Querying Large Text Databases for Efficient Information Extraction

Eugene Agichtein Luis Gravano

Columbia University
Computer Science Department
New York, NY 10027
{eugene, gravano}@cs.columbia.edu

Abstract

A wealth of data is hidden within unstructured text. This data 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. This exhaustive approach is not practical, or sometimes even feasible, for large databases. In this paper, we develop an efficient query-based technique to identify documents that are potentially useful for the extraction of a target relation. We start by sampling the database to characterize the documents from which an information extraction system manages to extract relevant tuples. Then, we apply machine learning and information retrieval techniques to derive queries likely to match additional useful documents in the database. Finally, we issue these queries to the database to retrieve documents from which the information extraction system can extract the final relation. Our technique requires that databases support only a minimal boolean query interface, and is independent of the choice of the underlying information extraction system. We report a thorough experimental evaluation over more than one million documents that shows that we significantly improve the efficiency of the extraction process by focusing only on promising documents. Our proposed technique could be used to query a standard web search engine, hence providing a building block for efficient information extraction over the web at large.

1 Introduction

Text documents often hide valuable *structured data*. For example, a database of newspaper articles might contain information on the *location* of the headquarters of a number of *organizations*. Many large text databases, including the set of publicly-accessible web pages, contain data that can be best exploited in structured form. The goal of information extraction is to produce a structured representation of the information that is “buried” in unstructured (text) documents.

As another example of an application that could benefit from automatic information extraction, consider a large database of customer feedback emails to a large manufacturer. Presumably, this manufacturer would want to address customer-reported problems effectively. A structured relation *CustomerComplaints(ProductName, ProductType, Retailer, Complaint)*, listing the details of the problems that a customer has encountered, would help analyze and address customer complaints. An example of a document “hiding” a tuple for this relation is in Figure 1. From this document, an information extraction system could extract a tuple $<$*“inkjet cartridge”, “printer cartridge”, “www.greatcartridges.com”, “it lacked two of the three colors”*$>$. Once the customer complaint information is in structured form, the manufacturer could run sophisticated queries over it, and even mine the data for interesting patterns (e.g., to identify particularly problematic retailers or products).

> [...] When I purchased **four inkjet cartridges** (two black and two color) from this company, I did not realize that they were remanufactured. Revisiting the site at **www.greatcartridges.com** after receiving the cartridges, [...]. My second mistake was not to try the **color cartridges** when I received them. When I did put one of them into my printer, I found that **it lacked two of the three colors** necessary for full-color printing and was useless. [...]

Figure 1: A document fragment that may be used to extract a tuple for the *CustomerComplaints* relation.

State-of-the-art extraction systems [7] are quite sophisticated and typically require labor-intensive training. After training, these systems 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.

In this paper, we address this scalability problem of all information extraction systems, and introduce the first query-based technique to identify the database documents that are potentially useful for the extraction of a target relation. Our technique makes it possible for an information extraction system of choice 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 minimal user-provided feedback. Specifically, a user needs to provide our system with only a handful of example tuples of the target relation. Our system then retrieves a sample of documents from the database where all attributes of at least one of the initial tuples can be found. We process these documents using the information extraction tool of choice, after which our system applies machine learning and information retrieval techniques to discover the features that make documents useful for extracting the desired relation. This information is used to generate queries that are likely to retrieve additional useful documents from the database. The retrieved documents are processed by the information extraction system to extract the final relation.

The key contribution of this paper is our query-based technique for identifying useful documents for information extraction from large text databases. Our technique expects document databases to support only a minimal boolean query interface, and is independent of the choice of information extraction system. Furthermore, our technique could be used to query a standard web search engine, hence providing critical *infrastructure for efficient information extraction from the web at large*. We report a large-scale evaluation of our technique over more than one million real documents, which shows that we significantly improve the efficiency and scalability of the extraction process by focusing only on promising documents.

Related Work

Information extraction has been the focus of active research for decades. The main emphasis of this research, notably in the context of the Message Understanding Conference (MUC), has been on the quality of the extracted relation [7]. In contrast, our work assumes a given information extraction system, and focuses on how to retrieve a relatively small set of documents that would allow the extraction of a close approximation of the target relation efficiently.

Two information extraction systems on which we focus in this paper are *DIPRE* [2] and *Snowball* [1]. These systems are attractive alternatives to traditional extraction systems because they require minimal human training to work. Both *DIPRE* and *Snowball* start with only a handful of user-provided examples of the tuples to be extracted, and proceed to extract the target relation by finding segments of text similar to those in which the seed tuples occur. New tuples are extracted from these text segments and become the new seeds for the next iteration of the system. In this paper, we experiment with both *DIPRE* and *Snowball* as the information extraction systems of choice.

The problem of retrieving documents that are "relevant" to a user's information need has been studied extensively in the information retrieval (IR) field [14]. Although our problem is different in nature, we exploit state-of-the-art term weighting and query expansion results [12] from IR in the design of one of our system's variants (Section 2.3.2). For this, we initially identify a set of "relevant" documents by sampling the database to discover documents that are useful to the information extraction system, and then extract the most important terms from these documents.

Alternatively, the characterization of the useful documents given an initial sample could be viewed as a traditional classification problem. In fact, we also explored a number of machine learning techniques [5, 9] in the design of other variants of our system (Section 2.3.2).

Our work is related to recent research on focused web crawling (e.g., [4, 3]), which addresses the problem of fetching web pages relevant to a given topic via focused crawling. Our proposed technique is tuned for information extraction, and operates over any searchable text database, whether its contents are "crawlable" or not.

Recent work [11] addresses the problem of crawling the "*hidden web*," the portion of the web hidden behind search forms. The authors report a method for crawling specialized data sources using a manually-constructed task-specific database of potential query terms, which is further expanded during crawling. In contrast, our goal is different: we attempt to extract the most complete *relation* from the text database while retrieving *as few documents as possible*.

The rest of the paper is organized as follows. Section 2 presents our new document retrieval method in detail. Then, Section 3 summarizes the general experimental setting, including the evaluation methodology, metrics, and databases we used for tuning and evaluation of our strategy. Section 4 describes how we estimated the best parameters for our system. Section 5 reports the results of an experimental evaluation of our technique (and several baseline strategies) on a large database of text documents. Finally, we conclude the paper in Section 6.

2 Retrieving Promising Documents from Text Databases

To extract a relation from a document database, all current state-of-the art information extraction systems require examining every document at least once, which is not practical or sometimes even feasible for large databases. In this section, we describe a new method for querying a text database to retrieve only the documents useful for extraction of the target relation, which can then be processed as usual by an information extraction system.

Specifically, the problem we address in this section is as follows. We are given an information extraction system E and a database D_{all}, together with a specification of the relation that we want to extract. Let R_{all} denote the actual instance of the relation that E would extract from the entire database D_{all}. Our goal is to construct a close approximation of R_{all}, R, by retrieving a small fraction of D_{all}, D, and then having E operate on D rather than on the much larger database D_{all}. Note that R_{all} may not contain *all* of the correct tuples that could be extracted from the database by a perfect system. Rather, we are limited by the best relation that a given extraction system can extract, and we try to approximate that relation in an efficient manner.

Section 2.1 provides an overview of how we fetch useful documents to feed a given arbitrary information extraction system. We first retrieve a small sample of documents and determine those from which the extraction system is able to extract tuples (Section 2.2). This sample is subsequently used to provide examples of useful and non-useful documents to our methods of generating queries to retrieve the rest of the useful documents in the database (Section 2.3), as we describe next.

2.1 System Overview

We would like to extract a specified relation from a database of text documents with a search interface, provided a small number of example tuples in the relation. Throughout the paper, as a running example and also in most

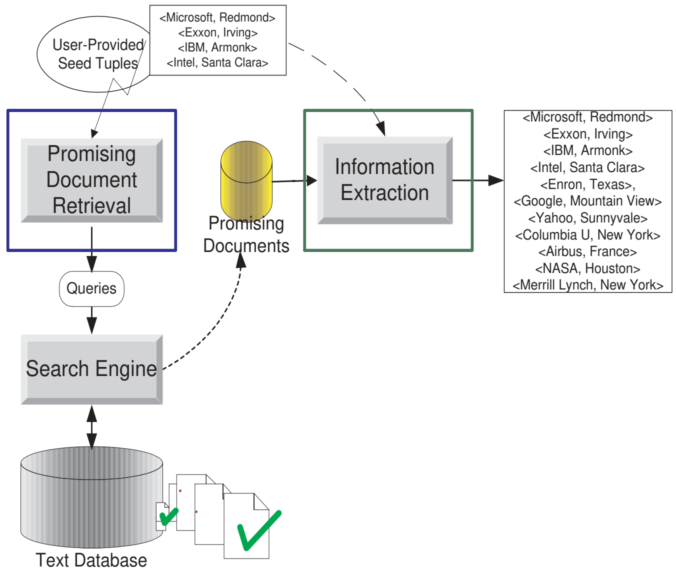

Figure 2: The architecture of an efficient information extraction system that identifies promising documents via querying.

Organization	Location of Headquarters
MICROSOFT	REDMOND
EXXON	IRVING
IBM	ARMONK
INTEL	SANTA CLARA

Figure 3: Example user-provided seed tuples for the *Headquarters* relation.

experimental results, we will use relation *Headquarters(organization, location)*, which contains a tuple $< o, l >$ if organization o has headquarters in location l. Figure 3 shows the example seed tuples that we assume a user specified to our system as the only human-generated input we require.

The text database from which we retrieve documents 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, or a web search engine such as Google). Only a small fraction of the database may be relevant to the relation of interest, so in this case it would be wasteful to run the information extraction system over every available document. If our database is the set of all web pages indexed by a search engine such as Google, then it is simply impossible to scan every page to extract tuples. For these reasons, our approach zooms in on the potentially useful documents, while ignoring the rest.

The main components of the system are shown in Figure 2. We interact with the text database through a search engine. We assume that the search interface supports simple boolean queries such as *"data AND mining AND NOT apriori"*. This query model provides sufficient expressiveness, and is widely supported: all of the major available text indexing tools (e.g., Glimpse [10]) and web search engines support such queries with minor variations in syntax. For example, Google's syntax for the above query is *"data mining -apriori"*, with the *AND* operator being implicit.

We can apply our general document retrieval approach to a database containing documents in any domain by using an appropriate underlying information extraction system, which can be arbitrarily specialized for that domain. For example, if we wanted to extract complex corporate acquisition events, might want to use the PET system [15], a powerful information extraction system created at NYU and specifically trained for this task. Alternatively, if we wanted to extract the relation of protein interaction with subcellar structures from MEDLINE, a database of more than 9 million abstracts of medical articles we could use the system described in [6].

For flexibility, we developed a modular architecture by treating the information extraction system as a black box, and interacting with it solely through a *wrapper* that exports a uniform interface to other components. The wrapper hides peculiarities of the information extraction system, and provides a single input/output interface:

- **Input:** A set of example tuples (*Seed*) in the target relation, and a set of documents D for the extraction system to process.
- **Output:** The set of new tuples (*Tuples*) extracted from D, and for each tuple $t_i \in Tuples$, the set of identifiers U_i of the documents from which t_i was extracted. The wrapper returns the identity of all the useful documents, computed as *Useful* $= U_1 \cup U_2 \cup ... \cup U_{|T|}$.
- **Optional Output:** *Confidence:* Some extraction systems are able to assign a *confidence* value to each extracted tuple. In this case, the wrapper also returns a list of weights $W = < \ldots, W_i, \ldots >$, where W_i is the confidence with which tuple t_i was extracted. *Matched Patterns:* Some systems may store additional information about each extracted tuple, such as the extraction pattern that produced the tuple. In this case, the wrapper also returns this information in a list $TP = < \ldots, TP_i, \ldots >$, where TP_i is the set of patterns used to extract tuple t_i. *All Patterns:* An extraction system may export just the set of all the extraction patterns P that it has available for extracting the target relation.

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 document retrieval process.

The only information that we assume we have initially is a set of user-provided tuples that are part of the target relation (Figure 3). Our method does not require any additional prior information about the documents in the database: we proceed to sample the database starting with only the example tuples. (Presumably, at least some of the seed tuples occur in the database.) The output of the document retrieval component (Figure 2) is a set of *promising documents*, which are then used as input to the information extraction system.

Before we delve into details of our approach, we specify the general class of relations we consider. R is an arbitrary relation with attributes $a_1, a_2, ..., a_n$. Our querying technique will be able to support extraction of R if:

1. An attribute of R is a key, i.e., it must uniquely determine the remaining attributes of the relation (e.g., $a_1 \rightarrow \{a_2, \ldots, a_n\}$).
2. A tuple $t \in R$ will only be extracted if all of its attributes $< a_1, a_2, \ldots, a_n >$ 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.)

For example, the *Headquarters* relation described earlier has the key attribute *organization* (an organization is assumed to have a single headquarters location), and both the *organization* and its associated *location* must appear within the same document in order for the tuple to be extracted.

We will rely on these assumptions (which may be relaxed as part of our future work) during querying. The information extraction system may impose additional internal constraints on the types of relations it can extract, but it does not change the promising document retrieval procedure. For example, *Snowball* [1] and *DIPRE* [2] both have been developed to extract binary relations. Our document retrieval procedure does not have these additional limitations.

The overall document retrieval process is shown in Figure 4. Starting with a set of user-provided seed tuples, we first use the seed sampling procedure described in Section 2.2 to retrieve a small sample of documents, likely to be useful to the extraction system for extracting the target relation. The information extraction system is run over this sample set, producing as output the set of extracted tuples, and the identifiers of useful documents. The document sample is converted to positive and negative examples, where the positive examples represent the documents in the sample that were determined to be useful for extraction. These examples allow us to derive queries targeted to match

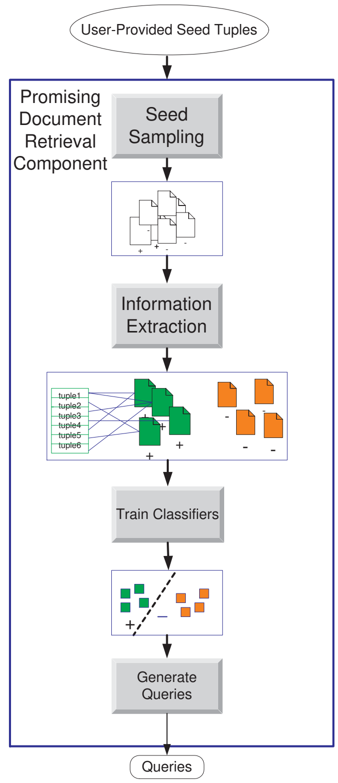

Figure 4: Promising document retrieval: detailed view.

–and retrieve– documents similar to the positive examples (Section 2.3). These queries are used to retrieve a set of promising documents from the database (Section 2.4), which are returned as input to the information extraction system.

2.2 Retrieving a Seed Document Sample 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 have at this point about the target relation is the example tuples provided by the user. Our goal is to retrieve enough useful documents to provide sufficient examples to the subsequent query training stage. To accomplish this, we use the *Seed Sampling* algorithm shown in Figure 5. Each round of sampling consists of two stages:

1. We retrieve a set of documents to be added to the sample D_{sample} by querying the search engine for occurrences of the current seed tuples, which initially are provided by the user as $Seed_1$ (line 0).

2. We run E over the documents in D_{sample}, extracting the current set of tuples. From these, we select a small number as new seed tuples and start a new sampling round.

To retrieve documents, we build queries with the attribute values of each tuple t in the current $Seed_i$ set (line 3). Each tuple t is used to construct two queries, $q^+ = t.a_1$ *AND* $t.a_2$ *AND ...* $t.a_n$, and $q^-= t.a_1$ *AND NOT* *(*$t.a_2$ *AND ...* $t.a_n$*)*, assuming a_1 is the key of the target relation. Query q^+ will retrieve documents where all the attributes of t appear somewhere within the same document. In principle, these are documents from which t could have been extracted by the information extraction system. In contrast, query q^- will retrieve documents where not

Procedure `SeedSample(`$Seed_1$`)`
0 $D_{sample} = \{\}$
1 `For` i `= 1 to` $maxSamplingRounds$
2 `For` $s \in Seed_i$
3 $Temp$ `= QuerySeed(`s, $maxSeedResults$`)`
4 $D_{sample} = D_{sample} \cup Temp$
5 `(`$Tuples$, $Useful$, W, TP, P`) =` E`.extract(`$Seed_i$, D_{sample}`)`
6 $topTuples$ `= getTopTuples(`$Tuples$, U, k`)`
7 $Seed_{i+1}$ `=` $Seed_i \cup topTuples$
8 $Useless$ `=` $D_{sample} - Useful$
9 `Return(`$Tuples, Useful, Useless, W, TP, P$`)`

Figure 5: The seed sampling algorithm.

all attributes of tuple t appear together. These documents are not useful with respect to the extraction of tuple t: although they mention t's key (i.e., a_1), they do not contain at least one of the remaining attributes and hence the information extraction system would not have found these documents useful for t. Intuitively, query q^+ will tend to retrieve useful documents for extraction, while query q^- will tend not to. We will need these *negative* examples to help derive queries for retrieving promising documents. We retrieve the first *maxSeedResults* matches returned by the database for each query. The query results are added to the set of documents retrieved so far, D_{sample} (line 4).

Clearly, not all documents retrieved by q^+ will be useful for extraction, and some of the documents retrieved by q^- may actually be useful and contain a different valid tuple. To determine which documents are useful, we run the information extraction system E over D_{sample} (line 5), which returns the extracted tuple set *Tuples* and identifiers of useful documents from which the tuples were extracted. Additionally, the wrapper returns the assigned confidences of the tuples W and the extraction patterns that the information extraction system used. Per our specification, W, TP, and P may all be empty if the extraction system does not export this information.

In line 6, we select k tuples from $Tuples$ into $topTuples$, which will be added to $Seed$ and used for the next round of sampling. The choice of seed tuples is critical, since it will strongly affect the resulting sample set. Specifically, we need to decide on the value of k, and on the criteria by which to order the extracted tuples in order to select the top-k tuples. We determine a good value for k as part of the system tuning (Section 4). To each of the extracted tuples $t_i \in Tuples$ we assign a score, equal to $|U_i|$, where U_i is the set of unique sample documents from which t_i was extracted, and order the tuples by this score. This favors selecting tuples that are likely to appear in many documents in the database.

To select the new seed tuples $Seed_{i+1}$ (line 7) for sampling round $i + 1$, we *expand* the set $Seed_i$ with the new tuples $topTuples$. The new set $Seed_{i+1}$ becomes the current seed and a new round of sampling starts. The number of sampling rounds, *maxSamplingRounds*, is a parameter that we tune during training in Section 4.

Unfortunately, we cannot simply continue the seed sampling process to retrieve all of the useful documents in the database. As we will show in Section 4, after a few sampling rounds only a small fraction of the target relation is extracted, and the process tends to converge, with no additional useful documents added in subsequent rounds. However, many more useful documents are typically still "hiding" in the collection at this point. These documents are not retrieved because they only contain tuples that have not yet been extracted. Therefore, we cannot retrieve the remaining documents using the attribute values of the extracted tuples alone so we need to resort to an alternative strategy.

Our key observation is that many useful documents share similarities in content. For example, useful documents for the *Headquarters* relation may contain terms or phrases such as "headquarters of," "the spokesperson for the company located in," "in the interview given in the secluded campus of," etc. These combinations of terms are more likely to occur in useful documents than in non-useful documents for extracting the *Headquarters* relation. Our goal now is to generate queries that would identify and use such terms to retrieve the documents similar to the ones that the extraction system marked as *Useful*. For this, we exploit the documents in D_{sample} as a training set with positive

Okapi	*Ripper*	*SVM*	*Patterns*
company	based AND company	laboratory	based AND losers AND fell
based	based AND largest AND leader	station	bloomberg AND oct
bloomberg	based AND bloomberg AND acquisition	maker	headquarters
percent	companies AND employees AND expected	based	shares
shares	analysts AND earnings	produces	based AND rosemont

Figure 6: Some top candidate queries produced by the different query generation techniques.

and negative examples of useful documents.

2.3 Learning Queries to Retrieve Promising Documents

Given a set of useful and non-useful 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 proceeds in three stages:

1. Convert positive and negative examples into an appropriate representation for training (Section 2.3.1).
2. Use the training examples to determine the distinguishing characteristics of the positive examples, and generate an ordered list of queries expected to retrieve new useful documents (Section 2.3.2).
3. Submit the queries to the database to retrieve new potentially useful documents (Section 2.4).

2.3.1 Representation of Training Examples

A good representation of the training examples is crucial in order for any learning mechanism to succeed. For our problem, the choice of representation must also be such that the training results can be used to generate database queries. For instance, we cannot use HTML tags as part of the example representation even if there is a combination of HTML tags that only appears in useful documents, since search engines typically do not allow querying for HTML tags.

Therefore, we must account for the method of querying that will be used. Since our goal is to design a *general* mechanism to query for promising documents, we will produce queries for a vanilla boolean query model, which is widely supported with clear semantics. For generality, we will not rely on more advanced query features such as the specification of a desired order among the query words, which would restrict the applicability of our results. Of course, if such advanced features (or alternative query models) are available, we could apply the same general approach that we present in this paper and tailor it to the query interface of choice.

An important decision that must be made is whether to focus our attention on *complete* documents or rather on just the *local context* around a tuple occurrence. While in some cases considering the content of the whole document as an example may introduce noise, in others this approach may prove helpful in identifying other potentially useful documents. Therefore, we experiment with both *document* and *local* contexts as possible "granularities" of a positive or negative example. In the *document* feature set, all terms from a useful document are considered during training. In the *local* feature set, only the terms in the same line as an extracted tuple –according to the document's original formatting– are considered during training. In either case, we eliminate so-called *stopwords* (e.g., "a" and "the") from consideration. Additionally, all tags inserted by the named entity tagger, and all the entities recognized by the tagger (including the values, of attributes of the extracted tuples) are discarded. In Section 4 we explore the choice of features to determine which representation works best.

2.3.2 Generation of Queries from Examples

After identifying a set of useful and non-useful documents, we now turn to the generation of simple queries that would hopefully retrieve most remaining 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. Specifically, we first consider query generation as an IR automatic query expansion problem, using a state-of-the-art term weighting scheme. Then, we consider query generation techniques that exploit the output of two machine-learning text classifiers. Finally, we present a simple query generation technique that builds queries from the extraction patterns built over the training documents by the information extraction system of choice.

Okapi: As a first query generation technique, we exploit a state-of-the-art term weighting scheme from IR, from the *Okapi* retrieval system [12]. To predict which terms are most likely to retrieve useful documents, we compute the *selection weight* [12] 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 non-useful ones.

First, each term t_i in the document is assigned the Robertson-Spark Jones term weight $w_i^{(1)}$ [13]:

$$w_i^{(1)} = \log \frac{(r+0.5)/(R-r+0.5)}{(n-r0.5)/(N-n-R+r+0.5)}$$

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 database, 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 relevance information in the training data. Then, we compute the *query selection weight* wtr_i of each term t_i as described in [12] for automatic query expansion:

$$w_i = tf_i \cdot w_i^{(1)}$$

where tf_i is the number of unique useful documents in which t_i appears and $w_i^{(1)}$ is as 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. Figure 6 shows some of the queries generated by this approach for the *Headquarters* relation, using the data for the experiments of Section 5.

Ripper: As a second query generation approach, we exploit a highly-efficient rule-based text document classifier, Ripper [5], developed at AT&T Research Labs. Ripper is trained with a set of useful and non-useful documents, where each document is represented as a bag of words. Ripper learns concise rules such as *"based AND company → 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 top $numQueries$ rules are then translated into conjunctive queries in the syntax accepted by the search engine. For example, the rule above might be translated to query *"based AND company."* Figure 6 shows some of the queries generated by this approach for the *Headquarters* relation.

Support Vector Machines (SVMs): As a third query generation approach, we exploit another family of classifiers, SVMs, which have been shown to perform well in text classification [9]. SVMs operate by finding an optimal hyperplane that separates the positive from the negative examples. To filter out noise, we prune the set of terms used in training by discarding those that occur in fewer than a minimum fraction of training examples. The result of training the SVM is an n-dimensional hyperplane, where n is the number of terms in the pruned feature set. The learned hyperplane constants are essentially weights assigned to each term. We use a freely-available efficient implementation of linear-kernel SVMs [9].

During classification, the score for a document is computed as the sum of the SVM weights of each term in the document. If the score exceeds a threshold, then the document is classified as useful. Because of the high dimensionality of the feature space, converting the resulting feature weights to queries is not trivial. To generate rules from SVM feature weights, we use an algorithm for extracting minimal rules from SVMs presented in [8]. We compute all *minimal* sets of terms that are collectively sufficient to imply a positive classification of a document. The result of this algorithm is a set of "rules" similar to the Ripper output. We calculate the expected precision of each rule and generate queries from these rules similarly to the way we process Ripper rules. Figure 6 shows some of the queries generated by this approach for the *Headquarters* relation.

Patterns: As a fourth and final query generation approach, we attempt simply to exploit the terms in the *extraction patterns* generated by the information extraction system over the training documents, if available. For example, from a *Snowball* pattern <{<``the'', 0.2>}, LOCATION, {<``-'', 0.5>, <``based'', 0.5>} ORGANIZATION, {}> for the *Headquarters* relation we can extract a query *"based"*, since this word will have to appear in any document that matches the extraction pattern. (Note that we do not use stopwords, punctuation, or the named-entity tags *LOCATION* and *ORGANIZATION* in the queries.) An advantage of this approach is its simplicity: For each pattern generated by the extraction system, we generate a conjunctive boolean query consisting of all non-stopword terms in the pattern.

A disadvantage of this approach is that the associated queries might be too broad, as in the example above. Also, extraction patterns vary considerably by information extraction system, which makes this approach not that 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 have implemented this approach for *Snowball* and *DIPRE* patterns. *Snowball* patterns have a confidence estimate associated with them, while *DIPRE* patterns do not. We order the *Snowball* patterns in descending order by confidence, and the *DIPRE* patterns in descending order by frequency of occurrence. Figure 6 shows some of the queries generated by this approach for the *Headquarters* relation, for *Snowball* as the underlying information extraction system.

2.4 Querying for Promising Documents

We described above how to generate queries that are likely to retrieve more useful documents than non-useful documents. These queries are used to extract 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 $maxTotalRetrieved$ for the number of documents that we are willing to extract. The higher this upper bound, the more complete the extracted relation is likely to be. We submit the top $numQueries$ queries (generated and ranked as in Section 2.3.2) 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, $maxResults$, is reached. (Section 4 explores the choice of values for $numQueries$ and $maxResults$.) We keep the running total of the documents retrieved so we stop before exceeding the upper bound $maxTotalRetrieved$. After this bound is reached (or there are no more documents to extract using our queries), all retrieved documents are returned as the output of the *promising document retrieval* component. These potentially useful documents are then input to the information extraction system to extract the final approximation of the target relation.

3 Experimental Setting

In this section, we first describe the metrics we use to evaluate the alternative methods for querying the database (Section 3.1). Then, we describe the training and test databases (Section 3.2) that we used for experiments in Section 5, as well as the information extraction systems with which we experiment (Section 3.3) and the various techniques that we compare for document retrieval (Section 3.4). We conclude by describing the two relations that we use in our experiments (Section 3.5).

3.1 Evaluation Methodology and Metrics

Our goal is to retrieve a promising set of documents D from the database, allowing us to extract from these documents a relation R, which should hopefully be a close approximation of the relation R_{all} that would have been extracted had we examined every document in the database. To evaluate our success in this task, we use a number of metrics, each measuring a different aspect of the overall system performance:

Usefulness: Measures the quality of the retrieved document set. The retrieved set D would ideally contain only documents that are useful to the extraction system. Furthermore, we want to extract R in the most efficient way possible, i.e., by retrieving few documents. To quantify how efficient D is for extracting R, we define *Usefulness* as the average number of extracted tuples per document retrieved:

$$Usefulness = \frac{|R|}{|D|} \tag{1}$$

F_{Ideal} : Measures the *absolute* accuracy of R, using the *Ideal* metric presented in [1]. This metric uses a large sample of known correct tuples R_{Ideal} that would be extracted for the target relation by an ideal extraction system. R_{Ideal} is chosen from an external source R_{ext} by selecting the tuples in R_{ext} that actually appear in the database. For example, to create R_{Ideal} for the *Headquarters* relation, we use a large (13,000 tuples), publicly available directory of organizations provided on the "Hoover's Online" website [1] as R_{ext} and select as the R_{Ideal} those organization and location pairs that occur within the same line in some document in the database. Given the relation R_{Ideal} and the extracted relation R, we can define the *Recall* and *Precision* of R with respect to R_{Ideal} more formally:

$$Recall_{Ideal} = \frac{|R \cap R_{Ideal}|}{|R_{Ideal}|} \cdot 100\% \tag{2}$$

$$Precision_{Ideal} = \frac{Correct(R \cap R_{Ideal})}{|R \cap R_{Ideal}|} \cdot 100\% \tag{3}$$

An extracted tuple t is considered *Correct* if there exists a tuple in R_{Ideal} that agrees with t on both the key attribute of t and the remaining attributes. We combine recall and precision into one number, the *F-measure* of R with respect to the R_{Ideal}, for easier analysis of the accuracy of R:

$$F_{Ideal} = \frac{2 \cdot Recall_{Ideal} \cdot Precision_{Ideal}}{Recall_{Ideal} + Precision_{Ideal}} \tag{4}$$

Intersection: Defined analogously to F_{Ideal}, *Intersection* measures how closely R approximates R_{all}, and is similarly defined as the *F-measure* of R, as evaluated with respect to R_{all}.

[1] `http://www.hoovers.com`

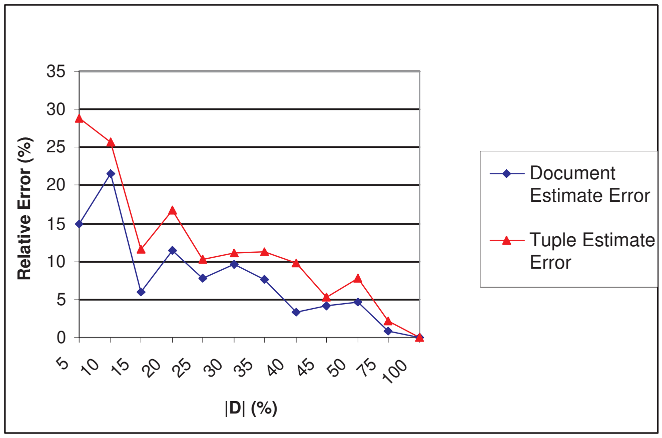

Figure 7: Estimate error (of number of tuples extracted) for varying sample size.

EUsefulness: Estimates *Usefulness* efficiently. Recall that *Usefulness* of a document set D is defined with respect to the extraction system that runs over D and extracts tuple set T. Evaluating the *Usefulness* of sets of documents repeatedly (e.g., during system tuning) may be prohibitively expensive. However, we can define a close approximation that does not require running the extraction system over different sets D as follows. First, we run the extraction system over *all* documents in database D_{all} to establish a set of useful documents U_{all}; for each document $d_i \in U_{all}$, store the number of tuples extracted from d_i as $n(d_i)$. Then, the *estimated* number of tuples T_{est} that would be extracted from an arbitrary subset of documents D in the database is $T_{est} = \sum_{d_i \in D \cap U_{all}} n(d_i)$. Now we can define *EUsefulness* of D as:

$$EUsefulness = \frac{T_{est}}{|D|} \tag{5}$$

Since this is an estimate, we do not expect perfect accuracy. The goal is to provide a quick method for comparing the alternative strategies during system tuning. We ran validation experiments to compare *EUsefulness* to *Usefulness*, and report results in Figure 7.

3.2 Training and Test Databases

Our *training* database, to which we will refer as NEWS, is a subset of 202,000 1996 documents from the North American News Text Corpus, available from LDC [2] and including articles from Los Angeles Times, The Wall Street Journal, and The New York Times.

The *test* database, to which we will refer as TREC, uses the documents from the TIPSTER Complete Collection used in the TREC evaluation task, available from the LDC website [3]. This database of 959,000 documents includes patents, foreign broadcast transcripts, and newspaper articles, comprising the bulk of the TIPSTER collection.

3.3 Underlying Information Extraction Systems

Our main goal is to retrieve a set of documents from which our information extraction system of choice will be able to extract our target relation. Our approach 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.

[2] `http://www.ldc.upenn.edu`
[3] `http://www.ldc.upenn.edu/Catalog/LDC93T3A.html`

For our experiments, we consider two information extraction systems that are attractive because they require minimal training:

- *DIPRE*, as described in [2].
- *Snowball*, an described in [1].

In principle, our document retrieval approach could be used in conjunction with any information extraction system. For example, we could plug in a manually-trained extraction system such as PET [15], which would require labor-intensive retraining for each new target relation. We plan to experiment with such an extraction system in future work.

3.4 Alternative Document Retrieval Methods

We experimentally compare a number of alternatives:

- *QXtract*: The algorithm described in Section 2, with parameter values from the tuning experiments that we report in Section 4 and a summary in Figure 13.
- *Baseline-Random*: A simple baseline technique that returns a random document subset of a given size from the database.
- *Baseline-Seed*: A second baseline technique that uses our seed sampling algorithm of Section 2.2 and returns the document sample D_{sample}. This baseline proceeds as *QXtract* but without the final query generation stage.
- *Patterns*: Extraction patterns exported by the information extraction system are used to generate queries to retrieve a set of promising documents as described in Section 2.3.2. This strategy is expected to produce a good approximation of the promising document set: for a tuple to be extracted from a document, the document must contain the terms in the pattern. However, the extraction patterns from the information extraction system may not always be available, or be amenable to conversion to queries (e.g., if patterns consist of HTML tags).

3.5 Target Relations for Extraction

We evaluate system performance on the extraction of two relations:

- *Headquarters*, as defined in Section 2.1.
- *Executive(organization, officer)*, where a tuple $< o, n >$ is in the relation if n is an executive officer such as the president, CEO, or chair of organization o. The initial set of seed tuples consisted of five CEOs or presidents of companies such as Microsoft and Intel.

4 Tuning our System

We explored the best parameter values for the document retrieval by running the system on the NEWS training database. For tuning, we used *Snowball* as the underlying information extraction system. As we will see in Section 5, the configuration derived with *Snowball* generalizes to work well with *DIPRE*, the other information extraction system that we consider in this paper. We tune the parameters of each document retrieval component individually, in order: **Seed Sampling** (Section 4.1), **Query Training** (Section 4.2), and **Document Retrieval** (Section 4.3).

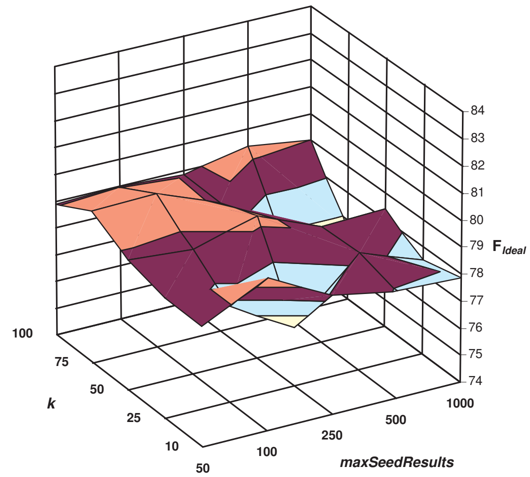

Figure 8: F_{Ideal} of the relations extracted from the NEWS database for varying k and *maxSeedResults*.

4.1 Seed Sampling

The goal of seed sampling is to generate the best possible training set for query generation (Section 2.2). The parameters that control the process are:

- k: Number of extracted tuples to add to the seed set.
- *maxSeedResults*: Limit on number of documents to retrieve for each seed tuple.
- *maxSamplingRounds*: Maximum number of iterations of seed sampling.

Parameters k and *maxSeedResults* control the breadth and size of the sample set coverage. To find a good combination of k and *maxSeedResults*, we try different combinations and retrieve a document sample for each. We train *Snowball* on the retrieved sample to generate extraction patterns, and then, using these patterns, run *Snowball* over all of the documents in the NEWS database to extract the final relation.

The extracted relations are evaluated for absolute accuracy using the F_{Ideal} metric (Figure 8). The F_{Ideal} surface represents the quality of tuples extracted for each combination of k and *maxSeedResults*. The combination of k=75 and *maxSeedResults*=100 produces the highest F_{Ideal} metric; we adopt these values for k and *maxSeedResults* for the remaining experiments.

Having fixed k and *maxSeedResults*, we now vary the *maxSamplingRounds* parameter to determine when to stop the seed sampling process. Figure 9 reports the F_{Ideal} results for the tuples extracted from the NEWS database. We found that most of the useful documents that contribute to extracting tuples are retrieved by round 5 of the sampling. Subsequently, the improvements are incremental and disappear completely by the tenth iteration of the sampling process. Based on these observations, we set the *maxSamplingRounds*=5.

4.2 Query Training

After seed sampling, we use the resulting document sample to generate queries to retrieve additional promising documents (Section 2.3). Thus, the overall goal is to evaluate the quality of the promising document set retrieved by each query generation method, and select the best method for our problem. In this step we will find the best configuration of the query generation component parameters:

- *Representation of Training Examples*: *document* vs. *local*.
- *Query Generation Technique*: Okapi, Ripper, and SVM.

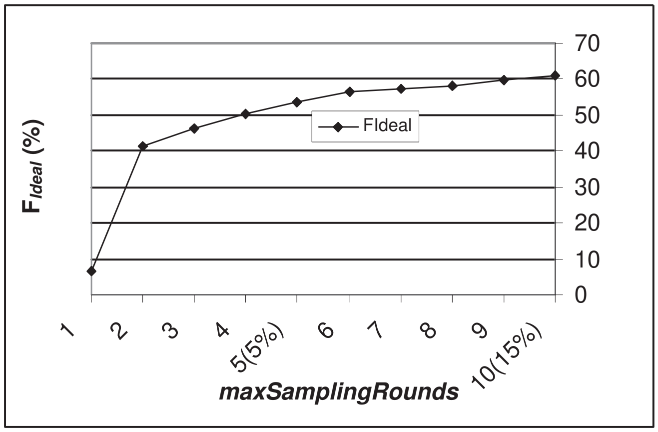

Figure 9: F_{Ideal} of the relations extracted from the NEWS database for varying *maxSamplingRounds*.

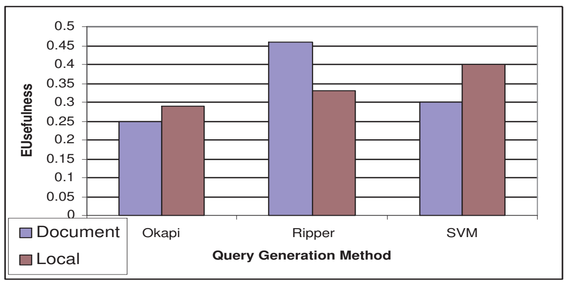

Figure 10: *EUsefulness* of documents retrieved by different classifiers and features for query generation.

We compare the *EUsefulness* of the document sets retrieved by generating queries using Okapi, Ripper, and SVM. Each method was trained using both *local* and *document* representations of training examples. The results on the NEWS database are in Figure 10, which shows that Ripper performs best for the *document* feature set, while SVM performs better using the *local* features. We conjecture that this is due to the simpler model used by SVM (i.e., linear kernel), since the dimensionality of the *local* feature space is likely to be lower than the dimensionality of the *document* feature space.

After tuning, we conclude that the best feature set for SVM and Okapi is *local*, while for Ripper it is *document*. At this point we only know the best representation for each query generation technique. We will know the best technique overall after the next tuning step, when we actually evaluate the usefulness of documents retrieved by each technique for a fixed upper bound on the fraction of the database that we are willing to retrieve.

4.3 Document Retrieval

As discussed in Section 2.4, we need to decide which query generation method produces the best documents for a given bound on the size of D, the set of documents retrieved from the database. This bound indicates the number of documents that we are willing to retrieve during extraction. The parameters for this step are:

- *maxSearchResults*: Limit on number of documents retrieved for each generated query.
- *numQueries*: Number of generated queries to use.

We use the *EUsefulness* metric to evaluate the documents retrieved using the top *numQueries* queries and retrieving at most *maxSearchResults* results per query. Parameter *numQueries* varies from 5 to 200, while *maxSearchResults* varies from 100 to 100,000. The upper bounds on search results depend on the size of the test database. While we

$\frac{\lvert D \rvert}{\lvert D_{all} \rvert} \cdot 100\%$	*SVM*	*Okapi*	*Ripper*
10	10/1%	10/1%	10/1%
25	25/1%	10/2.5%	25/1%
50	50/1%	10/5%	10/5%
75	15/5%	25/2.5%	15/5%

Figure 11: Best *numQueries/maxSearchResults* combination for SVM, Okapi, and Ripper on the NEWS database for different percentages of database D.

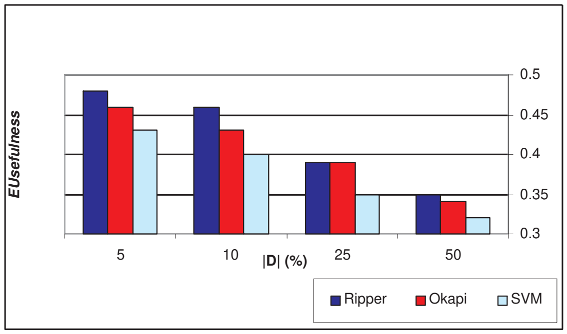

Figure 12: *EUsefulness* of documents retrieved by each query training method as a function of an upper bound on the database fraction retrieved.

do not need to know the *exact* size of the database, we use an approximate number of documents to *scale* the bound on the documents retrieved for each query proportionally.

We bucketize the results for each query generation method by the largest number of documents that could be retrieved using a particular combination of *numQueries* and *maxSearchResults*. For example, we could retrieve up to 50% of the 202,000 NEWS documents either with (*numQueries*=10 and *maxSearchResults*=10,000), or with (*numQueries*=50 and *maxSearchResults*=2,000). We estimate the usefulness of the documents retrieved, and pick the best combination of *numQueries* and *maxSearchResults* for each query generation method for each bucket. The best settings for SVM, Okapi, and Ripper are reported in Figure 11, where the *maxSearchResults* is reported as percentage of the documents in the NEWS database (e.g., 1% of the NEWS database corresponds to 2000 documents).

Using the best combinations of *numQueries* and *maxSearchResults* for SVM, Okapi, and Ripper, we can now compare the *EUsefulness* of documents retrieved by the best configuration of each query generation method. This will determine the best query generation method for a given upper bound on the fraction of the database that we are willing to retrieve. The results are shown in Figure 12. An *EUsefulness* value of 0.45 means that, on average, we should expect to extract a tuple from about every other document retrieved. If we are willing to retrieve 10% of the NEWS database using Ripper as the query generation method, we should expect to extract approximately 9,500 tuples from the retrieved document set. As we can see from Figure 12, Ripper performs the best for all database fractions. Figure 13 summarizes the final configuration of our system parameters.

5 Experimental Results

In this section, we first evaluate our techniques for the *Headquarters* relation on the training database NEWS (Section 5.1). Then, we compare the document retrieval strategies on the completely unseen test database TREC (Section 5.2). Additionally, we report results over the NEWS and TREC databases for a new relation (Section 5.3), to check the generality of our approach.

Parameter	*Value*	*Description*
k	75	Size of new seed
maxSeedResults	100	Maximum results returned for each q^+ or q^-
maxSamplingRounds	5	Seed sampling rounds
Feature Space	*document*	Example representation
Query Generator	Ripper	Query generation method
maxSearchResults	Figure 11	Results retrieved per query
numQueries	Figure 11	Queries to submit

Figure 13: Final configuration of our system used for evaluation on the test database.

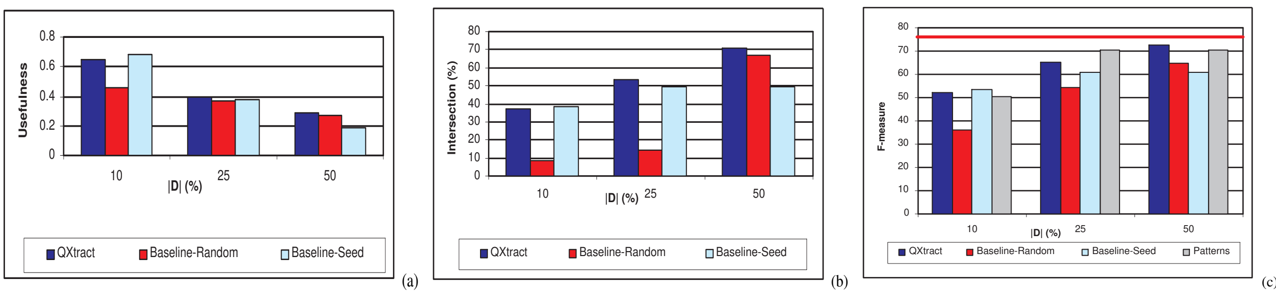

Figure 14: *Usefulness* (a), *Intersection* (b), and F_{Ideal} (c) of *QXtract*, *Baseline-Random*, and *Baseline-Seed*, on the NEWS database using *Snowball* as the extraction system.

5.1 Performance on the Training Database

We evaluate our document retrieval performance against alternative strategies using the configuration defined in Figure 13. Figure 14 reports the various evaluation metrics (Section 3.1) for our *QXtract* system and other retrieval systems (Section 3.4), over the NEWS database that was used for tuning *QXtract* and with *Snowball* as the underlying extraction system.

QXtract is consistently one of the best methods. For example, *QXtract* is able to extract more than 80% of the valid tuples in R_{all} (where the horizontal line in Figure 14(c) represents F_{ideal} of R_{all}), while retrieving only 25% of the documents in the database. Overall, *QXtract* appears as a robust technique with good performance across all database fractions retrieved. For example, at any fraction of documents retrieved, *QXtract*'s documents are as good as a randomly chosen document set of twice the size. (See results for *Baseline-Random*.)

Figure 15 summarizes the performance of the document retrieval system when we use *DIPRE* as the underlying information extraction system. These results are consistent with those for *Snowball*. The document set retrieved by *QXtract* is useful for *DIPRE*: *DIPRE* manages to extract over 70% of R_{all} from only the 25% of the NEWS database retrieved by *QXtract*. Also, *QXtract* manages to extract almost as many of the known *valid* tuples (derived from R_{Ideal}), as are contained in R_{all} (represented by the horizontal line in Figure 15(c)). In other words, not only does *QXtract* give a good approximation of R_{all} while only retrieving 25% of the documents, but also appears to focus in on documents containing *valid* tuples.

5.2 Performance on the TREC Database

QXtract is consistently either the best method or a close second for all metrics.

Usefulness: For example, when up to 10% of the database is retrieved, the *Usefulness* of the set of documents retrieved using *QXtract* is 0.46 (Figure 17(a)). This fraction reflects the fact that *Snowball* extracted 44571 tuples from 95900 documents that *QXtract* retrieved. By comparison, only 22843 tuples were extracted from the document

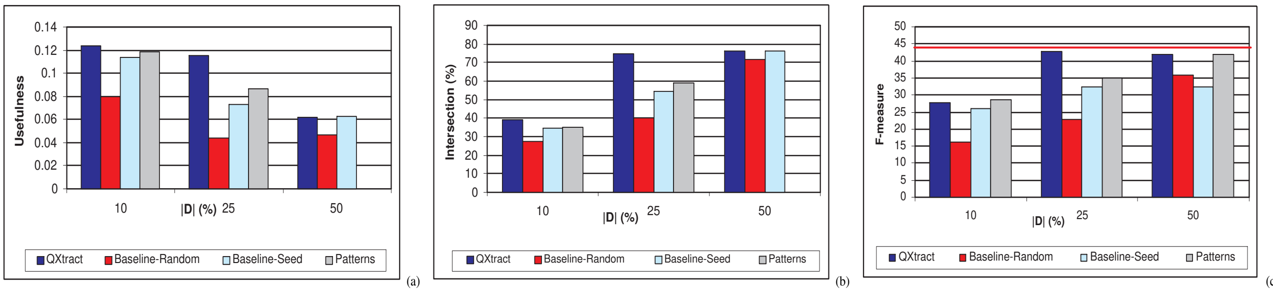

Figure 15: *Usefulness* (a), *Intersection* (b), and F_{Ideal} (c) of *QXtract*, *Baseline-Random*, *Baseline-Seed* and *Patterns* on the NEWS database using *DIPRE* as the extraction system.

set of the same size retrieved by the *Baseline-Random* strategy. On the other hand, *Patterns* was able to produce a set of documents that was almost as useful as the one produced by *QXtract* - with *Usefulness* $= 0.44$, 42157 tuples were extracted. Only 23879 tuples were extracted using the *Baseline-Seed* method.

Intersection: The other aspect of the comparison is how close R approximates R_{all}. On this metric (also for 10% as the upper bound on the fraction of database retrieved), *QXtract* scores 40.99, which is the best among all of the systems compared. *Intersection* was defined as the F-measure of R with respect to R_{all}. The *Intersection* measure of *Patterns* is 39.99, meaning that the relation R extracted from the documents retrieved by *Patterns* is a slightly less accurate approximation of R_{all} than the one produced by *QXtract*. By comparison, the quality of approximation produced by *Baseline-Random* is 22.1, which indicates that an approximation of R_{all} produced by both *QXtract* and *Patterns* is significantly closer to the real R_{all} than that of *Baseline-Random*.

F_{Ideal}: The third and final metric, F_{Ideal}, estimates the accuracy of extracted relation R with respect to a known correct set of tuples R_{Ideal}. The F_{Ideal} measure for *QXtract* (at 10% of database retrieved) is 43.19, which is slightly less than that of *Patterns*, with 45.21. We have seen *Patterns* sometimes performs as well as or slightly better than *QXtract*: *Patterns* specializes in retrieving documents containing text contexts that suggest the occurrence of tuple for the target relation. Unfortunately, as we discussed, *Patterns* is not a generally applicable strategy (it relies on information extraction systems exporting extraction patterns, which might not be always possible). Overall, *QXtract* appears as a robust technique with good performance across different database fractions retrieved. For example, the promising document set created by *QXtract* by retrieving 10% of the database is as good as a randomly chosen document set of more than twice that size as evaluated using the F_{Ideal} metric. (See results for *Baseline-Random*.)

DIPRE: *QXtract* was configured and tuned using *Snowball* as the underlying extraction system on the NEWS database. We demonstrate the generality of our approach by running the same *QXtract* configuration as before, but using *DIPRE* as the underlying information extraction system. Figure 18 summarizes these results, which are consistent with the results for *Snowball*. The promising document set retrieved by *QXtract* is useful for *DIPRE*: as shown in Figure 18(b), *DIPRE* manages to extract over 70% of R_{all} from only the 25% of the TREC database retrieved by *QXtract*. Also, note that *QXtract* allows *DIPRE* to extract most of the *valid* tuples in the target relation by retrieving only 5% of the documents in the database. Interestingly, the relation extracted by *DIPRE* by operating over the smaller document set (25% of the TREC database) retrieved by *QXtract*, has a higher value for the F_{Ideal} metric than the relation that *DIPRE* extracts from examining *all* documents in the TREC database. We attribute this surprising result to the lower level of noise present in the documents retrieved by *QXtract* as compared to all of the documents in the database. Consequently, the $Precision_{Ideal}$ of R produced by *QXtract* is 68.59%, compared to $Precision_{Ideal}$ of R_{all} of 61.33%. More interestingly, $Recall_{Ideal}$ of R is slightly higher than that of R_{all}, 30.8% and 28.98% respectively. Some of the valid tuples produced by *DIPRE* over the *QXtract* documents, but not over all the documents in the database as shown in Figure 16.

In summary, *QXtract* at least *doubles* the efficiency of information extraction: When using either *DIPRE* or

Organization	*Location*	
	QXtract (Correct)	*Scanning*
AETNA INC	HARTFORD	MINNEAPOLIS
MOTOROLA INC	SCHAUMBURG	CHICAGO
NINTENDO OF AMERICA INC	REDMOND	US
XEROX CORPORATION	STAMFORD	US

Figure 16: Some of the tuples correctly extracted by *DIPRE* from *QXtract* retrieved documents but incorrectly extracted when scanning the entire TREC database.

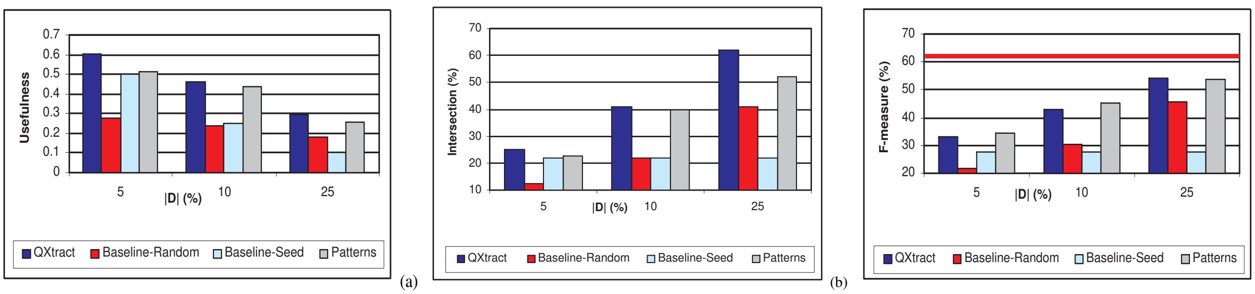

Figure 17: $Usefulness$ (a), $Intersection$ (b), and F_{Ideal} (c) of*QXtract*, *Baseline-Random*, *Baseline-Seed*, and *Patterns* on the TREC database using *Snowball* as the extraction system.

Snowball as the underlying extraction system, the target relation extracted by using *QXtract* is *in all cases* evaluated to be as good on the F_{Ideal} metric as, or better than, the relation extracted from the random sample of the database that is *more than twice as large* as the set of documents retrieved by *QXtract*.

5.3 Results on the *Executive* Relation

To further test the generality of our approach, we evaluated the performance of *QXtract* and the other systems on an additional relation, *Executive*, described in Section 3.5, over the NEWS database. *Snowball* was used as the underlying information extraction system, trained on 5 seed tuples only. Some of the extraction patterns produced by *Snowball* are shown in Figure 19.

The *Usefulness*, *Intersection*, and F_{Ideal} results are shown in Figure 20. *Usefulness* for *QXtract* at size of re-

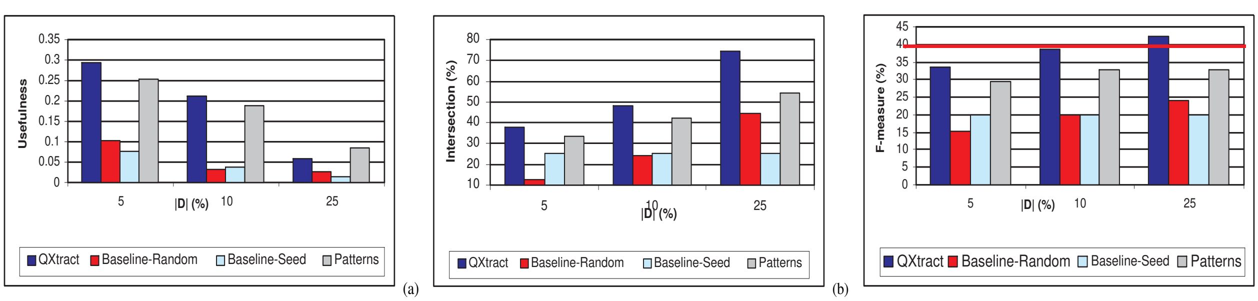

Figure 18: $Usefulness$ (a), $Intersection$ (b), and F_{Ideal} (c) of *QXtract*, *Baseline-Random*, *Baseline-Seed*, and *Patterns* on the TREC database using *DIPRE* as the extraction system.

```
<{}, PERSON, {<``,'' 0.48> <``CHAIRMAN'' 0.48> <``OF 0.48>}, ORGANIZATION, {}>
<{}, PERSON, {<``,'' 0.27> <``EXECUTIVE'' 0.39> <``OF'' 0.39>}, ORGANIZATION, {}>
<{}, PERSON, {<``,'' 0.51> <``HEAD'' 0.07> <``OF'' 0.51>} , ORGANIZATION, {}>
```

Figure 19: Some extraction patterns generated and used by *Snowball* for extracting the *Executives* relation.

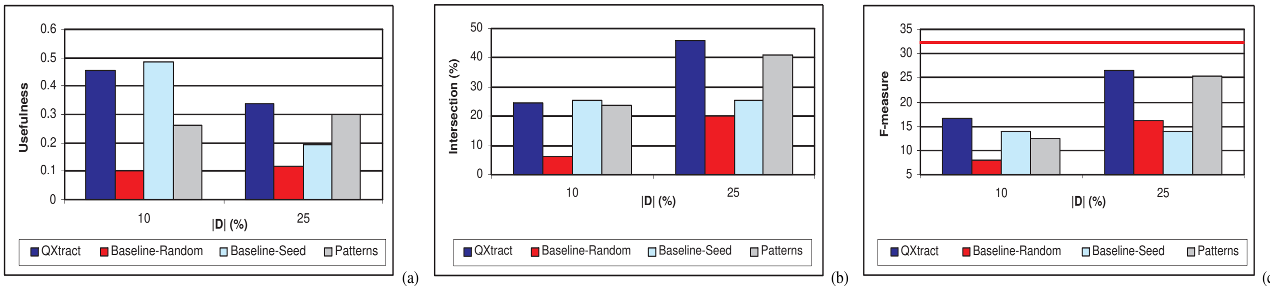

Figure 20: *Usefulness* (a), *Intersection* (b), and F_{Ideal} (c) of *QXtract*, *Baseline-Random*, *Baseline-Seed* and *Patterns* on the NEWS database and *Executive* relation using *Snowball* as the extraction system.

trieved sample 10% is 0.45, while that of *Baseline-Random* is only 0.11, which means that *Snowball* was able to extract more than four times as many tuples from the promising document set produced by *QXtract* than from the document set of the same size produced by *Baseline-Random*. For 10% of the collection, *Baseline-Seed* performs as well as *QXtract* on both the *Intersection* and the *Usefulness* metric. On the other hand, *Patterns* retrieves a significantly less useful set of documents: with *Usefulness* at 0.26, the document sample produced by *Patterns* results in almost half the productivity of tuples per document of *QXtract* or *Baseline-Seed*. Likewise, performance of our technique on the *Intersection* metric is consistently high, following the performance for the original *Headquarters* relation.

Note the generally low performance of all techniques on the F_{Ideal} metric. Unfortunately, extracting this relation is problematic for *Snowball* even when examining the whole database, which results in F_{Ideal} of 33.68. We conjecture that the reason for this is due to a limitation of *Snowball*: *Snowball* can only extract one executive for each organization. Similarly, this limitation prevents *Snowball* from performing well on the *Intersection* metric, since only one executive for each company is kept in the final extracted table.

In Figure 21 we present results for the extraction of the *Executive* relation from the TREC database using *Snowball* as the underlying information extraction system. The performance of *QXtract* on the *Usefulness* metric is high, and the *Intersection* and F_{Ideal} values are consistent with the corresponding values for the NEWS database.

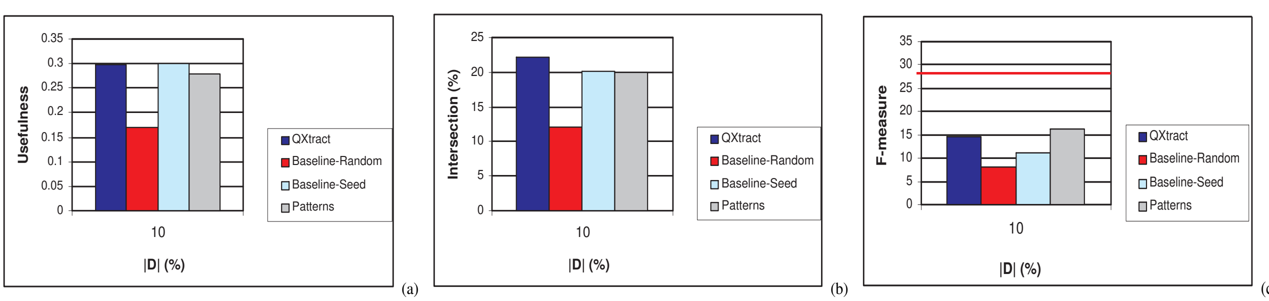

Figure 21: *Usefulness* (a), *Intersection* (b), and F_{Ideal} (c) of *QXtract*, *Baseline-Random*, *Baseline-Seed* and *Patterns* on the TREC database and *Executive* relation using *Snowball* as the extraction system.

6 Conclusions

In this paper, we developed a query-based technique to identify documents that are potentially useful for the extraction of a target relation. This technique allows existing information extraction systems to scale to much larger databases than previously possible: information extraction systems can now focus on the promising documents and not process every document in the database. We demonstrated that our method is general and efficient through a comprehensive experimental evaluation over more than one million real documents. Our new technique could be used to query a standard web search engine, hence providing a building block for efficient information extraction over the web at large.

Acknowledgement

We thank Nicolas Bruno and Amélie Marian for helpful comments on the earlier draft of this paper.

References

[1] 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.

[2] 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.

[3] S. Chakrabarti, M. van den Berg, and B. Dom. Distributed hypertext resource discovery through examples. In *Proceedings of the 25th Conference on Very Large Databases*, pages 375–386, 1999.

[4] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific web resource discovery. In *Proceedings of the 8th World Wide Web Conference*, Toronto, May 1999.

[5] W. W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995.

[6] M. Craven and J. Kumlien. Constructing biological knowledge-bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999.

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

[8] P. G. Ipeirotis, L. Gravano, and M. Sahami. Qprober: A system for automatic classification of hidden-web resources. Technical Report CUCS-010-01, Columbia University, 2001.

[9] T. Joachims. Making large-scale support vector machine learning practical, 1998.

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

[11] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th Conference on Very Large Databases*, pages 129–138, 2001.

[12] S. Robertson. On term selection for query expansion. In *Journal of Documentation*, volume 46, pages 359–364, 1990.

[13] S. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–146F, 1976.

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

[15] 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.