Extracting Relations from Large Text Collections

Yevgeny (Eugene) Agichtein

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2005

ABSTRACT

Extracting Relations from Large Text Collections

Yevgeny (Eugene) Agichtein

Advisor: Professor Luis Gravano

A wealth of information is hidden within unstructured text. Often, this information can be best exploited in structured or relational form, which is well suited for sophisticated query processing, for integration with relational database management systems, and for data mining. This thesis addresses two fundamental problems in extracting relations from large text collections: (1) *portability*: tuning extraction systems for new domains and (2) *scalability*: scaling up information extraction to large collections of documents. To address the first problem, we developed the *Snowball* information extraction system, a domain-independent system that learns to extract relations from unstructured text based on only a handful of user-provided example relation instances. *Snowball* can then be adapted to extract new relations with minimum human effort. *Snowball* improves the extraction accuracy by automatically evaluating the quality of both the acquired extraction patterns and the extracted relation instances. To address the second problem, we developed the *QXtract* system, which learns search engine queries that retrieve the documents that are relevant to a given information extraction system and extraction task. *QXtract* can dramatically improve the efficiency of the information extraction process, and provides a building block for extracting structured information and text data mining from the web at large.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I thank my Advisor, Professor Luis Gravano, for his wisdom, guidance, and infinite patience. No matter how busy, Luis always made time to discuss my research, providing insights, suggestions, and ideas that made all of my graduate work a reality, and spared no effort to help me improve my writing and presentation skills. He helped me develop as a researcher in more ways than I can list here. Luis is truly a great mentor and role model, and I am extremely fortunate to have had him as an advisor.

I would also like to thank my thesis committee: Ralph Grishman, Kathy McKeown, Ken Ross, and Torsten Suel. Professor Grishman of NYU introduced me to NLP research when I was an undergrad, and has been a source of advice and inspiration ever since. Professors Kathy McKeown and Ken Ross were always ready to discuss my research and provide great ideas and suggestions. Professor Torsten Suel's insightful comments improved my thesis significantly.

Large portions of this work are a result of collaboration. Parts of Chapter 3 are based on our work with Eleazar Eskin. The bulk of Chapter 5 is based on our work with Panos Ipeirotis. Many experiments in Chapter 4 were done using Mayer Crystal's Java implementation of *Snowball*. Jeff Pavel, Vika Sokolova, Alex Voskoboynik, and Vijay Sundaram helped develop beautiful demos and prototypes. David Kaufman made possible the evaluation of Chapter 4 by helping me navigate the complexities of human subjects research. Roman Yangarber of NYU provided the *Proteus* system used for some of the experiments in Chapters 4, 5, and 6.

Mark Yeun, Jeff Pavel, and Dennis Shim of CRF saved me from my own stupidity more

times than I care to remember. For many years, Tobias Hollerer was an ideal officemate from the time he helped me get acquainted with the department when I arrived. Nicolas Bruno, Panos Ipeirotis, Wisam Dakka, and Amelie Marian were always ready to chat and, sometimes, commiserate. Noemie Elhadad, Dave Evans, Sasha Blair-Goldensohn and Elena Filatova and other NLP-ers made 7th floor CEPSR a friendlier place. Remiko Moss, Twinkle Edwards, and Patricia Hervey went out of their way to help me with Columbia requirements to make the administrative side of things almost painless.

My friends and family keep me relatively sane. Mama and Papa – your unwavering support kept me from giving up. This is one is for you. Misha –the best Big Brother one could ask for– thank you for always being there for me in every way. Most importantly, I would not survive the last years of grad school, and beyond, without the tremendous support and understanding of my dearest wife Shoshana. And life would not be the same without our daughter Shiraz Abigail's ready smile.

To Mama and Papa.

Chapter 1

Introduction and Overview

Text documents convey valuable *structured* information. For example, the medical literature contains information about new treatments for diseases. Similarly, news archives contain information useful to analysts tracking financial transactions, or to government agencies that monitor infectious disease outbreaks. All this information could be managed and queried more easily if represented in a structured form. This task is typically called *information extraction*, and is the subject of this thesis.

In general, information extraction refers to automatic methods for creating a structured representation of selected information drawn from natural language text. More specifically, information extraction systems can identify particular types of entities (such as drug names) and relationships between entities (such as adverse interactions between medical drugs) in natural language text for storage and retrieval in a structured database [Gri97]. Once created, the database can be used to answer specific questions quickly and precisely by retrieving answers instead of complete documents, for sophisticated query processing, for integration with relational databases, and for traditional data mining tasks.

This thesis focuses on the extraction of such structured relations from large text collections. As an example, consider extracting the *CompanyHeadquarters(Organization:ORGANIZATION, Location:LOCATION)* relation, which contains a tuple $\langle o, l \rangle$ if organization o has headquarters in location l. More precisely, the relation *CompanyHeadquarters* rep-

resents the specific semantic relationship between companies and their locations, while an *instance* of *CompanyHeadquarters* contains a set of tuples (i.e., pairs) of individual company and location entities related as specified above [RG03]. For brevity, we will use the term "relation" to refer to both, unless the meaning is not clear from the context.

Figure 1.1 shows the basic stages in the extraction of a tuple from a document fragment. As one of the first stages of extraction, the input documents are typically passed through

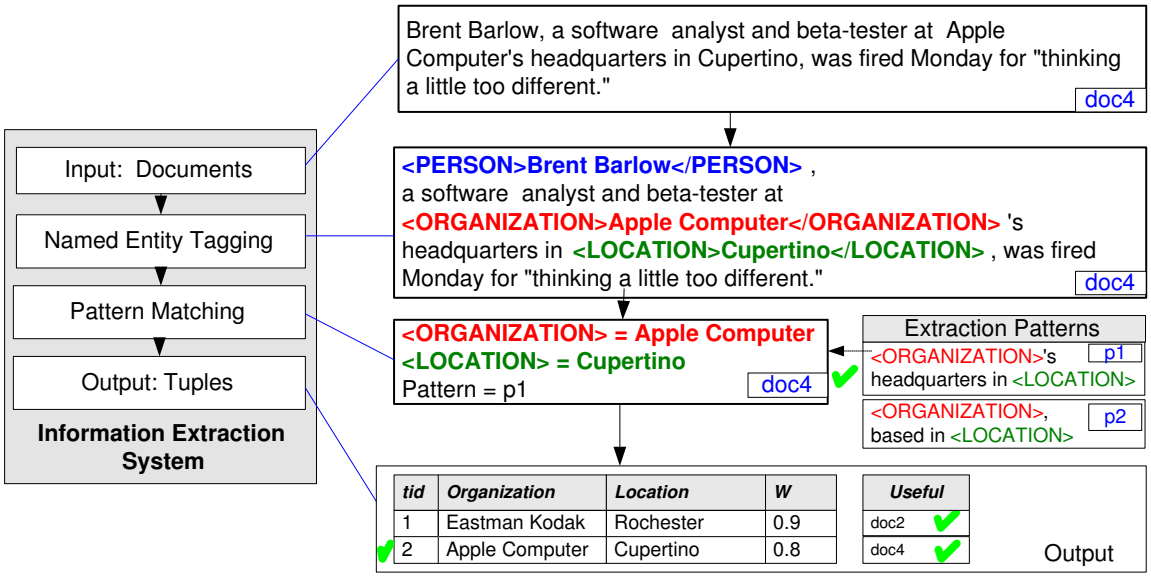


Figure 1.1: Extracting tuples for the *CompanyHeadquarters* relation.

a *named-entity tagger*, which is able to recognize entities such as organizations, locations, and persons. Named-entity tagging is a well studied problem, with tools publicly available for the most common entity types [DAH+97, BC]. 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 [YG98], automatically learned [YGTH00, AG00], 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. Our description omits 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 [Gri97] for an in-depth discussion.)

In the example in Figure 1.1, 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 1.1, 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.

A fundamental problem in information extraction is how to train an extraction system for an extraction task of interest. Traditionally, this training required substantial human effort and hence the development of information extraction systems was generally expensive and time consuming. An attractive approach to reduce the training cost, pioneered by Brin [Bri98], is to start with just a handful of "seed" tuples for the relation of interest, and automatically discover extraction patterns for the task. These patterns, in turn, help discover new tuples for the relation, which could be used as new seed tuples for a next iteration of the process. This could, potentially, reduce the effort needed to adapt an information extraction system for a new target relation to simply providing the initial seed tuples for the relation.

In practice, however, this bootstrapping approach requires solving a number of challenging and fundamental problems: how to define, represent, create, and evaluate extraction patterns; how to distinguish between valid and invalid tuples proposed by the system; and how to set the operating parameters (e.g., how closely a pattern must match a text fragment in order to generate a new tuple). Also, it is important to effectively evaluate the quality of the resulting (large) relations as well as understand the practical limitations of this general approach. In Chapters 3 and 4, we will discuss these problems in depth and propose our solutions, embodied in our *Snowball* information extraction system.

After training an information extraction system, running it over a large document collection is challenging, because the process of extracting structured information from text documents is usually computationally expensive. So it may not be feasible to process every document in a large collection to extract a relation of interest. Interestingly, processing every document in a collection is often not necessary, given that typically many documents do not contribute to the relation to be extracted. Therefore, identifying –via querying– the potentially useful collection documents to process becomes an efficient alternative to processing every document. Furthermore, some valuable document collections are only accessible via querying. For example, PubMed, a widely-used reference for medical professionals, is only accessible through a web query interface; therefore, to extract, say, a relation connecting diseases to their recommended treatments from PubMed, we must resort to querying and retrieving documents that contain the relevant information. Hence, for both efficiency and accessibility, query-based access to text databases is crucial for information extraction over (large) collections of documents. In Chapters 5 and 6, we present our *QXtract* system for efficient, query-based information extraction, as well as show how to analyze and model a large family of query-based techniques.

Overall, in this thesis we address two fundamental problems for extracting relations from large text collections: (1) adapting the extraction system to new domains and collections, and (2) scaling up information extraction to large collections of documents. Our main contributions in this thesis include:

- **Techniques for domain-independent, partially supervised information extraction**: We present and thoroughly evaluate *Snowball*, a minimally-supervised information extraction system for extracting structured relations from large text collections (Chapter 3). To achieve this goal, we develop: a novel strategy for defining and representing extraction patterns so that we capture many of the valid tuples in a text collection; algorithms for evaluating extraction patterns and the discovered tuples, which allow *Snowball* to assess the likelihood that each extracted tuple is valid; and a scalable evaluation methodology and metrics to evaluate *Snowball* over large document collections.

- **Methods for adapting information extraction systems to new domains**: An

important problem with a bootstrapping system such as *Snowball* is how to set the (often domain-dependent) operating parameters to produce reasonable results. In Chapter 4, we present techniques for *automatically* estimating important *Snowball* parameters, thus further facilitating porting of *Snowball* to new domains. We evaluate our techniques for diverse relations such as corporate acquisitions, disease outbreaks, and reported side effects of prescription drugs.

- **Techniques for modeling query-based access to text databases**: Understanding how to retrieve documents for information extraction via querying is central to scaling up information extraction to large document collections. We present a general model that can help understand and analyze an important class of query-based strategies (Chapter 5).

- **A robust and scalable query-based architecture for information extraction**: Finally, we present a domain-independent and general architecture for improving the efficiency of information extraction systems (Chapter 6). Our query-based system, *QXtract*, expects the document collection to support only a simple query interface, and is independent of the choice of the information extraction system. Furthermore, our method could be used to query a standard web search engine, hence providing the infrastructure for efficient information extraction from the web at large.

This thesis is organized as follows: In Chapter 2, we review related work. Then, in Chapter 3, we present *Snowball*, our partially supervised information extraction system. Later, in Chapter 4, we present techniques for adapting *Snowball* to new domains and collections of interest, as well as thoroughly evaluate *Snowball* over three substantially different domains and tasks. We then shift our focus to how to scale information extraction to large collections. Specifically, in Chapter 5, we present a general model for analyzing an important family of query-based document retrieval strategies for information extraction. Then, in Chapter 6, we present *QXtract*, a robust and domain-independent query-based system for scaling up information extraction. Finally, in Chapter 7, we conclude this thesis and describe some interesting open questions and future research directions.

Chapter 2

Background and Related Work

We can identify two major problems related to information extraction from large document collections: (1) learning extraction patterns for text, for extraction accuracy, and (2) retrieving the relevant documents for an extraction task from the collection, for extraction efficiency. Therefore, our work touches on two major research areas: information extraction and information retrieval. This chapter reviews related work in these two areas, as well as in the related area of question answering. First, we summarize the research on information extraction, emphasizing learning-based techniques that are most closely related to our work (Section 2.1). Then, we review recent work on question answering and discuss its connection to information extraction (Section 2.2). Finally, we summarize relevant work on information retrieval, and comment on its connection to our query-based extraction method of Chapter 6 (Section 2.3).

2.1 Information Extraction

Information extraction has long been a rich subject of research. An information extraction system typically examines every document in the collection and attempts to extract information for a given target relation by filling each attribute of the relation from the text of the document. The extraction task itself is usually subdivided into the named entity (NE) recognition task, and the scenario template (ST) filling or relation extraction task. In this thesis, we focus on the relation extraction task.

Starting from the 1980s, information extraction research has been largely encouraged, and driven, by the series of seven Message Understanding Conferences (MUCs). The MUC extraction tasks ranged from parsing naval dispatches [Rep87], to terrorist attacks in Latin America [Rep91], to joint ventures and company acquisitions [Rep93] as well as corporate management successions [Rep98]. Over the years, the emphasis of the MUC evaluation shifted to the rapid development of information extraction systems, where the participating teams were given four to six weeks to tune their system for a new scenario. The approaches for adapting information extraction systems evolved along two different paths: sophisticated language-engineering environment systems and learning-based systems.

The language-engineering approach has stood the test of time as the resulting systems performed best in the MUC relation extraction tasks. These systems can be viewed as a development environment for constructing, manipulating, testing, and combining rules, dictionaries, and extraction patterns for a variety of information extraction tasks. With enough domain knowledge, resources, and expert tuning, this approach can result in systems with high accuracy for the specified domain and document collection. We review some representative systems in Section 2.1.1. When such resources are not available, or a duly exhaustive analysis is not feasible, or if a previously tuned system is applied to a new document collection, these systems can suffer from low coverage and can only find a small fraction of the possible valid tuples.

In contrast, the learning-based systems attempt to discover the rules or extraction patterns from the text itself. This approach can require less tuning effort for each new task than the language-engineering approach does, but tends to result in lower accuracy as a result. However, as the amount of labeled and unlabeled text available on-line increases, the learning-based family of information extraction techniques become more competitive. We review the most relevant learning-based information extraction techniques in Section 2.1.2.

2.1.1 Manually Tuned Information Extraction Systems

One of the major challenges in information extraction is the necessary amount of expert tuning or programming involved in training a system for each new task. This challenge has been addressed in different ways. One approach is to build a powerful and intuitive

graphical user interface for training the system, so that domain experts can quickly adapt the system for each new task [YG98].

A prominent example of such a system is *Proteus* [YG98], developed at the New York University. *Proteus* allows a system user (typically a sophisticated domain expert or developer) to create, generalize, and test extraction patterns based on the manual examination of example text documents. One way of interacting with *Proteus* is for a user to provide example sentences or text fragments to the system, which then attempts to generalize the examples to create a new extraction pattern or to expand an existing pattern, while consulting with the user of the system throughout the process. Another example of a language-engineering environment is *GATE* [CMBT02], a system developed at the University of Sheffield. *GATE* provides generic components that help easily design, develop, and evaluate information extraction patterns for a given extraction task. *GATE* is an open system that allows plugging in and extending components for each stage of the extraction process. The emphasis is on flexibility and generality, and the burden is on the developer to adapt the individual components for each new task.

2.1.2 Learning-Based Information Extraction Systems

A promising research direction is to automatically train the information extraction systems and generate rules or extraction patterns for new tasks. With large amounts of text (annotated and otherwise) electronically available, the accuracy of the machine learning-based systems can rival that of manually-engineered systems. The machine learning approaches can be either *supervised* or, alternatively, *unsupervised* (or *partially supervised*). First, in Section 2.1.2.1, we review the supervised techniques and then, in Section 2.1.2.2, we review the unsupervised and partially supervised techniques, which are most closely related to our work.

2.1.2.1 Supervised Information Extraction

The general approach used by many supervised extraction systems is to train the system over a large *manually tagged* corpus, where the system can apply machine learning techniques to generate extraction patterns. Examples of such systems include the CRYSTAL

system [FSM+95], BWI [KWD97], and Rapier [CM98]. A drawback of this approach is the need for a large tagged corpus, which involves a significant amount of manual labor to create. To reduce the amount of required annotation, the AutoSlug system [Ril96] generates extraction patterns automatically by using a training corpus of documents labeled as either relevant or irrelevant for the topic. This requires less manual labor than annotating the documents, but nevertheless the effort involved is substantial. [CDF+99] describes machine learning techniques for creating a knowledge base from the web. Their approach requires training over a large set of labeled web pages, with relevant document segments manually labeled, as well as a large training set of page-to-page relations.

2.1.2.2 Unsupervised and Partially Supervised Information Extraction

With virtually infinite amounts of unlabeled text accessible on the web and other sources, learning methods that can exploit unannotated text become particularly interesting and valuable. Hence, bootstrapping techniques that start with a few labeled examples and exploit large amounts of unlabeled data are attractive.

For word-sense disambiguation, [Yar95] demonstrated a bootstrapping technique that starts with a small set of seed collocations for each word (e.g., seed collocation "life" to disambiguate the biological sense of the noun "plant"), and iteratively classifies the occurrences of the word into one of the appropriate senses. [CS99] used bootstrapping to classify named entities in text exploiting two orthogonal features: the spelling of the entity itself (e.g., having a suffix "Corp.") and the context in which the entity occurs. [RJ99] also presented a bootstrapping technique to recognize and classify named entities in text. Similarly, *KnowItAll* [ECD+04] learns instances of a class (e.g., a set of cities) by starting with a generic pattern (e.g., "is a"), which is expanded by iteratively finding additional instances on the web.

A bootstrapping approach for extracting relations has been used by *DIPRE* [Bri98]. *DIPRE* starts with a few user-provided examples, or *seed tuples*, for the target relation. To identify new tuples, the named entities of interest must be surrounded by *exactly* the same contexts as the seed tuples. While this requirement may not be problematic for some relations on the web (e.g., by matching the markup of HTML tables listing books and their

authors), *DIPRE* tends to miss many valid tuples for some relations when operating over plain-text documents. Our *Snowball* system [AG00], which builds on some of *DIPRE*'s ideas, addresses this problem by allowing for a more flexible pattern representation. Furthermore, *Snowball* introduces a method for automatically filtering the extracted tuples based on estimates of the correctness of each extracted tuple. As we will see, our *Snowball* system does not require extensive annotation or tuning for new extraction tasks. *Snowball* is designed to be flexible about variations inherent in natural language text and can in some cases discover a vast majority of the tuples in a collection while starting with just a few example tuples. (We will describe *Snowball* in detail in Chapter 3.) A similar approach was recently presented in [HSG04] for automatically clustering and classifying co-occurring pairs of entities, with no need for example tuples. [YS99] describes an extension of *DIPRE* to mining the web for acronyms and their expansions. [BM98] presents a methodology and theoretical framework for combining unlabeled examples with labeled examples to boost the performance of a learning algorithm for classifying web pages. [YGTH00] presents a bootstrapping-based method that starts with a few user-supplied extraction patterns and proceeds to automatically include new patterns discovered in the collection by labeling documents as relevant or non-relevant for the task.

2.2 Question Answering

There is a large body of research on question-answering, the task of returning short answers to natural language questions. For example, for the question "Where are the headquarters of Microsoft Corporation located?" a question-answering system might return the answer "Redmond, WA." Many state-of-the-art question-answering systems were recently represented in the Text Retrieval Evaluation Conference (TREC) Question-Answering track [Voo99], which involves retrieving a 50 or 250 byte snippet containing the answer (and most recently, the exact answer) to a set of test questions. Question answering presents interesting connections with information extraction, which we outline in this section.

A number of systems aim to extract answers from documents. For example, Abney et al. [ACS00] describe a system in which documents returned by the SMART information

retrieval system are processed to extract answers. Questions are classified into one of known "question types" that suggest the type of entity corresponding to the answer. The retrieved documents are tagged to recognize entities, and passages surrounding entities of the correct type for a given question are ranked to select the most likely answers. Ravichandran et al. [RH02] also present a system that learns patterns for extracting the answers from the retrieved documents. Moldovan et al., and Aliod et al. [MHP+99, ABH98] present systems that re-rank and postprocess the results of regular information retrieval systems with the goal of returning the best passages. Cardie et al. [CNPB00] describe a system that combines statistical and linguistic knowledge for question answering and employs sophisticated linguistic filters to postprocess the retrieved documents and extract the most promising passages to answer a question.

Other systems attempt to modify queries in order to improve the chance of retrieving answers (e.g., [LG98, ALG01, ALG04, RQZ+01, HPM00]). Additionally, the AskMSR system [BDB02] used manually-crafted question-to-query transformations to focus on promising documents. Recently, [RQZ+01] presented a method for automatically weighting query reformulation operators (e.g., insertions and deletions of terms in the original question), with the goal of transforming a natural language question into one "best" search engine query. Another system, *Mulder* [KEW01], also transforms questions into queries and extracts answers from the returned documents. *Mulder* uses a semi-automatically constructed taxonomy of questions in order to match a new question to a known question type, and to use pre-defined transformations to convert the question into a search engine query.

The main difference of these systems from our work on relation extraction is that question answering systems treat each question in isolation. In our scenario, we want to extract a complete *relation* (i.e., *all* valid tuples). 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., [ALG04, KEW01]), the problem of retrieving documents that collectively contain the complete relation has largely remained unaddressed in the question-answering literature. The work in the question answering community most closely related to our work is reported in [FHE03]. The authors consider one question type,

"Who is $\langle X \rangle$", and build a table with pairs of person names and corresponding descriptions using hand-crafted extraction patterns, with tuples subsequently filtered by a classifier. The system was specifically tailored for that question type, while our work is more general and domain-independent. Nevertheless, [FHE03] confirms our claim that extracting an entire relation at once and then using it to answer questions can be computationally more efficient than processing each individual question in isolation. Furthermore, we will show that by leveraging the redundancy and stylistic conventions often present in large document collections we can also improve the accuracy and coverage of the extracted relation.

2.3 Information Retrieval and Web Search

So far, we discussed work on learning and defining extraction patterns from text. We turn now to the complementary problem of identifying the documents worth processing for a given extraction task. This problem is motivated by efficiency, as discussed above: processing every document in a collection is often prohibitively expensive and, interestingly, unnecessary. The problem of retrieving "relevant" documents, for some definition of "relevance", from large text collections has been studied for decades in the information retrieval field. In this section, we discuss the relationship between this work and *QXtract*, our query-based information extraction system, which we introduce in Chapter 6.

Our *QXtract* work is related to recent research on focused web crawling (e.g., [CPS02]), 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 [RGM01] 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 [Rep98]. Documents can be filtered at various stages of the extraction process [LT92]. Some systems [CWG+92] 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 Chapter 6, we evaluate a related strategy that uses queries derived from extraction patterns. In contrast to these techniques, our *QXtract* system *automatically* generates standard search-engine *queries* (and not more general *filters* that cannot be processed by current search engines) that would retrieve only the relevant documents for an extraction task.

The evaluation presented in [GR97] uses ideas related to our *QXtract* 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, [KGC+00] uses a combination of focused crawling and manually generated queries to retrieve promising documents. In our *QXtract* experiments, 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 [Sal89]. Although our problem is different in nature, we exploit state-of-the-art term weighting and query expansion results [Rob90] from IR in the design of one variant of *QXtract*. Alternatively, the characterization of the useful documents could be viewed as a traditional classification problem. We explore a number of machine learning techniques [Coh95, Joa98] in the design of other variants of *QXtract*.

Several techniques use supervised learning to devise queries that match documents about a specific category of interest [IGS03]. [CS96] 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. Recently, Ghani et al. [GJM01] presented a technique that is similar in spirit to our *QXtract* 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 [GJM01], as well as other query generation strategies that exploit machine learning results. In the research area of specialized search engines, [FGLG02] present a method for learning topic-specific query modifications by starting with a set of example documents on the topic of interest, and training a classifier to recognize such documents. Then, salient features are selected from the example documents and tested on a search engine by retrieving documents using the candidate queries.

Chapter 3

Extracting Relations from Text

Text documents often hide valuable *structured data*. For example, a collection of newspaper articles might contain information on the *location* of the headquarters of a number of *organizations*. If we need to find the location of the headquarters of, say, Microsoft, we could try and use traditional information-retrieval techniques for finding documents that contain the answer to our query [Sal89]. Alternatively, we could answer such a query more precisely if we somehow had available a *table* listing all the organization-location pairs that are mentioned in our document collection. A *tuple* $\langle o,\ell \rangle$ in such a table would indicate that the headquarters of organization o are in location ℓ, and that this information was present in a document in our collection. Tuple $\langle$*Microsoft, Redmond*$\rangle$ in our table would then provide the answer to our query. In general, the web contains millions of pages whose text hides data that would be best exploited in structured form. If we could build structured tables from the information hidden in unstructured text, then we would be able to run more complex queries and analysis over these tables, and report precise results. In this chapter, we present the *Snowball* system for extracting structured data from unstructured text documents with *minimal human participation*.

Extracting structured information from text has long been a rich subject of research. An extraction system typically examines every document in a collection and attempts to extract a tuple for a target relation by filling each relation attribute with an extracted string. As outlined in [Gri97], information extraction systems typically use *extraction patterns* (e.g., rules, regular expressions or other mechanisms, which we reviewed in Chapter 2) to identify

strings from a document and assign them to the appropriate attributes. In this chapter, we present techniques for automatically learning extraction patterns and automatically assigning a *confidence* score to the extracted tuples. Our main contributions in this chapter include:

- **Techniques for generating extraction patterns and extracting tuples:** We develop a novel strategy for defining and representing patterns that is at the same time flexible, so that we capture most of the tuples that are hidden in a text collection, and selective, so that we extract few invalid tuples (Section 3.2).

- **Strategies for evaluating patterns and tuples:** Information extraction is an error-prone process, and some of the extracted tuples are likely to be invalid. A crucial feature of an information extraction system such as *Snowball* is being able to automatically assign a *confidence* to each extracted tuple to indicate the likelihood that the tuple is valid; similarly, assigning confidence scores to extraction patterns is also used to discard unreliable patterns and favor robust ones. We develop strategies for estimating the confidence in extracted patterns and tuples (Section 3.3).

- **Evaluation methodology and metrics:** Evaluating information extraction systems like *Snowball* is challenging: these systems are designed to work over large document collections, so manually inspecting all documents to build the "perfect" table that should be extracted is just not feasible. We introduce a scalable evaluation methodology and associated metrics (Section 3.4), which we use later for large-scale experiments.

We now state the problem we are addressing in more detail, and outline our general approach. The bulk of this chapter has been published as [AG00].

3.1 The Information Extraction Problem

As discussed above, text documents often embed valuable structured information. Our goal is to learn to extract this information with minimal human effort. Specifically, given the definition of a *relation* $R(A_1, \ldots, A_n)$ with attributes $A_1, \ldots, A_n$ and a collection of

text documents, our goal in this chapter is to learn to extract from the documents valid *tuples* with n attribute values $a_1, \ldots, a_n$, matching $A_1, \ldots, A_n$, to "populate" R. While –in principle– we could allow for relation attributes to have arbitrary text values, in this thesis we restrict our attention to the important case where the attribute values are *named entities* (e.g., a company name, a location name, or a name of a drug or a disease). These entities can be identified in a document by a *named entity tagger* (e.g., [DAH+97, BC]), which assigns a predefined *entity type* to each entity (e.g., all recognized "company" entities will be assigned the "ORGANIZATION" type).

A relation definition captures the desired relationship between the entities within each tuple. For example, we can define the relation *CompanyHeadquarters(Organization:ORGANIZATION, Location:LOCATION)*, where a tuple $t = \langle o, l \rangle$ means that company o has headquarters in location l. In addition to the "meaning" of each tuple, a relation definition includes the types of the attributes of the relation (i.e., types of named entities) and *integrity constraints* on the attributes. For example, we may define a *functional constraint* for *CompanyHeadquarters* to indicate that each company o has a unique location of headquarters l.

To determine if a tuple is a valid instance of a relation, we need to examine the documents –and associated *text contexts*– where the tuple occurs. For example, a text fragment "The headquarters of Google are in Mountain View" clearly supports the validity of tuple $t = \langle$*Google, Mountain View*$\rangle$ for the relation *CompanyHeadquarters*. Unfortunately, different documents in the collection may contain contradictory, outdated, or just plain inaccurate information. However, in this thesis we will consider a tuple to be "valid" if the relationship that it represents is explicitly stated in at least one document in the collection.

We now state the problem that we will address in this chapter:

Problem 1 *Given a relation $R(A_1, \ldots, A_n)$ and a document collection D, with* minimal human participation *extract all valid tuples for R from D, using the text contexts associated with each tuple to estimate its validity.*

Intuitively, a valid tuple will be supported by at least one "valid" text context. The problem then reduces to determining how to recognize valid text contexts for a given rela-

tion. A common solution is to manually create extraction patterns to explicitly list exact phrases, words, or regular expressions that indicate the desired relationship. Unfortunately, specifying such patterns by hand is difficult, time consuming, and does not scale to multiple relations, collections with different formatting styles, and domain-specific language variations. In contrast, we adapt an alternative approach that requires as input a handful of *seed* tuples for a relation of interest. Our extraction process uses these example tuples to "learn" how to recognize valid text contexts and extract the rest of the (valid) instances of the relation.

Note that the problem that we solve is somewhat different from that of traditional information extraction (e.g., [Gri97]), where the goal is to extract every instance of a tuple t in a relation R from every document in which t occurs. In contrast, our goal is to construct the relation R with one instance of each tuple t, each instance successfully extracted from at least one of the documents in which it occurs. This difference motivates our approach (described next), as well as the metrics that we use for evaluating the success of the relation extraction process.

In order to make the problem tractable, we make the following assumptions. While we introduce some limitations into the types of relations that we can support, the assumptions below still allow us to extract a wide range of relations with minimal human participation. In Section 3.7, we will discuss some future directions for relaxing these assumptions and restrictions.

1. We only attempt to capture information that is explicitly present in a given document collection; this collection is then our "closed world," and we do not rely on any additional external "context."

2. As mentioned above, the attributes of the relations are named entities, which we assume we can recognize using state-of-the-art named-entity taggers such as *Alembic* [DAH+97] or *LingPipe* [BC]. While this assumption excludes some interesting relations (e.g., book authors and titles), it captures a large family of natural and interesting relations, as we will see.

3. The entities for attributes of a tuple for R all appear within the same document

and, furthermore, within a localized, contiguous fragment of text. In other words, in this thesis we focus on "simple," "lexical" relationships that do not require analyzing relatively long text segments or multiple documents.

4. The document collection is consistent, in the sense that it does not contain contradictory or incorrect information. Resolving conflicts between contradictory tuple values (e.g., when two tuples occur in equally "valid" text contexts) is beyond the scope of this thesis.

The assumptions above allow us to build a general-purpose relation extraction system that requires minimal human participation, as we discuss next.

3.2 The *Snowball* System

We now present our *Snowball* system, which extracts a relation from text by starting with just a handful of example tuples for the relation. The main idea behind *Snowball* is that there is enough regularity in natural-language text that, by starting with a few initial "seed" tuples, we can discover the rest of the tuples by analyzing the context where the seed tuples occur. If the document collection is large, these seed tuples will occur in sufficiently many contexts for *Snowball* to derive *extraction patterns*, which in turn can let *Snowball* extract new tuples and iterate the process.

Our approach builds on the general idea of *Dual Pattern Tuple Expansion* (*DIPRE*) introduced by Brin [Bri98]. *Snowball* is a substantial generalization and extension of *DIPRE* as we will describe in this chapter.

The general structure of the *Snowball* system is shown in Figure 3.1. The process is initialized with a small number of examples of the tuples in the target relation. For example, if the target relation is *CompanyHeadquarters*, this process might start with just the handful of example tuples that are shown in Table 3.1. Starting with the seed tuples, *Snowball* finds occurrences of these tuples in the text collection. Then *Snowball* derives extraction patterns that match the text contexts in which the example tuples occur. The extraction patterns are used to extract new tuples, which are again used as seed tuples, and the process repeats.

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

Table 3.1: User-provided example tuples for *Snowball*.

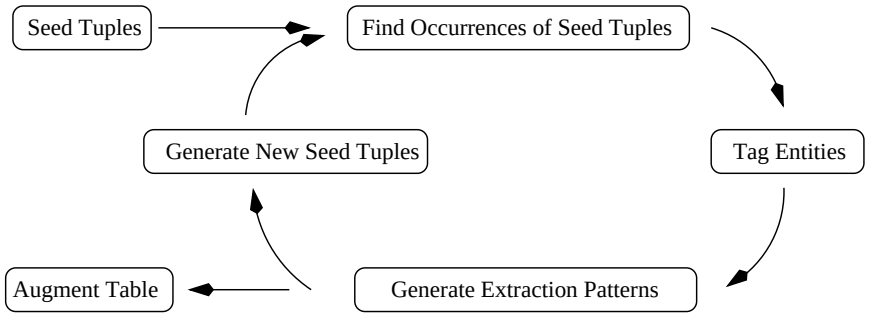


Figure 3.1: The main components of *Snowball*.

3.2.1 Generating Extraction Patterns

A crucial task in the relation extraction process is the generation of *extraction patterns* to find new tuples in text documents. For example, a pattern ⟨*LOCATION*⟩*-based* ⟨*ORGANIZATION*⟩[1] for the *CompanyHeadquarters* relation would extract a tuple ⟨Google, Mountain View⟩ from a sentence fragment "Mountain View-based Google announced today..." Ideally, we would like patterns both to be *selective*, so that they do not generate incorrect tuples, and to have high *coverage*, so that they identify many new tuples. In this section, we study different ways of generating such patterns from a set of seed tuples and a large unlabeled document collection.

The first step of the *Snowball* pattern generation step is recognizing in the documents the named entities that will make up the attribute values, because *Snowball* includes the

[1] As we will see, this is not strictly a *Snowball* pattern; however, we will use it in our initial discussion for simplicity, instead of using the somewhat more involved *Snowball* patterns that we define below.

named-entity tags in its extraction patterns. For example, the pattern ⟨*LOCATION*⟩-*based* ⟨*ORGANIZATION*⟩ will not match any pair of strings connected by "-based"; instead, ⟨*LOCATION*⟩ will only match a string identified by a tagger as an entity of type *LOCATION*, and, similarly, ⟨*ORGANIZATION*⟩ will only match a string identified by a tagger as an entity of type *ORGANIZATION*. To illustrate the impact of using named-entity tags in an extraction pattern, consider a hypothetical extraction pattern ⟨*STRING2*⟩-*based* ⟨*STRING1*⟩ for the *CompanyHeadquarters* relation. This pattern would match the text surrounding correct *ORGANIZATION-LOCATION* tuples (e.g., "Armonk-based IBM has introduced..."). Unfortunately, this pattern would also match any strings connected by "-based," like "*computer-based learning*" or "*alcohol-based solvents.*" This might result in the inclusion of invalid tuples ⟨*learning, computer*⟩ and ⟨*solvents, alcohol*⟩ in the resulting table. When used to generate patterns, these tuples may in turn result in wrong patterns. By using the version of the same pattern that involves named-entity tags, ⟨*LOCATION*⟩-*based* ⟨*ORGANIZATION*⟩, we have a better chance of avoiding this kind of spurious matches.

To recognize named entities, *Snowball* can use any named-entity tagger (e.g., *Alembic*, *LingPipe*, etc.). In addition to *ORGANIZATION* and *LOCATION* entities, these taggers can identify *PERSON*, *TIME*, *MONEY*, and other kinds of entities. Figure 3.2 shows some intuitive patterns that we could create using named-entity tags for extracting the *CompanyHeadquarters* relation. Once the entities in the text documents are tagged, *Snowball* can ignore unwanted entities. This allows *Snowball* to analyze the context that surrounds the entities of interest and check if they are connected by the right words and hence match our patterns.

⟨*ORGANIZATION*⟩ *'s headquarters in* ⟨*LOCATION*⟩
⟨*LOCATION*⟩-*based* ⟨*ORGANIZATION*⟩
⟨*ORGANIZATION*⟩, ⟨*LOCATION*⟩

Figure 3.2: Patterns that exploit named-entity tags.

Now we describe the *Snowball* pattern generation process in more detail. As we mentioned, *Snowball* analyzes the *text contexts* where the seed tuples occur in order to generate

extraction patterns. More specifically, *Snowball* considers an occurrence of a seed tuple $t = \langle a_1, \ldots, a_n \rangle$ within a text fragment if each attribute value a_i is recognized appropriately by the named-entity tagger. For example, a seed tuple $\langle$*Microsoft, Redmond*$\rangle$ might occur in a text fragment "servers at Microsoft's headquarters in Redmond were recently upgraded." From this text fragment, we can generate an *example* text context $C_{EX} = \langle$"servers at," *ORGANIZATION*, "'s headquarters in," *LOCATION*, "were recently upgraded."$\rangle$ for the example occurrence above, consisting of the text strings "servers at," "'s headquarters in," and "were recently upgraded." which surround the tuple attributes in the text fragment, as well as of the named-entity tags *ORGANIZATION* and *LOCATION* associated with the relation attributes, in the order and position in which they appear in the text.

More formally, we define the notion of text context as follows:

Definition 1 *Consider a tuple $t = \langle a_1, \ldots, a_n \rangle$ and a text fragment F where all attributes $a_1, \ldots, a_n$ of t occur. The text context associated with t and F is $\langle s_1, e_1, s_2, \ldots, e_n, s_{n+1} \rangle$, where (1) $F = s_1 a_1 s_2 a_2 \ldots s_n a_n s_{n+1}$ is a substring of the text fragment and (2) each named entity a_i is of named-entity type e_i, for $i = 1, \ldots, n$.*

For clarity, we refer to s_1 as the *Left text span* of a text context C_{EX}, to s_{n+1} as the *Right text span* of C_{EX}, and to $s_2, \ldots, s_n$ as the *Middle text spans* of C_{EX}. In the example text context above, "servers at" is the *Left* text span, "were recently upgraded." is the *Right* text span, and "'s headquarters in" is the *Middle* text span.

Following our "locality" assumption (i.e., that the information needed to extract a tuple is concentrated within a relatively short text fragment), we assume that the relevant information to extract a tuple is contained within small *Left* and *Right* spans. Hence, we limit the *Left* and *Right* spans of extraction patterns to a small window to the left and right of the leftmost and rightmost entity, respectively. Similarly, we limit the distance between entities to *MaxProximity* characters for some fixed –and relatively small– value. We will discuss the choice of the parameter values in Section 3.5.

Once we identify the text contexts associated with the seed tuples for the relation and their occurrences in the collection documents, we could use them directly to identify new relation tuples in the documents. For example, we could use the context $\langle$"servers at," *ORGANIZATION*, "'s headquarters in," *LOCATION*, "were recently upgraded."$\rangle$ to extract

a new tuple ⟨Oracle, Redwood⟩ from the text fragment "servers at Oracle's headquarters in Redwood were recently upgraded." Unfortunately, using text contexts directly is unsatisfactory for two reasons: (1) Text contexts might be too specific, and may hence match very few new tuples. In our hypothetical example, we were able to identify a new tuple ⟨Oracle, Redwood⟩ only because we found a text fragment "servers at Oracle's headquarters in Redwood were recently upgraded." that matched the earlier fragment for ⟨Microsoft, Redmond⟩ exactly, an unlikely event. (2) Text contexts might be too general, and may hence result in spurious "tuples" being extracted. For example, a text fragment "IBM, Armonk researchers reported..." would result in a text context ⟨"", *ORGANIZATION*, ",", *LOCATION*, "researchers reported"⟩, which could in turn match text fragments such as "... after vising Microsoft, California researchers reported ... ." This could result in a spurious tuple ⟨Microsoft, California⟩ being extracted. Because of these potential problems, *Snowball* does not use all text contexts as extraction patterns, but rather it synthesizes patterns from the contexts in a variety of ways, so that the patterns are general but not overly so.

Specifically, a *Snowball* pattern has the same general shape of the text contexts defined above, but each text span is replaced with an associated "acceptor," for flexibility, specifying when –and to what extent– a text substring matches the corresponding subcomponent of the pattern. Formally, we define a *Snowball* extraction pattern as follows:

Definition 2 *A Snowball pattern for an n-ary relation is defined as* $\langle acceptor_1, e_1, acceptor_2, e_2, \ldots, e_n, acceptor_{n+1} \rangle$*, where* $acceptor_1, \ldots, acceptor_{n+1}$ *are rules for determining whether a text span is "appropriate" or not, and* $e_1, \ldots, e_n$ *are the named-entity types or tags of the relation attributes, in the order in which the pattern expects them to appear.*

To match a text context, each acceptor, in effect, assigns a score to the respective text span, which determines the degree to which the text span is deemed appropriate for the target relation. For example, a text span "'s headquarters near" may be assigned a high score by an acceptor constructed from examples like "'s headquarters in."

We explored three complementary classes of acceptor rules for *Snowball*: an exact string-match model (*DIPRE*), a vector-space similarity model (*Snowball*), and classifier models

(namely, Naive Bayes classifiers, Support Vector Machines, and Sparse Markov transducers). We now give more details for each acceptor class and the associated scoring functions.

- *String Match Acceptors* (*DIPRE*): A string acceptor consists of a single rule comprised of a text string and a scoring function that returns 1 if a given text span matches the text string associated with the rule and 0 otherwise. For example, one rule may consist of the exact string "based in," indicating that only that string would result in a non-zero score, and all other strings would be rejected, with a score of zero. These acceptors are created by copying the exact text spans surrounding the example tuples. This approach to relation extraction is proposed in [Bri98], where extraction patterns equivalent to those using string-match acceptors were used for extracting relations from unstructured HTML documents. For example, *DIPRE* may generate string match patterns such as ⟨"servers at," *ORGANIZATION*, "'s headquarters in," *LOCATION*, "were recently upgraded."⟩. String-match acceptors (and the resulting patterns) are sometimes not effective on unstructured text, as we will show empirically in subsequent sections.

- *Vector Space Acceptors* (*VS*): Defining patterns that require exact string matches might be too rigid a choice for many relations. For example, text spans "renovated headquarters in," "'s spacious headquarters in," and many other analogous variations should be acceptable for the *CompanyHeadquarters* relation. Hence, it is critical to match the text spans surrounding the entities within a text context in a flexible way that is selective, yet has high coverage. As a result, minor variations such as an extra comma or a determiner will not stop us from matching contexts that are otherwise close to our patterns. More specifically, *VS* acceptors use a *feature vector* to describe valid text contexts in the same way as the documents are represented in the vector-space model of information retrieval [Sal89] or for text classification.

  Thus, the rules for the *VS* acceptors are vectors associating weights with features, which we define as contiguous sequences of non-space characters. The weights indicate the importance of each feature in the corresponding context. An example *VS* acceptor rule is [(-, 0.5), (based, 0.5)], which associates a weight of 0.5 with the features "-"

and "based," and can accept strings like ", based in" and ", which is based in." An example of a *Snowball* pattern using vector-space acceptor rules is $\langle$ [(the, 0.2)], *LOCATION*, [(-, 0.5), (based, 0.5)], *ORGANIZATION*, [] $\rangle$. This pattern will match a text context like "*the Irving-based Exxon Corporation,*" where the word "the" (*Left* span) precedes the *LOCATION* entity Irving, which is in turn followed by the strings "-" and "based" (*Middle* span) and the *ORGANIZATION* entity Exxon Corporation. The text to the right of the organization in the string is unimportant in this case, hence the empty *Right* acceptor in the pattern. As we mentioned, slight variations of the text span will be accepted with a lower score.

In order to match a text span s_i with the corresponding vector-space acceptor VS_i, we represent s_i as a vector of features v_i (described above) with associated weights just like we represent the *VS* acceptor rules. We compute the *score* of v_i and vector-space acceptor VS_i as the similarity [Sal89] between v_i and VS_i:

$$Score(VS_i, v_i) = \frac{VS_i \cdot v_i}{\|VS_i\| \cdot \|v_i\|}$$

The overall score of the text context and a vector-space pattern is a linear combination of the respective *Score* values.

- *Classifier Acceptors* (*CL*): *Snowball* can use as an acceptor rule a *classifier* trained to assign higher probability to the tuples extracted from "valid" contexts than to the tuples from "invalid" contexts. Here, the scoring function is just the score assigned to the text span by the respective classifier. In our model, a classifier instance is associated with each acceptor CL_i, so that an extraction pattern with $n+1$ acceptors will be associated with $n+1$ classifier instances. The classifiers are trained over respective text spans in the example text contexts. After training, classifiers can return a score between 0 and 1, estimating the likelihood that a text span s is part of a "valid" text context. Specifically, we experiment with Naive-Bayes (*NB*) classifiers.[2]

[2]We also experimented with using Support Vector Machines in [YA03] and Sparse Markov Transducers in [AEG00]. For brevity, we omit our experiments with these additional classifiers, and refer the interested reader to our work in [AEG00, YA03].

In the *NB* classifier acceptor, each feature f_i has an associated weight w_i, which estimates the probability $Prob(t \text{ is } valid|f_i)$, and is computed as:

$$w_i = \frac{Prob(f_i|C = positive) \cdot Prob(C = positive)}{Prob(f_i)}$$

where $Prob(f_i)$ is the probability of observing f_i in any fragment of a text document, and $Prob(f_i|C = positive)$ is the probability of observing f_i in a positive example text context. $Prob(C = positive)$ is the prior probability of seeing an example context and is estimated as the ratio of the number of observed example contexts to the total number of document fragments of the same size in the document sample. Then, the probability of a candidate context C_{CA} with features $f_1, \ldots, f_m$ being valid is $Prob(C_{CA}|f_1, \ldots, f_m) = \prod_i^m w_i$.

Having described the acceptor classes, we now turn to the task of creating and training the acceptors in order to generate extraction patterns. The procedure varies for each acceptor class:

- *DIPRE*: The method for creating *DIPRE* acceptors clusters together occurrences of seed tuples that share the same *Middle* text span into one "group." Thus, constructing a *DIPRE* acceptor for a *Middle* span is simple: the acceptor is the exact text string connecting the entities in at least two different example contexts. For a group of contexts that share the same *Middle* text span, *DIPRE* finds the maximal *Left* and *Right* strings that match all of the *Left* and *Right* spans in the group. This is done by starting with empty *Left* and *Right* acceptors, and extending them one character at a time while all contexts in the group match [Bri98]. The resulting pattern consists of the list of entity tags (which are the same for each group), and a list of strings to be matched exactly by the respective *Left*, *Right*, and *Middle* text spans.

- *VS*: In order to generate a pattern, *Snowball* groups occurrences of known tuples in documents, if the contexts surrounding the tuples are "similar enough." More precisely, for a text context $C^i_{EX} = \langle s^i_1, e^i_1, \ldots, s^i_k, e^i_k, \ldots, s^i_n, e^i_n, s^i_{n+1} \rangle$, *Snowball* generates feature vectors $v^i_1, \ldots, v^i_k, \ldots, v^i_n, v^i_{n+1}$ so that each text span s^i_k is represented by vector v^i_k. Similar text contexts are then clustered together using a simple single-pass clustering algorithm [FBY92]. In order to perform the clustering, we specify

the similarity measure between two text contexts $C^i_{EX} = \langle s^i_1, e^i_1, \ldots, e^i_n, s^i_{n+1} \rangle$ and $C^j_{EX} = \langle s^j_1, e^j_1, \ldots, e^j_n, s^j_{n+1} \rangle$ as:

$$Sim(C^i_{EX}, C^j_{EX}) = \begin{cases} \sum_{k=1}^{n+1} (v^i_k \cdot v^j_k) \cdot W_k & \textit{if } e^i_k = e^j_k, \textit{ for } 1 \leq k \leq n \\ 0 & \textit{otherwise} \end{cases}$$

where the weights W_k represent the relative importance of each text span (set during the tuning of *Snowball* as we will discuss below) and v^i_k and v^j_k are the weight vectors associated with s^i_k and s^j_k, respectively. For each created cluster, we maintain $n+1$ *centroids*, so that all vectors in the cluster in position k (e.g., all *left* vectors) are represented by the corresponding $centroid_k$. These $n+1$ centroids, together with the named-entity tags (which are identical for all example contexts in the cluster), are used to create a *VS* pattern, where each $acceptor_k$ is constructed from the corresponding $centroid_k$. Finally, the entity tags in the pattern are identical to the entity tags in the cluster.

- *CL*: Classifiers typically require labeled data for training. In our partially supervised scenario, we train the classifier on occurrences of seed tuples as positive examples, and example contexts automatically recognized as containing invalid tuples (Section 3.3) as negative examples. The actual training algorithms vary with each classifier, and are described at length in [Mit97] and other standard machine-learning texts.

To generate the extraction patterns with the selected acceptor class (*VS* or *CL*), we proceed as follows. As we discussed, we first find example text contexts where each seed tuple occurs. We then group together similar text contexts using the appropriate similarity metric. For each group of similar contexts, we create an extraction pattern and train the selected acceptors for the pattern. Finally, we post-process the generated patterns to discard empty and invalid patterns (e.g., those patterns without sufficient "support"). The resulting patterns are all initially assigned the same confidence, which will be adjusted later in the extraction process.

After generating and training the extraction patterns as described above, we can use the patterns to extract new tuples. More specifically, we use the extraction patterns to predict whether a candidate text context C_{CA} contains a valid tuple for the target relation.

This likelihood, or *Match* of a text context and extraction pattern, is computed as the linear combination of individual acceptor scores. Specifically, we define *Match* between a pattern $P = \langle acceptor_1, e_1^p, \ldots, acceptor_k, e_k^p, \ldots, e_n^p, acceptor_{n+1} \rangle$ and a candidate text context $C_{CA} = \langle s_1, e_1^c, \ldots, s_k, e_k^c, \ldots, e_n^c, s_{n+1} \rangle$ as:

$$Match(P,\ C_{CA}) \quad = \quad \begin{cases} \sum_{k=1}^{n+1} acceptor_k(s_k) \cdot W_k & if\ e_k^p = e_k^c,\ for\ 1 \leq k \leq n \\ 0 & otherwise \end{cases}$$

where W_k indicates the relative weight, or importance, of $acceptor_k$[3].

In summary, we have presented techniques for generating extraction patterns for *Snowball*. These extraction patterns will be used to extract new tuples from the collection by analyzing the entity tags and the surrounding text contexts. As we discussed, a *Snowball* extraction pattern consists of n entity tags corresponding to the relation attributes and $n+1$ acceptors to match the surrounding text spans. Each acceptor is trained to estimate the likelihood of the corresponding text span belonging to a "valid" text context, and individual acceptor scores are combined to estimate the validity of the complete text context. We discussed three different classes of acceptors –namely, a simple exact string match acceptor (*DIPRE*), a vector-space acceptor (*VS*), and classification-based acceptors (*CL*)– and outlined the corresponding methods for training each acceptor. We now describe how the patterns are used to extract and automatically evaluate new tuples for a target relation.

3.2.2 Extracting and Automatically Evaluating New Tuples

After generating extraction patterns for a target relation and a document collection, *Snowball* scans the collection documents to discover new tuples, using Algorithm 3.1. *Snowball* first identifies candidate text contexts that contain the target entities as determined by the named-entity tagger. A candidate tuple t_{CA} is generated from a candidate context C_{CA} if there is at least a pattern P such that $Match(P, C_{CA}) \geq \tau_{score}$, where τ_{score} is the *extraction similarity threshold* for the minimum degree of match between a text context and an extraction pattern. If the *BestScore* value of a candidate text context and an extraction

[3] From our experiments with English-language documents, we have found the *Middle* context to be the most indicative of the relationship between the elements of the tuple. Hence, we will typically assign the *Middle* acceptors higher weights than the *Left* and *Right* acceptors.

pattern is at least τ_{score}, we add the candidate tuple t_{CA} to the set of *CandidateTuples*. Each candidate tuple will, in general, have a number of patterns that helped generate it, each with an associated degree of match. *Snowball* uses this information, together with information about the "selectivity" of the patterns (discussed in the next section), to decide what candidate tuples to actually add to the table that it is constructing.

```
Algorithm ExtractTuples(Patterns, Documents)

  CandidateTuples = {};
  foreach document in Documents do
      candidateContexts = generateTextContexts(document);
      foreach context C_CA in candidateContexts do
          BestScore = 0;
          foreach pattern P in Patterns do
              Score = Match(P, C_CA);
              if Score ≥ BestScore then
                  BestPattern = P;
                  BestScore = Score;
              end
          end
          if BestScore ≥ τ_score then
              Create candidate tuple t_CA from C_CA;
              Associate pattern BestPattern with t_CA, with match degree
              BestScore;
              Add t_CA to CandidateTuples;
          end
      end
  end
  return CandidateTuples
```

Algorithm 3.1: Extracting Tuples from *Documents* using *Patterns* Algorithm

Snowball uses the extracted tuples as seed to further expand the set of extraction patterns. If undetected, invalid tuples can potentially be used as seed tuples for a later iteration of the algorithm. This, in turn, might result in even more invalid tuples being generated. To prevent using potentially incorrect tuples as examples, we only keep tuples that have a high *confidence*. Intuitively, the confidence of a tuple will be high if it is generated by several highly selective patterns. To compute the confidence of a tuple, we store the set of

patterns that produced the tuple, together with the degree of match between the context in which the tuple occurred and the matching pattern. Consider a candidate tuple t_{CA} and the set of patterns $\{P_1, \ldots, P_k\}$ that were used to generate it. Let us assume for the moment that we know the probability $Prob(P_i)$ with which each pattern P_i generates valid tuples. If these probabilities are independent of each other, then the probability that the candidate text context C_{CA} is valid can be calculated as:

$$Prob(C_{CA} \textit{ is valid}) \;=\; 1 - \prod_{i=1}^{k} (1 - Prob(P_i))$$

Since we usually do not know the value of *Prob(P_i)* exactly, we will approximate it as *Conf(P_i)* (Section 3.3.1). By substituting this approximation into the equation above, we estimate $Prob(C_{CA} \textit{ is valid})$ as:

$$Prob(C_{CA} \textit{ is valid}) \;\approx\; 1 - \prod_{i=1}^{k} (1 - Conf(P_i))$$

We also account for the cases where t_{CA} has occurred in contexts that did not match our patterns perfectly. Intuitively, the lower the degree of match between a pattern and a context, the higher is the chance of producing an invalid tuple. For this, we scale each $Conf(P_i)$ term by the degree of match of the corresponding pattern and context as follows:

Definition 3 *The* confidence *of a candidate tuple t_{CA} generated by extraction patterns $P_1, \ldots, P_k$ is defined as:*

$$Conf(t_{CA}) = 1 - \prod_{i=1}^{k} (1 - Conf(P_i) \cdot Match(P_i, C^i_{CA})) \tag{3.1}$$

where C^i_{CA} is the text context associated with the occurrence of t_{CA} that matched P_i with highest degree of match, adjusted by the degree of match between P_i and C^i_{CA}.

After specifying the confidence of the candidate tuples using the definition above, *Snowball* discards all tuples that have low confidence[4]. These tuples could add noise into the

[4]Note that the confidence of a tuple t, *Conf(t)*, is calculated differently from the confidence of a pattern P, *Conf(P)*. Hence, the definition above is not circular.

pattern generation process, which would in turn introduce more invalid tuples, degrading the performance of the system. Therefore we include as seed for the next iteration the tuples that have confidence of at least τ_{seed}, where τ_{seed} is a threshold that we discuss in Section 3.5.

3.3 Automatically Evaluating Extraction Patterns

Generating good patterns is a challenging task. For example, we may generate a *VS* pattern $\langle$ [], *ORGANIZATION*, [(",", 1)], *LOCATION*, [] $\rangle$ from a text occurrence like "Intel, Santa Clara, announced..." This pattern will be matched by any string that includes an organization, followed by a comma, followed by a location. Unfortunately, a sentence "It's a great time to invest in Microsoft, New York-based analyst Jane Smith said ..." might then generate an incorrect tuple $\langle$*Microsoft, New York*$\rangle$. Hence, we should assign a low "confidence" to this pattern, since it might generate incorrect tuples. *Snowball* will try to identify such patterns and not trust them, to instead focus on other patterns. The tuples generated by patterns with low confidence will be discarded, unless they are supported by other patterns with higher confidence. In this section, we present two different ways of automatically evaluating the quality of extraction patterns. The first approach, presented next (Section 3.3.1), exploits relation constraints (e.g., the key constraints) to automatically detect "negative" pattern matches. The second, more general EM-based approach (Section 3.3.2) does not require such constraints, focusing instead on the patterns that consistently match the seed tuples.

3.3.1 Constraint-Based Pattern Evaluation

As a first approach to evaluate extraction patterns, we exploit any known constraints on the relation R to be extracted. Specifically, we invalidate an extracted tuple if it violates any of the integrity constraints for R. In this case, we count the tuple as a "negative" match for the pattern(s) that generated it. Patterns with negative matches will then be discounted in favor of patterns without them. The constraints that we can utilize include key constraints (i.e., requiring key attributes of a tuple to uniquely determine the remaining attribute values),

domain constraints (i.e., the values of an attribute are restricted to a particular domain or range; for example, the value of an attribute must be, say, at least two characters long), and identity constraints (e.g., attributes of the same tuple must not be equal). Additional constraints for the target relation, if available, can be naturally incorporated into our model.

For concreteness, consider the key constraint in the *CompanyHeadquarters* relation, where each organization can only be associated with one location. If there is a known tuple $t' = \langle o, \ell' \rangle$ for the same organization o as a new tuple $t = \langle o, \ell \rangle$ generated by a pattern we are evaluating, then we compare locations ℓ' and ℓ. If the two locations are the same, then the new tuple t is considered a *positive* match for the pattern. Otherwise, the match is *negative*. As another example, consider the task of extracting the *Synonyms(Protein1:GENEPROTEIN, Protein2:GENEPROTEIN)* relation presented in [YA03]. This relation has an identity constraint that requires that no valid tuple $t = \langle s_1, s_2 \rangle$ is such that $s_1 = s_2$. Hence, any tuple that violates this constraint would be considered to be a negative match for every pattern that generated the offending tuple.

Intuitively, a selective pattern will have many positive matches, and few negative ones. Accordingly, we define the *confidence* of a pattern P as:

$$Conf(P) = \frac{P.positive}{P.positive + P.negative} \tag{3.2}$$

where P.positive is the number of positive matches for P and P.negative is the number of negative matches.

As an example of exploiting the key constraint to establish the confidence of a pattern, consider the pattern $P = \langle$ [], *ORGANIZATION*, [(“,”, 1)], *LOCATION*, [] $\rangle$ referred to above. Assume that this pattern only matches the following three lines of text:

> “**Exxon**, **Irving**, said” ...
>
> “**Intel**, **Santa Clara**, cut prices” ...
>
> “invest in **Microsoft**, **New York**-based, analyst Jane Smith said” ...

The first two lines generate candidate tuples ⟨*Exxon, Irving*⟩ and ⟨*Intel, Santa Clara*⟩, which match known seed tuples from Table 3.1. The third line proposes a candidate tuple ⟨*Microsoft, New York*⟩, and the location in this tuple conflicts with the location in tuple

⟨*Microsoft, Redmond*⟩, which we consider as a negative match for P. Then, pattern P has confidence $Conf(P) = \frac{2}{2+1} = 0.67$.

Our definition of confidence of a pattern above is only one among many possibilities. An alternative is to account for a pattern's "coverage" in addition to its selectivity. For this, we adapt a metric originally proposed by Riloff [Ril96] for the AutoSlug-TS information extraction system:

$$Conf_{RlogF}(P) = Conf(P) \cdot \log_2(P.positive)$$

Pattern confidences are defined to have values between 0 and 1. Therefore, we normalize the $Conf_{RlogF}$ values by dividing them by the largest confidence value of any pattern.

The techniques for automatic pattern evaluation described above rely on the existence of constraints (e.g., the key constraint) over the target relations. For some relations, such constraints might not be available. Therefore, we now present a more general EM-based approach for automatically evaluating extraction patterns and associated tuples that does not rely on the existence of relation constraints, and can thus be used for a larger family of target relations.

3.3.2 EM-Based Pattern Evaluation

So far, our pattern evaluation strategy relied on constraints on the tuples of the relation. Unfortunately, sometimes such constraints may not be available. For example, relations containing disease treatments do not have useful integrity constraints that we could exploit effectively. To handle such cases, and still be able to evaluate the quality of extraction patterns, we present in this section a partially supervised technique that borrows ideas from text classification.

Our technique is based on the observation that an extraction pattern can be regarded as a classifier that assigns some score to a text context. Intuitively, good patterns will assign high scores to valid text contexts, containing a tuple for the target relation, whereas bad patterns may assign high scores to invalid contexts. If labeled data were available, we could estimate the accuracy of the "classifiers" by using cross-validation, trusting their predictions accordingly. Unfortunately, in the common case where labeled data is not available, we cannot apply standard cross-validation techniques to estimate this accuracy. However,

recent developments in the area of *partially supervised* text classification, where positive and unlabeled examples are available but no labeled negative examples are [LLYL02], can help. Given a classifier and a set of positive examples, as well as a large number of unlabeled examples, we can use a variation of the classic *Expectation-Maximization* (*EM*) algorithm [DLR77] that is designed to optimize parameters of a model with missing (unlabeled) values, which naturally maps to our partially supervised setting.

The *EM* Pattern Evaluation Algorithm: We apply the general idea of EM to derive the confidence of extraction patterns, and consequently the confidence of the corresponding extracted tuples. The classic EM algorithm proceeds in two stages: in the "Expectation" (*E*) stage, the unknown values (i.e., the confidence scores for the extraction patterns) are predicted using the current confidence scores of the tuples generated by these patterns; in the "Maximization" (*M*) stage, the predicted pattern confidence scores are used to adjust the confidence scores of the extracted tuples[5].

Our *ScorePatternsEM* algorithm is shown in Figure 3.2. As input, the algorithm accepts a set of extraction patterns (*Patterns*), a set of positive seed tuples ($Seed_P$), a set of negative seed tuples ($Seed_N$, which could be an empty set), a set of all extracted tuples (*Tuples*, all generated by the *Patterns*), and the maximum number of iterations of the algorithm *MaxIteration*. In the initialization stage, all candidate tuples in *Tuples* are assigned a confidence score of 0, and all *Seed* tuples are assigned the confidence score of 1.

In the expectation stage of the algorithm, we update the confidence of each pattern P_j using the positive and negative counts of matching tuples (Equation 3.2). That is, if P_j has extracted a tuple with a confidence score of at least τ_t, this tuple is considered a positive match; otherwise, it is considered a negative match. Using the positive and negative match counts, we compute the confidence of P_j as in Equation 3.2. In the maximization stage of the algorithm, we compute the confidence of each tuple based on the newly computed confidence scores of the patterns (Equation 3.1). The original scores of the positive and negative seed tuples are preserved. This process is iterated *MaxIteration* times. Ideally, the

[5] An alternative formulation that predicts tuple confidence scores in the *E* stage and updates pattern confidence scores in the *M* stage could also be devised.

pattern and tuple confidence scores will converge to a "stable" combination of values, with those tuples that occur in contexts most similar to those of *SeedP* tuples being assigned the highest confidence scores.

```
Algorithm ScorePatternsEM(Patterns, Seed_P, [ Seed_N, ] Tuples, MaxIteration)

  initialize Conf(T_i) = 0 for each tuple T_i in Tuples;
  initialize Conf(T_s) = 1 for each tuple T_s in Seed_P;

  foreach iteration in 1, ..., MaxIteration do
      //E step: predict pattern confidence scores.
      foreach pattern P_j in Patterns do
          Pos = Neg = 0;
          foreach tuple T_i generated by P_j do
              if Conf(T_i)≥ τ_t then Pos=Pos + 1;
              else Neg= Neg +1;
          end
          UpdatePatternConfidence(P_j, Pos, Neg)        // Use Equation 3.2.
      end
      Normalize pattern scores;
      //M step: recompute tuple confidence scores.
      foreach tuple T_i in Tuples do
          if T_i ∈ Seed_P then Conf(T_i)=1;
          else if T_i ∈ Seed_N then Conf(T_i)=0;
          else UpdateTupleConfidence(T_i, Patterns)        //Use Equation 3.1.
      end
  end
```

Algorithm 3.2: The *EM* Pattern Evaluation Algorithm

The *EM-Spy* Pattern Evaluation Algorithm: The *ScorePatternsEM* algorithm above sometimes converges on excessively high confidence scores for patterns and tuples. This tends to happen because eventually many of the unlabeled candidate tuples are assigned high confidence values, which in turn causes *ScorePatternsEM* to overestimate the pattern confidence values. To address this problem, we adapt an extension of the classic *EM* classification algorithm, recently presented in [LLYL02] for document classification. The main idea is to use the "Spy" method of *hiding* the "labels" of some known valid seed tuples, thus

allowing the *ScorePatternsEM* algorithm to adjust the confidence of these tuples. After *ScorePatternsEM* completes, we can choose the τ_t and τ_N thresholds to expand the $Seed_P$ and $Seed_N$ sets with what are most likely to be valid and invalid tuples, respectively.

Our *ScorePatternsEM-Spy* algorithm is shown in Figure 3.3. The input to the algorithm is similar to that of *ScorePatternsEM*. During the initialization stage, a set of *Spy* tuples is created as a random subset (e.g., half) of the $Seed_P$ positive seed tuples. Then, the *ScorePatternsEM* is initialized with the remaining (non-*Spy*) $Seed_{PS}$ tuples, and an empty set of $Seed_N$ negative seed tuples. The *ScorePatternsEM* algorithm runs for *MaxIteration* iterations as described above, adjusting the confidence scores of all candidate tuples not included as $Seed_{PS}$ parameter to *ScorePatternsEM*, which includes our *Spy* tuples. We then use the distribution of confidence values for *Spy* tuples to learn the cutoff confidence value for the tuples that are most likely to be valid (i.e., those with the confidence scores in the top 10%), and the values indicating tuples most likely to be invalid (i.e., those with confidence scores in the bottom 10%). This allows us to automatically determine the appropriate threshold for extending the $Seed_{PS}$ and $Seed_N$ sets of positive and negative seed tuples. The *ScorePatternsEM* algorithm is then reinitialized with the extended $Seed_{PS}$ and $Seed_N$ sets. The additional negative seed tuples now prevent *ScorePatternsEM* from overestimating pattern and tuple confidence scores. Finally, when *ScorePatternsEM* completes, we recompute the τ_t threshold and can use it to select new reliable seed tuples for the next iteration of the overall *Snowball* training process.

The *ScorePatternsEM-Spy* algorithm we just presented is more robust than the previous *ScorePatternsEM* algorithm without the reinitialization. Additionally, *ScorePatternsEM-Spy* returns the *automatically* estimated τ_t threshold for selecting new seed tuples.

In summary, we described our *Snowball* system, which starts with a small number of seed tuples, discovers extraction patterns, and automatically evaluates the extracted patterns and the generated tuples. We now delve into the experimental setup and evaluation metrics for comparing variations of *Snowball* and other information extraction techniques. We will present our empirical evaluation in two stages. First, we present results for a few representative variations of *Snowball* (Section 3.6) that were tuned for extracting the *CompanyHeadquarters* relation. This evaluation lays the groundwork for the more comprehensive

Algorithm ScorePatternsEM-Spy(*Patterns, $Seed_P$, Tuples, MaxIteration*)

Spy = randomly selected half of $Seed_P$ tuples;
$Seed_{PS}$ = $Seed_P$ - *Spy*;
$Seed_N$ = {};

ScorePatternsEM(*Patterns, $Seed_{PS}$, $Seed_N$, Tuples, MaxIteration*);
Set τ_N such that 10% of *Spy* tuples have *Conf* $< \tau_N$;
Set τ_t such that 10% of *Spy* tuples have *Conf* $\geq \tau_t$;

foreach *tuple* T_i **in** *Tuples* **do**
    if *Conf*(T_i) $< \tau_N$ **then** Add T_i to $Seed_N$;
    if *Conf*(T_i) $\geq \tau_t$ **then** Add T_i to $Seed_{PS}$;
end

ScorePatternsEM(*Patterns, $Seed_{PS}$, $Seed_N$, Tuples, MaxIteration*);
Update τ_t such that 90% of the *Spy* tuples still have *Conf*$\geq \tau_t$;
return τ_t;

Algorithm 3.3: The *EM-Spy* Pattern Evaluation Algorithm

evaluation of *Snowball* that we report in Chapter 4.

3.4 Experimental Setup and Evaluation Metrics

In this section, we describe the data sets used for our illustrative evaluation (Section 3.4.1), the methodology used for evaluating the extraction output (Section 3.4.2), and the evaluation metrics (Section 3.4.3). Later, in Section 3.5, we describe the relevant Snowball tuning parameters.

3.4.1 Data Sets

Our experiments in this chapter are on the extraction of the *CompanyHeadquarters(Organization:ORGANIZATION, Location:LOCATION)* relation from a large collection of real newspapers from the North American News Text Corpus, available from LDC[6]. These collections include articles from the Los Angeles Times, The Wall Street Journal, and The New York Times, from 1994 to 1997. We split the corpus into two disjoint collections: the

[6] `http://www.ldc.upenn.edu`

training collection consists of 178,000 documents, all from 1996, while the *test* collection is composed of 142,000 documents, from 1995 and 1997.

3.4.2 Creating the *Ideal* Table

To evaluate *Snowball* over a *large* text collection, we had to adapt and extend appropriate evaluation approaches from the literature. The goal of *Snowball* is to extract as many valid tuples as possible from the text collection and to combine them into one table. As we have discussed, we do not attempt to capture every *instance* of such tuples. Instead, we exploit the fact that these tuples will tend to appear multiple times in the types of collections that we consider. As long as we capture one instance of a tuple, we will consider our system to be successful for that tuple. This is different from the goal of traditional information extraction [Rep95]. Traditional information extraction systems aim at extracting all the relevant information from *each document* as completely as possible, while our system extracts tuples from all of the documents in the collection and combines them into one table. To evaluate this task, we adapt the recall and precision metrics from information retrieval to quantify how accurate and comprehensive our *combined table of tuples* is [Sal89].

Our metrics for evaluating the performance of an extraction system over a collection of documents are based on *Ideal*, the set of all the tuples that "appear" in the collection. After identifying *Ideal*, we compare it against the tuples produced by the system, *Extracted*, using adapted precision and recall metrics. For small text collections, we could inspect all documents manually and compile the *Ideal* table by hand. Unfortunately, this evaluation approach does not scale, and becomes infeasible for the kind of large collections over which *Snowball* is designed to operate.

Interestingly, we can create an approximation of *Ideal* with manageable effort even for a large document collection. The overall approach is to use an external (structured) relation R_S, compiled independently of the document collection at hand, and then select from R_S the tuples that actually occur in the collection. This relation, though usually incomplete, nevertheless provides a useful sample of the correct tuples in the collection. Using this sample we can then estimate the recall and precision of an extraction system by computing the intersection of the extracted table with the *Ideal* table. Overall, this evaluation approach

demands only reasonable human and computational resources, as we demonstrate for the *CompanyHeadquarters* relation over a large collection of news documents. The resulting automatic accuracy estimation approach is also reasonable, allowing us to compare alternative techniques without the overhead of manually labeling a sufficient number of documents, a formidable task. As we will see, the results obtained with the *Ideal*-based methodology are consistent with those derived by manually checking the extracted table, which we did for completeness and to validate our (automated) evaluation methodology.

We now describe in more detail the construction of the *Ideal* table, outlined in Algorithm 3.4. As we mentioned, the general idea is to construct a set of known valid tuples for the relation of interest that also occur in the document collection at hand. Unfortunately, we cannot simply check for occurrence in the collection of every tuple in R_S because common entities (e.g., company names) have enough variations in a large collection to make this task difficult. Instead, we proceed as follows. We first create a table *AllCandidates* of all entities of the appropriate types (e.g., *ORGANIZATION* and *LOCATION* for the *CompanyHeadquarters* relation) that occur within a predefined distance *MaxProximity* in at least one document in D. We then *join AllCandidates* with R_S using an approximate string matching algorithm in order to collect attribute values in R_S together with their variations in the collection. For these experiments, we used Whirl [Coh98], a research tool developed at AT&T Research Laboratories for integrating similar textual information. Whirl allows us to match, for example, different references for the same organization, such as "Toshiba" and "Toshiba, Inc."

For *CompanyHeadquarters* we used a large, publicly available directory of organizations provided on the "Hoover's Online" web site[7]. Although the directory does not cover every organization there is, it is large enough for our purposes, covering over 13,000 mostly publicly traded corporations. From this well structured directory, we generate a table of organization-location pairs as the R_S table that Algorithm 3.4 expects. We then use the algorithm to construct the *Ideal* table, identifying which tuple in R_S (or slight variations thereof) appear in the corpora that we use for our experiments.

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

Algorithm CreateIdeal(R_S, D)

AllCandidates = {};
foreach *document* **in** D **do**
    Tag all named entities in *document*;
    Create *CandidateTuples* as n-tuples of entities that (1) appear within *MaxProximity* of each other and (2) match attributes of R_S;
    Add *CandidateTuples* to *AllCandidates*;
end
//Find tuples in R_S that match tuples in *AllCandidates*.
Ideal = *AllCandidates* $\bowtie_\phi$ R_S;
return *Ideal*

Algorithm 3.4: The *Ideal* Construction Algorithm

3.4.3 Evaluation Metrics

After creating the *Ideal* table, we can use it to evaluate the quality of the output of an extraction system, or the *Extracted* table. If the initial structured source R_S contained all possible organizations, then we could just measure what fraction of the tuples in *Extracted* are in *Ideal* to calculate precision. Unfortunately, a large collection will contain many more tuples than are contained in any single manually compiled directory. (For example, in our estimate, our training collection contains more then 80,000 valid tuples for the *CompanyHeadquarters* relation, while the R_S table that we used for this relation, from Hoover's, covers just over 13,000 organizations.) If we just calculated precision as above, all the valid tuples extracted by *Snowball*, which are not contained in *Ideal*, will unfairly lower the reported value of precision for the system. To address this problem, we create a new table, *ExtractedFiltered*, by filtering the extraction output with the values of *Ideal*. For example, for the *CompanyHeadquarters* relation we eliminate from *Extracted* any tuple whose organization is not in *Ideal*.

Given the table *Ideal* and the *ExtractedFiltered* table that we have just created, we can define recall and precision more formally. Specifically, we define *Recall* as:

$$Recall = \frac{|ExtractedFiltered \bigcap Ideal|}{|Ideal|} \tag{3.3}$$

where the intersection between relations is computed using an approximate string match

over attribute values. We give more details on how we handle variations of attribute names in [AG00].

Similarly, we define *Precision* as:

$$Precision = \frac{|ExtractedFiltered \bigcap Ideal|}{|ExtractedFiltered|} \tag{3.4}$$

An alternative to using our *Ideal* metric to estimate precision could be to sample the extracted table, and check each value in the sample tuples by hand. (Similarly, we could estimate the recall of the system by sampling documents in the collection, and checking how many of the tuples mentioned in those documents the system discovers.) This evaluation method is time consuming, potentially error-prone, and has to be redone for each new collection. Indeed, our system is specifically designed for large collections, where it is not possible for a human to manually examine any significant portion of the collection. In this sense, the sampling technique is inferior to the *Ideal* metric that we proposed. However, by sampling the extracted table we can detect invalid tuples not mentioned in *Ideal*. Similarly, we can detect invalid tuples that result from named-entity tagging errors. Hence, we will also report precision estimates using sampling, and refer to this method as *sample-based precision*. Furthermore, we also created a labeled document sample for several relations and document collections, and we will use these for the comprehensive evaluation that we discuss and report in the next chapter. However, because of the human effort involved in the labeling process, we were forced to restrict the sample size to under 100 documents for each relation-collection combination. In contrast, the *Ideal* table was automatically compiled from thousands of documents, making its associated recall and precision metrics a meaningful complement to the document sample-based results. For this reason, we will use both metrics for the illustrative experiments of this chapter.

3.5 *Snowball* Tuning Parameters

As we discussed, *Snowball* requires a handful of seed tuples as user input to extract a relation, as well as the entity types and integrity constraints associated with the relation. Additionally, *Snowball* has a number of parameters that can be tuned for each relation and collection of interest, most of which were discussed in Section 3.2:

- *Number of seed tuples NumSeed*: The minimum number of seed tuples to initialize *Snowball*.
- *Minimum seed confidence* τ_{seed}: The minimum confidence threshold to use a tuple as seed for subsequent *Snowball* iterations.
- *Minimum tuple confidence* τ_t: The minimum confidence to keep an extracted tuple.
- *Minimum pattern support* τ_{sup}: The minimum number of different seed tuples required to create a pattern.
- *Minimum clustering similarity* τ_{sim}: The minimum value of the *Sim* function for clustering together example contexts when creating extraction patterns.
- *Minimum extraction similarity* τ_{score}: The minimum value of the *Match* function to match candidate contexts to extraction patterns for extracting new tuples.
- *External context size WindowSize*: The maximum length (words) of *Left* and *Right* text spans.
- *Internal context size MaxProximity*: The maximum length in characters of the *Middle* text spans.
- *Number of Snowball iterations* I_{MAX}: The number of *Snowball* training iterations.
- *Relative span weights* $W_1, \ldots, W_{n+1}$: The relative "importance" of each acceptor.

We tuned the values for these parameters by running *Snowball* for the *CompanyHeadquarters* relation over the training collection, and comparing the extraction accuracy for different parameter values using the *Ideal* metric.

We also explored other implementation choices for *Snowball* by running the system on the training collection:

- *Punctuation*: We considered discarding punctuation and other non-alphanumeric characters from the contexts surrounding the named entities. Our hypothesis was that punctuation may just add noise and carry little content to help extract tuples.

We report results for *Snowball* and *Snowball-Plain*, where *Snowball* uses punctuation, and *Snowball-Plain* discards it.

- *Choice of pattern scoring strategies*: We tried minor variations on the basic constraint violation heuristic of Section 3.3, with or without using the *RlogF* metric of [Ril96].

Settings for other parameters listed above were also set based on the training collection experiments. We will present techniques to *automatically* set some of them in the next chapter. The values of the most important parameters are summarized in Table 3.2.

Parameter	*Value*	*Description*
NumSeed	5	Number of seed tuples
τ_{seed}	0.8	Minimum seed confidence
τ_t	0.2	Minimum tuple confidence
τ_{sup}	2	Minimum pattern support
τ_{sim}	0.6	Minimum clustering similarity
τ_{score}	0.6	Minimum extraction similarity
WindowSize	2	Maximum length (words) of *Left* and *Right* spans
MaxProximity	-	Maximum length (words) of *Middle* spans
I_{MAX}	3	Number of *Snowball* training iterations
$W_1, W_2, \ldots, W_n, W_{n+1}$	$0.2, 0.6, \ldots, 0.6, 0.2$	Relative span weights

Table 3.2: Parameter values used for evaluating *Snowball* on the test collection.

We now turn to the first in the series of experimental evaluations of *Snowball* and comparable systems. The experiments in the rest of this chapter are presented to highlight the major contributions of *Snowball*, and to illustrate the issues that arise with *Snowball* and related systems. We will present a more systematic and comprehensive evaluation of *Snowball* in the next chapter, together with techniques for adapting *Snowball* to new relations and collections of interest.

3.6 *CompanyHeadquarters*: Experimental Results

In this section, we experimentally compare alternative techniques for extracting the *CompanyHeadquarters* relation from news articles. Since *CompanyHeadquarters* has an associated key constraint, as we discussed, we will use the constraint-based pattern evaluation strategy for the experiments. Note that this is only the first in a series of *Snowball* evaluation experiments, which are completed in the next chapter. In this section we will focus on the implementation of *Snowball* using *VS* acceptors. First, we describe the techniques that we compare in the experiments (Section 3.6.1). Then, we present the actual experimental results (Section 3.6.2).

3.6.1 Techniques for Comparison

We compared *Snowball* with two other techniques, a *Baseline* method and our implementation of *DIPRE*. These two methods require minimal or no training input from the user, and hence are comparable with *Snowball* in this respect. In contrast, state-of-the-art information extraction systems require substantial manual labor [YG98], or involve creating a hand-tagged training corpus [FSM+95]. Therefore, in this chapter we compare *Snowball* only with techniques that require a similar amount of human-driven training.[8]

- *Baseline*: The first method, *Baseline*, relies purely on the frequency of co-occurrence of organizations and locations in the documents. Specifically, *Baseline* selects the location that co-occurs in the same line with each organization most often as the headquarters for this organization. This simple method has been found useful in real applications [FAFL+02]. *Baseline* uses as input lines of text in the collection, tagged with the Alembic named entity tagger [DAH+97], and creates an index of the organizations and locations that occur in the same line. Then, *Baseline* simply selects the most frequent location for each organization. Despite its simplicity, the method works surprisingly well in this setting.

[8] In the next chapter, we will compare *Snowball* with a state-of-the-art manually tuned information extraction system.

- *DIPRE*: The second method is *DIPRE*, described in Section 3.2.1. We did not have access to its original implementation, so we had to re-implement it and adapt it to our setting. The original *DIPRE* implementation uses the prefix of the document URLs to restrict pattern generation and application. Since all of our documents came from just three sources, *DIPRE* was not able to exploit this feature. The second, more important modification had to do with the fact that *DIPRE* was designed to extract tuples from HTML-formatted data, which is inherently more structured than the plain text that we used for these experiments. Without HTML tags, *DIPRE* could not find occurrences of the seed tuples in plain text that were surrounded by exactly the same non-empty contexts. To solve this problem, we used the named-entity tagger to pre-tag the input to *DIPRE*. This way, all the organizations and locations were consistently surrounded by named-entity tags. *DIPRE* could then incorporate these tags as part of the surrounding context, and generate patterns that take advantage of these tags. Therefore, the results we report are *not* for the original *DIPRE* implementation. Rather, we evaluate our adaptation of *DIPRE* that takes advantage of the named-entity tags as described above.

- *Snowball*: The *Snowball* system as described in Section 3.2, with the *VS* acceptors of Section 3.2.1 and using *constraint*-based pattern evaluation. The *VS* acceptors use as features words (non-space character sequences) and punctuation.

- *Snowball-Plain*: The *Snowball* system of Section 3.2, with *VS* acceptors that only considers words as features for the acceptor and ignores everything else (e.g., punctuation, numbers, etc.). The intuition behind using *Snowball-Plain* is to reduce noise by not considering what are likely to be irrelevant features (e.g., the feature "," is just as likely to occur in a valid context as in an invalid context).

All variations of *Snowball* learned the extraction patterns from scratch, starting with just the seed tuples in Table 3.1 and using the operational parameters listed in Table 3.2 as the default setting for the experiments. The normalized *RlogF* metric was used to score patterns for selecting the seed tuples to keep from one iteration to the next. We will focus on the *Snowball* system with *VS* acceptors, and explore other parameters that are orthogonal

to the choice of the acceptor class. Namely, we will consider the number of iterations of *Snowball*, the effect of collection properties such as "redundancy", and the effect of varying the minimum tuple confidence. The goal of this section is to obtain an initial understanding of *Snowball* and of the efficacy of our evaluation methodology. In the next chapter we will perform a more comprehensive *Snowball* evaluation over a variety of other domains and relations.

3.6.2 Experimental Results

In this section, we present experimental results for *Snowball*, *DIPRE*, and *Baseline* over the news data set described in Section 3.4.1. Both *Snowball* and *DIPRE* rely on tuples appearing multiple times in the document collection at hand. To analyze how "redundant" the training and test collections are, we report in Table 3.3 the number of tuples in the *Ideal* set for each frequency level. For example, 5,455 organizations in the *Ideal* set are mentioned in the training collection, and 3,787 of these organizations are mentioned in the same line of text with their location at least once. So, if we wanted to evaluate how our system performs on extracting tuples that occur at least once in the training collection, the *Ideal* set that we will create for this evaluation will contain 3,787 tuples.

	Organization-Location Pairs	
Occurrence	*Training Collection*	*Test Collection*
0	5,455	4,642
1	3,787	3,411
2	2,774	2,184
3	2,094	1,561
4	1,669	1,177
5	1,321	909
6	1,082	746
7	907	607
8	780	512
9	675	445
10	593	389

Table 3.3: Occurrence statistics of the *Ideal* tuples in the training and test collections.

The first row of Table 3.3, corresponding to zero occurrences, deserves further explanation. If we wanted to evaluate the performance of our system on *all* the organizations

that were mentioned in the corpus, even if the appropriate location never occurred near its organization name anywhere in the collection, we would include all these organizations in our *Ideal* set. So, if the system attempts to "guess" the value of the location for such an organization, any value that the system extracts will automatically be considered wrong in our evaluation.

The accuracy results for extracting *CompanyHeadquarters* using the *Ideal* metric are reported in Figure 3.3. The plot shows the performance of the systems as we attempt to extract test tuples that are mentioned more times in the corpus. As we can see, *Snowball* performs increasingly well as the number of times that the test tuples are required to be mentioned in the collection increases. *DIPRE* has better precision than *Snowball* at the 0-occurrence level (72% for *DIPRE* vs. 67% for *Snowball*). As we discussed, *DIPRE* requires the *exact* string match of a pattern with a text context for a tuple to be extracted. This technique is less likely to match spurious contexts, but may also miss valid contexts, resulting in relatively low recall. For all occurrence levels, *Snowball* has substantially higher recall than *DIPRE* and *Baseline* do. We also observe that punctuation matters. The recall of *Snowball-Plain* is lower than that of *Snowball* while precision is also somewhat decreased.

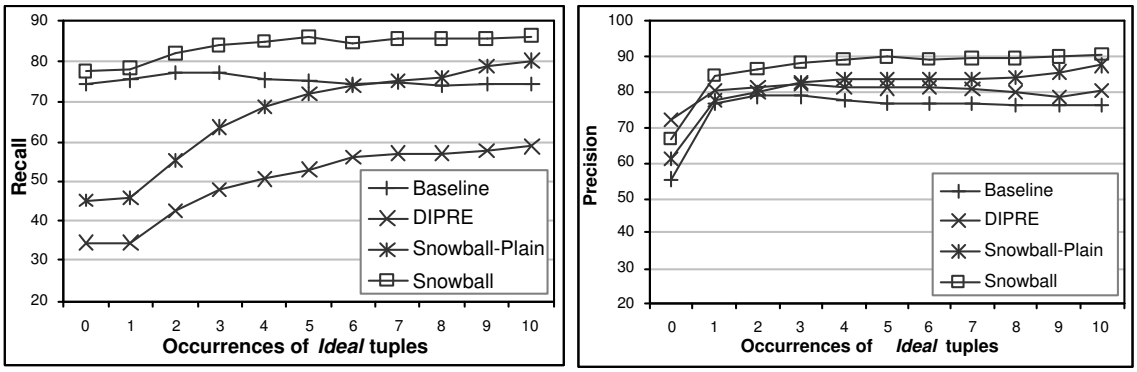


Figure 3.3: Recall (a) and precision (b) of *Baseline*, *DIPRE*, *Snowball*, and *Snowball-Plain* as a function of the number of occurrences of the *Ideal* tuples (test collection).

Figure 3.4 shows that *Snowball*'s results are stable over subsequent iterations of the algorithm. In contrast, *DIPRE* quickly diverges, since it has no way to prevent unreliable tuples from being seed for its next iteration. We report data for only two iterations for

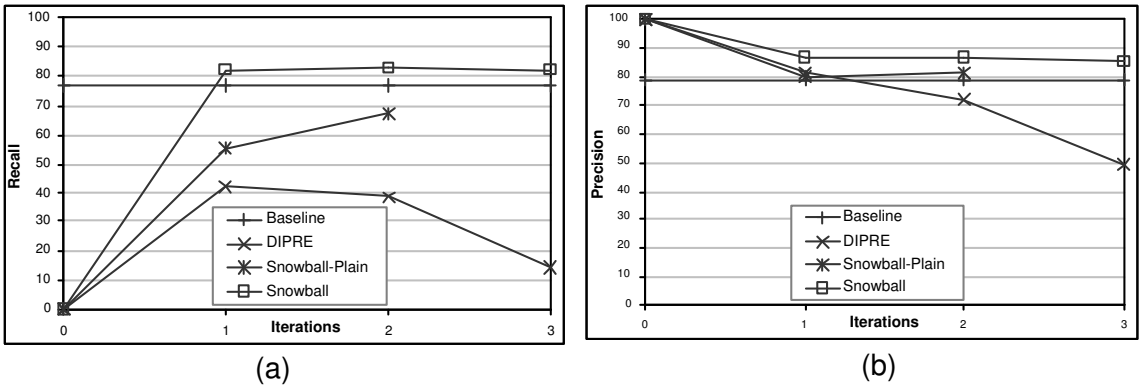


Figure 3.4: Recall (a) and precision (b) of *Baseline*, *DIPRE*, *Snowball*, and *Snowball-Plain* as a function of the number of iterations (*Ideal* tuples with at least 2 occurrences; test collection).

			Type of Error			
	Correct	*Incorrect*	*Location*	*Organization*	*Relationship*	P_{Ideal}
DIPRE	74	26	3	18	5	90%
Snowball (all tuples)	52	48	6	41	1	88%
Snowball ($\tau_t = 0.8$)	93	7	3	4	0	96%
Baseline	25	75	8	62	5	66%

Table 3.4: Manually computed precision estimate, derived from a random sample of 100 tuples from each extracted table.

Snowball-Plain because it converged after the second iteration (i.e., it did not extract any new tuples).

As discussed in Section 3.4.3, we complete our evaluation of the precision of the extraction systems by manually examining a sample of their output. For this, we randomly selected 100 tuples from each of the extracted tables, and checked whether each of these tuples was a valid organization-location pair or not. We separate the errors into three categories: errors due to mis-tagging a location and assigning it to a valid organization ("Location" error), errors due to including a non-existing organization ("Organization" error), and errors due to deducing an incorrect relationship between a valid organization and

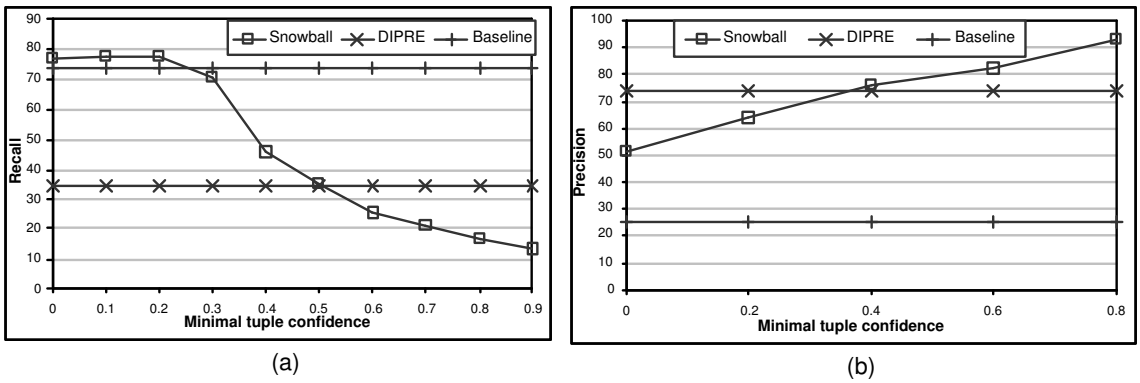


Figure 3.5: Recall (a) and *sample-based* precision (b) as a function of the threshold τ_t used for the last-step pruning of the *Snowball* tables (*Ideal* tuples with at least one occurrence; test collection).

location ("Relationship" error). These different types of errors are significant because they highlight different "culprits": the "Location" and "Organization" errors could be prevented if we had a perfect named-entity tagger, whereas the "Relationship" errors are wholly the extraction system's fault (Table 3.4).

The last column in Table 3.4 (P_{Ideal}) is precision, calculated by ignoring the "Organization" errors and computing the fraction of valid organizations for which a correct location was found. These values correspond to the values of precision we would have calculated if our *Ideal* table included all the valid organizations in the random samples. These figures, however, do not capture invalid tuples generated due to improper tagging of a string as an organization. From our manual inspection of a random sample of 100 tuples from each extracted table, we observed that *DIPRE*'s sample contained 74 correct tuples and 26 incorrect ones. *Snowball*'s sample contained 52 correct tuples and 48 incorrect tuples, while *Baseline* contained 25 valid and 75 invalid tuples. As we can see from the breakup of the errors in the table, virtually all of *Snowball*'s errors are tagging related (i.e., "Location" or "Organization" errors). If we prune *Snowball*'s final output to only include those tuples t with $Conf(t) \geq 0.8 = \tau_t$, then most of these spurious tuples disappear. A random sample of 100 tuples from this pruned table contained 93 valid tuples and only 7 invalid ones. Furthermore, none of the invalid tuples are due to "Relationship" errors (third row

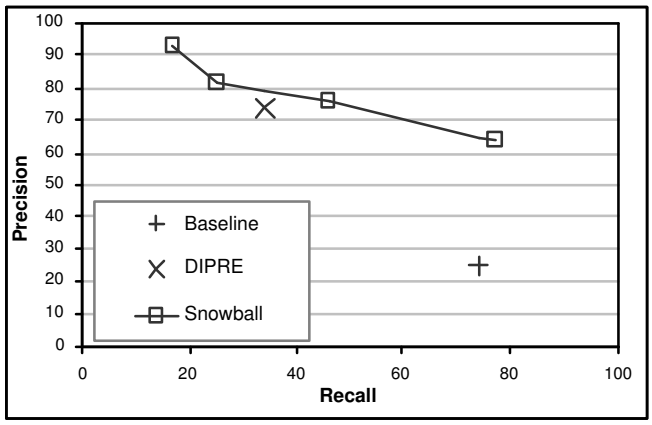


Figure 3.6: *Sample-based* precision vs. recall of *Baseline*, *DIPRE*, and *Snowball* (*Ideal* tuples with at least one occurrence; test collection).

of Table 3.4).

So far, the results that we have reported for *Snowball* are based on a table that contains all the "candidate" tuples generated during *Snowball*'s last iteration. As we saw in Table 3.4, the precision of *Snowball*'s answer varies dramatically if we prune this table using the tuple confidence threshold τ_t. Of course, this last-step pruning is likely to result in lower recall values. In Figure 3.5, we explore the tradeoff between precision and recall for different values of this last-step pruning threshold for *Snowball*, *DIPRE*, and *Baseline*. (We do not include results for *Snowball-Plain* as it behaves essentially the same as *Snowball*, but with lower recall.) A user who is interested in high-precision tables might want to use high values for this threshold, while a user who is interested in high-recall tables might want to use lower values of the threshold. For example, by setting $\tau_t = 0.4$ and filtering the *Extracted* table accordingly, we estimate the absolute precision of *Snowball*'s output to be 76% and recall to be 45%, both of which are higher than the corresponding metrics of *DIPRE*'s output.

The experimental results are summarized in Figure 3.6, which plots precision at each value of recall for each system. To produce this plot, we first sorted the extracted tuples in order of decreasing confidence and then computed precision and recall at the τ_t values of $0.9, 0.8, \ldots, 0.2$[9]. We report precision for each recall value. This plot is essentially a differ-

[9]Since the recall at $\tau_t = 0.2$ is the same as recall at $\tau_t = 0.1$ but the former has higher precision, by

ent way to present the results in Figure 3.5, providing insight into the precision-vs.-recall tradeoff. As *DIPRE* does not define a confidence for a tuple, there is no natural way to rank the extracted tuples. Similarly, by definition, *Baseline* returns the one most frequent location that co-occurs with an organization, and, as is, does not provide a confidence score either. (In the next chapter, we will present an extension of *Baseline* that provides a confidence score for each tuple.) While *Baseline* has high recall and low precision, and *DIPRE* has high precision and relatively low recall, the precision of *Snowball* gracefully degrades for increasing recall values. In summary, both *Snowball* and *DIPRE* show substantially higher precision than *Baseline*. In effect, *Baseline* tends to generate many tuples, which results in high recall at the expense of low precision. *Snowball*'s recall is at least as high as that of *Baseline* for most of the tests, with higher precision values. *Snowball*'s recall is generally higher than *DIPRE*'s, while the precision of both techniques is comparable.

3.7 Discussion

In this chapter, we presented *Snowball*, a system for extracting relations from large plain text collections. *Snowball* requires minimal training for each new scenario. Our contributions include novel strategies for automatically generating extraction patterns, a general method for automatic pattern and tuple evaluation, and a scalable experimental evaluation methodology that we used in an empirical evaluation of *Snowball* and related techniques.

We presented techniques for generating flexible and accurate extraction patterns by analyzing the text contexts around the occurrences of the initial seed tuples. Among the techniques we considered, the vector space model we adapted from information retrieval is the most successful, and appears to be the most promising for new extraction tasks. We showed that we can exploit integrity constraints on the relation, e.g., the key constraint, where a tuple is considered invalid if it violates any of the provided integrity constraints. This allows *Snowball* to reliably select seed tuples to start a new *Snowball* iteration, and to gracefully trade-off recall for precision. We introduced as well a more generally applicable technique for automatic *constraint-based* pattern and tuple evaluation, which allows *Snow-*

convention we report the precision at the higher value of τ_t.

ball to assign a confidence score to each extracted tuple. We also presented a more generally applicable EM-based technique for pattern evaluation. Our approach is based on the observation that good extraction patterns will match many known valid tuples, and few invalid tuples.

Evaluating a system like *Snowball* over a large text collection is not an easy task. We presented a scalable evaluation methodology based on the creation of the *Ideal* table from external data sources. Using the *Ideal* table we can automatically estimate the precision and recall of the extracted relation. Our large-scale experimental evaluation shows that *Snowball* can produce high-quality tables. Our experiments involved over 300,000 real newspaper articles.

In the next chapter, we describe techniques for adapting *Snowball* to new extraction tasks, including methods for automatically estimating many of the *Snowball* parameters to additionally reduce the amount of tuning required for porting *Snowball* to new domains. We will also complete the evaluation of *Snowball* by considering a number of new extraction tasks, and exploring more fully the *Snowball* variations that we have presented in this chapter. As we will see, our automatic tuning techniques will successfully discover values for some of the necessary *Snowball* parameters. Overall, we will see that *Snowball* performs competitively with state-of-the-art information extraction systems.

Chapter 4

Tuning *Snowball* for New Relations

In the previous chapter, we presented the *Snowball* system for extracting structured relations from unstructured text with minimal human participation. *Snowball* is initialized with a handful of seed tuples for a relation, and automatically generates extraction patterns to produce more relation tuples. Additionally, we introduced techniques for automatically evaluating patterns, which allow *Snowball* to assign a confidence score to the extracted tuples. We also presented a scalable evaluation methodology and metrics specifically tailored for the relation extraction task. In order to make the extraction problem more tractable, the previous chapter relied on a number of assumptions. We also made choices about the *Snowball* parameters based on a training collection for each new relation. We now turn to the important problem of adapting *Snowball* for new scenarios with as little manual re-tuning as possible, as well as to reporting a comprehensive experimental evaluation of *Snowball*. We show that we can obtain acceptable extraction accuracy using *Snowball* with minimal parameter tuning for new scenarios. To achieve this goal, we will present techniques for estimating some crucial *Snowball* parameter values at run time, embodied in our new, adaptive version of *Snowball*, which we call *Snowball-Auto*. As part of our empirical evaluation, we will use *Snowball-Auto* for extracting relations in the medical domain with no domain- or relation-specific tuning.

The contributions of this chapter include:

- A systematic and principled exploration of the *Snowball* design decisions and parame-

ter space (Sections 4.1 and 4.2).

- Automatic parameter estimation techniques that reduce *Snowball*'s dependence on heuristics and manual tuning, and significantly ease the process of adapting *Snowball* to new domains (Section 4.3).

- A comprehensive evaluation of *Snowball* and other extraction systems over a variety of domains and collections (Sections 4.4 and 4.5).

- A discussion of the applicability and limitations of *Snowball* and related techniques (Section 4.6).

The rest of this chapter is organized as follows. We first review the relevant *Snowball* parameters used in the previous chapter and their values (Section 4.1). Then, we revisit and empirically validate some of the design decisions for *Snowball* that we have not yet explored in depth –namely, the features and the feature selection algorithms to use for representing extraction patterns, the number and composition of the seed tuples, and the techniques for filtering "outliers" during the pattern generation step of *Snowball* (Section 4.2). Surprisingly, some mundane parameters, such as the size of the text context to consider, can have a significant effect on accuracy. Therefore, adapting *Snowball* to a new domain can require substantial tuning to find a good combination of parameter values for each collection and relation. Furthermore, the tuning performed over a training collection may not provide acceptable extraction accuracy on a different test document collection. To address this problem, we present techniques for automatically estimating some important *Snowball* parameters that can vary for each relation and document collection (Section 4.3). We evaluate the *Snowball* variations over four diverse relations extracted from the news and the medical literature (Section 4.4). Our experiments show that our parameter estimation techniques, and domain-independent pattern and tuple evaluation methods allow *Snowball* to perform competitively on some extraction tasks with very little relation– or collection–specific tuning (Section 4.5). We conclude the chapter with a discussion of the applicability and limitations of *Snowball* based on this evaluation and other experiments (Section 4.6).

4.1 The *Snowball* Parameter Space

As we discussed, *Snowball* relies on a number of parameters (Section 3.5) that control the *Snowball* behavior and may need to be tuned for each relation or collection of interest. The most important parameters are summarized below, together with the manually derived values used in the experiments of the previous chapter. Note that these parameter values were tuned specifically for extracting the *CompanyHeadquarters* relation from the training document collection of Chapter 3. These parameters include:

- *Number of seed tuples NumSeed*: The number of seed tuples to initialize *Snowball* was previously set to 5 for extracting the *CompanyHeadquarters* relation.

- *Minimum seed confidence τ_{seed} and minimum tuple confidence τ_t*: These parameters specify the minimum confidence to include a tuple as seed for the next *Snowball* iteration, and the minimum confidence to keep a tuple altogether. Our tuning suggested the values of 0.8 and 0.2 for these parameters, leaving only a small fraction of "reliable" tuples as seeds for the next iteration.

- *Minimum clustering similarity τ_{sim} and minimum extraction similarity τ_{score}*: These parameters are the minimum values of the *Sim* and *Match* functions for clustering example contexts, and matching text contexts to patterns, respectively. Both parameters were set to 0.6.

- *External and internal context sizes WindowSize* and *MaxProximity*: The maximum length (in words or characters) of the internal and external text spans. *WindowSize* was set to two words, and *MaxProximity* was unlimited as long as the related entities all appeared within the same line in the original document.

- *Relative span weights* $W_1, \ldots, W_{n+1}$: The relative "importance" of each text span. W_1 and W_{n+1} were set to 0.2, while $W_2, \ldots, W_n$ were set to 0.6 for the *CompanyHeadquarters* relation. While the experimental results of the previous chapter confirmed our intuition that the *Middle* text span was more important than the *Left* and *Right* contexts, these parameter values may not be appropriate for other scenarios.

As we discussed, *Snowball* relies on a number of seed tuples to start the extraction process. The quality and number of the seed tuples in turn can affect the overall extraction accuracy. Our earlier experiments in Section 3.4 assumed that the user provided a handful of "good" seed tuples (e.g., only five seed tuples were needed to extract the *CompanyHeadquarters* relation). Intuitively, the number of easily available seed tuples varies with the extraction task. In the next section, we will explore how the size and composition of the seed tuples affects the *Snowball* accuracy.

The parameters affecting the behavior of *Snowball* listed above fall into four categories: (1) selecting seed tuples to initialize the extraction process, (2) selecting the text contexts to consider for extracting new tuples, (3) choosing features to represent the text contexts, and (4) setting values of the *Snowball* operating parameters such as τ_{seed}. The first three categories are essentially *design* decisions, and we will explore these decisions in the next section. As for the last category, we will present techniques for automatically estimating the values of the *Snowball* operating parameters in Section 4.3.

4.2 Exploring *Snowball* Design Decisions

We now explore the system design decisions made implicitly in the *Snowball* implementation and evaluation described in the previous chapter. Specifically, we will choose the number of seed tuples *NumSeed*, the features (and feature classes) to use for representing patterns, and, finally, we will compare the efficacy of the pattern evaluation strategies of Section 3.2.1.

For this study, we will experiment with two target relations and associated document collections. The reason we use two different relations and document collections is to highlight the importance of techniques for automatically tuning *Snowball* to avoid re-tuning *Snowball* for every relation and collection individually.

- *CompanyHeadquarters(Organization:ORGANIZATION, Location:LOCATION)*: extracted from 40,000 New York Times articles selected from the *training* collection described in Section 3.4.1 [1].

[1] We used a subset of the original *training* collection for rapid experimentation.

- *MergersAcquisitions(Buyer:ORGANIZATION, Target:ORGANIZATION)*: extracted from 40,000 Wall Street Journal articles selected from the TIPSTER Complete collection, available from LDC[2].

Note that the *MergersAcquisitions* relation is a simplified version of a similar task that was used in the past in the MUC evaluation [Rep93]. As shown previously by others, even the simplified version of *MergersAcquisitions* is relatively "hard" for machine learning-based techniques (e.g., see [Fre98]). To evaluate the quality of the *Snowball* output for these relations, we used the evaluation methodology and metrics described in Section 3.4. As a departure from the automatic *Ideal* construction algorithm of Section 3.4.2, we manually tagged a sample of 65 documents for each relation, half of which were a random sample of the collection, and the other half were retrieved by intuitive queries (e.g., "headquarters AND located" for the *CompanyHeadquarters* relation, and "acquired" for the *MergersAcquisitions* relation). The manually tagged tuples were then inserted into a structured table, which became the "gold standard" for the evaluation. The reason for this change in evaluation methodology (i.e., creating the *Ideal* table manually) was that we did not have available a sufficiently large table with company mergers and acquisitions to create *Ideal*. For all of the experiments of this section, we used *Snowball* with *VS* acceptors, and, unless otherwise indicated, the parameter values specified in Table 3.2.

The rest of this section is organized as follows. First, we experiment with varying the number and composition of the seed tuples that *Snowball* needs to start the extraction process (Section 4.2.1). Then, we explore the effect of varying the choice of features for representing extraction patterns (Sections 4.2.2 and 4.2.3). Finally, we empirically study the choice of the pattern evaluation strategy (Section 4.2.4).

4.2.1 Choice of Seed Tuples

Snowball relies on the provided *seed* tuples to start the extraction process. The quality and number of the seed tuples can affect the accuracy of the extracted relation. Our earlier experiments in Section 3.4 relied on the user providing a handful of "good" seed tuples, with

[2] `http://www.ldc.upenn.edu`

the seed tuples chosen essentially by trial and error for the *CompanyHeadquarters* relation. Until now, we did not experiment with the choice of seed tuples in a controlled way. To reduce any potential experimental bias introduced by manually choosing seed tuples, our experiments on the number of seed tuples start with the seed tuples picked randomly from the *Ideal* table of each relation. This methodology models the realistic situation when we do not know a-priori the set of "good" seed tuples to initialize *Snowball* for extracting a new relation. Hence, starting with a set of tuples drawn at random from the *Ideal* table, with no preference given to frequent or "good" tuples, is a more rigorous test of the *Snowball* system.

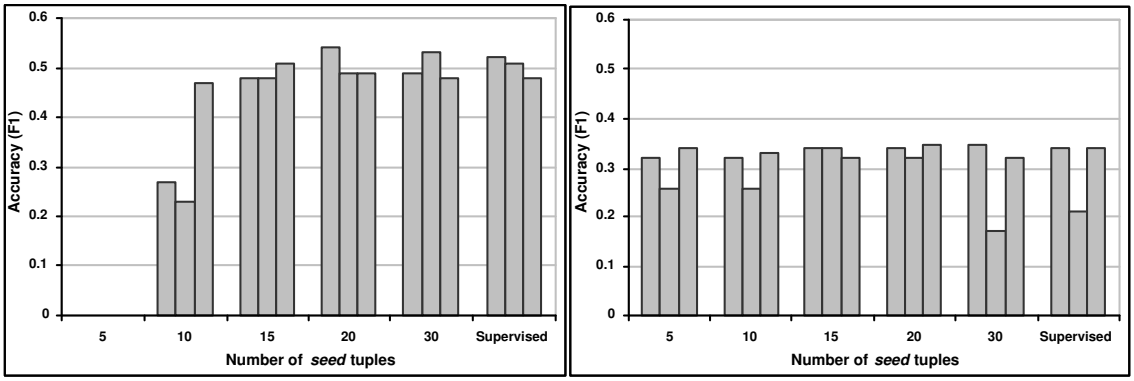


Figure 4.1: Accuracy (F-Measure) of the extraction of the *CompanyHeadquarters* (a) and *MergersAcquisitions* (b) relations for varying number of *seed* tuples *NumSeed* (three independent random samples).

Figure 4.1 reports the highest *F-measure* (defined as $\frac{2PR}{P+R}$ where P is precision and R is recall), obtained at the recall/precision break-even point achieved by *Snowball* for varying *NumSeed* values (all seed tuples drawn at random from *Ideal*). The test tables were created for each run as all of the tuples in *Ideal* except for the seed tuples, hence the "training" (seed) tuple sets were disjoint from the remaining "test" tuple sets[3]. For these and all the subsequent experiments, three samples of each size were taken.

Intuitively, when we increase *NumSeed*, we should observe an increase in the accuracy of *Snowball*. This intuition holds true for the *CompanyHeadquarters* relation (Figure 4.1(a)),

[3]Variations of the same tuple present in *Ideal* were treated as distinct tuples for all purposes.

but the accuracy does not increase for more than 15 randomly chosen seed tuples. Similarly, for the *MergersAcquisitions* relation, providing more than 15 randomly drawn seed tuples does not increase *Snowball*'s accuracy. In general, providing a large seed set puts a larger burden on the user. Therefore we want to start with as few seed tuples as possible without sacrificing the extraction accuracy. Determining such an optimum number of seed tuples automatically is difficult, and so we will heuristically set *NumSeed* to 15 based on the empirical evidence in Figure 4.1. We will use this setting for all of the subsequent experiments. In practice, requesting a user to provide 15 seed tuples appears to require only a relatively minor time investment, preserving the main *Snowball* feature of requiring minimal human effort.

4.2.2 Feature Selection

We now turn to the problem of deciding which example text contexts to use for generating extraction patterns, and which contexts to ignore as "noise." As we described in Section 3.2, the first step of the *Snowball* pattern generation process is finding occurrences of seed tuples in the text. Sometimes, attribute values of seed tuples co-occur in a document in non-representative ways or "by chance." We want to ignore such example contexts as they can lead to bad extraction patterns. We use the following procedure to identify outlier text contexts.

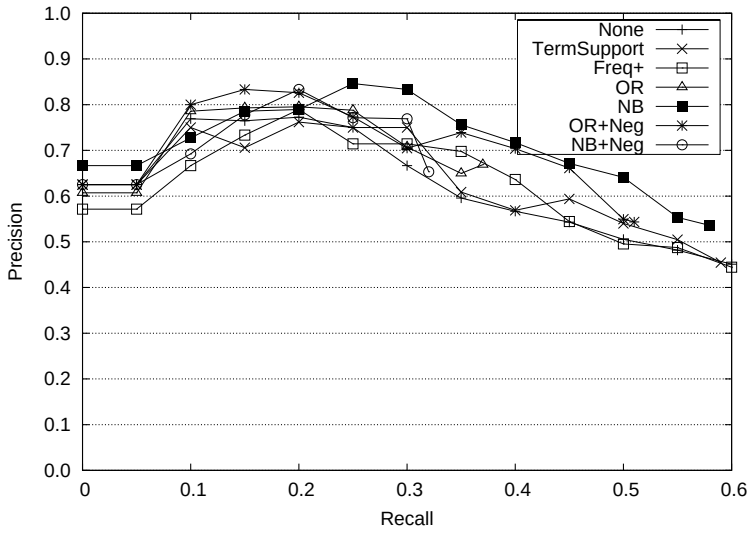
For each example context C^i_{EX} associated with a seed tuple, we compute the distance d^i_{nn} to the "nearest" example context using the *Sim* function defined in the previous chapter. This gives us the distribution of the "nearest neighbor" distances for the example text contexts. By considering this distribution, we can find "outliers" (i.e., example contexts that have no close "neighbors"). Specifically, we compute the mean (μ) and the standard deviation (σ) of the nearest neighbor distribution above. Any example context where d^i_{nn} is larger than $\mu + k \cdot \sigma$ is considered to be an "outlier" and is discarded. Based on our tuning experiments, we set $k = 2$. Note that because we are primarily interested in large collections (which often contain redundant information), discarding these outlier contexts from consideration should not adversely affect the coverage of the resulting extraction patterns.

Many of the features that occur in example text contexts may not be useful for dis-

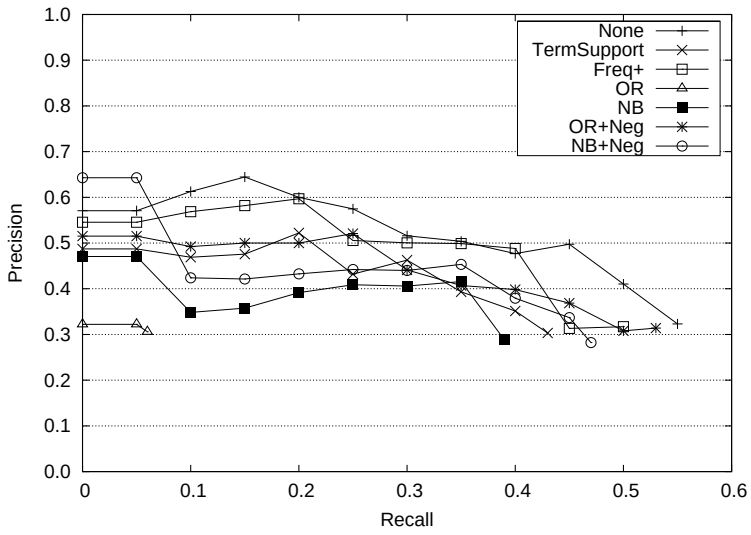
criminating between valid and invalid text contexts. By including extraneous features in the patterns we may match additional text contexts and potentially extract invalid tuples. Furthermore, keeping all features that appear in the example contexts can result in an extremely high dimensionality of the resulting feature space, which would tend to decrease the effectiveness of most classifiers. We start by considering all words and punctuation as candidate features, which could potentially result in a large feature space. To reduce this space, we exploit a variety of sophisticated feature selection –and weighting– techniques developed over the years in the machine learning and information retrieval communities, as outlined below:

- *OR*: Keep the features with the highest "Odds-ratio" weights from information retrieval [YP97, Mla98].
- *NB*: Keep the features with the highest Naive-Bayes weights [Mit97].
- *Freq+*: Keep the features with the highest positive frequency (i.e., the features that are most frequent within valid contexts).
- Term support: Keep the features that appear at least τ_{sup} times in the example contexts.
- *NB+NEG, OR+NEG, etc.*: Keep a mix of most strongly positive and most strongly negative features as proposed in the "balanced" feature selection scheme of [ZWS04].

We report results of our feature selection experiments in Figure 4.2 using the feature sets used in the previous chapter (namely, words and punctuation). Surprisingly, using all features (no feature selection, or *None* in the plot) results in the highest accuracy for the *MergersAcquisitions* relation. Unfortunately, this choice would not scale as the vocabulary of the text contexts grows. Among the feature selection methods, *NB* performed reasonably well for our problem. While *NB* is not a top performer for *MergersAcquisitions*, *NB+NEG* performs well, suggesting that *NB* is also a promising strategy. Furthermore, *NB* classifiers, while simple, have been extensively studied and perform competitively on many classification tasks. For these reasons, we will use the *NB* feature selection scheme for subsequent experiments.



(**a**)



(**b**)

Figure 4.2: Recall vs. precision of *Snowball* for different feature selection techniques for extracting *CompanyHeadquarters* (a) and *MergersAcquisitions* (b).

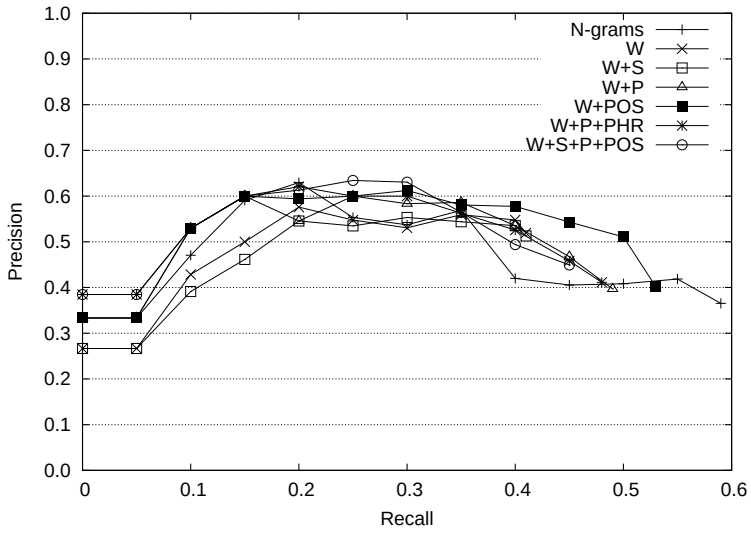
Some features are undoubtedly more important than others. For example, the word "based" occurring in a *Middle* context is a strong indication that the corresponding candidate tuple for *CompanyHeadquarters* is valid, whereas the presence of the feature "," is not as valuable. We considered the following alternatives for feature weighting: term frequency (TF), Naive-Bayes weights (described above), TF·IDF weights from information

retrieval [SB88], and a variation of these weights, TF·NB weights (where TF is the term frequency from information retrieval and NB is the Naive-Bayes feature weight described above). In all cases, the goal is to assign higher weight to the features that occur in valid contexts (but not in invalid ones), and to assign lower weights to the features that tend to occur in both valid and invalid contexts. We have experimented with assigning a different weight to the features instead of keeping the original feature selection score. Our experiments suggested that keeping the original (feature selection) weights was the most effective alternative from both design and accuracy perspectives. Consequently, we assign each feature a weight that is equal to the conditional probability of the context being valid if it contains the feature. This probability is estimated using the Naive Bayes formula of Section 3.2.1.

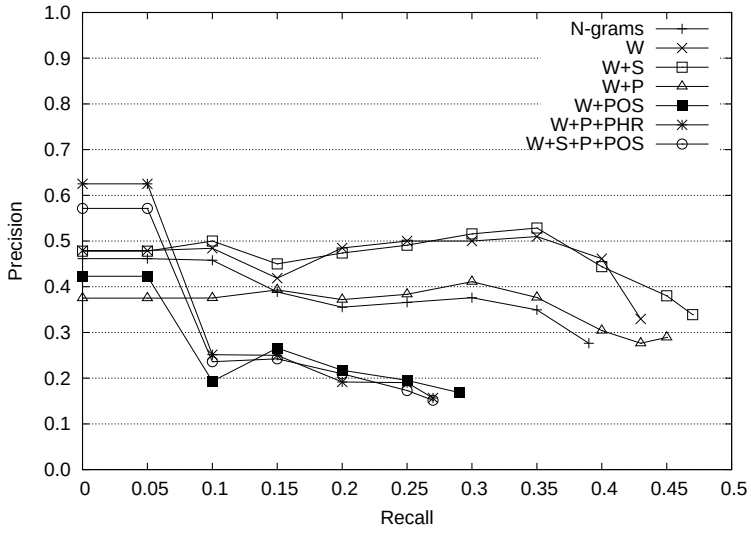
4.2.3 Choice of Features and Context Representation

Now we turn to the actual features used to represent the text contexts and the corresponding extraction patterns. Choosing the feature set is one of the most important decisions when designing an information extraction system. Strategically, we chose not to use linguistically-rich features (e.g., syntactic information) in order to keep *Snowball* as general as possible (depending heavily on standard English grammar or lexicon could prevent us from processing documents on the web or in highly technical domains). Therefore, we decided to use simple and domain-independent features, namely non-space terms (W) (e.g, "-based"), *N*-grams (e.g., "bas", "ase", "sed" for "based"), word stems (S) (e.g., "locat" for "located"), phrases (PHR) (e.g., "based in"), part-of-speech (POS) tags (e.g., "NOUN"), and punctuation (P).

We report the recall vs. precision results for each feature set (and some promising combinations) in Figure 4.3. Interestingly, the effectiveness of features varies with the relation. For *CompanyHeadquarters*, the best feature set is the combination of words and POS tags, while for *MergersAcquisitions*, the best features are words and word stems. We believe that the *MergersAcquisitions* relation is representative of the "interesting" relations we would like to extract. Therefore we will restrict the feature sets to words and word stems for the subsequent experiments. Note that while punctuation was found to be helpful for the



(**a**)



(**b**)

Figure 4.3: Recall vs. precision of *Snowball* when using combinations of N-grams, words (W), word stems (S), punctuation (P), phrases (PHR), and part-of-speech tags (POS) for extracting *CompanyHeadquarters* (a) and *MergersAcquisitions* (b).

CompanyHeadquarters experiments of the previous chapter, our feature selection techniques usually discarded punctuation as not discriminative.

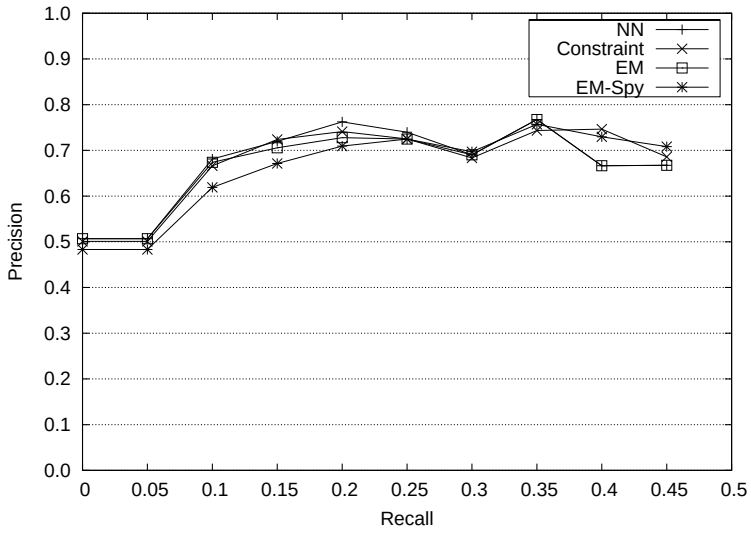
4.2.4 Choice of Pattern Evaluation Strategy

As we discussed in Section 3.3, some patterns (e.g., ⟨ [], *ORGANIZATION*, [(“, ”, 1)], *LOCATION*, [] ⟩ for the *CompanyHeadquarters* relation) are not accurate and tend to extract invalid tuples. In Section 3.3, we presented a technique for automatically evaluating these patterns based on our *constraint-violation* heuristic. Unfortunately, such constraints are not always available. For example, relations containing *drug interactions* can be “many-to-many” and may not have any constraints that we could exploit. To address this problem, we adapted a *partially supervised* technique from text classification to our relation extraction problem in Section 3.3.2.

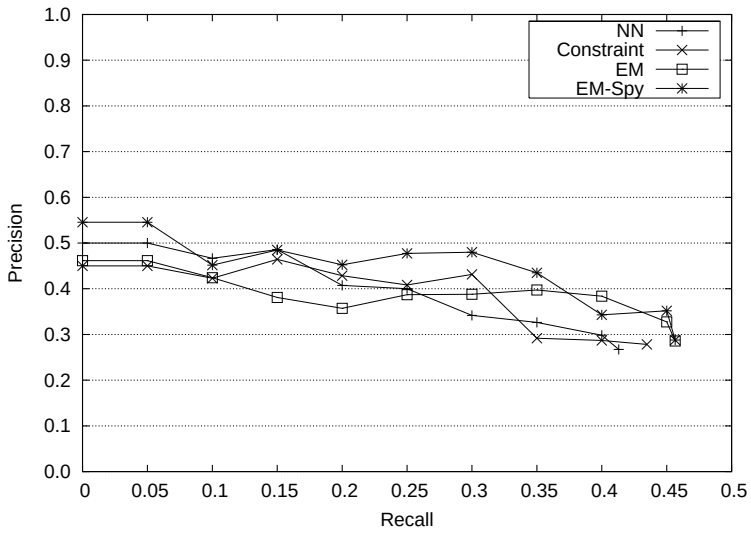
We now empirically evaluate the alternatives for automatically evaluating extraction patterns. We used the *recall* and *precision* metrics of Section 3.4 and the *Ideal* table constructed manually as described above. For all pattern evaluation techniques, we compute tuple confidence as described in Section 3.2.2. Specifically, we compare four methods of automatically evaluating extraction patterns:

- *NN*: All patterns are assigned the same score of 1, and the score of a text context is determined by the *Match* of the text context and its closest pattern.
- *Constraint*: This is the constraint violation heuristic described in Section 3.3. The score of each pattern is proportional to the fraction of “positive” matches (known seed tuples) and “negative” matches (tuples that violate integrity constraints).
- *EM*: This is the *ScorePatternsEM* algorithm presented in Algorithm 3.2.
- *EM-Spy*: This is the *ScorePatternsEM-Spy* algorithm presented in Algorithm 3.3.

As we can see in Figure 4.4, a pattern scoring method can noticeably affect the accuracy of an extracted relation. When the relation has available integrity constraints (e.g., *CompanyHeadquarters*), the *Constraint* method performs competitively with *EM* and *EM-Spy*. Interestingly, even the simplest *NN* method is competitive on *CompanyHeadquarters*, suggesting that pattern scoring is less important for “easy” relations. In contrast, on extracting *MergersAcquisitions*, *EM* and *EM-Spy* have substantially higher accuracy than other pattern evaluation methods (Figure 4.4(b)).



(**a**)



(**b**)

Figure 4.4: Recall vs. precision of *Snowball* using *NN*, *Constraint*, *EM*, and *EM-Spy* pattern scoring methods for extracting *CompanyHeadquarters* (a), and *MergersAcquisitions* (b).

Having revisited the *Snowball* design decisions in a systematic manner, we now present our techniques for automatically estimating the remaining *Snowball* operating parameters. In principle, our estimation techniques could allow *Snowball* to operate essentially "out-of-the-box" for a new target relation and a document collection of interest.

4.3 Automatic Parameter Estimation

In the last section, we have described important design decisions for implementing *Snowball*. Additional operating parameters that are needed by *Snowball* are listed and explained in Section 3.5. Choosing the right combination of parameter values can be tricky and time consuming. In this section, we address this problem by proposing techniques for *automatically* estimating *Snowball* parameter values for a given target relation and document collection. The techniques presented in this section can dramatically reduce the amount of manual tuning required to adapt *Snowball* for extracting new relations over new document collections.

Algorithm EstimateSnowballParameters(*Seed, Documents*)

(1) *MaxProximity* = estimateInternalContextSize(*Seed, Documents*)
(2) *initialTextContexts* = generateTextContexts(*Seed, Documents, MaxProximity*)
(3) *sizedTextContexts* = estimateExternalContextSize(*initialTextContexts*)
(4) *filteredTextContexts* = doFeatureSelection(*sizedTextContexts*)
(5) $W_1, \ldots, W_{n+1}$ = estimateContextWeights(*filteredTextContexts*)
(6) τ_{sim} = estimateClusteringSimilarity(*filteredTextContexts*)
(7) $\tau_{score} = \frac{1}{2} \cdot \tau_{sim}$

Algorithm 4.1: Estimation of *Snowball* Operating Parameters Algorithm

Our approach for automatically tuning *Snowball* depends on the main bootstrapping assumption of Section 3.1, namely that the seed tuples are representative of the other tuples in the target relation, so that the as-yet undiscovered valid tuples will appear in similar contexts as the seed tuples. Therefore, we can estimate many of the *Snowball* parameters by analyzing the text contexts where the seed tuples occur. The overall parameter estimation procedure is outlined in Figure 4.1.

An important parameter for relation extraction is the size of the text context around the entities to consider, and the maximum proximity of the tuple attributes in the text for which we still consider the attributes to form a valid tuple occurrence. We conjecture that entities for different relations will tend to occur in the text with different maximum proximity (e.g., a larger text context may be needed to reliably detect the *MergersAcquisitions* relation compared to the *CompanyHeadquarters* relation). Therefore, in Step (1)

we estimate *MaxProximity*, the maximum distance –in characters– that can separate the leftmost from the rightmost entity in a candidate tuple. As discussed earlier, this parameter captures the textual "locality" of the target relation. Intuitively, we will assume that seed tuples for a given relation and collection should occur in similar contexts (i.e., within a similarly-sized text fragment). Using the example contexts for the provided seed tuples, we can compute the distribution of proximity values between the leftmost and the rightmost entity in each text context. By using the average μ and the standard deviation σ of the distribution, we can discard all "outlier" contexts that have proximity greater than $\mu+2\cdot\sigma$. Therefore, we estimate *MaxProximity* as $\mu + 2 \cdot \sigma$. Any example context with proximity greater than *MaxProximity* is considered an outlier and is discarded.

In the next step, Step (2), we generate a vector representation of the example contexts in order to estimate *WindowSize*, the number of words to consider for the *Left* and the *Right* text spans. We estimate the *WindowSize* parameter in Step (3) as follows. We repeatedly shorten the *Left* and the *Right* contexts by one word, and re-compute *Coherence* of all of the example text contexts, defined as the sum of pairwise similarities between the respective *Left* and *Right* text spans across all available example text contexts, when up to *WindowSize* words are considered. For example, *Coherence(Left)* at *WindowSize*=3 is computed as $\sum_i \sum_j Sim(C^i_{EX}, C^j_{EX})$, or the sum of the pairwise similarities of all the rightmost three words across all available *Left* text spans. Intuitively, there should be a setting of *WindowSize* that maximizes the similarity between all corresponding text contexts. We conjecture that this value is a good indicator of the size of the external text context to consider for the given relation and document collection. As a result, the value of *WindowSize* with the highest computed *Coherence* value is chosen.

In Step (4), we perform feature selection to reduce the dimensionality of the example contexts, to make them more manageable, using the *NB* feature selection of Section 4.2. Intuitively, the contexts that are most uniform between the example contexts would be the most reliable indicators of validity of the text contexts for the relation (and hence, the validity of the resulting candidate tuple). Therefore, we estimate relative context weights as the relative *Coherence* of the text span compared to the remaining spans of the example context. For example, the relative weight W_1 for the *Left* span for the *CompanyHeadquarters*

relation is estimated as:

$$W_1 = \frac{Coherence(Left)}{Coherence(Left) + Coherence(Middle) + Coherence(Right)}$$

```
Algorithm EstimateClusteringSimilarity(textContexts, K, δ)

  bestError = bestSimilarity = 0;
  foreach sim in δ, ..., 1; sim = sim + δ do
      currentError = 0;
      foreach c in 1...K; c = c + 1 do
          TrainingSet = select random sample of size 2/3 · |textContexts|;
          TestSet =textContexts - TrainingSet;
          Clusters = bucketAverageCluster( TrainingSet);
          foreach context in TestSet do
              currentError = currentError + NearestClusterDistance(Clusters, context)
          end
      end
      if currentError < bestError then
          bestSimilarity = sim; bestError = currentError;
      end
  end
  return bestSimilarity;
```

Algorithm 4.2: Clustering Similarity Estimation Algorithm

Having estimated the relative weight of each text span, we can now compute the similarity of two example contexts, and so we can estimate the minimum clustering similarity τ_{sim} for grouping together similar example text contexts (Step (6)). The procedure for estimating the best value for τ_{sim} is outlined in Algorithm 4.2. Our approach uses cross-validation: we first randomly partition the example text contexts into "training" and "test" sets using a $\frac{K-1}{K}$ split (e.g., for 5-fold cross validation we would set K to 5). We then cluster the training contexts using the current value of τ_{sim}, and then test the resulting clusters on the remaining test contexts. The error is calculated as the sum of distances from each example context in the test set to nearest cluster. Finally, in Step (7) we estimate the minimum extraction similarity threshold τ_{score} as $\frac{1}{2} \cdot \tau_{sim}$. This heuristic is designed to force *Snowball* to match only "reliable" candidate contexts and works well in practice.

We now empirically validate the effectiveness of our parameter estimation techniques.

Tables 4.1 and 4.2 show that our automatically estimated parameter values are similar to the manually selected values. Note that the automatically estimated parameter values can change somewhat with each run of the system (due to the randomly chosen seed tuples used to initialize *Snowball*), but the results are typically close to the manually-tuned values. Also note that setting the minimum tuple confidence τ_t is not necessary as it is already estimated by the *ScorePatternsEM-Spy* algorithm. Furthermore, as we will see in Figures 4.6 and 4.7, the automatically tuned *Snowball* with EM-based pattern scoring noticeably outperforms the manually tuned version of *Snowball*.

Parameter	*Description*	*Manual*	*Estimated*
MaxProximity	Maximum distance (in characters) between tuple entities	150	200
WindowSize	Size of external text context (in words) to consider	2	3
τ_{sim}	Minimum clustering similarity	0.6	0.52
τ_{score}	Minimum extraction similarity	0.6	0.3
$W_2, \ldots, W_n$	Relative weight of the *Middle* acceptors	0.6	0.5
W_1, W_{n+1}	Relative weights of the *Left* and *Right* acceptors	0.2, 0.2	0.3, 0.2

Table 4.1: The manually tuned and the automatically estimated parameter values for *CompanyHeadquarters*.

Parameter	*Description*	*Manual*	*Estimated*
MaxProximity	Maximum distance (in characters)between tuple entities	150	122
WindowSize	Size of external text context (in words) to consider	2	5
τ_{sim}	Minimum clustering similarity	0.6	0.4
τ_{score}	Minimum extraction similarity	0.6	0.2
$W_2, \ldots, W_n$	Relative weight of the *Middle* acceptors	0.6	0.4
W_1, W_{n+1}	Relative weights of the *Left* and *Right* acceptors	0.2, 0.2	0.3, 0.3

Table 4.2: The manually tuned and the automatically estimated parameter values for *MergersAcquisitions*.

In summary, we have presented techniques for *automatically* estimating *Snowball* parameters. We will use these techniques, embodied in an "adaptive" system, *Snowball-Auto*,

to apply them for new, unseen relations without additional manual tuning of the *Snowball* parameters.

4.4 Evaluating *Snowball* over New Domains

We now evaluate *Snowball* over domains that are substantially different from the domains that we explored earlier. Specifically, we apply *Snowball* for extracting a simplified table containing information about infectious disease outbreaks, and for extracting three important medical relations. We first describe the experimental setup and the data sets that we used for the experiments, and then present results of the evaluation of several *Snowball* variations as well as of other techniques.

4.4.1 Test Data Sets and Relations

We used four relations (each extracted from a different collection) in two different general domains. The collection and relation statistics are summarized in Table 4.3. The relations to extract are:

- *DiseaseOutbreaks(Disease:DISEASE, Location:LOCATION)*: Name and location of a reported disease occurrence (whether a full-fledged outbreak or a single occurrence).
- *DrugSideEffects(Drug:DRUG, Effect:FINDING)*: Known side effects that can occur because of a treatment.
- *RecommendedTreatment(Treatment:TREATMENT, Condition:FINDING)*: Recommended treatments for a condition or disease.
- *DrugInteractions(Drug1:DRUG, Drug2:DRUG)*: Pairs of known drugs that can interact with each other.

As we discussed, *Snowball*, *DIPRE*, and *Baseline* all rely on a named-entity tagger to identify the entities in the document. For this, we used LingPipe [BC], which can be extended with custom gazetteers. For example, we could feed LingPipe a list of all known diseases, allowing LingPipe to now tag a new entity type *DISEASE*.

Domain	*Relation*	*Collection*	*Total Number of Documents*	*Total Number of Annotated Documents*	*|Ideal|*
News	*DiseaseOutbreaks*	NYT 1996	145,000	78	125
Medicine	*DrugSideEffects*	PDRHealth	535	99	4,972
	RecommendedTreatment	Harrison's	2,638	97	1,950
	DrugInteractions	MICROMEDEX	1,339	98	2,372

Table 4.3: Statistics on the test data sets.

For this evaluation, human annotators with the appropriate domain knowledge created an *Ideal* table manually for each relation. We created a simple web interface for document annotation that allowed volunteer and for-pay annotators to copy and paste attribute values from the document into fields corresponding to attributes of the target relation. An example document fragment and the annotation interface are shown in Figure 4.5. In total, over forty annotators contributed to creating the *Ideal* tables. Of those, 11 were 3rd- and 4th-year students, specifically recruited (for pay) to help creating the *DrugSideEffects*, *RecommendedTreatment*, and *DrugInteractions Ideal* tables. Another 9 were other medical professionals who were specifically recruited through Columbia's NSF DLI2 project PERSIVAL[4]. The rest of the volunteers were friends and colleagues in health-related professions.

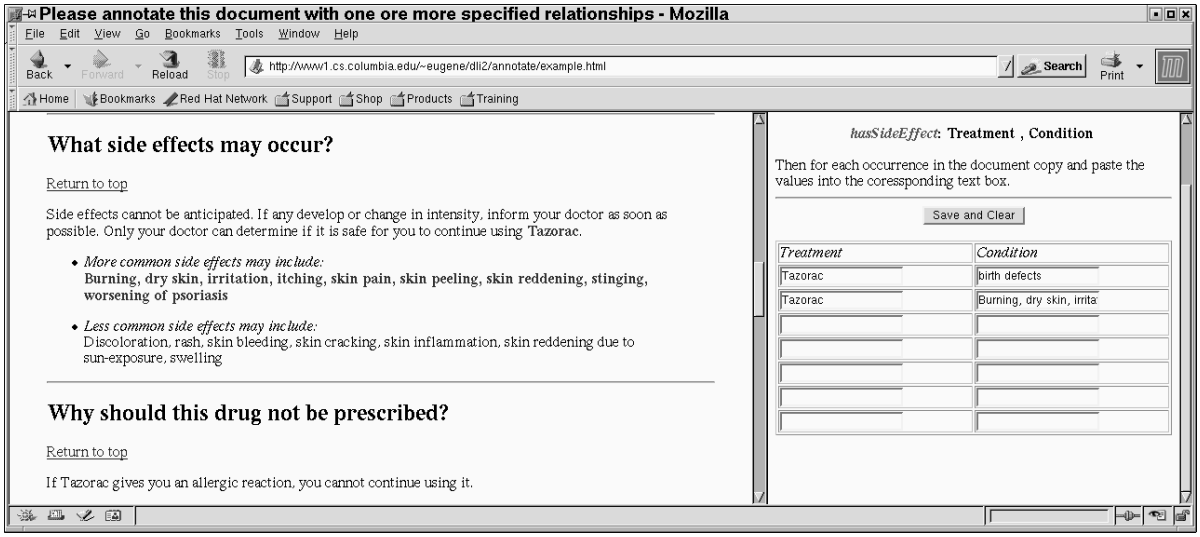


Figure 4.5: Sample test document and annotation interface (*DrugSideEffects*).

[4] `http://persival.cs.columbia.edu`

We use the *Ideal*-based metrics of Section 3.4.2 for evaluation. Because our *Ideal* tables were constructed by copying attribute values from the documents verbatim, the flexible matching described in Chapter 3 was less crucial than it was for the (highly redundant) *CompanyHeadquarters* relation. Additionally, since we now have available a thoroughly labeled set of test documents, we also report the *absolute* recall (i.e., the fraction of labeled tuples correctly extracted by the extraction system), and absolute precision (i.e., the fraction of extracted tuples that matched the manually labeled ones). Unfortunately, since we did not have the resources to have multiple annotators label every document, the test set has some inconsistencies and omissions. Nevertheless, we will treat the human annotator output as our "gold standard" and compute the extraction precision and recall accordingly.

4.4.2 Extraction Systems Compared

We compared three variations of *Snowball* with two other systems, namely *Baseline* and our implementation of *DIPRE*. These last two methods require minimal or no training input from the user, and hence are comparable with *Snowball* in this respect. In contrast, state-of-the-art information extraction systems require substantial manual labor for training.

- *Baseline*: This is our frequency-only baseline with co-occurrence statistics collected over the whole collection (Section 3.6.1). This version of *Baseline* provides weights for the extracted tuples that are proportional to the co-occurrence count of the entities.
- *DIPRE*: This is our implementation of the *DIPRE* system with named-entity tags (Section 3.6.1).
- *Snowball-Manual*: This is the basic *Snowball* system as described in Chapter 3.2, with *constraint*-based pattern evaluation and parameter values tuned over the *Company-Headquarters* and *MergersAcquisitions* relations, as reported in Table 3.2.
- *Snowball-Auto-NB*: This is the *Snowball* system with Naive-Bayes classifier acceptors, using *ScorePatternsEM* with automatic parameter estimation (Section 4.3).
- *Snowball-Auto-EM*: This is the *Snowball* system with *VS* acceptors, using *ScorePatternsEM-Spy* with automatic parameter estimation (Section 4.3).

Additionally, we compare *Snowball* with *Proteus*, a state-of-the-art information extraction system developed at New York University and tuned for extracting a superset of the *DiseaseOutbreaks* relation [GHY02][5]. *Proteus* was tuned on a different document collection from our test collection. Nevertheless, we believe this is a fair comparison since none of the systems were specifically tuned for the test collection.

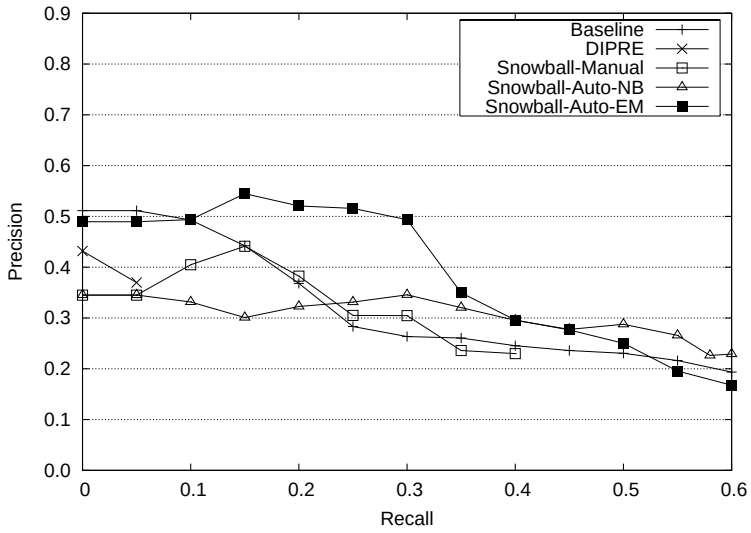
4.5 Experimental Results

We first report the accuracy of all systems over the tuning relations and collections. Figure 4.6 reports the system accuracy of extracting the *CompanyHeadquarters* relation using our *absolute* (a) and *Ideal* (b) precision and recall definitions. As we can see, both *Snowball-Auto-NB* and *Snowball-Auto-EM* perform on par with the manually tuned systems for this task. Also note that the relative performance of the systems as determined by the *absolute* version of the metrics is largely consistent with the *Ideal*-based conclusions. Observe also the strong performance of *Baseline*: for frequently occurring tuples (e.g., those occurring 10 or more times in the collection), *Baseline* rivals and sometimes exceeds in accuracy the "real" extraction systems that consider the text contexts around the tuples.

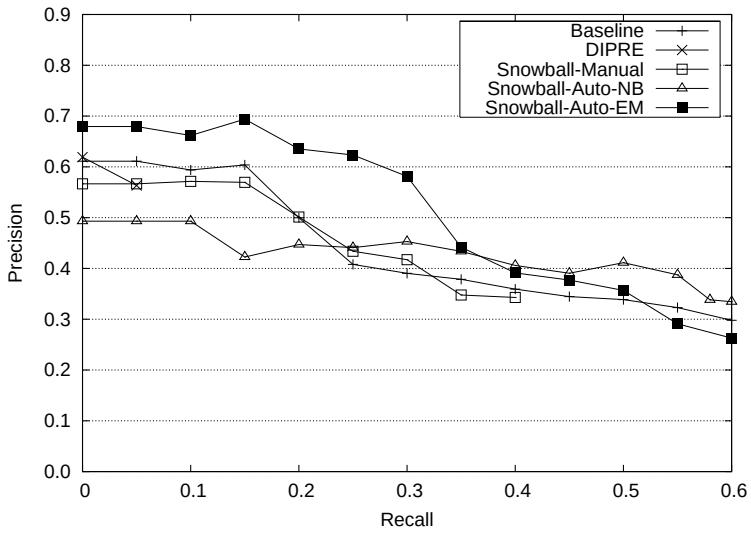
We now turn to the more "difficult" of the two relations, namely *MergersAcquisitions*. Figure 4.7 reports the extraction accuracy on both sets of metrics. As we conjectured, the *EM*-based systems have higher precision than the constraint-violation based *Snowball-Manual*. For *Snowball-Manual*, we used both the key and the identity constraints: namely, we used the *Target* attribute as key, with the intuition that a target company is typically only acquired by one buyer company; we also enforced the identity constraint, stating that a company cannot buy itself.

This is a harder task than *CompanyHeadquarters*, since many mergers or acquisitions are not clearly stated, often have "hedging" clauses in case the merger does not go through, and frequently enough the news item reports that the merger will *not* be happening. Our performance on the *absolute* version of the metrics is comparable to the accuracy of previously

[5] 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.



(**a**)

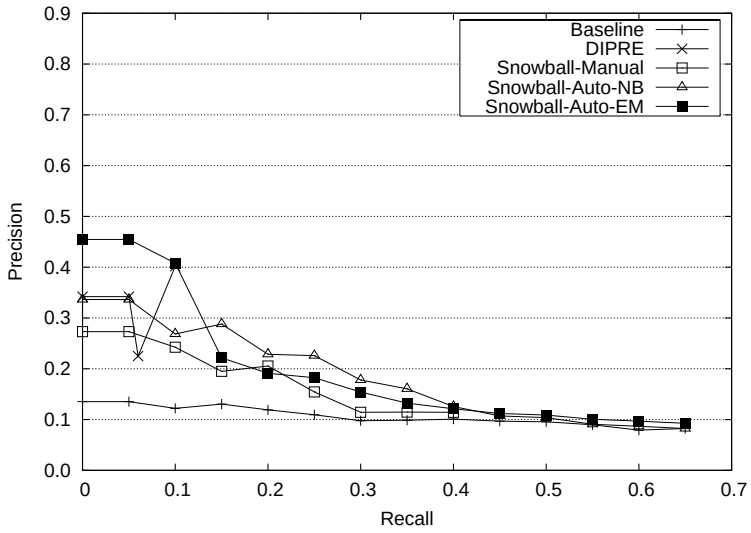


(**b**)

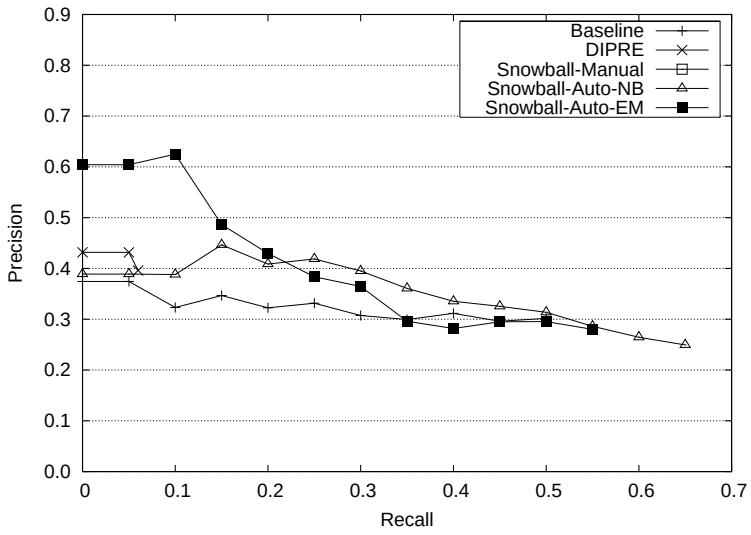
Figure 4.6: *Absolute* (a) and *Ideal* (b) recall vs. precision of *Baseline*, *DIPRE*, *Snowball-Manual*, *Snowball-Auto-NB*, and *Snowball-Auto-EM* (*CompanyHeadquarters*).

reported supervised machine learning approaches for this domain (e.g., [Fre98]).

Some text contexts containing "valid" and "invalid" contexts for the *MergersAcquisitions* relation are listed in Figure 4.8. Note that both kinds share a large fraction of terms. This makes the task harder than the *CompanyHeadquarters* relation. We report two example patterns automatically generated by *Snowball-Auto-EM* for extracting *MergersAcquisitions*



(**a**)



(**b**)

Figure 4.7: *Absolute* (a) and *Ideal* (b) recall vs. precision of *Baseline*, *DIPRE*, *Snowball-Manual*, *Snowball-Auto-NB*, and *Snowball-Auto-EM* (*MergersAcquisitions*).

in Figure 4.9. For clarity, we omit terms with low weights.

We now evaluate the extraction systems accuracy over the *DiseaseOutbreaks* relation. We compare the extraction systems above and *Proteus*, as discussed. We report sample valid and invalid text contexts in Figure 4.11. *Snowball-Auto-EM* is able to generate reasonable patterns for extracting *DiseaseOutbreaks*, with accuracy competitive with that of

PNC Financial Corp. said the Comptroller of the Currency approved its Pittsburgh National Bank unit's acquisition of Bridgeville Trust Co., Bridgeville, Pa.

Texas Commerce Bancshares will have to pay Chemical New York Corp. $24 million in cash if Texas Commerce stockholders don't approve its proposed $1.19 billion acquisition by Chemical.

UNITED NEWSPAPERS PLC offered to acquire EXTEL GROUP PLC, a British financial-information company, in a bid that values Extel at $430 million. Extel's board, however, rejected United's offer.

Figure 4.8: Some valid and invalid text contexts for the *MergersAcquisitions* relation.

Conf(P)	*Left*	*Middle*	*Right*
0.96	[(docid, 0.49), (stake, 0.17)]	[(of, 0.23), (acquisition, 0.21)]	[(securing, 0.5), (unit, 0.33)]
0.8	[(acquisitions, 0.49), (another, 0.19)]	[(took, 0.05), (over, 0.13)]	[(announced, 0.5)]
0.72	[(with, 0.57), (agreed, 0.14)]	[(unit, 0.43), (of, 0.33)]	[(restructure, 0.59), (to, 0.07)]

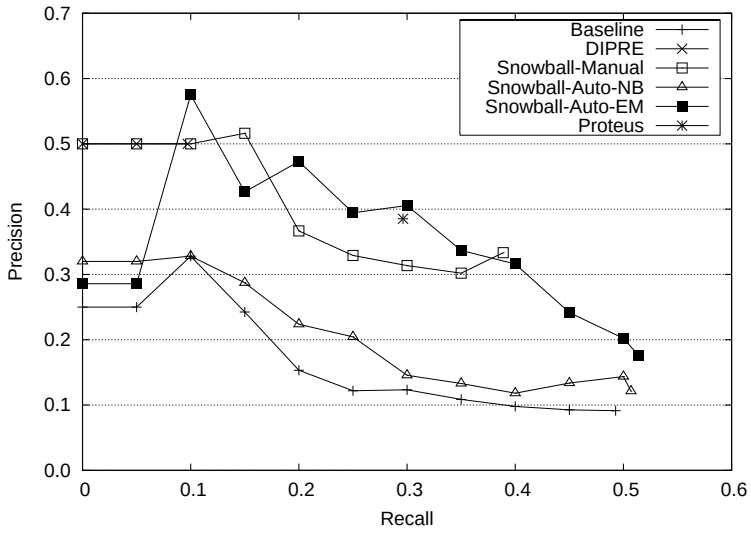
Figure 4.9: Some sample patterns for extracting the *MergersAcquisitions* relation (features with highest weights only).

Proteus. We report some sample patterns in Figure 4.12. *Snowball* performs relatively well on this task. As we can see in Figure 4.10, *Snowball* variations with no specific tuning for *DiseaseOutbreaks* produce results that are comparable with those for *Proteus*. In fact, *Snowball* manages to achieve recall that is substantially higher than that of *Proteus* while maintaining competitive precision.

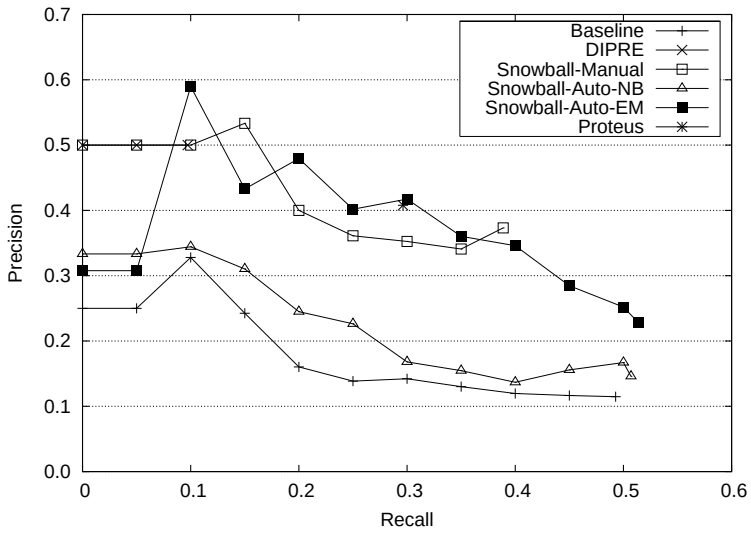
We now consider the *Snowball* accuracy for the extraction of relations from the medical literature. The *Snowball* variations were not tuned for this task, and use parameters obtained earlier from the tuning collections (*Snowball-Manual*) or automatically estimated at run time (*Snowball-Auto-EM, Snowball-Auto-NB*). The entities, namely drug and disease names, were recognized by providing a large list of disease names and drug names adapted from UMLS[6] as domain-specific dictionaries to the LingPipe system.

We first present results for the *DrugSideEffects* relation. An excerpt from the PDRHealth

[6] `http://www.nlm.nih.gov/research/umls/`



(**a**)



(**b**)

Figure 4.10: *Absolute* (a) and *Ideal* (b) recall vs. precision of *Baseline*, *DIPRE*, *Snowball-Manual*, *Snowball-Auto-NB*, *Snowball-Auto-EM*, and *Proteus* (*DiseaseOutbreaks*).

collection is shown in Figure 4.5. We report the accuracy of the systems in Figure 4.13. *Snowball-Auto-NB* has higher accuracy than the other systems. This is because correct tuples are primarily contained within a block on the page under a heading "Drug Side Effects". Instead of learning multiple patterns to "understand" arbitrary text contexts, *Snowball-Auto-NB* effectively trains a small classifier to recognize the block in the page. A

A diphtheria epidemic sweeping the independent states of the former Soviet Union has been declared an international health emergency by the WHO and UNICEF.

ABIDJAN, Ivory Coast: In one of West Africa's worst outbreaks of infectious disease in recent memory, bacterial meningitis has infected more than 100,000 people in the last three months, killing more than 10,000, international health care workers say.

At a House budget hearing, the Atlanta [CDC] centers were alternately praised for leadership in eradicating smallpox and driving polio from the Western Hemisphere and condemned for trying to change "society's attitudes so that it becomes socially unacceptable to own handguns."

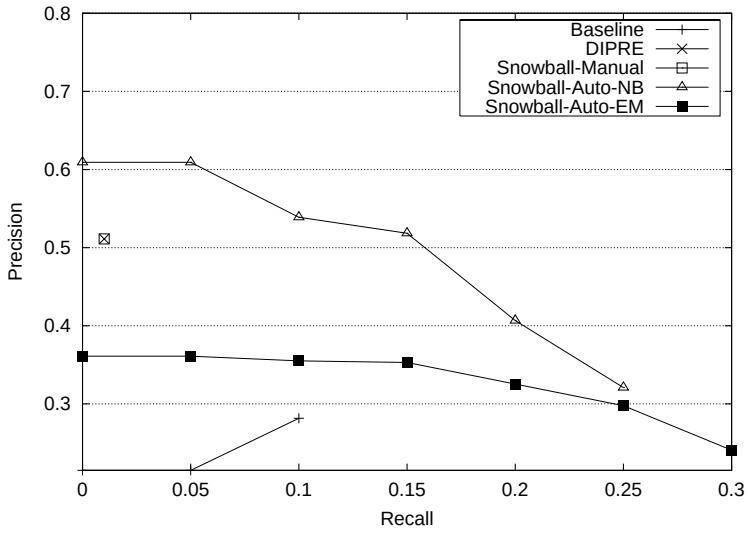
Figure 4.11: Some valid and invalid text contexts for the *DiseaseOutbreaks* relation.

Conf(P)	*Left*	*Middle*	*Right*
0.99	[(the, 0.24)]	[(virus, 0.67), (in, 0.29), (the, 0.25)]	[(transferred, 0.11)]
0.59		[(in, 0.47), (beef, 0.28), (humans, 0.21), (transmitted, 0.17)]	
0.43		[(the, 0.58), (epidemic, 0.16), (in, 0.41)]	

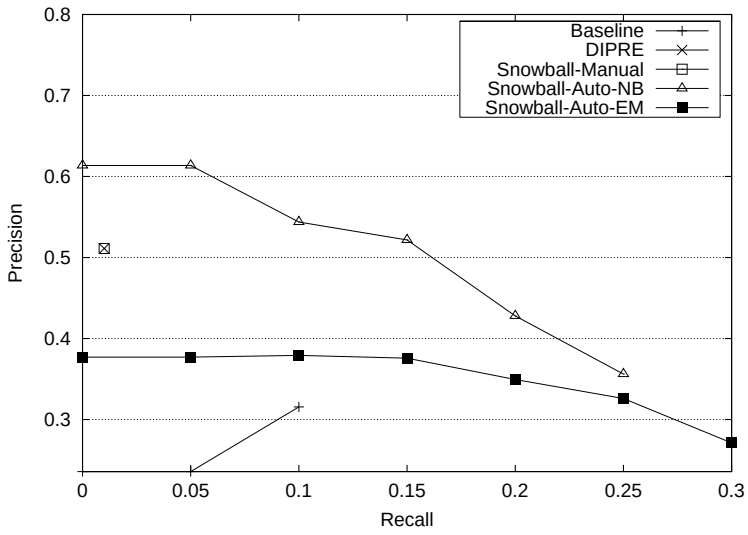
Figure 4.12: Some sample patterns for extracting the *DiseaseOutbreaks* relation (features with highest weights only).

Snowball-Auto-NB pattern with the highest-weighed features is reported in Figure 4.14. As we can see, *Snowball-Auto-NB* discovers intuitive keyword features that allow it to achieve reasonable accuracy on this task.

Unfortunately, the highest recall achieved is 30%. This is because there is little "crossover" of tuples that appear in multiple documents. Therefore, it is likely that the majority of the initial seed tuples will only appear in one document each, and will be in relatively local text contexts. Hence, our parameter estimation procedure imposes the *MaxProximity* threshold that disregards potentially valid tuples because the attributes are listed too far from each other on the page. With some minor tuning, recall can be improved by explicitly setting *MaxProximity* to some large value. Furthermore, the almost regular structure of the documents (e.g., the heading "What side effects may occur?" immediately precedes the paragraph with valid tuples) suggests using larger phrases than the 2- and 3-term phrases considered during our exploration over the less structured newspaper text.



(**a**)



(**b**)

Figure 4.13: *Absolute* (a) and *Ideal* (b) recall vs. precision of *Baseline*, *DIPRE*, *Snowball-Manual*, *Snowball-Auto-NB*, and *Snowball-Auto-EM* (*DrugSideEffects*).

The next relation we consider is *DrugInteractions*, which is to be extracted from the MICROMEDEX collection. The corresponding results are shown in Figure 4.15. As we can see, accuracy is low for all systems. The main reason is the structure of the documents; see Figure 4.16 for an example: the drug discussed in a document tends to be mentioned near the top of the document, and the interacting drugs are mentioned near the end. *Snowball* is

Left	*Middle*	*Right*
[(h2, 0.02), (possible, 0.06), (combining, 0.01), (following, 0.01)]	[(effect, 0.07), (altered, 0.04), (taken, 0.06), (serious, 0.04)]	[ ]

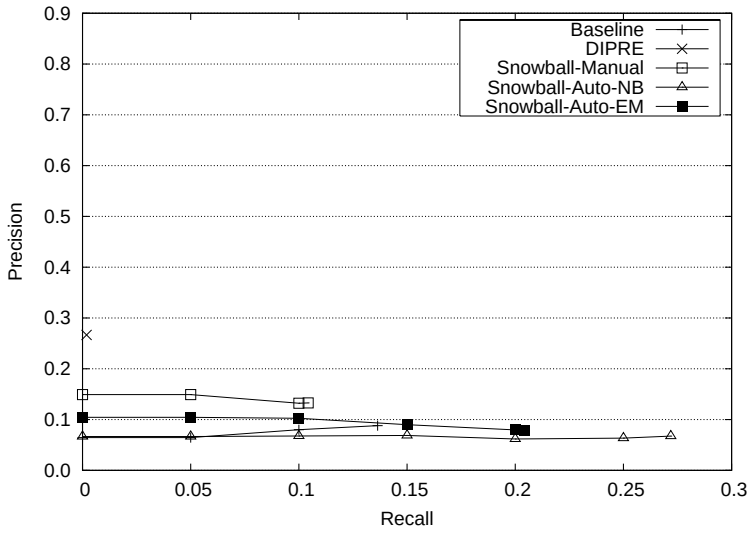
Figure 4.14: A sample *Snowball-Auto-NB* pattern for extracting the *DrugSideEffects* relation (features with highest weights only).

not designed to handle text spans of such length. We may be able to remedy this problem in future work by taking the structure of the document into account.

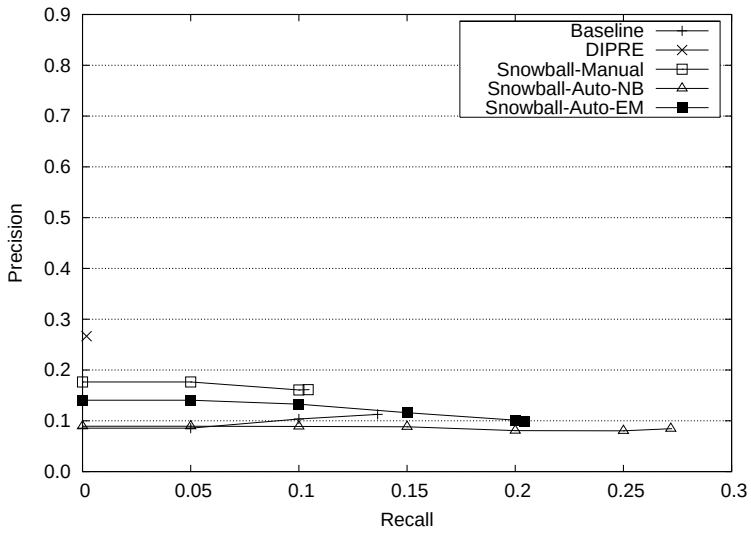
In Figure 4.17, we report extraction results over the *RecommendedTreatment* relation. Sadly, none of the systems appear successful for this relation. Based on anecdotal evidence (e.g., complaints of the volunteer annotators), this relationship is hard to precisely define and detect consistently and reliably even for trained medical professionals. In Figure 4.5, we report some "easy" and "hard" text contexts, all of which contain valid tuples for the *RecommendedTreatment* relation. We can (and should) be able to recognize automatically the first two contexts, while the last context requires being able to follow the cross-reference to a description of a heart defect similar to the one in question. As we will discuss, incorporating co-reference resolution and other post-processing information extraction techniques into *Snowball* may help address some of these issues. Additionally, many of the terms used in the Harrison's textbook do not appear in our gazetteers, which LingPipe –our named-entity tagger– uses to recognize drug names, and diseases and conditions. In summary, *Snowball*'s performance on this domain could be significantly improved by incorporating sophisticated pre- and post-processing information extraction techniques.

4.6 Discussion

In this chapter, we presented techniques that can significantly ease the adaptation of *Snowball* to new domains by automatically estimating relevant parameters. Having been initially designed with newspaper text in mind, *Snowball* does not perform as well as we had hoped on the more complex, more dense, and more highly structured medical documents without additional tuning. Nevertheless, the results are promising, considering the complex nature of the domains we attempted to tackle. In our experiments, we considered a spectrum of



(**a**)



(**b**)

Figure 4.15: *Absolute* (a) and *Ideal* (b) recall vs. precision of *Baseline*, *DIPRE*, *Snowball-Manual*, *Snowball-Auto-NB*, and *Snowball-Auto-EM* (*DrugInteractions*).

techniques ranging from the simple unsupervised *Baseline* technique to the state-of-the-art manually tuned *Proteus* system. In all cases, *Snowball* outperformed *Baseline* while requiring little additional overhead for training. Additionally, our results show that different variations of *Snowball* are more appropriate for some tasks than for others.

We now summarize the lessons learned from applying *Snowball* over a variety of relation

Relation	*Domain*	*Formal Evaluation?*
CompanyHeadquarters(Organization, Location)	News	Chapters 3 and 4
MergersAcquisitions(Buyer, Target)	News	Section 4.5
DiseaseOutbreaks(Disease, Location)	News	Section 4.5
Synonyms(Protein1, Protein2)	Biology	[YA03]
DrugSideEffects(Drug, Effect)	Medical	Section 4.5
RecommendedTreatment(Treatment, Condition)	Medical	Section 4.5
DrugInteractions(Drug1, Drug2)	Medical	Section 4.5
Aliases(RealName, AliasName)	News	No
CEOs(Person, Company)	News	No
VIPs(Person, Company)	News	No
Employees(Person, Company)	News	No
BasketballPlayers(Person, BasketballTeam)	News	No
Ages(Person, Age)	News	No
Birthdates(Person, Birthdate)	News	No
Spouses(Person, Spouse)	News	No

Table 4.4: Summary of some relations extracted by *Snowball*, with corresponding formal and informal evaluations.

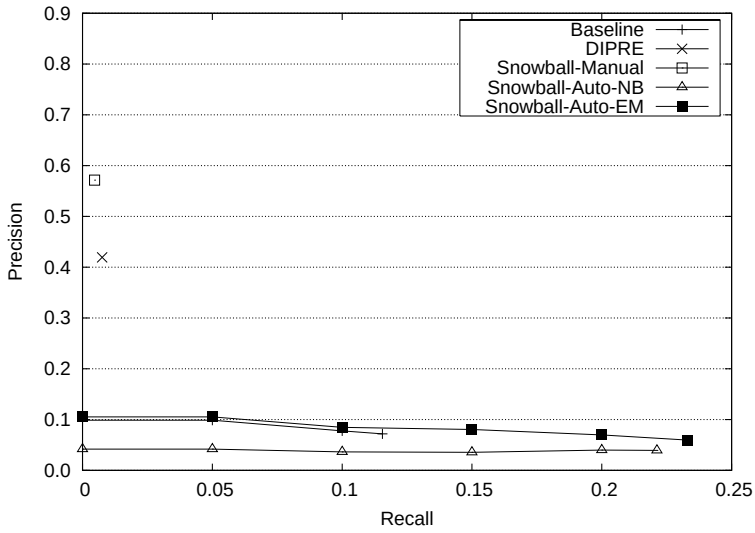
ACARBOSE *Systemic*
Introduction
Revised: 07/31/1998
VA CLASSIFICATION (Primary)? HS504
Commonly used brand name(s): Precose.
... [3+Kb of text] ...
Drug interactions and/or related problems

The following drug interactions and/or related problems have been selected on the basis of their potential clinical significance (possible mechanism in parentheses where appropriate)? not necessarily inclusive (>> = major clinical significance):
...
>> **Adsorbents, intestinal**, such as **activated charcoal**...

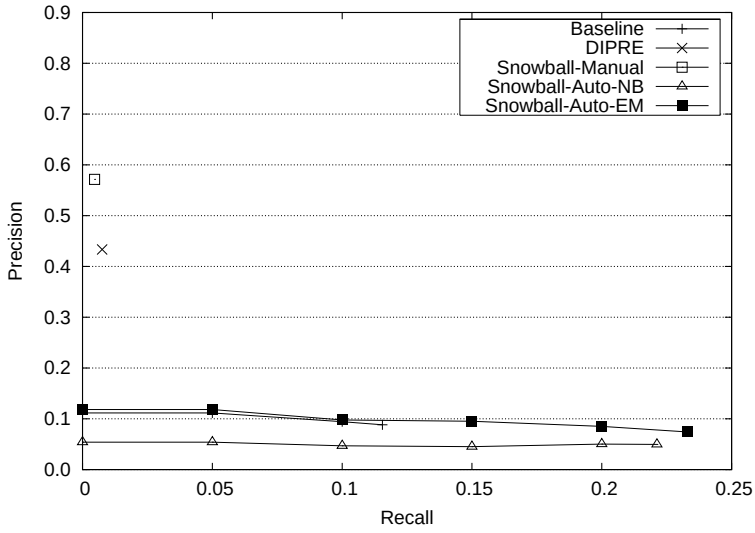
Figure 4.16: A valid text context for the *DrugInteractions* relation.

extraction tasks, which are listed in Table 4.4. The extraction results for some of these tasks were not formally evaluated, but the extracted tuples were manually examined to understand the strengths and limitations of *Snowball*. *Snowball* was most successful for extracting the *CompanyHeadquarters*, *DiseaseOutbreaks*, *Employees*, *Ages*, and *Birthdates* relations and, to a lesser extent, for extracting the *Aliases* and *Spouses* relations. In all of these cases, the relationships between entities were typically expressed in small, localized contexts, the document text was highly edited and grammatical, and the information was largely static and consistent across the collection. Hence, *Snowball* was able to exploit the collection redundancy to find multiple occurrences of seed tuples, and consequently, to derive good patterns.

Interestingly, some *Snowball* variations perform better than others in the news domain. *Snowball-Auto-EM* performs best for *CompanyHeadquarters* and *DiseaseOutbreaks*, while *Snowball-Auto-NB* becomes competitive on *MergersAcquisitions*. *MergersAcquisitions* is an example of an extraction task where localized patterns are not sufficient because they do not consider larger contexts. For this task, when the relationship is indeed expressed in



(**a**)



(**b**)

Figure 4.17: *Absolute* (a) and *Ideal* (b) recall vs. precision of *Baseline*, *DIPRE*, *Snowball-Manual*, *Snowball-Auto-NB*, and *Snowball-Auto-EM* (*RecommendedTreatment*).

a relatively short text fragment, the simpler *Snowball-Auto-NB* model of identifying and exploiting the salient keywords appears just as effective as the more sophisticated pattern weighting strategies employed by the other *Snowball* variations.

In addition to the applications of *Snowball* described in the last two chapters, we have also successfully used *Snowball* for an important bioinformatics task, which consisted of con-

The most effective treatment of atrial flutter is direct-current cardioversion ...

A number of well-designed and -executed large-scale clinical trials have now shown
that treatment with statins reduces recurrent myocardial infarction, reduces strokes,
and lessens the need for revascularization or hospitalization for unstable angina pectoris.

While often asymptomatic, junctional premature complexes may be associated with palpitations
and cause cannon *a* waves, which may result in distressing pulsations in the neck.
When symptomatic, they should be treated like VENTRICULAR PREMATURE COMPLEXES.

Figure 4.18: Some valid "easy" and "hard" contexts for the *RecommendedTreatment* relation.

structing a *Synonyms* relation, with gene and protein synonyms [YA03] from the biological literature. As part of that work, we showed that *Snowball* can be successfully combined with a hand-crafted, domain-specific system, resulting in recall and precision exceeding that of the original systems. We have also experimented with combining different pattern acceptor classes, namely, Sparse Markov Transducers (SMTs), which model sparse sequences of tokens, and *VS* acceptors. Intuitively, because SMTs and *VS* acceptors use different representations, the combination of the two should result in higher accuracy than either of the individual systems [AEG00].

Unfortunately, *Snowball* did not perform well for the relations in the medical domain, and for the *CEOs*, *VIPs*, and *BasketballPlayers* relations in the news domain. *Snowball* failed for a variety of reasons.

In the medical domain, the main difficulty derived from not taking advantage of the document structure. As we can see in the example document in Figure 4.5, for extracting the *DrugSideEffects* relation it is often sufficient to recognize the appropriate paragraph headings. The additional text before and after each heading is effectively just noise. However, these noise terms can cause the *VS* acceptors to assign low scores to correctly related entities. Interestingly, the additional noise terms have no effect on the *Snowball-Auto-NB* acceptors (as long as the entities are within the maximum distance from each other) and hence *Snowball-Auto-NB* performs better on these tasks than other *Snowball* variations. Unfortu-

nately, simply relying on the structure is also not sufficient: we experimented with wrapper induction systems from the literature, namely *RoadRunner* [CMM01] and *Rapier* [CM99], which rely exclusively on detecting the explicit structure in the documents; these systems did not produce usable output. Hence, *Snowball* is still the most viable partially-supervised approach for such (mostly) unstructured domains.

In the case of the *CEOs*, *VIPs*, and *BasketballPlayers* relations, *Snowball* suffered from a "concept drift" problem that was caused by the seed tuples not specifying the target relationship unambiguously. In this case, the most frequent patterns generated using the seed tuples were overly general. As a result, many of the newly extracted tuples were not related as intended. For example, instead of extracting only basketball players and their team names for the *BasketballPlayers* relation, *Snowball* extracted a more general *Employees* relation.

In general, for an extraction task to be appropriate for *Snowball*, the following characteristics of the relationship and document collection associated with the task should be met:

- Seed tuple entities play unambiguous roles in the relationship: As discussed by Barzilay [Bar03] and others, some entities tend to be "dominantly" related. For example, companies tend to have clear "headquarters" relationships with geographic locations, and people tend to have "employee" relationships with companies. This explains *Snowball*'s only partial success on the *MergersAcquisitions* task, where the seed entities can be related in multiple ways and the same companies can be described as competitors, participants in a merger, and subsidiaries, all in the same collection.

- The relationship is usually expressed completely in short text fragments: *Snowball* performs best when related entities tend to appear within a few words of each other, and no additional "global" context is needed.

- The document collection is "sufficiently redundant," with the seed tuples occurring in multiple contexts: *Snowball* requires the seed tuples to appear in similar contexts for patterns to be generated. Redundancy improves the chance of finding text contexts that are shared between multiple seed tuples, which in turn allows *Snowball* to generate

good extraction patterns.

- The document collection is uniform in style: A document collection created with strict style guidelines is particularly appropriate for *Snowball*, because *Snowball* can generate high-confidence patterns that will tend to have high recall. For example, news sources like The Wall Street Journal tend to express the age of a person in a highly regular and consistent way (e.g., "Predictably, **Bill Gates**, **45**, of Microsoft continues to top the list... He is followed by investment guru **Warren Buffet**, **72**, of Berkshire ..."), which simplifies the generation of patterns for the extraction of the *Ages* relation of Table 4.4.

While we have not developed automatic techniques for verifying if an extraction task is appropriate for *Snowball*, the above characteristics can be easily recognized by a human being, especially if presented with an interactive version of *Snowball* such as the one that is described in [AGP+01]. By observing the number and quality of the generated patterns, as well as the number and diversity of contexts in which the seed tuples occur, a human can easily make a good prediction on whether *Snowball* is likely to succeed for the extraction task.

For appropriate extraction tasks, *Snowball* can rival state-of-the-art supervised learning systems in terms of accuracy. For example, *Snowball* achieved a peak F-measure of 0.37 for *MergersAcquisitions*, compared to the highest F-measure between 0.37 and 0.42 achieved by a fully supervised machine learning system [Fre98] on a comparable task. Both systems are outperformed by the best MUC systems that were specifically tuned for the task, with a best reported F-measure of 0.52 during the formal MUC-5 evaluation.[7] However, as we have shown, in important and practical scenarios *Snowball* can outperform even the manually tuned systems, specifically if both systems are to operate over a new document collection. As we reported earlier, *Snowball* achieved a peak F-measure of 0.36 for *DiseaseOutbreaks*, slightly exceeding the accuracy of the manually tuned *Proteus* system (with an F-measure of 0.35).

[7] In the years since the formal MUC evaluation, the improved manually-tuned systems have reached F-measures of over 0.6.

The extensive evaluation of *Snowball* described in this chapter suggests important ways in which *Snowball* can be further improved:

- *Negations*: We assumed that text contexts either reported an event (tuple) of interest, or nothing at all. We did not consider "non-events" (e.g., a report that a company merger did *not* occur). This is one of the cases where syntactic information could be beneficial, as we could incorporate (or automatically derive) syntactic rules (e.g., to ignore subordinate clauses of negations).

- *Document Formatting and Layout*: In some cases, markers such as section and paragraph headings could be exploited more aggressively. By incorporating wrapper induction techniques (e.g., [CMM01, BFG01]) for such domains, we could exploit additional features of a text context such as its nearest heading or the position of the context within the document.

- *Document Structure*: Additional information about text contexts is available from considering the DOM-tree (or other tree-like) representations of the documents. For example, immediate "parents" of a text context containing a candidate tuple can be weighed higher than the "siblings". We describe some preliminary experiments along these lines in [AHJG03].

- *Co-reference and Anaphora Resolution*: Often a text context can refer to an entity of interest simply as "it," "the company," or "the disease," with the actual entity name stated elsewhere in the document. By incorporating well studied co-reference and anaphora resolution techniques, we could rewrite these contexts so that entities are mentioned explicitly, and then use our techniques as before.

These extensions fit naturally into the *Snowball* framework, and could be incorporated without significant changes to the model.

As we have shown over the last two chapters, large text collections contain valuable structured information. However, extracting this information from text documents typically requires sophisticated and computationally intensive techniques, which make processing every document in a large document collection not feasible. In the next two chapters, we will address the problem of scaling up information extraction to large text collections.

Chapter 5

Query-based Access to Text Databases

As we have shown over the last two chapters, large text collections contain valuable information that can be better exploited in structured form. In general, state-of-the-art extraction systems [Gri97] 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 would be required to process a modestly sized 135K-document news 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.

Interestingly, such brute-force processing of every document is generally unnecessary, since only relatively few documents might contribute to the extraction task at hand. For

example, documents in a news archive such as The New York Times would not be relevant for the extraction of the *DiseaseOutbreaks* relation in the previous chapter (as articles about sports, politics, and business, which comprise a large portion of the archive, are not likely to mention disease outbreaks). Furthermore, some potentially useful document collections are only accessible via querying. For example, a widely-used reference for medical professionals called PubMed is primarily accessible through a web query interface. In order to be able to extract structured relations from PubMed, such as the *RecommendedTreatment* relation described previously, we must be able to query and retrieve the documents that contain the relevant information. Therefore, for both efficiency and accessibility reasons, query-based access to text databases is crucial for scaling up information extraction to large collections of documents.

Understanding and improving query-based access to text databases for information extraction is the subject of this chapter. Specifically, we analyze a simple and intuitive querying strategy, *Tuples*, which constructs queries from attribute values of extracted relation tuples. We present the conditions under which this class of strategies is likely to succeed, and validate our model on real document collections. Similarly, we discuss the scenarios when *Tuples* and related querying strategies might miss a large portion of the target relation. By detecting these scenarios, we can adjust the querying strategy. We present an empirical validation of our analytical model, and will show that we can predict the success of *Tuples* and similar strategies efficiently and accurately.

Our contributions in this chapter include:

- The first principled analysis of query-based access to text databases for information extraction, focusing on an intuitive and powerful querying strategy.
- Empirical validation of our analytical model on real text databases.
- Efficient estimation techniques that allow us to predict whether a querying strategy such as *Tuples* will succeed for a given relation and document collection.

The rest of this chapter is organized as follows. In Section 5.1 we formalize an intuitive and powerful query-based information extraction strategy, *Tuples*, which constructs queries based on attribute values of extracted tuples for a target relation. In Section 5.2 we present

an analytical model that allows us to identify the conditions when a querying strategy such as *Tuples* is likely to succeed. In Section 5.3 we validate our model and assumptions on real document collections. Then, in Section 5.5 we present efficient estimation techniques that allow us to predict whether *Tuples* or a similar querying strategy will succeed. Finally, in Section 5.6 we discuss applications of the modeling and estimation techniques presented in this chapter. The bulk of this chapter has been published as [AIG03].

5.1 A Simple Query-Based Strategy

Various query-based methods (e.g., [CC01, RGM01, Sha97]) have been proposed in the past for retrieving and extracting the information stored in databases, with different tasks in mind. These algorithms share the general approach of starting with a small set of queries to a database, retrieving some "objects" from the database, generating new queries from the retrieved objects, and iterating the process. For the task of information extraction, we can adapt this simple query-based strategy to extract the relation tuples as follows. Given an information extraction system already trained to extract a relation, as well as a few tuples from the relation, first construct queries using the tuples and use these queries to retrieve matching documents. Intuitively, the documents that contain a query tuple are expected to contain other previously unseen tuples as well. We can then run the information extraction system over these retrieved documents and, hopefully, identify new unseen tuples for the relation. These new tuples can in turn be transformed into new queries and the process iterates until no new tuples remain to be extracted. This simple strategy is outlined in Algorithm 5.1, to which we refer as *Tuples*.

Algorithm *Tuples* (*Seed*)

while *Seed* has an unprocessed tuple t **do**
    use t to generate a query q;
    retrieve up to *MaxResults* documents matching q;
    extract new tuples t_e from retrieved documents;
    augment *Seed* with t_e;
end
return *Seed*;

Algorithm 5.1: The *Tuples* Querying Strategy Algorithm

Thus, new tuples are discovered by querying for the known tuples, and when the algorithm terminates, all of the tuples "reachable" from the original seed tuples will be discovered. While in some cases this simple strategy will succeed in reaching most of the useful documents in the database (and thereby in extracting most of the tuples in the target relation), in other cases this algorithm may only discover a small fraction of the available tuples. We will present an analytical model that allows us to identify and analyze the properties of text databases that determine the success of this general approach for an extraction task.

5.2 Model

In order to better understand the *Tuples* strategy above, we will abstract away some of the specifics of querying. In this section, we present our graph-based model of query-based access to text databases, capturing the relationship between the documents, the tuples that they embed, and the queries (derived from the tuples) used to retrieve the documents. We first provide the intuition behind our model (Section 5.2.1) and develop this intuition into a model using the "reachability" graph formalism (Section 5.2.2). We then present a key observation about the general structure of the reachability graphs that emerge when querying text databases using *Tuples* (Section 5.2.3).

5.2.1 Querying Text Databases Revisited

Consider again the *Tuples* algorithm above. Recall that we start with a few seed tuples for the target relation, convert them to queries, and retrieve new documents using these tuples (as queries), which in turn allows us to extract new tuples for the relation. We now present our graph-based model that allows us to describe the process more precisely.

Consider the query-based relation extraction process illustrated in Figure 5.1 for a database, and assume that the initial set of seed tuples consists of one tuple, t_1. The database is queried using t_1 as described in Section 5.1. As a result, document d_1 is retrieved. From this document, we extract the new tuple t_2. After adding t_2 to *Seed*, and sending it as a query to the database, we retrieve documents d_2 and d_3, and extract the new tuple t_3, which in turn retrieves the document d_4, containing the tuple t_4. Note that t_3 also retrieves document d_2, which "rediscovers" tuple t_2. As we can see, tuples t_2, t_3, and t_4 are all "reachable" from t_1, and this is indicated in the "reachability" graph on the right side of Figure 5.1. In contrast, tuple t_5 is not reachable from any of these tuples. As a result, t_5 will not be discovered by the algorithm if t_1 is the only "seed" tuple.

The procedure described above can be summarized using the graphs in Figure 5.1. By analyzing the structure of these graphs we can get a better understanding of the process and predict whether a query-based access method can succeed in reaching all (or a significant fraction of) the content of interest stored in the database. With this goal in mind, we now formally present our query-based reachability model.

5.2.2 Querying and Reachability Graphs

In this section, we define our query-based reachability model formally. In our model, the query that is derived from a tuple is simply the conjunction of the tuple attributes. For example, the query derived from the *CompanyHeadquarters* tuple ⟨*Microsoft, Redmond*⟩ is simply *"Microsoft" AND "Redmond."* In the rest of the discussion, by querying for a tuple t we mean querying for the conjunction of the attributes of t.[1] Using this notion of queries, we can now define the querying graph more formally.

[1] In [AIG03], we extend this analysis to any query derived from the retrieved documents in an application-specific way.

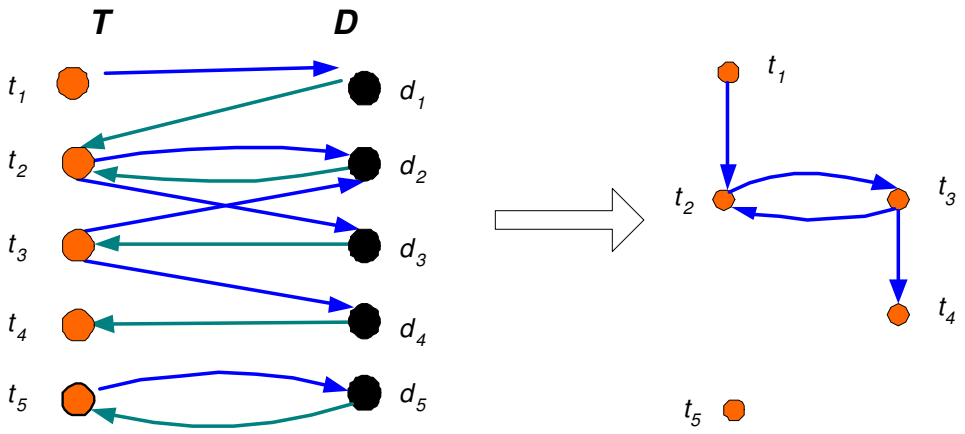


Figure 5.1: Portion of the querying and reachability graphs of a database.

Definition 4 *We define the* querying graph $QG(T, D, E)$ *of a database for the extraction of a relation* T *as a bipartite graph containing tuples* T *and the database documents* D *as nodes, and a set of edges* E *between* T *and* D *nodes. A directed edge from a document node* d *to a tuple node* t *means that* t *occurs in* d*. A directed edge from a tuple node* t *to document node* d *means that* d *is returned from the database as a result to a query derived from* t*.* □

For example, suppose the tuple t_1=⟨*flu, aspirin*⟩ for the *RecommendedTreatment* relation retrieves a document d that also contains another tuple t_2=⟨*cold, tylenol*⟩. Then, we insert an edge into QG from t_1 to d, and also an edge from d to t_2. We consider an edge $d \rightarrow t$, originating from a document node d and pointing to a tuple node t, as a "*contains*" edge and an edge $t \rightarrow d$, originating from a tuple node t and pointing to a document node d, as a "*retrieves*" edge. Notice that the existence of the edge $t \rightarrow d$ in the graph does not imply the existence of the edge $d \rightarrow t$, and the existence of the edge $d \rightarrow t$ in the graph does not imply the existence of the edge $t \rightarrow d$.

The *querying graph* representation can accommodate constraints that appear while querying real text databases. For example, a database might have an upper bound *MaxResults* on the number of documents returned as a result to a query. This is modeled by constraining any tuple vertex in the querying graph to have outdegree no larger than

MaxResults. Another real-life constraint for a query-based algorithm might be an upper limit *MaxDocs* on the total number of documents retrieved. In this case, the algorithm to find all tuples that are reachable from an initial seed set (which can be considered as a walk on the graph) should not cross more than *MaxDocs* edges of the type $t \rightarrow d$.

While the querying graph thoroughly describes the querying process, what we are really interested in is the *reachability graph* of the database, which is derived directly from the querying graph.

Definition 5 *We define the* reachability graph $RG(T, E)$ *of a database for the extraction of a relation T as a graph whose nodes are the tuples T that occur in the database, and whose edge set E is such that a directed edge $t_i \rightarrow t_j$ means that t_j occurs in a document that is retrieved by t_i.* □

In Figure 5.1(b) we show the reachability graph derived from the underlying querying graph, illustrating how its edges are added. Since tuple t_2 retrieves document d_3 that contains tuple t_3, the reachability graph contains the edge $t_2 \rightarrow t_3$. Intuitively, a *path* in the reachability graph from a tuple t_i to a tuple t_j means that there is a set of queries that start with t_i and lead to the retrieval of a document that contains the tuple t_j. In the example in Figure 5.1, there is a path from t_2 to t_4, through t_3. This means that query t_2 can help discover tuple t_3, which in turn helps discover tuple t_4. The absence of a path from a tuple t_i to a tuple t_j in the complete reachability graph means that we cannot discover t_j starting from t_i. This is the case, for example, for t_2 and t_5 in Figure 5.1.

The reachability graph is a directed graph, and its connected components can be described using the "bowtie" structure in [BKM+00]. Consider a strongly connected component *Core* in a reachability graph. By definition, every tuple in the *Core* is reachable via a directed path from every other tuple in the *Core*. Additionally, the tuples in the *Core* are reachable via a directed path from other tuples not in the *Core*, to which we refer as the *In* component (Figure 5.2). Finally, other nodes not in the *Core* or the *In* components are reachable via a directed path from the *Core* tuples. We refer to these nodes as the *Out* component (Figure 5.2). For example, consider the strongly connected component that consists of nodes t_2 and t_3 in Figure 5.1(b). The *In* component for this *Core* consists of the

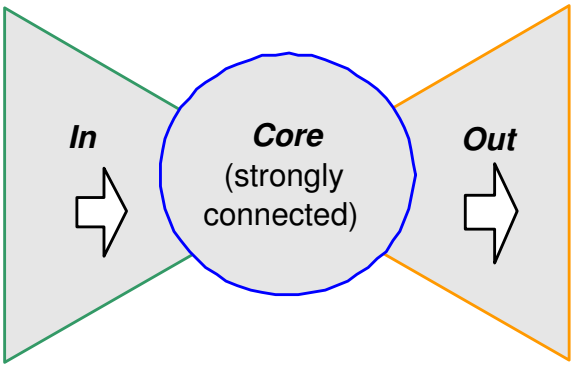


Figure 5.2: Structure of connected components in directed graphs.

node t_1, while the *Out* component contains the tuple t_4.[2]

Having described the general shape of connected components in the reachability graph, we now turn to a quantitative analysis of the relative sizes of the different parts of the graph. We conjecture that the reachability graph of a database for information extraction belongs to the well-studied family of power-law graphs. Power-law distributions have been known to arise in text domains [Zip49]; additionally, power-law graphs have recently been observed to be a good model for graphs in related domains such as the web [BKM+00] and the Internet [FFF99] graphs. One property of interest of power-law graphs is that the size of their connected components can be estimated using only a small number of parameters, as we describe next.

5.2.3 Reachability of Power-Law Graphs

A power-law graph [ACL00] is a graph that has vertices with degrees that follow a *power-law* distribution. The power-law distribution states that the expected number of vertices y with degree k is:

$$y = e^{\alpha} \cdot k^{-\beta} \qquad \Rightarrow \qquad ln(y) = \alpha - \beta \cdot ln(k) \tag{5.1}$$

[2] Additionally, other nodes not in the *In*, *Out*, or *Core* might still be connected to these components (e.g., the nodes in the "tendrils" of the component [BKM+00]). These additional nodes do not help in our reachability analysis, and hence we do not consider them further.

where the parameters α and β are the intercept and the slope of the line with best fit to the degree distribution plotted on the log-log scale.

Power-law graphs are actively studied in the graph theory community. Recent results [ACL00, CL02] allow to efficiently estimate properties of a given power-law graph. Specifically, Aiello, Chung, and Lu [ACL00, CL02] show that by using the average degree of the vertices in a random undirected power-law graph and the parameters α and β of the power-law distribution, it is possible to predict the size of the biggest connected component C_G of a graph, also called the "*giant component*". If the giant component emerges, then the remaining connected components are expected to be *small*, with a size distribution that also follows the power law.

We conjecture (and we study this experimentally in Section 5.3) that the reachability graphs in real text databases for our task can be modeled as *directed* power-law graphs. We use the giant *strongly connected* component *Core* to define the giant component C_{RG} in the reachability graph RG. More specifically, we define C_{RG} as the biggest strongly connected component *Core*, with its associated *In* and *Out* components in the "bowtie" structure described above. According to the power-law, if a giant strongly connected *Core* component exists, then the remaining strongly connected components (and their associated *In* and *Out* components) are expected to be small.

We can now define the *reachability* of a text database for the querying strategy associated with an extraction task. As we discussed, in order to successfully reach all the needed documents, the extracted tuples must help discover other new tuples, which by definition are reachable from the previously discovered tuples. As we mentioned above, if a large component C_{RG} exists in the reachability graph RG, then a tuple not in C_{RG} necessarily belongs in a *small* component and can help discover only a small number of new tuples. On the other hand, any tuple in the *In* or *Core* components of C_{RG} will allow the querying strategy to discover all of the tuples in the *Core* and *Out* components of C_{RG}. Hence, the relative size of the *Core* and *Out* can be used to predict the fraction of the tuples in T reached by our querying strategy, if at least one of the initial seed tuples is part of the *In* or *Core*. Interestingly, this event is very likely for a wide range of *In* and *Core* sizes. Specifically, the probability that at least one of the initial seed tuples belongs to either *In*

or *Core* components of C_{RG} is

$$1 - \left[1 - \frac{|In(C_{RG})| + |Core(C_{RG})|}{|T|}\right]^{|seed|}$$

which quickly approaches 1 for even a small number (e.g., five) of initial seed tuples, for a wide range of C_{RG} sizes. Similarly, if all connected components including *In* and *Core* are *small*, the number of tuples in T that we can reach is limited to the sizes of the connected components touched by querying for the initial seed tuples. This would not allow us to discover a substantial number of tuples for the relation beyond our initial seed tuples. Hence, for our query strategy to reach a large fraction of the relation tuples, *RG* must contain large connected components, which, according to our power-law conjecture, will be the dominated by the *In*, *Core* and *Out* components. Thus, we define the *reachability* of a text database as the fraction of the nodes T of the reachability graph that belong to the *Core* and *Out* portions of the giant component C_{RG}:

$$reachability = \frac{|Core(C_{RG})| + |Out(C_{RG})|}{|T|} \tag{5.2}$$

In the rest of the chapter, we turn to the problem of how to efficiently approximate the reachability of a database for query-based information extraction.

5.3 Reachability of Real Databases

In this section, we show that the reachability graphs constructed over real text databases for extracting a relation have an approximate power-law degree distribution. Hence, we conjecture that power-law graphs can model the structure of the reachability graphs for query-based information extraction. We support our conjecture with observations made over two real text databases. Observations over additional databases are consistent with this conjecture, and so we will focus on just these two representative databases.

NYT: This database is a collection of 135,000 newspaper articles from The New York Times from 1995, a superset of the collection used for the *DiseaseOutbreaks* experiments in Chapter 4. We use the *Proteus* system (Section 4.4.2) for this task. A total of 8,859 tuples were extracted from *NYT* using an exhaustive scan of the database (which required over two weeks to complete). The queries are constructed using the conjunction of the first two

attributes of each tuple (e.g., *"Typhus" AND "Belize"*). The complete reachability graph *RG* is computed by querying the database with all the 8,859 tuples extracted beforehand from the collection. To simulate constraints that search engines might impose, we limit the maximum number of documents, *MaxResults*, retrieved for each query.

We report the degree distribution of the resulting reachability graphs of the NYT database for extracting the *DiseaseOutbreaks* relation in Figures 5.3(a)-(c) when *MaxResults* is set to 10, 50, and 200 respectively. We show the power-law distribution that best fits the data. As we can see, the outdegree distribution is closely related to the fitted distribution. We report the size distribution of *connected components* in Figure 5.4, which also agrees with the size distribution expected for power-law graphs.

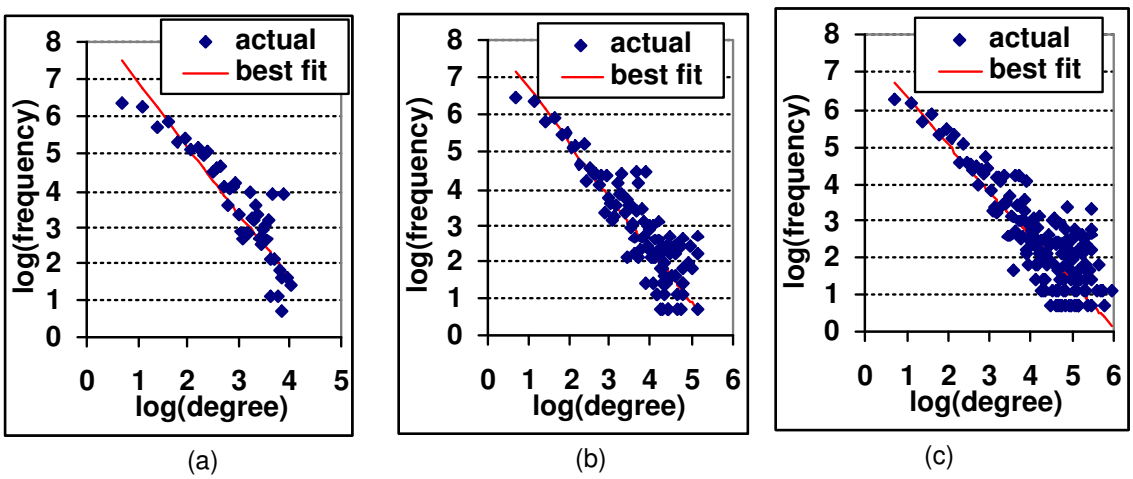


Figure 5.3: The outdegree distribution of the NYT reachability graph for the *DiseaseOutbreaks* relation when (a) *MaxResults* = 10, (b) *MaxResults* = 50, and (c) *MaxResults* = 200.

AP: This database consists of 242,911 AP newswire articles from 1988-1990, available as part of the TREC Tipster collection. We use this database to retrieve all of the tuples for the *DiseaseOutbreaks* relation extracted by the *Proteus* system [YG98], as well as for the *CompanyHeadquarters* relation, extracted by *Snowball*. A total of 9,280 tuples for the *DiseaseOutbreaks* relation, and 23,809 tuples for the *CompanyHeadquarters* relation were extracted from *AP* by using an exhaustive scan of the database by *Proteus* and *Snowball*,

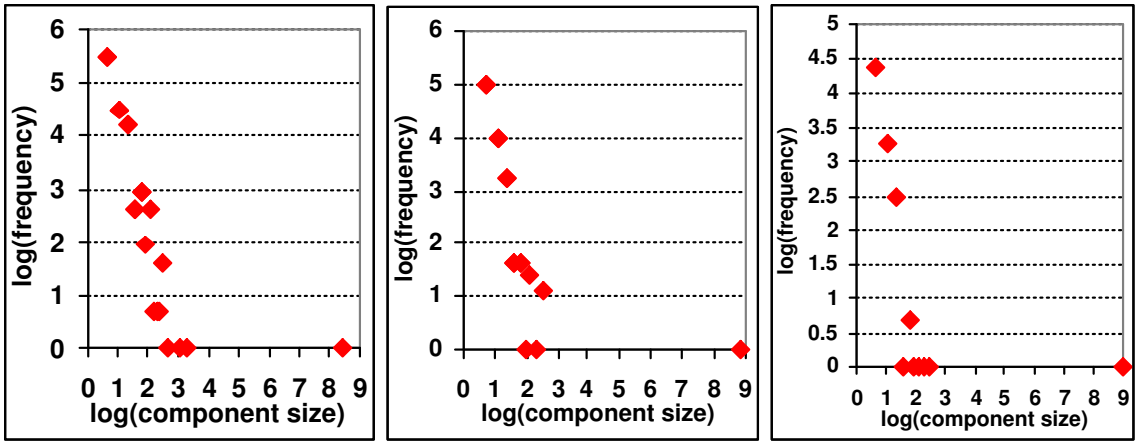


Figure 5.4: The component size distribution of the NYT reachability graph for the *DiseaseOutbreaks* relation when (a) *MaxResults* = 10, (b) *MaxResults* = 50, and (c) *MaxResults* = 200.

respectively. Figures 5.5 (a) and (b) report the degree distribution and the connected component sizes of the reachability graph for the *DiseaseOutbreaks* relation. Figures 5.6 (a) and (b) report the corresponding results for the *CompanyHeadquarters* relation, for *MaxResults* = *50*. Note that the shapes of the degree distribution on this database for both target relations clearly follow our power law conjecture, as do component sizes. The same conclusions hold for the other sub-collections (*WSJ*, *REUTERS*) of the TREC Tipster collection.

Based on the observation that our power-law conjecture holds for all of the text databases and relations we examined, we believe that the reachability graphs constructed over other text databases will have similar shapes of outdegree and component size distributions. For the text databases where our conjecture holds, we will be able to predict the reachability of the databases (and consequently the performance of query-based algorithms for relation extraction) without constructing the complete reachability graph. Instead, we can simply estimate the parameters of the outdegree distribution of the corresponding reachability graphs, as discussed next.

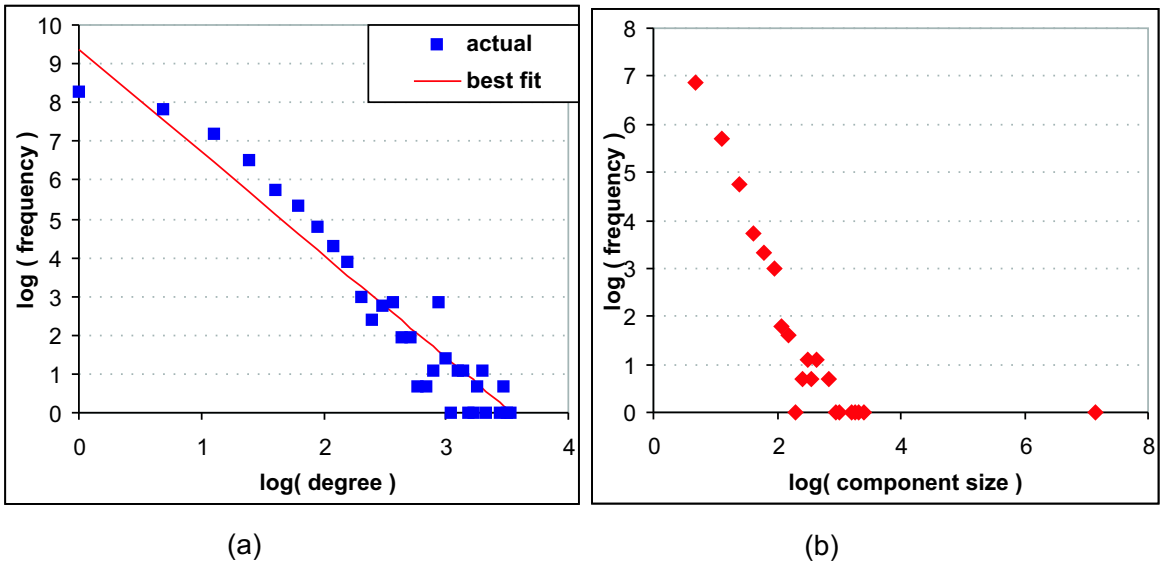


Figure 5.5: The outdegree distribution (a) and the connected component size distribution (b) of the AP reachability graph for the *DiseaseOutbreaks* relation when *MaxResults* = 50.

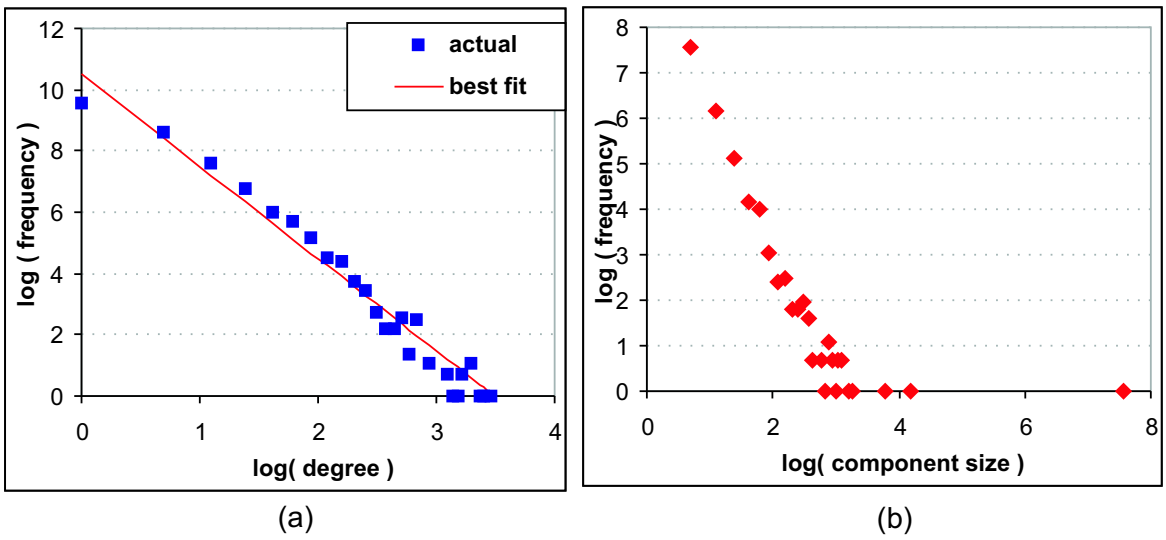


Figure 5.6: The outdegree distribution (a) and the connected component size distribution (b) of the AP reachability graph for the *CompanyHeadquarters* relation when *MaxResults* = 50.

5.4 Estimating Graph Properties

In this section, we describe how to estimate the reachability of a database for a query strategy. Specifically, Section 5.4.1 shows that we can base our estimates on the average node outdegree d in the reachability graph, which in turn we can estimate using a small document sample from the database. Section 5.4.2 describes our document sampling method.

5.4.1 Estimating Reachability

As we discussed, in a power-law graph at most one giant connected component is expected to emerge, and the rest of the connected components are expected to be small. Thus, the reachability estimation for power-law random graphs reduces to estimating the size of the giant component, if any.

Chung and Lu [CL02] showed that for an undirected power-law graph G with values of $\beta < \beta_0 \quad (\beta_0 \approx 3.475)$[3], a giant component emerges, while for larger values of β all components are expected to be small. More specifically, [CL02] estimates the relative size of the giant component C_G when $\beta < \beta_0$ as follows:

$$\frac{|C_G|}{|T|} \geq \begin{cases} 1/d \cdot (1 - \frac{2}{\sqrt{de}}) & \textit{if} \quad d \geq e \\ 1/d \cdot (1 - \frac{1+\log d}{d}) & \textit{if} \quad 1 < d < e \\ 0 & \textit{if} \quad 0 < d \leq 1 \end{cases} \tag{5.3}$$

where d is the average degree of G. Note that the estimate *depends only on d*.

We apply Equation 5.2 to estimate the size of the giant component C_{RG} in our *directed* reachability graph RG, and consequently to estimate the *reachability* of a database with respect to a given extraction task, where d in the equation above is the average *outdegree* of RG. Note that while *reachability* (Equation 5.2) is defined in terms of the *Core* and *Out* portions of C_{RG}, the equation above predicts the relative size of the *complete* giant component in the undirected graph. Therefore, the value predicted by Equation 5.3 will overestimate the *reachability*. Unfortunately, we are not aware of any similarly compact theoretical results to estimate the sizes of the specific portions of the giant component in

[3] β is the absolute value of the slope of the linear fit to the degree distribution. See Section 5.2.3.

directed graphs. As such results are developed, we could use them to improve the quality of our estimation.

Recall that by applying the results in [CL02] we can estimate the reachability of a database for extracting a given relation by estimating the average outdegree d. We now describe an efficient document sampling technique that can be used to estimate d based on an observed (small) sample of the reachability graph.

5.4.2 Sampling Text Databases for Estimating Reachability

In order to estimate d, we retrieve a small document sample D_S and construct a reachability graph RG_S for D_S. By definition, RG_S is a subgraph of the complete reachability graph RG. We construct RG_S as follows. We start with a small number (e.g., 50) of seed tuples $T_{seed} \subset T$. We then query the database for each tuple $t_i \in T_{seed}$, retrieving up to *MaxResults* documents for each query. For each document d retrieved by a tuple t_i, we extract each tuple t_j in d. For each t_j, we insert an edge $t_i \rightarrow t_j$ into RG_S.

Having obtained our sample subgraph RG_S, we can use it to estimate parameters of interest for the complete reachability graph RG. Specifically, we estimate d, the average outdegree of RG, as the average outdegree of the nodes in T_{seed}. Note that the outdegree of each node $t_i \in T_{seed}$ in RG_S is equal to the outdegree of t_i in RG. Since we draw T_{seed} randomly, we expect that the average outdegree of the corresponding vertices in the partial reachability graph will reflect the average outdegree of the complete RG.

The estimate of the average outdegree is used to predict the relative size $|C_{RG}|/|T|$ of the giant component as described in Equation 5.3, which approximates the reachability of the text database in question (Equation 5.2).

5.5 Experiments

We now report experimental results for the technique that we described in Section 5.4. In Section 5.5.1 we describe the experimental setup, and in Section 5.5.2 we report our empirical results.

5.5.1 Experimental Setup

To evaluate the accuracy of our estimation technique, we constructed the reachability graph for the *NYT* and *AP* databases (Section 5.3) and for relations *DiseaseOutbreaks* and *CompanyHeadquarters*, respectively. For each reachability graph, we compute the size $|C_{RG}|$ of its giant component using the STRONGLY-CONNECTED-COMPONENTS algorithm from [CLR90] to compute the *Core*, and subsequently the *In* and *Out* components. Using $|Core(C_{RG})|$ and $|Out(C_{RG})|$ we compute the reachability values (Equation 5.2), which are reported in Figure 5.7, and serve as the "gold standard" for evaluating the estimation accuracy of our method. Observe that a large fraction of the tuples for *DiseaseOutbreaks* over the NYT database are not reachable for values of *MaxResults* below 200 (i.e., by following the *Tuples* algorithm in Algorithm 5.1 we will not be able to reach a large fraction of the tuples in the target relation no matter how many queries we issue to the database).

For each database and each value of *MaxResults*, we apply the sampling technique of Section 5.4.2 by starting with a different number of seed tuples. Specifically, we sample the corresponding database using S randomly chosen *seed* tuples[4] from T to construct RG_S. We experimented with $S = 10, 50, 100$, and 200, which means that we would send $10, 50, 100$, and 200 tuple queries to the database to estimate its reachability. The maximum number of documents retrieved during sampling has a strict upper bound equal to $MaxResults \cdot S$.

5.5.2 Experimental Results

We report the estimation results for the NYT database for retrieving tuples for the *DiseaseOutbreaks* relation. As we can see, our method is able to estimate the reachability of the database with varying success. For example, when $MR = 1$ the actual reachability is 0.001. For sample size S of 100 and 200 queries, we estimated the reachability to be 0.015 and 0.002, respectively. While this estimate has high relative error, it is *good enough*, as it indicates that an algorithm for extracting *DiseaseOutbreaks* tuples will not succeed in retrieving all necessary documents if *MaxResults = 1*. For the remaining values of *MaxResults*, we

[4] We assume that we can somehow obtain (e.g., as user input) an initial seed set of the appropriate size S. When this is not possible, we can repeatedly obtain random *document* samples until S seed tuples are extracted.

MaxResults	*Core*	*In*	*Out*	*reachability*
1	0.001	0.001	0	0.001
10	0.078	0.156	0.063	0.141
50	0.260	0.160	0.200	0.460
100	0.334	0.132	0.274	0.608
200	0.388	0.102	0.334	0.722
1,000	0.477	0.036	0.429	0.906

Figure 5.7: The relative size of the subcomponents of C_{RG} for the NYT database for the *DiseaseOutbreaks* relation, for different values of *MaxResults*.

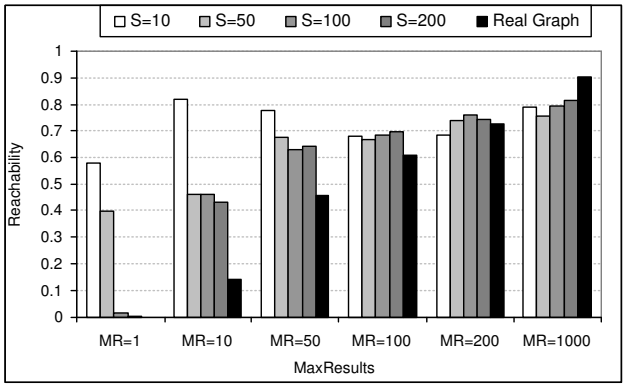


Figure 5.8: The reachability estimates for the NYT database for the *DiseaseOutbreaks* relation, for different values of *MaxResults* and seed sample size *S*.

still generally overestimate the reachability of the database, but our estimates are closer to the real value. As we discussed above, the overestimates are partly caused by using the prediction of Equation 5.3 to estimate the relative size of only the *Core* and *Out* portions of the giant component. Another possible cause of error is the divergence of real reachability graphs from the idealized, pure power-law model, and may be remedied by extending our work to include refinements of the power-law model. While not exact, our estimates are still indicative of the general structure of the reachability graph, and consequently can be used as a rough predictor of expected performance of querying strategies for retrieving tuples, as we will discuss in the next section.

5.6 Discussion

We presented a model for query-based access to text databases for information extraction. To the best of our knowledge, this is the first attempt to model query-based access to text databases for information extraction, and our work parallels the efforts of modeling the web [BKM+00] and Internet [FFF99] topologies. Based on this model, we developed the *reachability* metric, which indicates the expected performance of the *Tuples* query-based algorithm for information extraction. To complement our model, we presented an efficient technique for computing approximate values of reachability for a database with respect to the desired extraction task.

As we discussed, exploiting search engines can dramatically improve the efficiency of information extraction and the documents that can be reached by the information extraction system. However, as we will show in the next chapter, the best querying strategy to use may vary with the target relation and the text database of interest. In some cases, the *Tuples* algorithm might "stall" and fail to retrieve documents with "new" tuples. In the next chapter, we will present a different querying strategy that does not have this limitation, but may have additional overheads. Hence, it is crucial to be able to model, and predict, the behavior of a querying strategy for extracting a relation of interest from a text database.

Additionally, we believe that our graph-based abstraction of the querying process can be used to model a variety of query-based access methods for different tasks. Currently,

most query-based algorithms are only empirically tested, with limited theoretical justification. The techniques presented in this chapter are a significant step towards more rigorous analysis of strategies for querying text databases. We believe that our model provides useful abstraction tools for the study the general family of bootstrapping-based algorithms. In fact, we were able to successfully apply our techniques to modeling the process of indexing the "hidden web" databases, as described in [AIG03].

Other properties of the reachability graph are also of interest and can be estimated using a small number of parameters. The edge density of the reachability graph is an indication of the rate at which an algorithm can obtain new information by querying a text database. The diameter of the graph shows the minimum required effort to retrieve all the information stored in the database. Additionally, the *querying* graph can be a useful tool for studying the relative efficiency of different querying algorithms. In this chapter, we have shown that the *reachability* graph can be used to predict whether a method can succeed in retrieving all the tuples stored in a database. However, the reachability graph cannot reveal how many queries are required to retrieve these tuples. Reaching all the tuples that can be extracted from the database may require issuing thousands of queries and retrieving the resulting documents. This can be modeled by crossing thousands of edges in the corresponding querying graph.

Chapter 6

QXtract: Scalable Information Extraction

As we described in the previous chapter, information extraction systems typically require computationally expensive processing. At the same time, often only a small fraction of the documents in a collection are relevant to an extraction task. Therefore attempting to extract information from every document in a collection is both sometimes not feasible and unnecessarily expensive. Previous approaches for addressing the high computational cost of information extraction resort 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 use keywords or phrases as filters (which could be converted to queries) that are manually crafted and tuned by the information extraction system developers, as we will discuss. Finally, the simple *Tuples* strategy that we analyzed in the previous chapter can (as we showed) run out of queries and leave a large portion of the target relation undiscovered.

In this chapter 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 the 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.

Our contributions in this chapter include:

- An 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.

- A thorough evaluation of our query generation techniques over real document collections, showing, in some cases, dramatic efficiency improvements over naive approaches.

- Techniques for estimating text database characteristics to facilitate the adaptation of *QXtract* to new collections and extraction tasks.

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.

The rest of the chapter is organized as follows. Section 6.1 presents our new document retrieval method in detail. Then, Section 6.2 summarizes the general experimental setting, including the evaluation methodology, metrics, and databases we used for tuning and evaluating our strategy. Section 6.3 reports the results of an experimental evaluation of our technique (and several other strategies) on a large text database. Section 6.4 presents techniques for choosing the best query-based strategy for a new document collection and relation. The bulk of this chapter has been published as [AG03].

6.1 Retrieving Promising Text Documents

In this section, we present an *automatic* method for generating queries to match the documents that are useful for the extraction of a target relation. We start by defining the problem and the overall architecture of our *QXtract*[1] system (Section 6.1.1). *QXtract* first retrieves a small sample of documents from a text database and determines those from which the extraction system is able to extract tuples for the target relation (Section 6.1.2). 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 6.1.3).

6.1.1 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, as we have mentioned. 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 chapter. 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).

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

[1] *QXtract* stands for **Q**uerying for e**Xtract**ion, originally presented in [AG03].

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 6.1. We interact with the target information extraction system through a unified 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 1.1.
- **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 1.1, $Useful = \{doc2, doc4\}$.

[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.

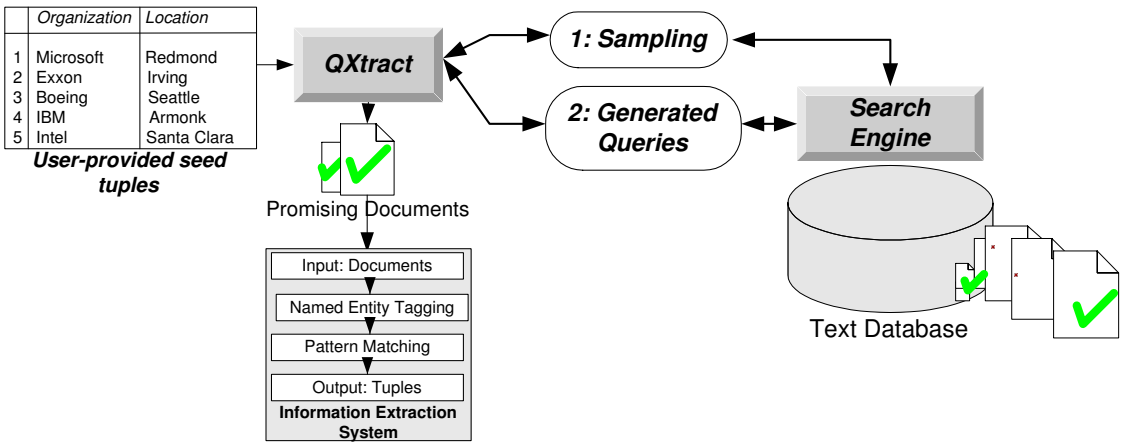


Figure 6.1: The architecture of an efficient information extraction system that identifies promising documents via querying.

- **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 1.1).

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 [MW94], *Lucene*) 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 6.2. Starting with a set of user-provided seed tuples, we first use the sampling procedure described in Section 6.1.2 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.

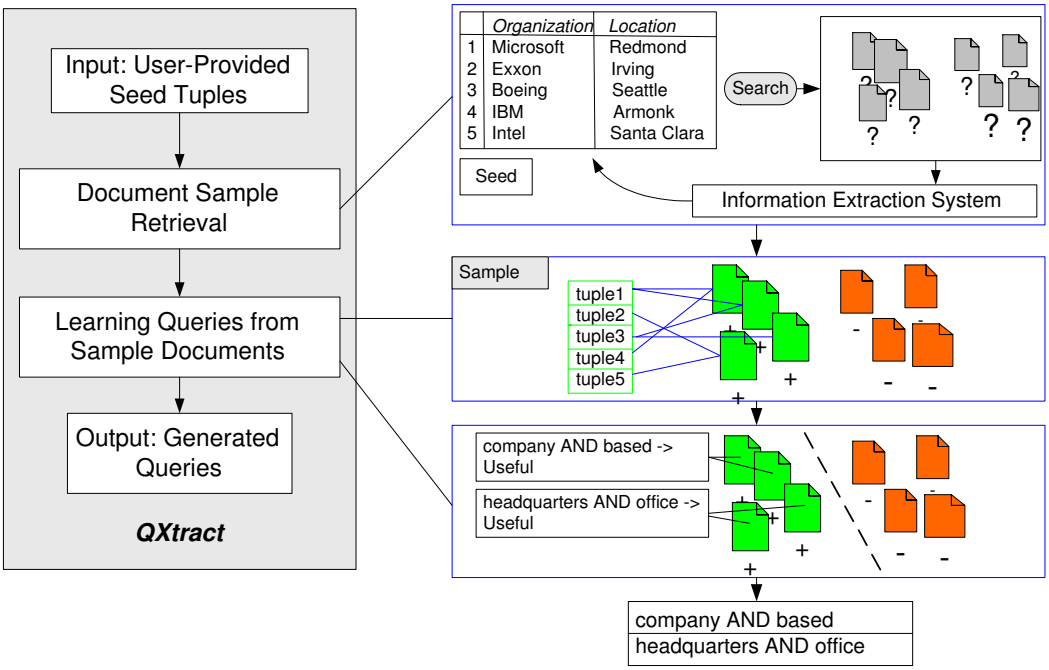


Figure 6.2: *QXtract*: Promising document retrieval.

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 6.1.3). These queries are used to retrieve a set of promising documents from the database (Section 6.1.4), to be returned as *QXtract*'s output and finally processed by the information extraction system.

6.1.2 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 6.1. 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 Algorithm 6.1), 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.

[3] Independently, Ghani and Jones [GJ02] have recently introduced a similar strategy to construct a training corpus for a bootstrapping-based general entity tagger.

```
    Algorithm 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,
      //which will likely be "negative" examples for training.
1     Sample = RetrieveRandom(MaxSampleSize / 2);

      //Identify the useful documents in the random sample.
      //Augment the Seed set with the extracted tuples T.
2     (T, Useful) = E.Extract(Sample);
3     Seed = Seed ∪ T;

      //Now, retrieve MaxSampleSize / 2 documents with tuple attributes,
      //likely to be "useful", to provide positive examples for training.
4     while |Sample| < MaxSampleSize do
5       LikelyUseful = ∅;
6       foreach t in Seed do
7         q = t.a1 AND t.a2 AND ... t.an;
8         LikelyUseful =
              LikelyUseful ∪ RetrieveSeedDocuments(q, MaxSeedResults);
          //Skip to line 9 if MaxSampleSize exceeded.
        end
9       (T, U) = E.Extract(LikelyUseful);
        //Set Seed tuples for next iteration.
10      Seed = PickBestTuples(T, U);
11      Useful = Useful ∪ U;
12      Sample = Sample ∪ LikelyUseful;
      end

      //Sample now consists of MaxSampleSize documents.
13    Useless = Sample - Useful;
14    return Useful, Useless;
```

Algorithm 6.1: The *QXtract* Document Sampling Algorithm.

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* ...

$t.a_n$, 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 "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 discussed in Chapter 5, the *Tuples* strategy, which in essence corresponds to the above procedure, might "stall" and fail to reach most of the relation tuples. As we will show, *Tuples* behaves exactly as predicted by our model in Chapter 5. Specifically, as we will show in Section 6.3, 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 *CompanyHeadquarters* 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 *CompanyHeadquarters* relation. Our goal now is to automatically generate queries to retrieve the documents similar to those that the

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

extraction system marked as *Useful*. Hence, the *Useful* and *Useless* documents returned by *DocumentSample* serve as the training set to learn queries for promising document retrieval.

6.1.3 Learning Queries for Promising Document Retrieval

Given a training set of useful and useless documents for an information extraction system E, our goal now is to generate queries to retrieve the documents that E finds useful and as few useless documents as possible. 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 6.1.4, we will see how to submit the queries to the database to retrieve promising documents.

6.1.3.1 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.

6.1.3.2 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 [Rob90]. While there are many promising alternatives to this weighting scheme in the IR literature (e.g., [XC00, Roc71]), 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* [Rob90] 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 $w_i^{(1)}$ [RJ76]:

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

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 [Rob90] 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 [Coh95]. 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 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 ex-

ploit another family of classifiers, SVMs, which have been shown to perform well in text classification [Joa98]. 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 [Joa98]. 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 [IGS03]. 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 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].

6.1.4 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 6.1.3.2) 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

[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.

[6]More sophisticated ways of combining the generated queries are possible (e.g., to eliminate redundancy across queries).

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.

6.2 Experimental Setting

We now report the metrics we use to evaluate the alternative query methods (Section 6.2.1). Then, we describe the information extraction systems (Section 6.2.2) and the two target relations that we use in our experiments (Section 6.2.3). Later, we specify the training and test databases (Section 6.2.4), and conclude by describing the various querying techniques that we compare (Section 6.2.5).

6.2.1 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 6.1.1). 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*=

[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.

$\frac{|D \cap U|}{|D|} \cdot 100\%$, 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 6.3 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.

6.2.2 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 6.1. For our experiments, we consider three extraction systems:

- *DIPRE* [Bri98], which we described in Section 3.6.1.
- *Snowball*, which we presented in Chapters 3 and 4.
- *Proteus* [GHY02], which we described in Section 4.4.2.

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*).

6.2.3 Target Relations for Extraction

We evaluate the performance of *QXtract* on the extraction of two relations, with the initial seed tuples of Figure 6.3:

- *CompanyHeadquarters(Organization:ORGANIZATION, Location:LOCATION)*, as defined in Section 3.1. The attributes *Organization* and *Location* are used for querying for sample documents. We use *DIPRE* and *Snowball* with *VS* acceptors and parameter values of Table 3.2 to extract this relation.

- *DiseaseOutbreaks(DiseaseName, Location, Country, Date, ...)*. Each tuple $\langle n, l, c, d, \ldots \rangle$ (e.g., $\langle$"Mad Cow Disease", "The U.K.", "U.K.", "3/27/1996", $\ldots\rangle$) 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.

CompanyHeadquarters		*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 6.3: Initial seed tuples provided to *QXtract* for extracting the *CompanyHeadquarters* and *DiseaseOutbreaks* relations.

6.2.4 Training and Test Databases

Our **training** database for this evaluation consists of 137,893 New York Times documents selected from the training collection of Section 3.4.1[9]. The **test** database consists of 135,438 documents from the 1995 New York Times documents selected from the test collection of Section 3.4.1. 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[10].

6.2.5 Alternative Document Retrieval Methods

We experimentally compare a number of alternatives to retrieve promising documents:
QXtract: This is the technique described in Section 6.1, whose parameters (tuned using the **training** database and summarized in Figure 6.5) include *MaxSampleSize*, *MaxSeedResults*, and the *query generation strategy*.
Tuples: This technique, presented and analyzed in Chapter 5, uses tuples to retrieve promising documents.
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[11]. The filters were

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

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

[11] 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

converted to the closest phrase and Boolean queries. In Figure 6.11, 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 *CompanyHeadquarters* 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 *CompanyHeadquarters* relation in Figure 1.1 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*, as we are using *VS* acceptors, 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. For example, the extraction pattern ⟨[], *ORGANIZATION*, [(based, 0.70), (in, 0.70)], *LOCATION*, [] ⟩ would be converted to the query *"based" AND "in"*. 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 6.1.1), which makes this approach 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.

---

was acceptable).

6.3 Experimental Results

In this section, we evaluate our techniques on the **test** database (Section 6.2.4). The statistics for the occurrence of tuples in the target relations in these databases are summarized in Figure 6.4. 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 *CompanyHeadquarters* 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 6.5. This configuration was determined by tuning the system over the **training** database (Section 6.2.4). 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 6.5.

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

Figure 6.4: The **test** database statistics.

Extraction of the *CompanyHeadquarters* relation: Figure 6.6 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 6.6(a) shows that when *QXtract* returns 10% of the database

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 6.5: Final configuration of *QXtract* as used for evaluation on the **test** database.

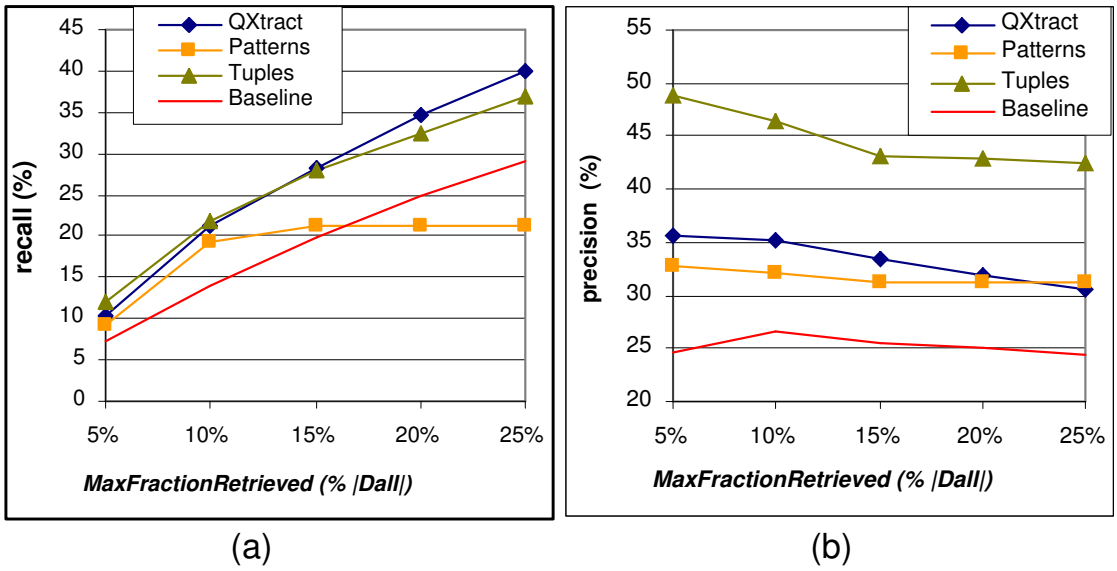


Figure 6.6: Recall (a) and precision (b) of *QXtract*, *Patterns*, *Tuples*, and *Baseline* over the **test** database using *Snowball* as the information extraction system (*CompanyHeadquarters*).

documents, *Snowball* manages to extract about 21% of the tuples in R_{all} from this reduced document set. Furthermore, Figure 6.6(b) shows that 35% of the retrieved documents are actually useful, meaning that *Snowball* managed to extract *CompanyHeadquarters* tuples from them. By comparison, *Tuples* exhibits a higher precision of 46%, while the recall re-

mains 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 *CompanyHeadquarters* tuples, allowing the tuple-based querying strategy to retrieve many of the useful documents in the database. However, as we showed in Chapter 5, such tuple distribution cannot be generally expected and, as we will see, *QXtract* is the most robust method overall. Specifically, *Tuples* will only reach a substantial portion of the relation if there are enough co-occurring tuples for the giant *Core* and *In* components to emerge. Interestingly, the *Patterns* strategy[12] 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 *CompanyHeadquarters* 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 to retrieve all matched documents. Figure 6.7 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 *CompanyHeadquarters* 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 6.5, but now using *DIPRE* as the underlying information extraction system. Figure 6.8 summarizes the recall and precision results, which are consistent with the results for *Snowball*. Figure 6.9 reports some *Pattern* and *QXtract*

[12] Many automatically generated *Snowball* patterns rely on stopwords and punctuation to extract *CompanyHeadquarters* 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 6.7: Some *Snowball Patterns* and *QXtract* queries (the *CompanyHeadquarters* relation; $MaxFractionRetrieved = 10\%$ of $|D_{all}|$).

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 6.9, *DIPRE*'s *Patterns* queries are highly specific. This prevents *Patterns* from retrieving more than 10% of the database, resulting in relatively low recall (Figure 6.8(a)), while maintaining relatively high precision (Figure 6.8(b)).

To summarize the results for the dense *CompanyHeadquarters* 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 *CompanyHeadquarters* 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 6.2.3). The results on the **test** database are shown in

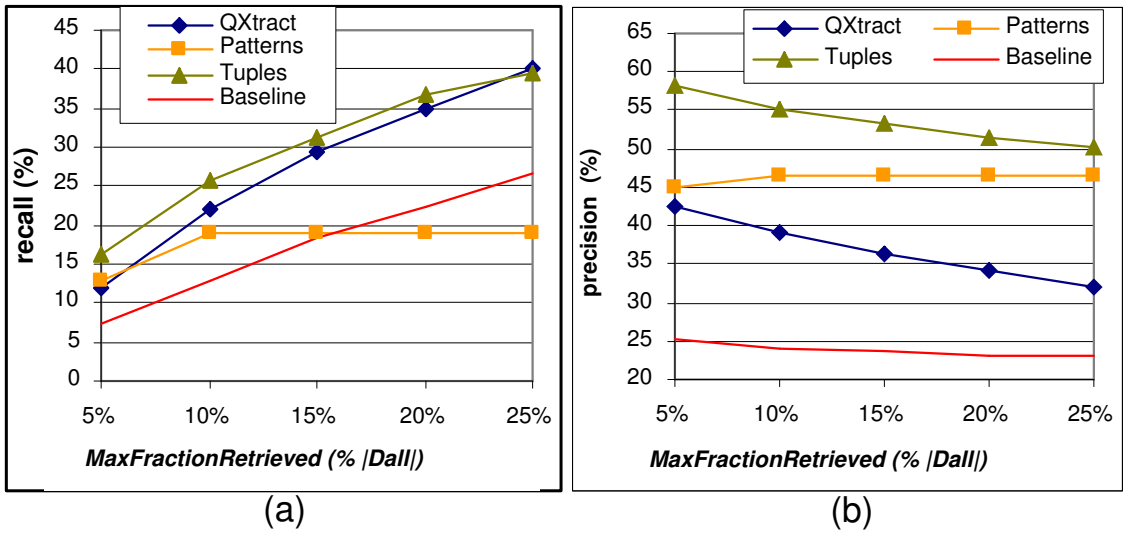


Figure 6.8: Recall (a) and precision (b) of *QXtract*, *Patterns*, *Tuples*, and *Baseline* over the **test** database using *DIPRE* as the information extraction system (*CompanyHeadquarters*).

	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 6.9: Some *DIPRE Patterns* and *QXtract* queries (the *CompanyHeadquarters* relation; $MaxFractionRetrieved = 10\%$ of $|D_{all}|$).

Figure 6.10[13]. *DiseaseOutbreaks* is a "sparse" relation, with tuples occurring in a much smaller fraction of the test database than is the case for *CompanyHeadquarters*: less than 4% of the test database documents are useful for extracting this relation (Figure 6.4). The performance of *QXtract* for this scenario is remarkable: for example, when *QXtract* retrieves only 5% of the database documents (i.e., *MaxFractionRetrieved* = 5%), *Proteus* manages to extract 48% of the tuples in R_{all} (Figure 6.10(a)). Figure 6.10(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 *CompanyHeadquarters* 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 46% recall. However, as we have shown in Chapter 5, *Tuples* would be able to retrieve a larger fraction of the relation if the *MaxSeedResults* parameter was increased to 100, 200, or even 1,000 documents per tuple-derived query. Since, during tuning, we optimized *MaxSeedResults* for precision (i.e., for creating a good set of useful documents), we did not explore larger values of *MaxSeedResults* as they tend to reduce precision of the retrieved document sets. Therefore, for different databases, collections, or parameter settings, the performance of *Tuples* might rival that of *QXtract*, and we explore the implications of having multiple good querying alternatives in Section 6.4.

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 6.10(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 6.11 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.

[13] This figure is a revised version of the one that appeared in the ICDE 2003 proceedings.

As we showed, our techniques can successfully predict the performance of *Tuples*. In particular, the behavior of *Tuples* for *DiseaseOutbreaks* over the NYT database with *MaxResults=50* agrees with our reachability-based predictions. All but a handful of the tuples retrieved by *Tuples* are in the *Core* ∪ *Out* portions of the giant component of the corresponding reachability graph, and the resulting recall of the strategy is therefore correctly predicted by the reachability value of 0.46 (Figure 5.7). We observed a similar effect with the *CompanyHeadquarters* relation. If higher recall is desired, our reachability predictions could be used to either increase –if possible– the maximum number of documents, *MaxResults*, returned for each query (at the expense of precision), or choose an alternative, more sophisticated querying strategy such as *QXtract*.

The tradeoff between the completeness of the extracted relation and the number of documents retrieved is apparent in Figure 6.10(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 6.10(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 *CompanyHeadquarters* relation and on the sparse *DiseaseOutbreaks* relation.

As we discussed, for a different combination of *MaxSeedResults* or other collection-specific parameters, *Tuples* might outperform *QXtract*. This observation motivates the estimation techniques that we will present in Section 6.4, which, together with our analysis in Chapter 5, will allow us to predict which query strategy is best for a given extraction task.

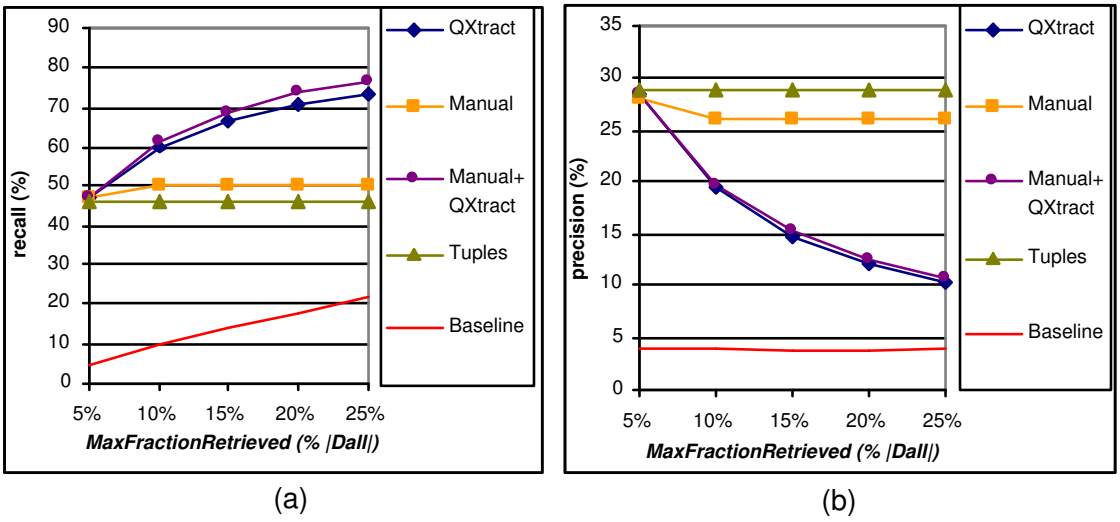


Figure 6.10: Recall (a) and precision (b) of *QXtract*, *Tuples*, *Manual*, *Manual+QXtract*, and *Baseline* over the **test** database using *Proteus* as the target information extraction system (*DiseaseOutbreaks*).

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

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 *CompanyHeadquarters* 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.

6.4 Adapting *QXtract* to New Tasks

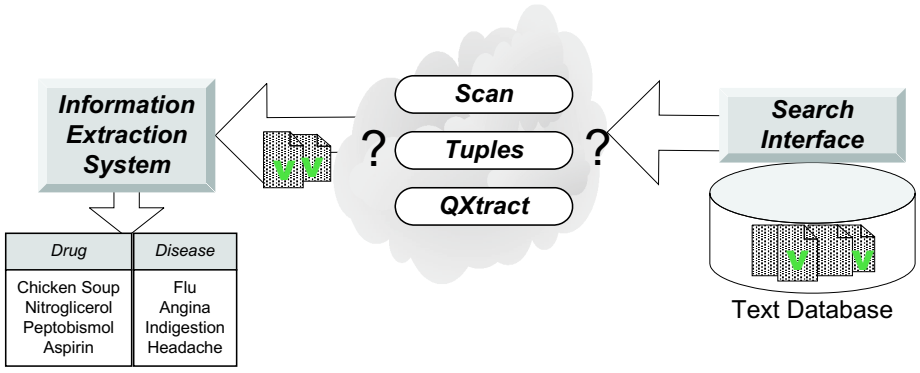


Figure 6.12: Document retrieval alternatives for extracting a relation from a text collection.

In the previous sections, we considered three general approaches for retrieving documents from a text collection for information extraction. As we saw, the effectiveness of the alternative document retrieval strategies can vary for the different tasks and constraints imposed by the search interface to the collection. Hence, an important problem is how to choose the best retrieval strategy for a new extraction task, or for a new document collection with associated search interface constraints. The problem is illustrated in Figure 6.12, which highlights the choices of scanning the collection exhaustively (*Scan*), using the *Tuples*

algorithm, or running *QXtract* for extracting the *RecommendedTreatment* relation from a new document collection.

We now summarize the main differences between the *Scan*, *Tuples*, and *QXtract* strategies with respect to extraction efficiency:

- *Scan*: Scan the complete collection, feeding every document to the information extraction system. This approach works well for relatively small collections, when most of the documents contain tuples for the target relation, and when processing each document is inexpensive relative to the time needed to retrieve the document using a search engine. In contrast, when extracting the information from a document is computationally expensive, processing every document might not be feasible for large collections. Additionally, if most documents in a collection *do not* contain tuples for the relation, *Scan* is not efficient as we would retrieve and process many useless documents. An additional consideration is accessibility: some valuable document collections, such as PubMed, are only available via querying. Hence, in these situations, only *Tuples* and *QXtract* would be viable options for document retrieval.

- *Tuples*: Use the attributes of extracted tuples to iteratively retrieve additional documents (Chapter 5). This approach is inherently more efficient than *Scan*, since the documents that *Tuples* retrieves are likely to contain tuples for the relation. As we discussed in Chapter 5, if a large enough fraction of the relation tuples are connected –via querying– we will be able to retrieve many of the tuples. In contrast, if the tuples are not connected and there is no giant *Core* component in the corresponding reachability graph, *Tuples* will "stall", retrieving only a small fraction of the relation tuples.

- *QXtract*: Derive queries to retrieve "relevant" documents (i.e., documents with tuples for the target relation) as described in this chapter. We have shown that *QXtract* is robust, as it does not depend on the existence of the giant component in the reachability graph as *Tuples* does, or on a large fraction of documents to be useful as *Scan* does. However, *QXtract* incurs the additional overhead of retrieving and processing a training document sample, while *Tuples* does not require retrieving any

"useless" documents for training.

Choosing the most appropriate document retrieval strategy is the focus of this section. In Section 6.4.1 we identify the characteristics of a document collection that determine whether *Scan*, *Tuples*, or *QXtract* should be used for retrieving the documents in the collection. Then, in Section 6.4.2 we consider different methods for retrieving a representative collection sample for estimating these properties. Later, in Section 6.4.3 we present and validate our estimation procedure. Finally, in Section 6.4.4 we discuss how to choose the best document retrieval strategy once the necessary collection characteristics are known.

6.4.1 Collection Properties

We now list the properties of a collection and extraction task that we believe have the highest impact on the choice of a document retrieval strategy:

- *Density*: This is the fraction of documents in the collection that embed tuples for the target relation. If this fraction is high (i.e., many documents are useful), the querying and retrieval overhead incurred by *Tuples* and *QXtract* is not required and would exceed the cost of simply scanning the collection and processing every document. In contrast, if *density* is low, using the *Scan* strategy and processing every document would be inefficient.

- *Reachability*: This is the expected number of documents in the collection (and target tuples hidden therein) that can be reached using the *Tuples* strategy of the previous chapter. As discussed in Chapter 5, *Tuples* iteratively queries the collection using the attributes of the extracted tuples as queries, starting from some small set of initial tuples. In order for this strategy to succeed, queries must continue to return enough documents with new tuples, until all tuples are reached. If *reachability* is high, *Tuples* would be more efficient than *QXtract*; in contrast, if *reachability* is low, *Tuples* will fail to retrieve all tuples for the target relation and *QXtract* would then be a more appropriate alternative.

In Chapter 5, we showed efficient techniques for estimating the *reachability* of a collection D for the task of extracting a relation with an information extraction system E. We now

estimate both the *density* and *reachability* using the procedure outlined in Algorithm 6.2. Our approach depends on obtaining a representative random sample D_R of documents from collection D (line 1). This sample is processed by the extraction system E to determine which documents in D_R are useful, and to extract the initial seed tuples *Seed* (line 2). This information is used to estimate the *density* of the collection for the extraction task (line 3). In the next step, we construct a small portion of the reachability graph (line 4) by starting with the *Seed* tuples extracted from D_R, which in turn allows us to estimate *reachability* (line 5) as described in Chapter 5. Note that since *QXtract* also requires a sample of useful and useless documents for training, this random sample can also be used for training *QXtract* and hence processing D_R has minimal additional overhead. We now describe the steps of Algorithm 6.2 in detail.

Algorithm EstimateCharacteristics(*E, D*)

1 D_R = RandomSample(*D*);
2 (*Seed, Useful*) = *E*.Extract(D_R);
3 *density* = EstimateDensity(D_R, *Useful*);
4 RG_S = ConstructReachabilitySubgraph(*Seed, D*);
5 *reachability* = EstimateReachability(RG_S);
return *density, reachability*;

Algorithm 6.2: Sampling-based Collection Characteristics Estimation Algorithm

6.4.2 Obtaining a Random Document Sample

We now focus on the first step of the *EstimateCharacteristics* algorithm, namely obtaining a random document sample of the collection. Obtaining a random document sample requires specifying (1) a method for choosing the documents for the sample, and (2) the size of the sample. For simplicity, we will assume that it is possible to simply request a random sample of documents from a collection (e.g., by selecting random entries from the document index). If such functionality is not available, we can follow the methodology of [CC01] and submit queries consisting of common words drawn at random from a dictionary. This strategy has been shown to retrieve a representative sample of documents from a database

and hence can serve as an approximation of a random document sample[14]. Similarly, determining the appropriate random sample size for estimating a given property is a well studied problem in the statistics, databases, and information retrieval research communities. Unfortunately, we cannot fix the sample size a-priori, since this size should be different for each collection or target relation of interest. Hence, we will use adaptive sampling techniques to automatically determine the appropriate sample size for each new scenario. Also, for the *QXtract* training to be meaningful, and for *Tuples* to operate, we need at least *tMin* tuples in the random document sample D_R, where we set *tMin*=50 based on statistical considerations. We will consider three representative adaptive sampling strategies, *MINIMAL*, *DOUBLE*, and *ADAPTIVE*, for creating our random sample D_R:

- *MINIMAL*: The *MINIMAL* sampling strategy retrieves the smallest (randomly drawn) set of documents from the collection with at least *tMin* tuples embedded in the documents. To accomplish this goal, *MINIMAL* repeatedly retrieves a small number (e.g., 10) of randomly chosen documents to be included in D_R until D_R contains at least *tMin* tuples.

- *DOUBLE*: The *DOUBLE* strategy is well studied in the statistics literature (e.g., see [Tho92]) and has been used successfully for estimating the expected size of query results [HÖD91, LS95]. We start by taking a small, *pilot* random sample of size m_1 (typically m_1 is between 100 and 200 documents) and, based on the estimated density over that pilot sample, calculate the size of the final sample. The *pilot* sample is drawn from the original database of size N and is classified into u_1 useful and $m_1 - u_1$ useless documents. The initial estimate for density, $\hat{p}$, is then $\frac{u_1}{m_1}$. Using the results in [HÖD91], we can estimate the optimal sample size S_{opt} as:

$$|S_{opt}| = \frac{(\frac{t_{1-\alpha}}{\epsilon})^2 \cdot (1-\hat{p})}{\hat{p}} + \frac{3}{\hat{p} \cdot (1-\hat{p})} + \frac{t_{1-\alpha}^2}{\epsilon^2 \cdot \hat{p} \cdot m_1} \tag{6.1}$$

  where ϵ is the desired relative accuracy, $\alpha = 0.05$, and $t_{1-\alpha}$ is the value of the t-distribution at $1-\alpha$.

[14] Our experiments with sampling document collections using the methodology of [CC01] confirm this observation.

- *ADAPTIVE*: A drawback of the *DOUBLE* method above is that it is sensitive to the composition of the pilot sample. One solution is to run multiple trials of *DOUBLE*. A more general adaptive sampling approach is proposed in [CMN98] as a robust method for constructing histograms. Specifically, *ADAPTIVE* starts with a small initial sample s_0 and proceeds to retrieve larger samples $s_1, s_2, ..., s_n$ such that $s_i = s_{i-1} \cdot 2^{i-1}$ for $i = 1, ..., n$, until the last sample s_n provides the density estimate $\hat{p_n}$ that is within ϵ of the previous estimate $\hat{p_{n-1}}$. We modified the original procedure to require the convergence criteria to be met at least twice. As described in [HS92, LS95], requiring the algorithm to converge at least two times will, most of the time, result in more accurate density estimates.

Having described our techniques for obtaining a representative random sample of the document collection, we now turn to estimating *density*. This is an appropriate place to point out a limitation of our random sampling approach to initialize the estimation procedure. Unfortunately, our approach may not work for collections with extremely low *density* (e.g., *density* < 0.01), since to acquire a random sample with sufficiently many useful documents would be prohibitively expensive. For example, for a collection with a *density* of 0.005, constructing a random sample with approximately 50 embedded tuples will, on average, require retrieving and processing more than 7,000 documents[15]. Estimating *density* and *reachability* for collections with density lower than 0.01 will require adapting more sophisticated sampling techniques such as graph sampling, or relaxing our restrictions on the composition of the random sample (e.g., lowering the *tMin* threshold to, say, 5, 10, or 20 seed tuples).

6.4.3 Estimating Density

Having obtained our random document sample D_R, we extract the *Seed* tuples for the target relation, which also gives us the number of useful documents *Useful* (See Algorithm 6.2).

[15]The required sample size can be estimated as the expected size of a random document sample that will contain at least 50 useful documents with probability of at least 0.99.

We can now estimate *density* (line 3) as the fraction of the useful documents in D_R:

$$density = \frac{|Useful|}{|D_R|} \tag{6.2}$$

If there are no useful documents in the sample, we could compute a one-sided confidence interval following [Cox92]. For a 1-α confidence interval given no useful documents in our sample $|D_R|$, the confidence interval for *density* is:

$$0 \leq density \leq (1-\alpha)^{|D_R|} \tag{6.3}$$

Unfortunately, if no useful documents are present in D_R, then the sample is useless for estimating *reachability* and for training *QXtract*. Therefore, we consider D to be so sparse with respect to our extraction task that our estimation techniques are not applicable.

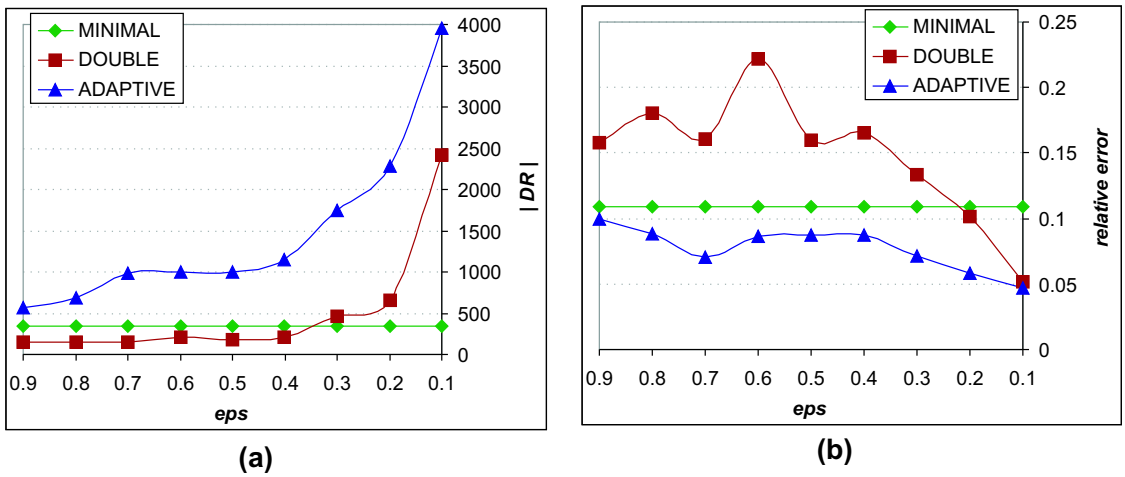


Figure 6.13: The average sample size $|DR|$ vs. ϵ *eps* (a), and the average relative error vs. ϵ *eps* (b) for *MINIMAL*, *DOUBLE*, and *ADAPTIVE* sampling methods (the *DiseaseOutbreaks* and *CompanyHeadquarters* relations, 10 trials).

We now empirically compare the alternative random sampling methods for our *density* estimation task. We report the results in Figures 6.13 (a) and (b). Figure 6.13 (a) reports the sample size, averaged over the two relations, for values of ϵ ranging from 0.9 to 0.1. Figure 6.13 (b) reports the mean relative error for varying ϵ, reported as *eps* in the plot. The *ADAPTIVE* strategy achieves the highest accuracy, with the lowest mean relative error across trials. In contrast, *MINIMAL* and *DOUBLE* are more economical, as they retrieve

fewer documents on average, but tend to have higher estimation error. As we discussed in Chapter 5, we can estimate *reachability* by starting with the *Seed* tuples included in the random sample D_R.

To summarize, we presented an automatic procedure for adaptively sampling a document collection for estimating its *density* and *reachability* for an extraction task. We now describe how these values can be used to choose the best document retrieval strategy for the collection and extraction task.

6.4.4 Choosing the Best Document Retrieval Strategy

As we discussed, the *density* and *reachability* of a database can help us choose the most promising document retrieval strategy for an extraction task over the database. Specifically, if the *density* of the database is high (e.g., 0.5 or higher), it might be best to simply scan the database. During the scan, we could potentially filter the documents using text filtering and classification techniques, as described in Chapter 2, before feeding the remaining documents to the information extraction system. This scan would avoid the additional overhead of querying the database to retrieve documents. Since the precision of *QXtract* and *Tuples* rarely exceeds 50% in our experiments, scanning a collection with a 0.5 density or higher is likely to be the most efficient document retrieval alternative. Similarly, if the *reachability* of the database is high enough, then *Tuples* would be the document retrieval strategy of choice, as it incurs relatively low overhead while maintaining high precision of the retrieved promising document set. However, if the *density* of the collection is low (e.g., less than 0.1) and the estimated *reachability* is not sufficiently high, we can use *QXtract* as the document retrieval method of choice. While *QXtract* does require an additional training step, as we showed above, *QXtract* is general, scalable, and robust in that it is able to perform competitively on all extraction tasks on which it was evaluated. Finally, note that our estimation techniques require almost no additional overhead beyond what is required for initializing and training *QXtract* and *Tuples*. Specifically, the random sample that we use to estimate *density* can be reused by *QXtract* for query training; similarly, the tuples extracted from the random sample can be used to initialize *Tuples*. Hence, our techniques are a significant step towards adaptive, scalable, query-based information extraction that

can be easily applied to new extraction tasks and databases of interest.

6.5 Discussion

In this chapter, 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 a significant improvement in extraction efficiency in the number of documents processed: for the *CompanyHeadquarters* 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. We also presented techniques for estimating the *reachability* and the *density* of a document collection with respect to a relation extraction task. These properties can suggest the best document retrieval strategy to use for a given (new) document collection and extraction task. In addition to improving the 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 [LLC]. 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 believe that the techniques presented here and embodied in *QXtract* can be of significant help for such scenarios.

Chapter 7

Conclusions and Future Work

In this thesis, we presented techniques for extracting structured relations from large text collections. The extracted relations can be used for sophisticated query processing, for integration with relational databases, and for traditional data mining tasks. To construct such relations, and as a key contribution of this thesis, we developed *Snowball*, a partially supervised, domain-independent information extraction system. *Snowball* needs only a handful of example relation instances as input, and proceeds to automatically acquire extraction patterns for expanding the target relation. If implemented naively, this approach might suffer from poor accuracy. We developed methods for improving the extraction accuracy by evaluating the quality of the extraction patterns, and by combining evidence on the correctness of the extracted relation instances from multiple text occurrences. The *Snowball* approach is general and can be applied for diverse applications. For example, we have used *Snowball* for mining the biological literature for synonymous gene and protein terms [YA03], for extracting information about infectious disease outbreaks, and for extracting a relation associating prescription drugs with their side effects, among other tasks.

Extracting structured information from text documents often requires sophisticated and computationally intensive techniques. Processing every available document is not feasible for the web or for large searchable databases such as PubMed. It is also highly inefficient, since only a small fraction of documents in a collection are often useful for most information extraction tasks. Hence, to extract information from large text collections *efficiently*, we must be able to zoom in quickly on the relevant documents. As another contribution of

this thesis, we analyzed *Tuples*, a simple query-based document retrieval strategy that exploits a collection's search interface to retrieve documents. To complement our analysis, we presented efficient approximation techniques that can predict whether such a query-based strategy could be successful for retrieving a complete target relation from a document collection.

Complementing our analysis of the *Tuples* querying strategy, we introduced *QXtract*, a query-based system that is based on machine learning and information retrieval results. In many cases the *Tuples* strategy will successfully reach the majority of the relation tuples. However, due to either restrictions of the search interface on the number of submitted queries, or if the relation tuples do not tend to co-occur in individual documents, *Tuples* can fail to retrieve new documents, and hence new tuples for the relation. Therefore, *QXtract* is a valuable complementary technique as it does not rely on tuple co-occurrence, and instead learns general queries that are expected to retrieve useful documents with many embedded tuples. By exploiting existing search engines, *QXtract* can operate over local document collections, over the web at large, and over remote searchable databases. 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.

The specific contributions presented in this thesis include:

- **Techniques for domain-independent, partially supervised information extraction**: We presented and thoroughly evaluated *Snowball*, a minimally-supervised information extraction system for extracting structured relations from large text collections (Chapters 3 and 4).

- **Methods to help adapt information extraction systems to new domains**: We presented techniques for *automatically* estimating important *Snowball* parameters, thus further easing the effort of porting *Snowball* to new domains. We evaluated our techniques over diverse relations including corporate acquisitions, disease outbreaks, and reported side effects of prescription drugs (Chapter 4).

- **The first principled analysis of query-based access to text databases for**

information extraction: Understanding how to retrieve documents for information extraction via querying is central to scaling up information extraction to large document collections. We presented a general model that can help understand and analyze an important class of query-based strategies (Chapter 5).

- **A robust and scalable query-based system for information extraction**: Finally, we presented a domain-independent and general architecture for improving the efficiency of information extraction systems (Chapter 6). Among other contributions, *QXtract* could be used to query a standard web search engine, hence providing infrastructure for efficient information extraction from the web at large.

Our work suggests interesting directions for future research, outlined below. First we discuss some immediate extensions of our work. Then we suggest longer-term research directions that naturally build on techniques developed in this thesis.

- **Extending supported attribute types:** As we discussed, *Snowball* is designed primarily to extract relations between named entities. As we showed, this allows *Snowball* to extract a variety of relations. We could expand the applicability of *Snowball* by relaxing this named-entity assumption, allowing *Snowball* to extract relations between arbitrary "entities" such as book and movie titles and descriptive phrases.

- **Exploiting explicit inter-document relationships:** In our work on partially supervised relation extraction, we focused on extracting tuples from each individual document in isolation. However, additional evidence supporting the validity of a candidate tuple for a relation can be available from other documents or external sources. For example, a drug often mentioned in cancer-related websites is more likely to be a cancer-related drug. Thus, a promising research direction would be to enrich the *Snowball* extraction model with "global" information about documents, links between them, and collections as a whole.

- **Robust handling of n-ary relations:** While the ideas behind *Snowball* are general, we have so far evaluated *Snowball* only on binary relations. Handling n-ary relations in a robust way may require relaxing our assumptions that all tuple attribute values

occur within the same document and that all of the values for a tuple can be extracted by a single pattern. One direction to explore would be to decompose n-ary relations into more manageable binary "chunks", which (after filling the attribute values) could be composed together to construct the target (n-ary) relation.

- **General cost models for information extraction:** As we have seen in Chapters 5 and 6, document retrieval strategies can have similar behaviors making it sometimes difficult to select the best strategy. Furthermore, with rapid advances in hardware and computing, cost assumptions (e.g., that scanning a collection is more expensive than, say, querying a search engine over the collection) can easily become obsolete, which could reverse conclusions about the best querying strategy. Hence, a promising and important direction is to develop parameterized cost models for information extraction, which could then be easily updated as needed.

- **Integrating structured and unstructured information via information extraction:** To support sophisticated querying and analysis of text data in relational databases and data warehouses, it would be extremely valuable to integrate information extraction functionality into relational database systems. The wealth of text information contained in relational databases and data warehouses can only be used in limited ways by current query processing engines and data mining algorithms. By presenting a structured view of the unstructured (text) data, a user could issue sophisticated queries over these "views" in combination with querying regular tables in the database. For example, it would be possible to support queries such as *"Select all employees under 35 who like to play soccer from tables Employee and Extracted-Hobbies,"* where the hobbies of employees are extracted from text postings in the company newsgroup archive. Maintaining the illusion of structured views over the unstructured data raises a variety of new issues: how to define and create such views with minimal overhead for the user; how to select which views are particularly beneficial to materialize; and how to efficiently use and update the structural view of the data for changing document collections.

In this thesis, we addressed two fundamental problems in extracting relations from large

text collections: *portability* and *scalability*. To address the first problem, we developed the *Snowball* information extraction system, a domain-independent system that learns to extract relations from unstructured text based on only a handful of user-provided example relation instances. *Snowball* can then be adapted to extract new relations with minimum human effort. To address the second problem, we developed the *QXtract* system, which learns search engine queries to retrieve the documents that are relevant to a given information extraction system and extraction task. *QXtract* can dramatically improve the efficiency of the information extraction process. Together, *Snowball* and *QXtract* provide building blocks for extracting structured information and text data mining from the web at large. We hope that the techniques and ideas presented in this thesis can be beneficial to the research community.

Bibliography

[ABH98] Diego Aliod, Jawad Berri, and Michael Hess. A real world implementation of answer extraction. In *Proceedings of the Workshop on Natural Language and Information Systems (NLIS-98)*, 1998.

[ACL00] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the ACM Symposium on Theory of Computing (STOC 2000)*, 2000.

[ACS00] Steven Abney, Michael Collins, and Amit Singhal. Answer extraction. In *Proceedings of the Applied Natural Language Processing Conference (ANLP)*, 2000.

[AEG00] Eugene Agichtein, Eleazar Eskin, and Luis Gravano. Combining strategies for extracting relations from text collections. In *Proceedings of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, 2000.

[AG00] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the ACM International Conference on Digital Libraries*, 2000.

[AG03] Eugene Agichtein and Luis Gravano. Querying text databases for efficient information extraction. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2003.

[AGP+01] Eugene Agichtein, Luis Gravano, Jeff Pavel, Viktoriya Sokolova, and Alexander Voskoboynik. Snowball: A prototype system for extracting relations from

plain-text collections (demonstration). In *Proceedings of the ACM SIGMOD Conference*, 2001.

[AHJG03] Eugene Agichtein, C. T. Howard Ho, Vanja Josifovski, and Joerg Gerhardt. Extracting relations from XML documents. In *Proceedings of the International Workshop on XML Schema and Data Management (XSDM)*, 2003.

[AIG03] Eugene Agichtein, Panagiotis Ipeirotis, and Luis Gravano. Modelling query-based access to text databases. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, 2003.

[ALG01] Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning search engine specific query transformations for question answering. In *Proceedings of the International World Wide Web Conference (WWW)*, 2001.

[ALG04] Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning to find answers to questions on the web. In *ACM Transactions on Internet Technology (TOIT)*, 2004.

[Bar03] Regina Barzilay. *Information Fusion for Multidocument Summarization*. Ph.D. Thesis, Columbia University, 2003.

[BC] Breck Baldwin and Bob Carpenter. The lingpipe system. Available at `http://www.alias-i.com/lingpipe/index.html`.

[BDB02] Erik Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[BKM+00] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph

structure in the web. In *Proceedings of the International World Wide Web Conference (WWW)*, 2000.

[BM98] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 1998 Conference on Computational Learning Theory (COLT)*, 1998.

[Bri98] Sergey Brin. Extracting patterns and relations from the World-Wide Web. In *Proceedings of the International Workshop on the Web and Databases (WebDB)*, 1998.

[CC01] Jamie Callan and Margaret Connell. Query-based sampling of text databases. *ACM Transactions on Information Systems (TOIS)*, 2001.

[CDF+99] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Sean Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 1999.

[CL02] Fan Chung and Linyuan Lu. Connected components in random graphs with given degree sequences. *Annals of Combinatorics*, 2002.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. McGraw-Hill Science, March 1990.

[CM98] Mary E. Califf and Ray J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, 1998.

[CM99] Mary E. Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Sixteenth National Conference on Artificial Intelligence*, 1999.

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[CMN98] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: How much is enough? In *Proceedings of the ACM SIGMOD Conference*, 1998.

[CNPB00] Claire Cardie, Vincent Ng, David Pierce, and Chris Buckley. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. In *Proceedings of the Applied Natural Language Processing Conference (ANLP)*, 2000.

[Coh95] William Cohen. Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1995.

[Coh98] William Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[Cox92] C. Phillip Cox. *A Handbook of Introductory Statistical Methods.* Wiley Series in Probability and Mathematical Statistics, 1992.

[CPS02] Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the International World Wide Web Conference (WWW)*, 2002.

[CS96] William Cohen and Yoram Singer. Learning to query the web. In *Proceedings of the AAAI Workshop on Internet-Based Information Systems*, 1996.

[CS99] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[CWG+92] Jim Cowie, Takahiro Wakao, Louise Guthrie, Wang Jin, James Pustejovsky, and Scott Waterman. The Diderot information extraction system. In *Proceedings of the 4th Message Understanding Conference*, 1992.

[DAH+97] David Day, John Aberdeen, Lynette Hirschman, Robyn Kozierok, Patricia Robinson, and Marc Vilain. Mixed-initiative development of language processing systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, April 1997.

[DLR77] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 1977.

[ECD+04] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in KnowItAll: (preliminary results). In *Proceedings of the International World Wide Web Conference (WWW)*, 2004.

[FAFL+02] Ronen Feldman, Yonatan Aumann, Michal Finkelstein-Landau, Eyal Hurvitz, Yizhar Regev, and Ariel Yaroshevich. A comparative study of information extraction strategies. In *Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, 2002.

[FBY92] William B. Frakes and Ricardo Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of the ACM International Conference on Data Communication (SIGCOMM)*, 1999.

[FGLG02] Gary Flake, Eric J. Glover, Steve Lawrence, and C. Lee Giles. Extracting query modifications from nonlinear SVMs. In *Proceedings of the International World Wide Web Conference (WWW)*, 2002.

[FHE03] Michael Fleischman, Eduard Hovy, and Abdessamad Echihabi. Offline strategies for online question answering: Answering questions before they are asked. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2003.

[Fre98] Dayne Freitag. *Machine Learning for Information Extraction in Informal Domains*. Ph.D. Thesis, Carnegie Mellon University, 1998.

[FSM+95] David Fisher, Stephen Soderland, Joseph McCarthy, Fangfang Feng, and Wendy Lehnert. Description of the UMass systems as used for MUC-6. In *Proceedings of the 6th Message Understanding Conference*, 1995.

[GHY02] Ralph Grishman, Silja Huttunen, and Roman Yangarber. Real-time event extraction for infectious disease outbreaks. In *Proceedings of Human Language Technology Conference (HLT)*, 2002.

[GJ02] Rayid Ghani and Rosie Jones. Automatic training data collection for semi-supervised learning of information extraction systems. Technical report, Accenture Technology Labs, 2002.

[GJM01] Rayid Ghani, Rosie Jones, and Dunja Mladenic. Mining the web to create minority language corpora. In *Proceedings of the International Conference on Knowledge Management (CIKM)*, 2001.

[GR97] Robert Gaizauskas and Alexander 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.

[Gri97] Ralph Grishman. Information extraction: Techniques and challenges. In *Information Extraction (International Summer School SCIE-97)*. Springer-Verlag, 1997.

[HÖD91] Wen-Chi Hou, Gultekin Özsoyoglu, and Erdogan Dogdu. Error-constraint count query evaluation in relational databases. In *Proceedings of the ACM SIGMOD Conference*, 1991.

[HPM00] Sanda M. Harabagiu, Marius A. Pasca, and Steven J. Maiorano. Experiments with open-domain textual question answering. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, 2000.

[HS92] Peter J. Haas and Arun N. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the ACM SIGMOD Conference*, 1992.

[HSG04] Takaaki Hasegawa, Satoshi Sekine, and Ralph Grishman. Discovering relations among named entities from large corpora. In *Proceedings of the Annual Meeting of Association of Computational Linguistics (ACL)*, 2004.

[IGS03] Panagiotis G. Ipeirotis, Luis Gravano, and Mehran Sahami. Qprober: A system for automatic classification of hidden-web resources. *ACM Transactions on Information Systems (TOIS)*, 2003.

[Joa98] Thorsten Joachims. Making large-scale support vector machine learning practical. *Advances in Kernel Methods: Support Vector Machines*, 1998.

[KEW01] Cody C. T. Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the web. In *Proceedings of the International World Wide Web Conference (WWW)*, 2001.

[KGC+00] Andries Kruger, C. Lee Giles, Frans Coetzee, Eric J. Glover, Gary William Flake, Steve Lawrence, and Christian W. Omlin. DEADLINER: Building a new niche search engine. In *Proceedings of the International Conference on Knowledge Management (CIKM)*, 2000.

[KWD97] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.

[LG98] Steve Lawrence and C. Lee Giles. Context and page analysis for improved web search. *IEEE Internet Computing*, 1998.

[LLC] BrightPlanet.com LLC. The Deep Web: Surfacing hidden value. Available at `http://www.completeplanet.com/Tutorials/DeepWeb/index.asp`.

[LLYL02] Bing Liu, Wee Sun Lee, Philip S Yu, and Xiaoli Li. Partially supervised classification of text documents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2002.

[LS95] Yibei Ling and Wei Sun. An evaluation of sampling-based size estimation methods for selections in database systems. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 1995.

[LT92] David Lewis and Richard Tong. Text filtering in MUC-3 and MUC-4. In *Proceedings of the 4th Message Understanding Conference*, 1992.

[MHP+99] Dan Moldovan, Sanda Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Girju, and Vasile Rus. Lasso: A tool for surfing the answer net. In *Proceedings of TREC-8*, 1999.

[Mit97] Tom Mitchell. *Machine Learning.* McGraw Hill, 1997.

[Mla98] Dunja Mladenic. Feature subset selection in text-learning. In *Proceedings of the European Conference on Machine Learning (ECML)*, 1998.

[MW94] Udi Manber and Sun Wu. Glimpse: A tool to search through entire file systems. In *Proceedings of the 1994 Winter USENIX Conference*, January 1994.

[Rep87] Defense Advanced Research Projects Agency Report. *Proceedings of the 1st Message Understanding Conference (MUC-1).* Morgan Kaufman, 1987.

[Rep91] Defense Advanced Research Projects Agency Report. *Proceedings of the 3rd Message Understanding Conference (MUC-3).* Morgan Kaufman, 1991.

[Rep93] Defense Advanced Research Projects Agency Report. *Proceedings of the 5th Message Understanding Conference (MUC-5).* Morgan Kaufman, 1993.

[Rep95] Defense Advanced Research Projects Agency Report. *Proceedings of the 6th Message Understanding Conference (MUC-6)*. Morgan Kaufman, 1995.

[Rep98] Defense Advanced Research Projects Agency Report. *Proceedings of the 7th Message Understanding Conference (MUC-7)*. Morgan Kaufman, 1998.

[RG03] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2003.

[RGM01] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[RH02] Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[Ril96] Ellen Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[RJ76] Stephen E. Robertson and Karen Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 1976.

[RJ99] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[Rob90] S.E. Robertson. On term selection for query expansion. In *Journal of Documentation*, 1990.

[Roc71] J. Rocchio. Relevance feedback in information retrieval. In *SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[RQZ+01] Dragomir R. Radev, Hong Qi, Zhiping Zheng, Sasha Blair-Goldensohn, Zhu Zhang, Waiguo Fan, and John Prager. Mining the web for answers to natural

language questions. In *Proceedings of the International Conference on Knowledge Management (CIKM)*, 2001.

[Sal89] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[SB88] Gerald Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 1988.

[Sha97] Mehul A. Shah. *ReferralWeb: A Resource Location System Guided by Personal Relations*. M.S. Thesis, M.I.T., 1997.

[Tho92] Steven K. Thompson. *Sampling*. Wiley Series in Probability and Mathematical Statistics, 1992.

[Voo99] Ellen M. Voorhees. The TREC-8 question answering track report. In *Proceedings of TREC-8*, 1999.

[XC00] Jinxi Xu and W. Bruce Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Transactions on Information Systems (TOIS)*, 2000.

[YA03] Hong Yu and Eugene Agichtein. Extracting synonymous gene and protein terms from biological literature. *Bioinformatics*, 2003.

[Yar95] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 1995.

[YG98] Roman Yangarber and Ralph 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.

[YGTH00] Roman Yangarber, Ralph Grishman, Pasi Tapanainen, and Silja Huttunen. Unsupervised discovery of scenario-level patterns for information extraction. In *Proceedings of the Applied Natural Language Processing Conference (ANLP)*, 2000.

[YP97] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1997.

[YS99] Jeonghee Yi and Neel Sundaresan. Mining the web for acronyms using the duality of patterns and relations. In *Proceedings of the International Workshop on Web Information and Data Management (WIDM)*, 1999.

[Zip49] George K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[ZWS04] Zhaohui Zheng, Xiaoyun Wu, and Rohini Srihari. Feature selection for text categorization on imbalanced data. *SIGKDD Explorations*, 2004.