Querying Text Databases for Efficient Information Extraction

Eugene Agichtein Luis Gravano
Columbia University
E-mail: {eugene,gravano}@cs.columbia.edu

Abstract

A wealth of information is hidden within unstructured text. This information is often best exploited in structured or relational form, which is suited for sophisticated query processing, for integration with relational databases, and for data mining. Current information extraction techniques extract relations from a text database by examining every document in the database, or use filters to select promising documents for extraction. The exhaustive scanning approach is not practical or even feasible for large databases, and the current filtering techniques require human involvement to maintain and to adopt to new databases and domains. In this paper, we develop an automatic query-based technique to retrieve documents useful for the extraction of user-defined relations from large text databases, which can be adapted to new domains, databases, or target relations with minimal human effort. We report a thorough experimental evaluation over a large newspaper archive that shows that we significantly improve the efficiency of the extraction process by focusing only on promising documents.

1 Introduction

Text documents often hide valuable *structured data*. For example, a newspaper archive contains data that might be useful to analysts who want to track mergers and acquisitions, or to government agencies that are interested in monitoring and tracing back infectious disease outbreaks as reflected in the news. Information extraction systems produce a structured representation of the information that is "buried" in unstructured text documents. Improving the efficiency of information extraction systems over large text databases is the focus of this paper.

In general, state-of-the-art extraction systems [16] apply many rules over each available text segment to determine whether the segment can be used to fill a value of an attribute in a tuple. Therefore, processing each document is relatively expensive, and typically involves several steps such as named-entity tagging (e.g., identifying person names or dates), syntactic parsing, and finally rule matching. This approach is not feasible for large databases, or for the web, when it is not realistic to tag and parse, or even simply scan, every available document. For example, one highly optimized state-of-the-art information extraction system requires over 9 seconds to process an average-sized newspaper article on a high-end workstation. As a result, over 15 days of processing time are required for a 135K document archive. With document database size commonly exceeding millions of documents, processing time is becoming a bottleneck when exploiting information extraction technology for any time-critical applications or for leveraging extracted information with relational databases.

Previous approaches for addressing the high computational cost of information extraction resorted to document *filtering* to select the documents that deserve further processing by the information extraction system. This filtering still requires scanning the complete database to consider every document. Alternative approaches used keywords or phrases as filters (which could be converted to queries) that were manually crafted and tuned by the information extraction system developers, as we will discuss.

In this paper we address the scalability of information extraction systems in a principled and general manner. We introduce *automatic* query-based techniques to identify the database documents that are promising for the extraction of a relation from text by an arbitrary information extraction system, while assuming only a minimal *search* interface to the text database. Our techniques make it possible for an information extraction system to operate over large text databases, or even the web, by first retrieving the set of documents worth analyzing, and then proceeding with the usual extraction process over this smaller document set.

Our approach automatically discovers the characteristics of documents that are useful for extraction of a target relation, starting with only a handful of user-provided examples of tuples of the relation to extract. Using these tuples, our system retrieves a sample of documents from the database. By running the information extraction system over the documents, we identify which documents are useful for the extraction task at hand. Then, we apply machine learning and information retrieval techniques to learn queries that will tend to match additional useful documents. Finally, the documents that are retrieved with these learned queries are processed by the information extraction system to extract the final relation.

Contributions: The key contribution of this paper is our unsupervised query-based method for retrieving useful documents for information extraction from large text databases. Our method requires the document database to support only a minimal query interface, and is independent of the choice of information extraction system. Furthermore, our method could be used to query a standard web search engine, hence providing infrastructure for efficient information extraction from the web at large.

Related Work: Information extraction has long been the focus of active research. The main emphasis of this research, notably in the context of the Message Understanding Conferences (MUCs) [1], has been on the quality of the extracted relation. In contrast, our work assumes a given information extraction system, and focuses on retrieving a relatively small set of documents that would allow the extraction of a close approximation of the target relation efficiently.

A related problem is *question answering*. Question answering systems attempt to find answers to natural language questions in collections of text documents. These systems typically process each question individually [28]. In our scenario, we are interested in extracting a complete *relation* (i.e., *all* tuples in the relation). The extracted relation can be viewed as a set of prepared "answers" for a particular class of questions (e.g., "What is the location of the headquarters of X?"). While question answering techniques may be useful for retrieving specific tuples in the target relation (e.g., [4, 20]), the problem of retrieving documents that collectively contain the complete relation has not been addressed in the question-answering literature, to the best of our knowledge.

Our work is also related to recent research on focused web crawling (e.g., [6]), which addresses the problem of fetching web pages relevant to a given topic. Our proposed technique is tuned for information extraction, and operates over any searchable text database, whether its contents are "crawlable" or not. Recent work [23] addresses the problem of crawling the "*hidden web*," the portion of the web hidden behind search forms. Our goal is different: we attempt to extract the most complete *relation* from the text database while retrieving *as few documents as possible*.

One subtask of the MUC evaluations, *text filtering*, is relevant to our work. In that task, each participating system would judge which of the documents are relevant for the extraction scenario [1]. Documents can be filtered at various stages of the extraction process [21]. Some systems [9] classified input documents based on single words and word n-grams prior to doing any further processing, while others used manually constructed keywords and phrase filters to discard documents. The classification-based approach required manually labeled documents for training the classifiers. Other systems developed filters from the extraction patterns devised to extract the target relation. In Section 4, we evaluate a related strategy that uses queries derived from extraction patterns. In contrast to these techniques, our goal is to *automatically* generate standard search-engine *queries* (and not more general *filters*) that would retrieve only the relevant documents for an extraction task.

The evaluation presented in [12] uses ideas related to our work. The authors consider 9 manually generated keywords originally used for compiling the 100 test documents used in the MUC-6 evaluation. These keywords were submitted to a web search engine and the resulting documents were processed by the extraction system and evaluated for relevance for the extraction scenario. In a different setting of compiling conference "Calls for Papers" extracted from web documents, [19] uses a combination of focused crawling and manually generated queries to retrieve promising documents. In Section 4, we evaluate a related manual query strategy to compare against our automatic query generation method.

The problem of retrieving documents that are "relevant" to a user's information need has been the core focus of the information retrieval (IR) field [27]. Although our problem is different in nature, we exploit state-of-the-art term weighting and query expansion results [24] from IR in the design of one variant of our system (Section 2.4.2). Alternatively, the characterization of the useful documents could be viewed as a traditional classification problem. We explore a number of machine learning techniques [8, 18] in the design of other variants of our system (Section 2.4.2).

Several techniques use supervised learning to devise queries that match documents about a specific category of interest [15]. [7] constructed topic-specific directories on the web by training a classifier with a labeled set of documents and then deriving queries to retrieve additional documents. Flake et al. [11] extracted category-specific query modifications from a non-linear SVM classifier. Recently, Ghani et al. [14] presented a technique that is similar in spirit to our current work, but for a different task: identifying web pages in a "minority" language (e.g., Slovenian) by querying a search engine. Their technique starts with a set of web pages that are fed to a language identifier and labeled as positive or negative examples. This set of pages is then used to derive Boolean queries that will tend to identify more pages in the language of choice, and the process iterates. In our work, on efficient information extraction, we consider query generation techniques based on a term weighting scheme that is related to some of the techniques in [14], as well as other query generation strategies that exploit machine learning results.

The rest of the paper is organized as follows. Section 2 presents our new document retrieval method in detail. Then,

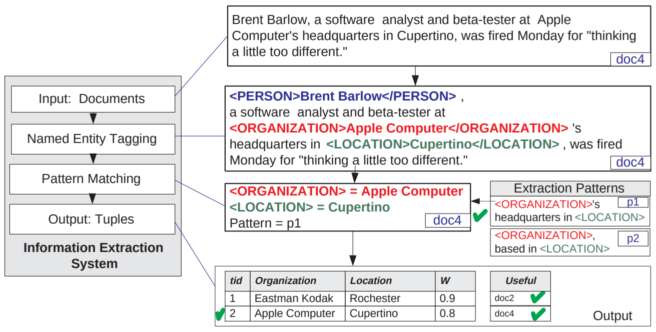

Brent Barlow, a software analyst and beta-tester at Apple Computer's headquarters in Cupertino, was fired Monday for "thinking a little too different."
doc4
Input: Documents
Named Entity Tagging
Pattern Matching
Output: Tuples
Information Extraction System
<PERSON>Brent Barlow</PERSON> , a software analyst and beta-tester at <ORGANIZATION>Apple Computer</ORGANIZATION> 's headquarters in <LOCATION>Cupertino</LOCATION> , was fired Monday for "thinking a little too different."
doc4
<ORGANIZATION> = Apple Computer
<LOCATION> = Cupertino
Pattern = p1
doc4
Extraction Patterns
<ORGANIZATION>'s headquarters in <LOCATION>
p1
<ORGANIZATION>, based in <LOCATION>
p2
tid
Organization
Location
W
1
Eastman Kodak
Rochester
0.9
2
Apple Computer
Cupertino
0.8
Useful
doc2
doc4
Output

Figure 1. Extracting tuples for the *Headquarters* relation.

Section 3 summarizes the general experimental setting, including the evaluation methodology, metrics, and databases we used for tuning and evaluating our strategy. Section 4 reports the results of an experimental evaluation of our technique (and several other strategies) on a large text database. Finally, Section 5 concludes the paper.

2 Retrieving Promising Text Documents

In this section, we present a new, *automatic* method for generating queries to match the documents that are useful for extraction of a target relation. Before describing our idea, we present a brief overview of how information extraction systems work (Section 2.1). Then, we define the problem on which we focus in this paper and the overall architecture of our *QXtract*[1] system (Section 2.2). *QXtract* starts by retrieving a small sample of documents from a text database and determining those from which the extraction system is able to extract tuples for the target relation (Section 2.3). This sample is used to provide examples of useful and useless documents to our methods of generating queries to retrieve the remaining promising documents in the database (Section 2.4).

2.1 Overview of Information Extraction

Information extraction usually refers to identification of instances of particular events and relationships in unstructured natural language text documents. The extracted *structured* records can be used to populate a relational table for answering queries and running data mining tasks. Thus, information extraction is a crucial step for fully exploiting natural language documents.

As an example of information extraction, consider extracting a *Headquarters(Organization, Location)* relation, which contains a tuple $< o, l >$ if organization o has headquarters in location l. Figure 1 shows the basic stages in the extraction of a tuple from a document fragment. We omit the more sophisticated post-processing and analysis performed by many state-of-the-art information extraction systems, as this is beyond the scope of this discussion. (Refer to [16] for an in-depth discussion.)

[1]*QXtract* stands for **Q**uerying for e**Xtract**ion.

As one of the first stages of extraction, the input documents are typically passed through a *named-entity tagger*, which is able to recognize entities (e.g., organizations, locations, and persons). Named-entity tagging is a well studied problem, with tools publicly available for the more common entity types [10]. These entities are potential values of attributes in the target relation. To find *related* entities, the tagged documents are processed by applying extraction *patterns* in the *pattern matching* step. These patterns may be manually constructed [30], automatically learned [31, 3], or created by using a combination of the two methods. Each pattern is applied to each text fragment, instantiating appropriate slots in the pattern with entities from the document. These entities are combined into candidate tuples, and after filtering and post-processing are returned as extracted tuples.

In the example in Figure 1, a sample document *doc4* is first passed through a named-entity tagger that recognizes *person*, *organization*, and *location* entities. The text fragment containing entities of interest, namely *organization* and *location*, is matched with one of the extraction patterns, "<ORGANIZATION>'s headquarters in <LOCATION>", instantiating the generic entity types with entities *Apple Computer* and *Cupertino*, respectively. Finally, the tuple $<$*Apple Computer, Cupertino*$>$ is generated. In Figure 1, a check-mark next to a document indicates that a tuple was successfully extracted from the document. We consider such documents *useful* for the extraction task. Also note that additional information may be available from the extraction system. For example, the information extraction system may assign a weight or confidence (W) to each extracted tuple.

The extraction process outlined above is too expensive to perform on every document in a large database. By focusing only on potentially useful documents, we can dramatically improve the efficiency and scalability of the information extraction process. Next, we introduce *QXtract*, our *automatic* technique for retrieving such "promising" documents.

2.2 Problem Statement and Notation

Consider the problem of extracting a relation from a large database of text documents. Often, only a small fraction of the documents contain information that is relevant to the extraction task. Hence it is not necessary for extraction completeness – or desirable from an efficiency viewpoint – to run the information extraction system over every database document. Furthermore, if our database is the set of all web pages indexed by a search engine such as Google, then it is virtually impossible to scan every page to extract tuples. For these reasons, our approach zooms in on the promising documents, while ignoring the rest. We now state the problem that we are addressing, and introduce the notation that we will use in the rest of the paper. For our purpose, a database can be either local (e.g., a company's archive of legal documents or customer e-mails) or remote (e.g., the web-accessible and searchable archive of a newspaper).

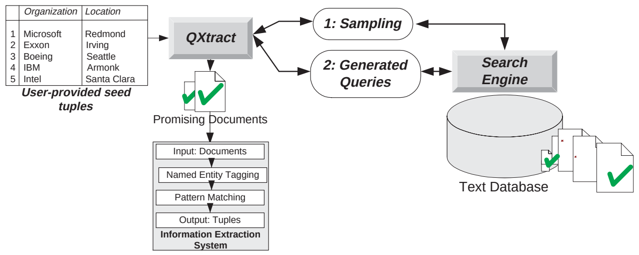


Figure 2. The architecture of an efficient information extraction system that identifies promising documents via querying.

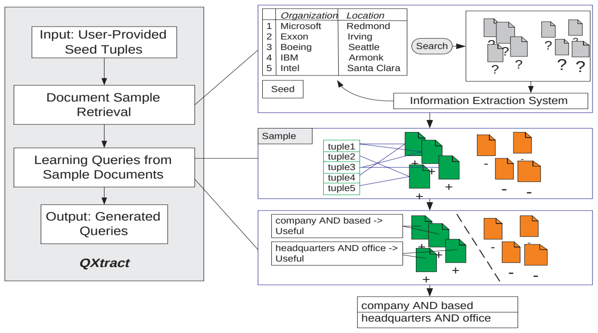

Input: User-Provided Seed Tuples
Document Sample Retrieval
Learning Queries from Sample Documents
Output: Generated Queries
QXtract
Organization
Location
1 Microsoft
Redmond
2 Exxon
Irving
3 Boeing
Seattle
4 IBM
Armonk
5 Intel
Santa Clara
Search
Seed
Information Extraction System
Sample
tuple1
tuple2
tuple3
tuple4
tuple5
company AND based -> Useful
headquarters AND office -> Useful
company AND based
headquarters AND office

Figure 3. *QXtract*: Promising document retrieval.

Problem Statement: We are given an information extraction system E and a text document database D_{all}, together with a few example tuples of the relation to be extracted. Let R_{all} denote the instance of the relation that would be extracted from the entire database D_{all}. Our goal is to construct a close approximation of R_{all}, R, by retrieving a small subset D of the document database D_{all}, and then having the extraction system operate on D rather than on the much larger original database D_{all}. We assume that the user specifies the maximum fraction of D_{all} that can be retrieved to extract R [2]. This parameter, *MaxFractionRetrieved*, would vary depending on the needs of the user, and on the size of the original database.

Note that R_{all} may not contain *all* of the correct tuples that could be extracted from the D_{all} database by a perfect extraction system. Rather, we are limited by the best relation that system E can extract, and we try to approximate that relation in an efficient manner. We also assume that E can only extract a tuple t if all of t's attributes occur within the same document. (In other words, we assume that the information extraction system does not "glue" together pieces of a tuple from multiple documents.)

Efficient Information Extraction Architecture: The overall architecture of the efficient information extraction system that we envision is shown in Figure 2. We interact with the target information extraction system through a unified information extraction system interface, which we describe next. The text database is accessible through a search engine interface. As we will discuss, *QXtract*, the promising document retrieval component, interacts with the extraction system and the database to retrieve promising documents.

Information Extraction System Interface: To handle a variety of arbitrary information extraction systems, we treat them as "black boxes" and interact with them through simple *extraction system wrappers*. These wrappers can be easily built to support the following unified interface:

- **Input:** A set of documents D for the extraction system to process, as shown in Figure 1.

[2]If the database size is unknown, then an absolute number of documents can be specified instead to control the efficiency of the extraction process.

- **Output:** The set of tuples extracted from D, *Tuples*, and for each tuple $t_i \in Tuples$, the set of identifiers U_i of the "useful" documents from which t_i was extracted. The wrapper returns the identity of all the useful documents, defined as *Useful* $= U_1 \cup U_2 \cup ... \cup U_{|Tuples|}$. In the example in Figure 1, $Useful = \{doc2, doc4\}$.
- **Optional Output:** *Patterns:* An extraction system may export the set of all the extraction patterns that it has available for extracting the target relation (e.g., the extraction patterns in Figure 1).

Designing for a minimal, uniform interface to the extraction system allows us to plug in any information extraction system to take advantage of our querying techniques, without any changes to the *QXtract* system.

Text Database Search Interface: We assume that the search interface of the database supports simple Boolean queries such as *"data AND mining AND text,"* as well as phrase queries. This query model provides sufficient expressiveness, and is widely supported: all of the major available text indexing tools (e.g., Glimpse [22]) and web search engines support such queries with minor variations in syntax.

The *QXtract* System: In the rest of this section, we will describe *QXtract*, whose overall architecture is shown in Figure 3. Starting with a set of user-provided seed tuples, we first use the sampling procedure described in Section 2.3 to retrieve a small sample of documents, likely to be useful to the extraction system for extracting the target relation, as well as other randomly chosen documents, likely to be useless to the extraction system. The information extraction system is run over this sample set, producing as output a set of extracted tuples and the identifiers of useful documents. The documents in the sample are thus labeled automatically as either positive or negative examples, where the positive examples represent the documents in the sample from which the information extraction system was able to produce tuples. These examples allow us to derive queries targeted to match –and retrieve– documents similar to the positive examples (Section 2.4). These queries are used to retrieve a set of promising documents from the database (Section 2.5), to be returned as *QXtract*'s output and finally processed by the information extraction system.

2.3 Retrieving Documents for Query Training

At the initial stage of the overall document retrieval process, we have no information about the documents that might be useful for extraction. The only information we require about the target relation is a set of user-provided example tuples, including a specification of the relation attributes to be used for document retrieval. Our goal is to retrieve a document sample of size specified by the *MaxSampleSize* parameter with a good mix of useful and useless documents for the subsequent query training stage. To accomplish this, we use the *DocumentSample* algorithm[3] shown in Figure 4. After initialization, each round of sampling consists of two stages:
(1) We retrieve documents for the sample by querying the search engine with the attribute values of the current seed tuples, which initially are provided by the user as the input parameter *Seed*.
(2) We run the information extraction system E over the documents in the current sample, extracting a new set of tuples. A subset of these tuples is selected as the new *Seed* tuples to start a new sampling round.

We initialize the document sample with randomly picked documents (line 1 in Figure 4), the majority of which are likely not to be useful for extraction and will be used as "negative" examples for training. The rest of the *Sample* documents will be retrieved using attributes of the *Seed* tuples, which results in documents that are likely to be useful for extraction and will be used as "positive" examples for training.

The initial *Seed* tuples are provided by the user, and are augmented by the tuples extracted by the extraction system E from *Sample* (lines 2 and 3). To retrieve additional sample documents, we build queries with the attribute values of each tuple t in the current set of *Seed* tuples. Each tuple t is used to construct a query $q = t.a_1$ *AND* $t.a_2$ *AND* $\ldots$ $t.a_n$, where $a_1, \ldots, a_n$ are the searchable attributes in the relation (line 7). Query q will retrieve documents where all attributes of t appear within the same document. In principle, these are documents from which t could have been extracted by the information extraction system. We retrieve the first *MaxSeedResults* matches returned by the database for each query (line 8). The query results are added to the set of *LikelyUseful* documents, retrieved during the current sampling round.

Clearly, not all documents in the *LikelyUseful* set will actually be useful for extraction. To determine which documents are indeed useful, we run the information extraction system E over *LikelyUseful* (line 9), returning the extracted tuples *T* and identifiers of useful documents *U* from which the tuples were extracted. In line 10, we choose the most

```
Procedure DocumentSample(Seed, MaxSampleSize)
//Seed is a set of example tuples. MaxSampleSize is the maximum
//number of documents to retrieve as a training sample.
     //First, retrieve a random sample of MaxSampleSize / 2 documents,
     //that will likely be "negative" examples for training.
1    Sample = RetrieveRandom(MaxSampleSize / 2)
     //Identify the minority of useful documents in the random sample.
     //Augment the Seed set with the extracted tuples T.
2    (T, Useful) = E.Extract(Sample)
3    Seed = Seed $\cup$ T

     //Now, retrieve MaxSampleSize / 2 documents with tuple attributes,
     //likely to be "useful", to provide positive examples for training.
4    while |Sample| $<$ MaxSampleSize
5         LikelyUseful = $\emptyset$
6         for each t in Seed:
7              $q = t.a_1$ AND $t.a_2$ AND ... $t.a_n$
8              LikelyUseful =
                   LikelyUseful $\cup$ RetrieveSeedDocuments(q, MaxSeedResults)
              //Skip to line 9 if MaxSampleSize exceeded.
9         (T, U) = E.Extract(LikelyUseful)
          //Set Seed tuples for next iteration.
10        Seed = PickBestTuples(T, U)
11        Useful = Useful $\cup$ U
12        Sample = Sample $\cup$ LikelyUseful

     //Sample now consists of MaxSampleSize documents.
13   Useless = Sample - Useful
14   Return(Useful, Useless)
```

Figure 4. Retrieving sample documents for subsequent query training.

"robust" tuples in *T* into *Seed* for the next round of sampling: this choice can be based on the number of documents in U from which the tuples were extracted, which favors selecting "popular" tuples that are likely to appear in many documents in the database[4]. The total size of the retrieved training set, *MaxSampleSize*, is a parameter that we tune during training.

Unfortunately, we cannot simply continue the sampling process to retrieve all of the useful documents in the database. As we will show in Section 4, if only a small fraction of the documents contain tuples for the target relation, or very few tuples tend to appear together in the same document, *DocumentSample* would not be able to discover a significant fraction of tuples that could otherwise be extracted.

Our key observation is that useful documents share similarities in content. For example, useful documents for the *Headquarters* relation may contain combinations of terms or phrases so that they match queries such as "headquarters of," "company AND based", "area AND companies," etc. These combinations of terms are more likely to occur in useful documents than in useless documents for the *Headquarters* relation. Our goal now is to automatically generate queries to retrieve the documents similar to those that the extraction system marked as *Useful*. Hence, the *Useful* and *Useless* documents returned by *DocumentSample* serve as the train-

[3]Independently, Ghani and Jones [13] have recently introduced a similar strategy to construct a training corpus for a bootstrapping-based general entity tagger.

[4]Alternatively, we could use the extraction confidence associated with the tuples, if this information is exported by the information extraction system.

ing set to learn queries for promising document retrieval.

2.4 Learning Queries for Promising Document Retrieval

Given a set of useful and useless documents as the training set, our goal now is to generate queries that would retrieve many documents that the information extraction system E will find useful, and few that E will not be able to use. The process consists of two stages: **(1)** Convert positive and negative examples into an appropriate representation for training, and **(2)** use the training examples to generate an ordered list of queries expected to retrieve new useful documents. Later, in Section 2.5 we will see how to submit the queries to the database to retrieve promising documents.

2.4.1 Representation Features for Training Examples

For training, we remove any extracted *tuple attributes* from the documents. For the current experiments, we use *words* as features to represent the training examples, and will not rely on other more advanced query features such as proximity operators or word "stems." Of course, if such advanced features (or alternative query models) are available, we could apply our same general approach and tailor it to the query interface of choice. In the future, we plan to experiment further with alternative document features, such as word phrases or sets of words within a window.

2.4.2 Generation of Queries from Examples

We now turn to the generation of queries to retrieve useful documents from the database. The problem of retrieving documents similar to a given set of "relevant" examples has been studied extensively in both the information retrieval and the machine learning communities. In this section, we discuss how we adapt well-established solutions from both communities to our (non-standard) problem. We first consider query generation as an IR automatic query expansion problem, using a state-of-the-art term weighting scheme. We then introduce query generation techniques that exploit the output of two machine-learning text classifiers. Finally, we present a hybrid query generation technique that combines the learned queries from all of the above methods.

Okapi: As a first query generation strategy, we exploit a state-of-the-art term weighting scheme from IR, from the *Okapi* retrieval system [24]. While there are many promising alternatives to this weighting scheme in the IR literature (e.g., [29, 26]), we chose Okapi because it has been demonstrated to perform well, is naturally well suited to our task, and is relatively straightforward to implement. Incorporating alternative information retrieval techniques into our system is easy and does not require changes to our model.

To predict which terms are most likely to retrieve useful documents, we compute the *term selection weight* [24] of each term in the training set. The terms with the highest positive weight are most likely to appear in useful documents and not in the useless ones. First, each term t_i in the document is assigned the Robertson-Spark Jones term weight $w_i^{(1)}$ [25]:

$$w_i^{(1)} = \log \frac{(r+0.5)/(R-r+0.5)}{(n-r+0.5)/(N-n-R+r+0.5)}$$

where a document is relevant if it was marked *useful* by the extraction system, r is the number of relevant documents containing t_i, N is the number of documents in the document sample, R is the number of relevant documents, and n is the number of documents containing t_i. Intuitively, this weight is high for terms that tend to occur in many relevant documents and few non-relevant documents, and is smoothed and normalized to account for potential sparseness of the training data. Then, we compute the *query selection weight* w_i of each term t_i as described in [24] for automatic query expansion, $w_i = r \cdot w_i^{(1)}$, where r and $w_i^{(1)}$ are defined above. The terms are sorted in descending order by w_i, and finally we define one-word queries consisting of each top-ranked term individually.

Ripper: As a second query generation strategy, we exploit a highly-efficient rule-based text document classifier, Ripper [8]. Ripper learns concise rules such as "*based AND company* $\rightarrow$ *Useful*," which indicates that if a document contains both term *based* and term *company*, then it should be declared "useful." After Ripper generates classification rules, we sort the rules in descending order of their expected precision, calculated as the ratio of positive examples to the total examples that match the rule. (This information is part of the Ripper output.) The rules are then translated into conjunctive queries in the search engine syntax. For example, the rule above might be translated to query "*based AND company.*"

Support Vector Machines (SVMs): As a third query generation strategy, we exploit another family of classifiers, SVMs, which have been shown to perform well in text classification [18]. To filter out noise, we prune the set of words used in training by discarding those that occur in fewer than 1%, or in more than 99% of the training examples[5]. We use a freely-available efficient implementation of linear-kernel SVMs [18]. To generate rules from SVM feature weights, we compute the *minimal* sets of words that are collectively sufficient to imply a positive classification of a document [15]. The result of this process is a set of "rules" similar to the Ripper output.

QCombined: Okapi, Ripper, and SVM all use different learning models, and the queries that they generate often have little overlap across techniques. We can exploit this observation to improve the robustness of *QXtract* by *combining* the ranked query sets generated by each query generation strategy. Specifically, we merge the query ranks generated

[5]Based on our experiments with linear-kernel SVMs on the training database, we additionally restrict the document features to the words in the immediate context (i.e., within the same line in the original document formatting) of the extracted tuples.

by Okapi, Ripper, and SVM in a round-robin fashion: The first query in the merged rank is the highest-ranked query generated by Ripper, followed by the highest-ranked query generated by SVM, and so on[6].

2.5 Querying for Promising Documents

We described above how to generate queries that are used to retrieve the final set of promising documents from which the information extraction system of choice will extract tuples. The size of this document set has a direct impact on the quality of the extracted relation.

We assume that, for efficiency considerations, we have a predefined upper bound $MaxFractionRetrieved$ on the fraction of the database D_{all} that we are willing to retrieve. The higher this upper bound, the more complete the extracted relation is likely to be. We submit the queries (generated and ranked as in Section 2.4.2) to the document database, one at a time. For each query, the database returns the document identifiers (e.g., URLs) of the matching documents. We retrieve the previously unseen documents until the maximum number of results per query, $MaxSearchResults$, is reached[7]. We keep the running total of all documents retrieved to avoid exceeding $MaxFractionRetrieved$. After this bound is reached (or there are no more documents to retrieve using our queries), all retrieved documents are returned as the output of *QXtract*. These promising documents are then used as input to the information extraction system, which extracts the final approximation of the target relation.

3 Experimental Setting

We now report the metrics we use to evaluate the alternative query methods (Section 3.1). Then, we describe the information extraction systems (Section 3.2) and the two target relations that we use in our experiments (Section 3.3). Later, we specify the training and test databases (Section 3.4), and conclude by describing the various querying techniques that we compare (Section 3.5).

3.1 Evaluation Methodology and Metrics

As we discussed, our goal is to approximate the R_{all} relation with all tuples that could be extracted through an exhaustive scan of the database D_{all} (Section 2.2). In contrast to exhaustive scanning, *QXtract* retrieves a promising set of documents D, from which the information extraction system obtains a relation R to approximate R_{all}. We then evaluate the document retrieval method based on R and D. Our evaluation focuses on: (1) how closely R approximates R_{all}, and (2) how "useful" the documents in D are on average[8]:

Recall: The percentage of the R_{all} tuples that were captured in R is *Recall* $= \frac{|R \cap R_{all}|}{|R_{all}|} \cdot 100\%$. R_{all} is computed by running the information extraction system over *every* document in the D_{all} database.

Precision: The percentage of documents in D that were useful for extracting R is *Precision*$= \frac{|D \cap U|}{|D|} \cdot 100\%$, where U is the subset of D from which the extraction system managed to extract tuples.

Note that we are not using *Recall* and *Precision* in a strictly standard way. Our recall measure is based on the percentage of the *tuples* in R_{all} correctly extracted, while our precision measures the percentage of useful *documents* within the retrieved document set. Intuitively, the document retrieval method has two purposes: the first is to extract a close approximation of R_{all}, while the second is to do so *efficiently*, i.e., by retrieving few documents. The most direct measure of success in the first task is the percentage of R_{all} tuples that are actually extracted from the documents retrieved by *QXtract*. The success in the latter task is naturally measured at the document level, as we feed the information extraction system one document at a time, and gain one or more tuples for R if the document is *useful*. If the document is *useless* (i.e., no tuples are extracted), the resources required to retrieve and process the document are wasted. Therefore, a larger fraction of the *useful* documents retrieved translates to a higher efficiency of extraction, as quantified by our precision measure.

To complement the study of the *efficiency* of our techniques, in Section 4 we also report on the *actual time* required to extract an approximation of R_{all} from the documents retrieved by *QXtract*, as compared to the time required to extract R_{all} from the complete database.

3.2 Target Information Extraction Systems

The design of *QXtract* is general in that we can use any information extraction system as long as it supports (through a wrapper) the simple interface described in Section 2. For our experiments, we consider three extraction systems:

(1) *DIPRE* [5], which uses a simple bootstrapping algorithm starting with a handful of user-provided seed tuples.

(2) *Snowball* [3], which is an extension of *DIPRE* that includes automatic pattern and tuple evaluation to improve the quality of the extracted table.

(3) *Proteus* [17], which is a sophisticated, manually trained information extraction system from NYU[9].

[6]More sophisticated ways of combining the generated queries are possible (e.g., to eliminate redundancy across queries), and we plan to explore some of them in our future work.

[7]Many search engines have a predefined limit on the maximum number of results per query that they return.

[8]We do not consider the absolute accuracy or "quality" of the extracted tuples. Rather, we focus on how closely we approximate the best possible relation that can be produced by a given information extraction system, if it had examined every document in the database.

[9]While the *Proteus* system is not publicly distributed, we were allowed to use an instance that was tuned for extracting infectious disease outbreaks, with kind help and permission from Roman Yangarber, Ralph Grishman, and Silja Hattunen.

Headquarters		*DiseaseOutbreaks*	
Organization	*Location*	*DiseaseName*	*Location*
Microsoft	Redmond	Malaria	Ethiopia
Exxon	Irving	Typhus	Bergen-Belsen
Boeing	Seattle	Flu	The Midwest
IBM	Armonk	Mad Cow Disease	The U.K.
Intel	Santa Clara	Pneumonia	The U.S.

Figure 5. Initial seed tuples provided to *QXtract* for extracting the *Headquarters* and the *DiseaseOutbreaks* relations.

The three systems above are representative of the state of the art in information extraction, and range from a simple strategy with minimal manual training (*DIPRE*) to a highly sophisticated strategy with extensive manual training (*Proteus*).

3.3 Target Relations for Extraction

We evaluate the performance of *QXtract* on the extraction of two relations, with the initial seed tuples of Figure 5:
Headquarters*(Organization, Location)*, as defined in Section 2.1. The attributes *Organization* and *Location* are used for querying for sample documents. We use *DIPRE* and *Snowball* to extract this relation.

DiseaseOutbreaks*(DiseaseName, Location, Country, Date, ...)*. Each tuple <*n, l, c, d, ...* > (e.g., <"Mad Cow Disease", "The U.K.", "U.K.", "3/27/1996", ... >) corresponds to an outbreak of a disease *n* in a location *l* of a country *c*, on date *d*, and other attributes that we do not discuss here for brevity. The attributes *DiseaseName* and *Location* are used for querying for sample documents. We use *Proteus* to extract this relation.

3.4 Training and Test Databases

Our **training** database consists of 137,893 documents from the first nine months of the 1996 New York Times archive[10]. The **test** database consists of 135,438 documents from the 1995 New York Times archive. For our experiments, we indexed the training and test databases using the Glimpse search engine. Glimpse supports a Boolean retrieval model with no document ranking. Queries specify either exact phrases (which do not ignore punctuation) or single words[11].

3.5 Alternative Document Retrieval Methods

We experimentally compare a number of alternatives to retrieve promising documents:
QXtract: This is the technique described in Section 2, whose parameters (tuned using the **training** database and summarized in Figure 7) include *MaxSampleSize*, *MaxSeedResults*, and the *query generation strategy*.
Tuples: This technique uses tuples to retrieve promising documents. *Tuples* proceeds essentially as procedure *DocumentSample* in Section 2.3, with the difference that *Tuples* does not retrieve the random document sample in Step 1. The modified version of *DocumentSample* continues to extract tuples to use as queries until a fraction *MaxFractionRetrieved* of the database has been retrieved. We include this technique in the experimental evaluation to study the benefits of *QXtract*'s query generation stage, which is missing in *Tuples*.
Baseline: This simple baseline technique returns a randomly chosen fraction *MaxFractionRetrieved* of the database documents, and processes them using a previously trained extraction system. The document sample, if any, that was used to train the extraction system, is not counted towards the retrieved document quota for *Baseline*.
Manual: This technique is based on hand-crafted filters. We implement this strategy only for extraction of the *DiseaseOutbreaks* relation, using manually constructed queries based on the filters provided to us by the developers of the *Proteus* system. These are the current filters that are applied to documents before running *Proteus* on them[12]. The filters were converted to the closest phrase and Boolean queries. In Figure 13 we show a sample of the automatically learned queries generated by *QXtract* that were somewhat close to the *Manual* queries. We do not apply this technique for the *Headquarters* relation, for which we did not have an externally provided set of manually constructed queries or filters.
Patterns: This technique exploits the terms in the *extraction patterns* generated by the information extraction system over the training documents, if available. For instance, one of the example patterns for extracting the *Headquarters* relation in Figure 1 is "<ORGANIZATION>, based in <LOCATION>". We can construct a query *"based in"* from this pattern, since this phrase will have to appear in any document that matches the extraction pattern. (Note that we cannot use the named-entity tags *LOCATION* and *ORGANIZATION* in the queries since such tags are typically not accepted as query features by standard search engines.) For *Snowball*, the extraction patterns are not phrases, but rather unordered vectors of terms. In this case, each pattern is converted to a conjunctive query with all the terms in the pattern. An advantage of the *Patterns* strategy is its simplicity: queries are readily derived from the extraction patterns, without further training. A disadvantage of this approach is that the associated queries might be too broad, as in the example above, or too specific, and retrieve too few useful documents. Also, extraction patterns vary considerably by information extraction system (Section 2.2), which makes this approach

[10]Available as part of the *North American News Text Corpus* from the Linguistic Data Consortium at `http://www.ldc.upenn.edu`.

[11]Glimpse also supports limited *regular expression* matching, which we do not exploit.

[12]Note that these filters were originally conceived to be applied to *every* available news document before running *Proteus*. The filters were designed primarily to maximize the *recall* of the extraction system, with less importance given to precision of the resulting document set (i.e., processing potentially useless documents was acceptable).

Relation and Extraction System	% |Useful|	$\|R_{all}\|$	$\|D_{all}\|$
Headquarters: Snowball	23	24,536	
Headquarters: DIPRE	22	20,952	135,438
DiseaseOutbreaks: Proteus	4	8,859	

Figure 6. The **test** database statistics.

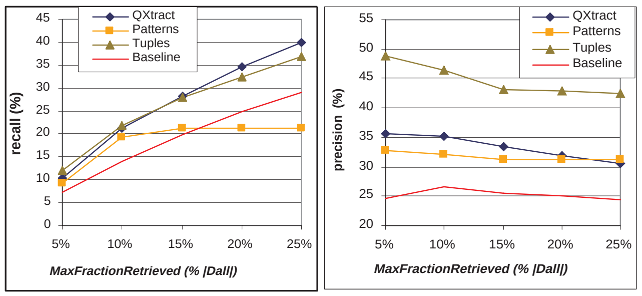
Parameter	*Value*	*Description*
MaxSampleSize	2,000 or 5,000	if *MaxFractionRetrieved* $\leq$ 5% then 2,000; otherwise 5,000
MaxSeedResults	50	Max. documents retrieved per individual seed tuple
NumSeeds	1,000	Max. new seed tuples picked
Query strategy	QCombined	Query generation strategy
MaxSearchResults	1,000	Max. documents retrieved per individual query

Figure 7. Final configuration of *QXtract* as used for evaluation on the **test** database.

not generally applicable. For example, sophisticated information extraction systems incorporate syntactic information into the extraction patterns (e.g., parsing information), which typically cannot be used for querying. We implemented the *Patterns* strategy only for *DIPRE* and *Snowball*: the *Proteus* patterns exploit specific syntactic relationships between terms, so they are not easily converted to regular search engine queries.

4 Experimental Results

In this section, we evaluate our techniques on the **test** database (Section 3.4). The statistics for the occurrence of tuples in the target relations in these databases are summarized in Figure 6. These statistics were generated by running each extraction system over the complete database in order to generate R_{all}. This exhaustive extraction process lasted for many days for one of the extraction systems. As we can see, the tuples in the *Headquarters* relation as extracted by *Snowball* from the **test** database are relatively frequent: tuples for this relation occur in approximately 23% of all of the documents in the database. In contrast, the *DiseaseOutbreaks* tuples occur in less than 4% of the documents. As we will see, *QXtract* exhibits the greatest gains in extraction efficiency for this kind of sparse relation, where it is challenging to identify the few documents with useful information. However, *QXtract* significantly improves the extraction performance on both types of relation, demonstrating the robustness of our techniques. The experiments in this section were run using the *QXtract* configuration summarized in Figure 7. This configuration was determined by tuning the system over the **training** database (Section 3.4), and we do not report further details due to space limitations. Note that the document sampling and query learning are automatically performed from scratch on the **test** database as part of the *QXtract* method, while the purpose of the system tuning over the **training** database was solely to set the best values for the parameters in Figure 7.



(a) (b)

Figure 8. Recall (a) and precision (b) of *QXtract*, *Patterns*, *Tuples*, and *Baseline* over the **test** database using *Snowball* as the information extraction system (*Headquarters* relation).

Extraction of the *Headquarters* relation: Figure 8 shows the performance of the alternative document retrieval methods on the **test** database with *Snowball* as the target information extraction system. Overall, the *QXtract* and *Tuples* strategies have the best recall. For example, Figure 8(a) shows that when *QXtract* returns 10% of the database documents, *Snowball* manages to extract about 21% of the tuples in R_{all} from this reduced document set. Furthermore, Figure 8(b) shows that 35% of the retrieved documents are actually useful, meaning that *Snowball* managed to extract *Headquarters* tuples from them. By comparison, *Tuples* exhibits higher precision of 46%, while the recall remains similar to that of *QXtract*: many of the tuples used for querying for new documents by *Tuples* tend to occur in *multiple* documents, causing *Snowball* to extract these tuples repeatedly from the retrieved documents. As a result, 2,276 out of the 5,339 tuples extracted from the documents retrieved by *Tuples* were extracted repeatedly from multiple documents. In contrast, only 1,292 of the 5,213 tuples extracted from the documents retrieved by *QXtract* came from multiple documents. The reason for the high recall of the *Tuples* strategy is that many of the documents contain multiple *Headquarters* tuples, allowing the tuple-based querying to retrieve many of the useful documents in the database. However, such tuple distribution cannot be generally expected and, as we will see, *QXtract* is the most robust method overall. Interestingly, the *Patterns* strategy[13] is only able to retrieve 11% of the documents in the **test** database, which results in the maximum recall of 21%. *Snowball* generates about 50 patterns for extracting the *Headquarters* relation, and approximately half of these resulted in valid queries. Furthermore, no query was allowed to retrieve more than 1,000 documents (simulating a common policy of web search engines), which prevented some of the potentially productive *Snowball Patterns* queries

[13]Many automatically generated *Snowball* patterns rely on stopwords and punctuation to extract *Headquarters* tuples, and the words that comprise the patterns are not ordered, or contiguous. As a result, *Snowball* patterns are not expressible as phrase queries. Since our search engine, *Glimpse*, does not support proximity queries, we omit the stopwords and punctuation when querying for *Snowball* patterns.

	Patterns	*QXtract*
High recall	office	releases
	unit	company AND unit AND week
	headquarters	companies AND largest AND including
High precision	brokerage AND chatting AND office	companies AND based AND assets AND rose
Low precision	including AND supermarkets	issues

Figure 9. Some *Snowball Patterns* and *QXtract* queries (*Headquarters* relation; *MaxFractionRetrieved* = 10% of $|D_{all}|$).

to retrieve all matched documents. Figure 9 reports some *Patterns* and *QXtract* queries. *QXtract* queries appear to be more topical, resulting in higher recall of *Snowball* over the retrieved documents. For this extraction task, the *Baseline* strategy performs relatively well, because *Headquarters* is a "dense" relation with 23% of the documents in the database being useful, as discussed.

To evaluate the generality and robustness of our approach, we run the same *QXtract* configuration reported in Figure 7, but now using *DIPRE* as the underlying information extraction system. Figure 10 summarizes the recall and precision results, which are consistent with the results for *Snowball*. Figure 11 reports some *Pattern* and *QXtract* queries. Note that the *Patterns* queries for *DIPRE* were treated as conjunctions of one or more *phrases*, requiring exact string equality (including punctuation) for a successful match. As we see in Figure 11, *DIPRE*'s *Patterns* queries are highly specific. This prevents *Patterns* from retrieving more than 10% of the database, resulting in relatively low recall (Figure 10(a)), while maintaining relatively high precision (Figure 10(b)).

To summarize the results for the dense *Headquarters* relation, *QXtract* and *Tuples* performed best overall: the document sets that they retrieve result in significantly better precision and recall than random document samples. This holds for both the *Snowball* and *DIPRE* information extraction systems. However, the recall values of all techniques seem low when they retrieve modest fractions of the test database: as discussed, this low recall is due to the fact that a large fraction of the documents in the test database are useful for extracting the *Headquarters* relation.

Extraction of the *DiseaseOutbreaks* relation: To further test the generality of our approach, we evaluated the performance of *QXtract* and the other techniques on the *DiseaseOutbreaks* relation (Section 3.3). The results on the **test** database are shown in Figure 12. *DiseaseOutbreaks* is a "sparse" relation, with tuples occurring in a much smaller fraction of the test database than is the case for *Headquarters*: less than 4% of the test database documents are useful for extracting this relation (Figure 6). The performance of *QXtract* for this scenario is remarkable: for example, when

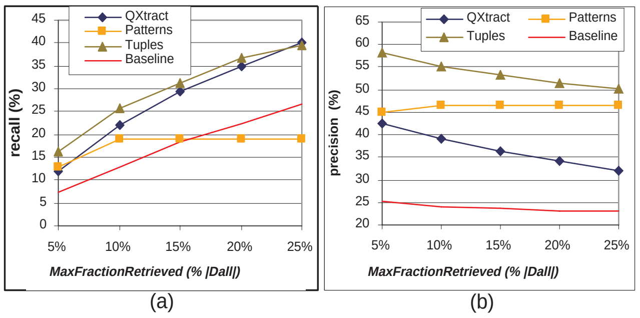


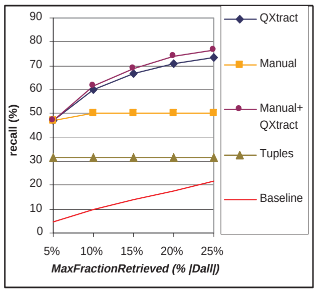
Figure 10. Recall (a) and precision (b) of *QXtract*, *Patterns*, *Tuples*, and *Baseline* over the **test** database using *DIPRE* as the information extraction system (*Headquarters* relation).

	Patterns	*QXtract*
High recall	", based in "	releases
	"is based in "	company AND based AND stock AND agreed
	", of "	company AND based AND acquisition AND buy
High precision	", a biotechnology company based in"	company AND based AND largest AND share AND buy AND big
Low precision	", the"	reports

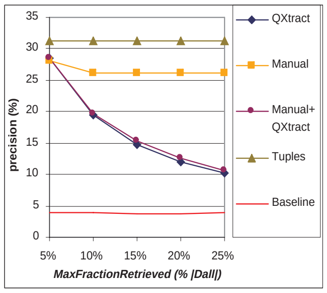
Figure 11. Some *DIPRE Patterns* and *QXtract* queries (*Headquarters* relation; *MaxFractionRetrieved* = 10% of $|D_{all}|$).

QXtract retrieves only 5% of the database documents (i.e., *MaxFractionRetrieved* = 5%), *Proteus* manages to extract 48% of the tuples in R_{all} (Figure 12(a)). Figure 12(b) shows that 29% of the documents retrieved by *QXtract* are actually useful, meaning that *Proteus* managed to extract *DiseaseOutbreaks* tuples from them. In contrast, the *Baseline* strategy exhibits the expected recall and precision for a random sample of 5% of the database documents. Unlike the results for the *Headquarters* relation, the *Tuples* strategy performs worse than *QXtract* for *DiseaseOutbreaks*. Using the same seed tuples as the *QXtract* strategy, *Tuples* is able to retrieve less than 5% of the documents in the **test** database, resulting in 31% recall.

QXtract also performs better than the *Manual* strategy, and requires no human involvement to generate these queries. For example, when *Manual* retrieves 10% of the database documents, *Proteus* manages to extract 50% of the tuples in R_{all} (Figure 12(a)). In comparison, *Proteus* manages to extract 60% of the R_{all} tuples from the document set of the same size retrieved by *QXtract*. Figure 13 reports some *Manual* and *QXtract* queries. For this extraction task, *QXtract* approximates closely some of the *Manual* queries, but includes more specific words discovered during the *DocumentSample procedure* (e.g., "ebola"). This results in higher recall than manually constructed queries, which were developed without the benefit of analyzing the **test** database.



(a)



(b)

Figure 12. Recall (a) and precision (b) of *QXtract*, *Tuples*, *Manual*, *Manual+QXtract*, and *Baseline* over the **test** database using *Proteus* as the target information extraction system (*DiseaseOutbreaks* relation).

The tradeoff between the completeness of the extracted relation and the number of documents retrieved is apparent in Figure 12(a): retrieving a larger set of documents allows *Proteus* to extract a closer approximation of R_{all}. For example, to extract an additional 12% of the R_{all} tuples after extracting 48% of R_{all} from 5% of the database requires that *QXtract* retrieve an additional 5% of the database documents. This tradeoff between relation completeness and extraction efficiency can be managed by the user to incrementally extract the target relation by increasing the value of the *MaxFractionRetrieved* parameter (thereby retrieving additional documents) until the extracted relation is "sufficiently complete."

We also explored combining the existing *Manual* queries with *QXtract* queries, resulting in a hybrid strategy to which we refer as *Manual+QXtract*. We submit these queries one at a time, alternating between *QXtract* and *Manual* queries until the desired fraction of the database is retrieved. Figure 12(a) shows that adding *Manual* queries to *QXtract* results in only a slight improvement in the quality of the retrieved documents.

In summary, the results for the sparse *DiseaseOutbreaks* relation show that *QXtract* accurately identifies the useful documents in the **test** database, allowing *Proteus* to extract the closest approximation of R_{all} for all fractions of the database retrieved. Depending on the *MaxFractionRetrieved* parameter, *QXtract* allows *Proteus* to extract between 48% and 74% of the tuples in R_{all}, while retrieving only between 5% and 25% of the documents in the test database. Overall, *QXtract* emerges as the most robust technique, performing well on both the dense *Headquarters* relation and on the sparse *DiseaseOutbreaks* relation.

Actual Running Times: To further illustrate the performance advantage of using *QXtract*, we computed the actual running times for extracting tables with and without *QXtract*. The experiments were run on 1 GHz Pentium III machines running Linux RedHat 7.1. The speedup achieved by *QXtract* is remarkable: Running the *Proteus* system to extract the *DiseaseOutbreaks* relation from the complete **test** database required over 15 days to finish. In contrast, our *QXtract*-based approach required 0.83 and 1.45 days to extract, respectively, 48% and 60% of the R_{all} tuples from the 5% and 10% retrieved fractions of the **test** database. Using *QXtract* for extracting the *Headquarters* relation using *Snowball* and *DIPRE* also resulted in efficiency gains. Overall, *QXtract* emerges as a highly *effective* and *efficient* technique for extracting relations from large text databases.

	Manual	*QXtract*
High recall	virus infected infection	disease AND died virus infected
High precision	"outbreak had spread"	virus AND ebola AND recent
Low precision	"new case of"	health

Figure 13. Some *Manual* and *QXtract* queries (*DiseaseOutbreaks* relation, *MaxFractionRetrieved* = 10% of $|D_{all}|$).

5 Conclusions

In this paper, we developed an automatic technique for generating queries to retrieve documents that are promising for the extraction of a target relation. We demonstrated that our method is general and efficient through a comprehensive experimental evaluation over more than 270,000 real documents. Our techniques allow for significant improvement in extraction efficiency in the number of documents processed: for the *Headquarters* relation, *QXtract* retrieves document sets allowing *Snowball* and *DIPRE* to approximate the target relation significantly better than *Baseline*. For the *DiseaseOutbreaks* relation, the improvement is dramatic: *QXtract* allows *Proteus* to extract 48% of the tuples in the target relation when retrieving only 5% of the documents in the test database. Hence *QXtract* will help deploy existing information extraction systems at a larger scale and for a wider range of applications than previously possible. In addition to improving efficiency of extraction, our automatic querying techniques can be used to extract a target relation from an arbitrary "hidden-web" database accessible only via a search interface [2]. As another example, our techniques could be used to exploit information behind a standard web search engine, hence providing a building block for scalable and portable information extraction over the web at large. We plan to evaluate *QXtract* in these scenarios as part of our future work.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grants No. IIS-9733880 and No. IIS-9817434. We thank Ralph Grishman and Roman Yangarber for fruitful discussions, and also Silja Hattunen for allowing us to experiment with their *Proteus* information extraction system. We also thank Regina Barzilay, Nicolas Bruno, Eleazar Eskin, and Amélie Marian for their helpful comments.

References

[1] *Proceedings of the 7th Message Understanding Conference, 1998*. Accessible at `http://www.itl.nist.gov/-iaui/894.02/related_projects/muc/-proceedings/muc_7_toc.html`.

[2] *The Deep Web: Surfacing hidden value*. BrightPlanet.com LLC, July 2000. Available at `http://www.complete-planet.com/Tutorials/DeepWeb/index.asp`.

[3] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, June 2000.

[4] E. Agichtein, S. Lawrence, and L. Gravano. Learning search engine specific query transformations for question answering. In *Proceedings of the 10th World Wide Web Conference (WWW-10)*, 2001.

[5] S. Brin. Extracting patterns and relations from the World-Wide Web. In *Proceedings of the 1998 International Workshop on the Web and Databases (WebDB'98)*, Mar. 1998.

[6] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th World Wide Web Conference (WWW)*, 2002.

[7] W. Cohen and Y. Singer. Learning to query the web. In *Proceedings of the AAAI Workshop on Internet-Based Information Systems*, 1996.

[8] W. W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, 1995.

[9] J. Cowie, R. Wakao, L. Guthrie, W. Jin, J. Pustejovsky, and S. Waterman. The Diderot information extraction system. In *Proceedings of the 4th Message Understanding Conference*, 1992.

[10] D. Day, J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. Mixed-initiative development of language processing systems. In *Proceedings of the Fifth ACL Conference on Applied Natural Language Processing*, Apr. 1997.

[11] G. Flake, E. J. Glover, S. Lawrence, and C. L. Giles. Extracting query modifications from nonlinear SVMs. In *Proceedings of the 11th World Wide Web Conference (WWW-2002)*, 2002.

[12] R. Gaizauskas and A. M. Robertson. Coupling information retrieval and information extraction: A new text technology for gathering information from the web. In *Proceedings of RIAO 97: Computer-Assisted Information Searching on the Internet*, 1997.

[13] R. Ghani and R. Jones. Automatic training data collection for semi-supervised learning of information extraction systems. Technical report, Accenture Technology Labs, May 2002.

[14] R. Ghani, R. Jones, and D. Mladenic. Mining the web to create minority language corpora. In *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM)*, 2001.

[15] L. Gravano, P. Ipeirotis, and M. Sahami. QProber: A system for automatic classification of hidden-web databases. *ACM Transactions on Information Systems (TOIS), accepted for publication*, 2002.

[16] R. Grishman. Information extraction: Techniques and challenges. In *Information Extraction (International Summer School SCIE-97)*. Springer-Verlag, 1997.

[17] R. Grishman, S. Huttunen, and R. Yangarber. Real-time event extraction for infectious disease outbreaks. In *Proceedings of Human Language Technology Conference (HLT)*, 2002.

[18] T. Joachims. Making large-scale support vector machine learning practical. *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, December 1998.

[19] A. Kruger, C. L. Giles, F. Coetzee, E. J. Glover, G. W. Flake, S. Lawrence, and C. W. Omlin. DEADLINER: Building a new niche search engine. In *International Conference on Knowledge Management (CIKM)*, 2000.

[20] C. C. T. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. In *Proceedings of the 10th World Wide Web Conference (WWW-10)*, 2001.

[21] D. Lewis and R. Tong. Text filtering in MUC-3 and MUC-4. In *Proceedings of the 4th Message Understanding Conference*, 1992.

[22] U. Manber and S. Wu. Glimpse: A tool to search through entire file systems. In *Proceedings of the 1994 Winter USENIX Conference*, 1994.

[23] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th Conference on Very Large Databases*, 2001.

[24] S. Robertson. On term selection for query expansion. In *Journal of Documentation*, volume 46, 1990.

[25] S. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27, 1976.

[26] J. Rocchio. Relevance feedback in information retrieval. In *SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[27] G. Salton. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley, 1989.

[28] E. Voorhees. The TREC-8 question answering track report. In *Proceedings of TREC-8*, 1999.

[29] J. Xu and W. B. Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems (TOIS)*, 18(1), 2000.

[30] R. Yangarber and R. Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*. Morgan Kaufman, 1998.

[31] R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. Unsupervised discovery of scenario-level patterns for information extraction. In *Proceedings of Conference on Applied Natural Language Processing ANLP-NAACL*, 2000.