining the two lists, DeWild still reported 58 names which we couldn't find in the combined list. This experiment shows that our queries can be used to compile a reasonably good list of resources which can be further edited for correctness.

7. 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 terms of a question and a possible answer sentence (e.g. [16, 11, 20]). Despite some overlap, there are fundamental differences between our work and the work on question answering. The size of the target set for question answering systems is typically one or only a few, whereas our goal is to use DeWild queries for large-scale data extractions. 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 [6] 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. Given a small set of examples, the semantics of the query sometimes is not also clearly defined.

KnowItAll [12] 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. This is useful when a concise class description is not available. Last but not least, our approach of porting link-based ranking algorithms for assessing extraction re-

[8] `www.track0.com/ogwc/authors`
[9] `www.umanitoba.ca/canlit/authorlist`

sults from text is novel and performs better than the one used in KnowItAll.

Related to our query rewritings is the work on query expansion [4] and query transformation for question answering [2]. Query expansion has shown to be difficult for phrase queries. Also query expansion and transformation techniques are not directly applicable to queries with wild cards. Our queries can benefit from inverted indexes on terms, phrases, and N-grams and there are already some works in these areas (e.g. [7, 9]). Other related work includes the work on Web query languages and wrappers (see [13] for a survey of this area before 1998). These works can be used to extract data from a specific site or a set of pages with similar structures but are not generally applicable to free text. 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).

8. 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.

9. REFERENCES

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

[2] E. Agichtein, S. Lawrence, and L. Gravano. Learning search engine specific query transformations for question answering. In *Proc. of the WWW Conf.*, pages 169–178, 2001.

[3] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. In *Proc. of the WWW Conf.*, pages 245–256, 2005.

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

[5] A. Borodin, J. S. Rosenthal, G. O. Roberts, , and P. Tsaparas. Link analysis ranking: algorithms, theory and experimets. In *ACM Transactions on Internet Technologies*, volume 5 of *1*, pages 231–297, 2005.

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

[7] M. J. Cafarella and O. Etzioni. A search engine for natural language applications. In *Proc. of the WWW Conf.*, pages 442 – 452, 2005.

[8] S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[9] J. Cho and S. Rajagopalan. A fast regular expression indexing engine. In *Proc. of the ICDE Conf.*, pages 419–430, 2002.

[10] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. In *Proc. of the Computational Linguistics Conf.*, pages 76–83, 1989.

[11] 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.

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

[13] 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.

[14] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proc. of the Computational linguistics Conf.*, pages 539–545, 1992.

[15] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[16] 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.

[17] H. Li. Data extraction from text using wild card queries. MSc thesis, U. Alberta, 2006.

[18] 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).

[19] D. Lin and P. Pantel. DIRT - discovery of inference rules from text. In *Proceedings of the SIGKDD Conference*, pages 323–328, 2001.

[20] G. Ramakrishnan, S. Chakrabarti, D. Paranjpe, and P. Bhattacharyya. Is question answering an acquired skill? In *Proc. of the WWW Conf.*, pages 111–120, 2004.

[21] D. Ravichandran and E. H. Hovy. Learning surface text patterns for a question answering system. In *Proc. of the ACL Conf.*, pages 41–47, 2002.

[22] RTE. www.pascal-network.org/Challenges/RTE/.

[23] E. M. Voorhees. Overview of the TREC 2001 question answering track. In *Text REtrieval Conf.*, 2001.

[24] E. M. Voorhees. Overview of the TREC 2004 question answering track. In *Text REtrieval Conf.*, 2004.

[25] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006.