Query Expansion Techniques for Question Answering

by

Matthew W. Bilotti

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .....
Boris Katz
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Query Expansion Techniques for Question Answering

by

Matthew W. Bilotti

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Query expansion is a technique used to boost performance of a document retrieval engine, such as those commonly found in question answering (QA) systems. Common methods of query expansion for Boolean keyword-based document retrieval engines include inserting query terms, such as alternate inflectional or derivational forms generated from existing query terms, or dropping query terms that are, for example, deemed to be too restrictive. In this thesis, I present a quantitative evaluation against a test collection of my own design of five query expansion techniques, two term expansion methods and three term-dropping strategies. I present results that show that there exist best-performing query expansion algorithms that can be experimentally optimized for specific tasks. My findings pose questions that suggest interesting avenues for further study of query expansion algorithms.

Thesis Supervisor: Boris Katz
Title: Principal Research Scientist

Acknowledgments

I am indebted to many people who have helped to make this thesis possible:

- To Boris Katz, for his guidance and support, for his boundless patience, and for welcoming me into his group in January of 2001; and
- To Jimmy Lin, Gregory Marton and Sue Felshin, for teaching me much of what I know about science and language processing research; and
- To Kimberle Koile and Patrick Winston, for their encouragement and timely advice; and
- To Stefanie Tellex and Aaron Fernandes, whose previous work in this area benefited this thesis enormously; and
- To Leah Oats, for working by my side for hours at a time, and for her dedication to this project; and
- To Caitlin Dwyer-McNally, for taking care of me, and for always having a smile and some warm words for me whenever I needed them; and
- To William and Celeste Bilotti, and to the rest of my family, who have made uncountable sacrifices on my behalf, without which I could have never gotten this far; and
- To Alexander Bilotti, for always understanding me and for always being a good listener; and
- To Vincent Bilotti and Candido Pensotti, whom I don't remember, but without whom none of this could have been possible.

This thesis could not have been completed were it not for the selfless contributions of the above named individuals, and it stands as a monument to their generosity. To each, I offer my most humble thanks, and to them, I dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The development of systems that interact with human users in natural language has long been a goal of the artificial intelligence research community. Since the 1960s, when the field was in its infancy, a variety of natural language database front-ends, dialog systems, and language understanding systems have been created. Each subsequent system has demonstrated mastery over a slowly increasing subset of the English language, and has proposed a solution to the problem of mediating human access to electronic information in some limited domain. Available processing power and linguistic resources have improved markedly since the time of the early natural language understanding systems, and interest in general purpose natural language interfaces has scaled commensurately. Perhaps the best example of this is an ambitious task known as question answering.

1.1 What is Question Answering?

Open-domain question answering (QA) is an area of natural language processing research aimed at providing human users with a convenient and natural interface for accessing information. QA is often viewed as a combination of two related, more established information access tasks known as information retrieval (IR) and information extraction (IE), but unlike them, the goal of QA is to provide exact, precise answers to human users' questions posed in natural language. For more information

on these related tasks, see Section 2.2.

Recently popularized by efforts to provide large-scale evaluation of QA systems, the field is quickly growing. A large number of research groups are interested in developing QA systems, in both academia and industry, and from around the world. The QA research community's vision is aggressive; placing heavy emphasis on accuracy of correct, unambiguous answers, and on systems that can understand how confident they are in their answers, and whether there are no answers found among the documents to which they have access.

A parallel movement in the QA community is highly interested in studying the user's interaction with the system, and the system's role in the user's work environment. Some feel that for a QA system to be useful to human users, it has to be cognizant of the context in which the user is asking the question, and of his or her purpose in asking it. An eventual goal of the community is to be able to build systems that support follow-up questions from the user, or requests for clarification. To be able to deliver a response tailored precisely to the information the user is seeking, the system must understand the context of and motive for the questioning dialog and have a model of what the user already knows.

Still another subarea of interest to some QA researchers is the indexing and retrieval of mixed media. As media access technologies improve, QA systems will eventually be searching not only text documents, such as those found on the web and in corpora of newswire articles, but also clips of video and audio, such as broadcast news. These forms of media contain information that can be used as source material for answers to users' questions, or as supporting evidence for answer justification. For more information about the vision of the QA research community, see the document developed by the QA Roadmap Committee [4].

1.2 Approaches to Question Answering

The general QA approach is prescribed by the nature of the task itself. Systems must provide some facility for analyzing the question, to understand what is being

asked for. They must also be able to quickly and efficiently search for documents or passages relevant to the question, in order to search for candidate answers. Finally, systems need to locate the extents of these answers and choose the best of them to present to the user.

This approach is often described as a QA pipeline, in which natural language questions flow into the first module, which is responsible for question analysis, and answers flow out of the last module, in which the answer is extracted and packaged into a response for the user. Modules are chained such that the output of an upstream module is connected directly to the input of the next adjacent downstream module. This general approach is known to have solid performance on answering short-answer, factual questions such as those focused on by the first few QA tracks of the Text REtrieval Conferences (TREC) [35].

A variety of research groups are augmenting this minimal approach with techniques such as question type ontologies, databases of external knowledge, heuristics for extracting answers of certain types, generation of answers, answer justifications, inference rules, feedback loops, machine learning and even logical analysis.

1.3 The Role of Document Retrieval

Whatever QA architecture is chosen, answering questions over a closed corpus or even the web almost always involves some kind of searching for and retrieval of documents as a first step to narrow the search space for the answer to the question [10].

Within the context of a pipelined QA system, this retrieval step generally takes the form of an upstream IR module that extracts relevant documents from the corpus of interest, prior to sending them to a downstream answer extraction module, which is responsible for generating candidate answers from them. The IR component in this context is known as a document retrieval module. Many pipelined systems also have a passage retrieval stage, interposed between the document retrieval and answer extraction components, which can be thought of as a second, smaller scale IR module. My colleagues and I, in the Infolab group of the MIT Computer Science and

Artificial Intelligence Laboratory, have a great deal of experience with pipelined QA architectures, as our submissions to the TREC 2002 and 2003 tracks indicate [21, 18]. For more information about these systems, see Chapter 3.

A previous study conducted by Tellex *et al.* [31] of the Infolab group has identified the importance of high-quality document retrieval to the performance of downstream passage retrieval modules, and of pipelined QA systems in general. The supposition is that maximal performance of the system as a whole depends on high recall in upstream stages, specifically document retrieval, and high precision in downstream stages, such as passage retrieval and answer extraction. Intuition confirms that recall is essential in the early stages of a QA pipeline because documents that are not retrieved can never be analyzed for relevant passages, and those passages can never be searched for reasonable answer candidates. Precision is clearly important throughout the whole question answering process, especially since the most recent TREC QA tracks required a single, exact answer for each question, but the claim is that precision should be allowed to decrease in favor of improved recall at the document retrieval stage.

1.4 What Query Expansion can Contribute

Query expansion is a name given to a class of techniques in which a query serving as input to a document retriever is evolved in some way with the intent that the change will improve the document retriever's performance, according to some metric. Query expansion is particularly applicable to document retrieval components that provide a Boolean query model, because of the expressiveness of the syntax and ease of modifying existing queries. This thesis will focus on query expansion techniques for a Boolean keyword and phrase document retrieval engine, which, as demonstrated in previous work by the Infolab group, can perform as well if not better when retrieving documents for question answering than other, more sophisticated retrieval methods, even though those methods outperform it in terms of raw retrieval [31].

The hypothesis is that cleverly designed query expansion techniques will improve recall of documents that are relevant to the query. The intention is that there will

be more relevant documents in the list of retrieved documents, and they will be more highly ranked, with query expansion than without it. Improving document retrieval in this way would provide the best possible input to a downstream passage retrieval or analysis module in a pipelined QA system.

1.5 Contributions of this Thesis

This thesis is the direct result of research I have done toward the development of a high-performance document retrieval module for incorporation in a pipelined QA system, one which will form the foundation for the Infolab's TREC 2004 competition system. The work I have done offers the following contributions toward the problem of improving document retrieval performance for question answering:

- Design and development of a test collection and accompanying suite of evaluation tools used to measure document retrieval performance. The test collection provides lists of documents known to be relevant, unsupported or irrelevant for selected TREC 2002 questions and is intended to provide better coverage than the relevant document lists provided by the TREC organizers.
- Discussion of evaluation techniques for query expansion algorithms and the outline of an approach for building an optimized query expansion algorithm given a specific document retrieval engine.
- Comprehensive evaluation of query expansion techniques such as morphological and derivational query term expansion, and term-dropping strategies, applied to a Boolean keyword and phrase document retrieval engine.

1.6 Outline of this Thesis

The remaining chapters of this thesis are organized as follows:

- In **Chapter 2**, I summarize the historical origins of research in natural language understanding systems, and trace the development of question answering as a

task of interest to the research community.

- In **Chapter 3**, I describe two systems central to Infolab's involvement in TREC-style QA: Aranea, developed for TREC 2002, and Pauchok, developed for TREC 2003.

- In **Chapter 4**, I tell of my work on Pauchok II, a second system based on the TREC 2003 version of Pauchok. Pauchok II is the system that underlies the work done in this thesis, and that will provide document retrieval support for Infolab's TREC 2004 QA effort.

- In **Chapter 5**, I discuss issues of evaluating document retrieval for question answering, and the need for a quality test collection. I describe the building of such a test collection, one that I later use in this thesis to evaluate my work.

- In **Chapter 6**, I give the details of my approach to the task of evaluating the performance of a variety of components of query expansion algorithms provided by the Pauchok II system, and I share the results of my experiments. The query expansion algorithm that will be used for TREC 2004 will be built from the best performing components available in Pauchok II.

- In **Chapter 7**, I make some concluding remarks about improving document retrieval for question answering, identify the primary contributions of this thesis, and propose promising avenues for future investigation into this problem.

Chapter 2

Background Information

In this chapter, I outline the foundations of question answering (QA) and explore the historical roots of QA research. The purpose of this chapter is to give the reader a grounding in the history and current directions of QA research.

2.1 The History of Question Answering

Research and development of systems capable of answering questions in natural language dates back to 1959 [28], but the notion of a question answering system was born in 1950, when Turing offered a solution to the question of whether or not machines can think. He proposed a task he called an "Imitation Game," which has eventually become known as the famous "Turing Test," in which a human communicates with a machine via a teletype interface and asks questions of it. Turing would have deemed the machine "intelligent" if the human interrogator could not tell the difference between the responses of the machine and the responses of another human, also communicating via teletype [32].

The situation that Turing envisioned, that of a user seated at a console typing questions to a machine, anticipated the mode of interaction for QA systems, and his Turing Test spawned the early research into systems that could pass it. Very early systems relied on identifying word patterns in the user input, drastically restricting the domain of discourse, or matching simple syntactic structural templates. Examples

of such systems include the "Conversation Machine," by Green *et al.* [8], and ELIZA by Weizenbaum [39]. While each of these systems might be able to fool a human operator into attributing intelligence to it for a short time by generating reasonable replies to user input, the limitations of the subset of English that each was able to handle eventually betrayed to the user the artificial mechanism of dialog generation that the system was using.

In the early 1960s, there was interest in developing natural language front-ends for database query systems. One of the most successful systems of its time was BASEBALL, by Green *et al.* [9]. The system was able to answer narrow-domain questions about statistics compiled over a season of American League play by using shallow parsing techniques on the natural language query to identify the teams and statistics in question. It was also able to handle some more comprehensive queries that involved collating data found in different records of the baseball database, and return the appropriate answer. Another example of a natural language database front-end was the LUNAR system, by Woods, which provided access to two databases containing information about moon rock samples [41]. The system worked by translating natural language questions into one or more queries in the database engine's query language. While these systems were excellent at responding to specific classes of questions within their domain of expertise, the systems are incapable of responding to any natural questions that might suggest themselves during the dialog with the user, but which happen to be outside of the set of questions that the system was specifically engineered to be able to process. The types of questions these systems can handle is largely constrained by the structure of their underlying databases.

Work carried out in the 1970s showed the research community making steps toward an understanding of human dialog. Early dialog systems were built in which human operators could ask a series of questions about a narrow domain. The SHRDLU system, built in 1972 by Winograd, offered users the opportunity to discuss the state of an imaginary blocks world with the system [40]. A later system, GUS, applied a similar dialog model to the domain of making air travel reservations [2]. Both systems demonstrated remarkable capacity to understand natural language, especially where

anaphora resolution or inference was required to carry out the user's instructions. They also displayed flexibility when working with a slightly uncooperative user who, for example, might answer a question from the system with information other than that for which it asked, or with another question, altering the flow of the dialog. Although these systems represented advances in interactive dialog systems, they showed that building a system able to pass the Turing Test was more difficult than it seemed. There has not yet been a machine built that can converse like a human, but steps are still being taken in the right direction. Interactive dialog systems are a fascinating area of work that has resulted in many useful applications available to the public, such as the telephone-based information systems developed by the Spoken Language Systems group at MIT.[1]

Attempts to build systems that were capable of basic reading comprehension arose in the mid-to-late 1970s. The aim was to be able to evaluate a machine's ability to understand language much in the same way as is done for that of a human. MARGIE, a system by Schank *et al.* [27], was capable of reading and interpreting a document, and answering a series of questions about it in a way that is reminiscent of standardized reading tests for children. MARGIE understood texts by parsing them into a semantic representation that is motivated by theories about how human memory is organized. It was an attempt to emulate what a human does when reading and understanding text. This work was taken to the next level by Lehnert and Dyer, whose BORIS system had a repertoire of representations for common plot elements of a written story, including themes, emotions and relationships among characters [19, 7]. Although the task of story understanding shares some features with that of interactive dialog systems, namely anaphora resolution and the use of context to understand questions, it is more of a precursor to modern-day QA than the other tasks described here in that it relies on understanding unstructured text to find answers to user queries.

For more information about some of the systems named above, or for more examples of early natural language systems, see the surveys by Simmons [28, 29]. For details about some of the later systems named, and for a good discussion of the

[1]See `http://www.sls.csail.mit.edu/sls/whatwedo/applications.html`

current state of QA research, see Hirschman and Gaizauskas [10].

2.2 Tasks Related to Question Answering

Many researchers see the modern QA task, described in Section 1.1, as a combination of two more established natural language processing tasks, information retrieval (IR) and information extraction (IE), which are discussed in detail in this section. The relationships of these tasks to that of QA is also covered.

2.2.1 Information Retrieval

The traditional task known as information retrieval can be considered to be similar to what a web search engine, such as Google, does, although the original IR engines predate the existence of the web, searching locally-stored collections of documents instead. An IR engine takes as input a query expressed in the engine's query syntax, which can be as simple as a "bag of words" or as complicated as that of a system such as INQUERY, which allows the user to query on phrases, sets of synonyms and keywords in strict order over windows of text [5].

As output, an IR engine provides a ranked list of documents drawn from the collection it has previously indexed that are relevant to the user's query, for some definition of relevance. IR engines generally rely on statistical measures to retrieve the documents that most closely match the user query. A popular model for building an IR engine is known as the vector space model (VSM), which represents both documents and user queries as vectors of terms in a high-dimensional space.

The most common term-weighting strategy for a VSM is known as the *tf-idf* strategy, which stands for term frequency and inverse document frequency. Term frequency refers to the number of times a term appears within a document. The inverse document frequency of a term is a measure of how rare the term is across the entire corpus. The insight is that if a term occurs frequently in a document, but not frequently in the corpus considered as a whole, then that term does a good job of semantically describing that document. In *tf-idf* weighting, each term is weighted by the product

of its term frequency and its inverse document frequency. It is customary to normalize the term weights against document length to avoid preferentially retrieving very long documents, which contain more terms and have higher term frequencies for those terms than do shorter documents.

Assuming that terms occur independently of each other, *tf-idf* turns out to be a fairly good term weighting strategy. Relating user queries to similar documents in the corpus is simply a matter of computing the cosine of the angle between the query vector and the projections of document vectors onto the hyperplane containing the query vector. Performance of this kind of retrieval algorithm can be improved by filtering out *stopwords*, which are words such as articles and prepositions that are so frequent in the entire corpus that their presence in a document does not contribute to that document's relevance to the query. For more information about term weighting as it pertains to IR, see Salton [26].

In IR, the input query is expressed in the engine's query language, and the output consists of a ranked list of documents that are presumably relevant to the user's query. The user is then responsible for reading the documents to learn whatever it is that he or she wants to know. QA is different from IR in that the user is allowed to ask his or her question directly to the system in natural language, without having to translate it into some query syntax. The system then answers the question in the form of an exact answer extracted from a source document. However different the two tasks are, the fate of QA is tied to that of IR. In QA systems engineered to answer questions over a corpus of documents, some provision for a coarse, first-pass search over the entire set of available documents is necessary, and for that many QA systems turn to an IR engine.

2.2.2 Information Extraction

Formerly known as message understanding, the general goal of information extraction is to locate information within free text that matches prepared templates. Templates can represent events, references to objects or entities, business deals, movements of military resources, or anything else of interest to the system. Each template, like a

frame, contains a number of slots that the IE system would like to fill. In the example of a business deal, a template might have slots for which corporations were involved, whether the deal was a sale, a merger or some other type of transaction, and the amount of stock or money that changed hands. When an IE system locates some text matching one of its templates, it uses as much context as it can to fill out all of the slots in the template.

Named Entity (NE) Recognition is a specialized form of the IE task dedicated to identifying phrases in text that refer to entities like people, organizations and facilities, and extracting their semantics. It is not enough for an NE recognizer to be able to identify that the phrase "Pope John Paul II" refers to a person; the system must be able to fill out a template of information, such that the person is male, his first name is "John Paul", his title is "Pope" and his generation is "II". Examples of NE extraction systems include the popular BBN Identifinder [1], and also Sepia, developed at MIT primarily by Marton [22].

Automated Content Extraction (ACE) is a large-scale evaluation effort for IE systems run by the National Institute of Standards and Technologies (NIST). ACE challenges participating systems to locate references of people, geo-political entities such as cities, states and nations, locations with physical extent, organizations and facilities within newswire text and broadcast news transcripts. Additional goals of the ACE program are to be able to track mentions of entities throughout larger bodies of text, and to recognize relationships among entities.[2] Prior to ACE, standardized IE evaluation opportunities were provided by the various Message Understanding Conferences (MUCs) held between 1987 and 1998, the proceedings of which are available from NIST.

Having extracted information from a large body of text, a database can be compiled about the various types of events or entity references extracted, and such a database can be combined with modern natural language database query front-ends to make a kind of narrow-domain QA system. Limitations of IE systems include the fact that the templates have to be hand-edited by humans, which can take signifi-

[2]See `ftp://jaguar.ncsl.nist.gov/ace/doc/ACE-EvalPlan-2002-v06.pdf`

cant effort that is usually not transferable across domains. As are natural language database front-ends, IE systems are constrained in the kinds of questions they can answer by the structure of their database templates. Just as with IR engines, however, a good IE system can be an enormously useful resource for a high-quality QA system to have. IE can assist with question analysis, helping the system understand what type of entity it is looking for, and also with answer extraction, identifying entity references of the desired type among passages retrieved by upstream passage analysis and document retrieval modules.

2.3 New Interest in Question Answering

Early explorations into question answering systems were concerned with producing natural language query front-ends for databases, dialog systems, reading comprehension programs and the like. Interest in the QA task in its current form did not really take root among the research community until the 1990s.

The START system, developed at MIT by Katz, was the first QA system deployed via the web and made available for public use. START works by matching the input question against schemata that break down the question focus into an object-attribute-value triple that becomes a query into the system's knowledge base. START's knowledge base is not only capable of checking local databases for assertions that match the query, but also able to retrieve information from web databases through the same uniform access model. After START retrieves the answer from its knowledge base, it packages it into a response paragraph of generated English that includes a link to the source of the information, and in some cases, relevant pictures or video clips.

Since 1993, START has answered hundreds of thousands of questions and has provided users with answers ranging from facts about geography, weather and movies to useful information such as distances between major cities, conversions between units of measure, and definitions of words. START's popularity among users from around the world helped to bring QA to the forefront of the research community's

attention in the early-to-mid 1990s. For more information about the system, see Katz [15].

NIST, with support from the Defense Advanced Research Projects Agency (DARPA), started an annual conference in 1992 to promote research in natural language technologies and in information retrieval. This Text REtrieval Conference, or TREC,[3] as it is referred to, organizes competitive tasks and comprehensive evaluation for natural language systems. Since 1999, TREC has offered a QA track in which the task was to answer specific questions over a closed corpus. Each year, TREC provides large-scale evaluation on increasingly difficult QA tasks, comparing systems from a growing community of research groups against a common metric, and raising the standards for the state of the art in QA.

The early QA tracks of TREC-8 and TREC-9 required systems to return 50-byte or 250-byte text windows extracted from the documents in the TREC corpus that contained the answers to factual, short-answer questions termed "factoid" questions. Systems were required to return between one and five answers per question, where answer consisted of a string of the required length and the identification number of the corpus document from which the answer was extracted. The evaluation metric was the average of the reciprocal of the rank at which the first correct answer appeared in the list of answers for each question. The difference between the TREC-8 and TREC-9 tasks was in the question set: while TREC-8 questions were specifically designed to be answerable, TREC-9 questions came from web search engine logs and were considerably more difficult [38].

The TREC 2001 QA track featured the introduction of the list question task in addition to the main (factoid question) task. The list task challenged systems to aggregate an answer from information present in several documents. This inaugural list task always specified the number of items requested in the question, for example, "Name 4 U.S. cites that have a 'Shubert' theater." Evaluation metric for the list task focused on accuracy, which was defined as the number of different, correct responses returned divided by the number requested in the question.

[3]See `http://trec.nist.gov`

TREC 2001 also introduced a context task, in which systems answered a series of questions with the same context. The task was subsequently discontinued because it did not do a good job of evaluating a system's ability to keep track of the context of a running dialog. The TREC 2001 main task was very similar to that of previous TRECs, except that all answers were required to be in 50-byte windows, and that not all questions were guaranteed to be answerable. Systems were given credit for returning 'NIL' as their answer for the 49 questions that had no answer in the corpus [34]. For more information about evaluating QA systems in the early TRECs, see [35].

The new challenge imposed by the TREC 2002 QA track was that systems had to return exact answers on the main and list tasks. Answer strings containing characters beyond the extent of the correct answer were judged "inexact." These strings, while not "incorrect," did not help a system's score. In the main task, systems were required to provide exactly one answer to each question, and to rank their responses in order of confidence. Evaluation was by means of a confidence-weighted score that gave systems credit for being sure of their answers. The TREC 2002 list task was evaluated using the same accuracy measure as that used in the list task from TREC 2001. TREC 2002 also saw a switch in the corpus used for finding answers, moving from the TREC corpus to the Linguistic Data Consortium's[4] AQUAINT corpus [36].

The TREC 2003 QA track contained two tasks, a passages task and a main task. The passages task was similar to earlier QA tracks, in which 250-byte passages were to be returned as answers, one per question. The main task was composed of three subtasks, one for each type of question: factoid, list and definition. Recognizing the need for a QA system to succeed at many question types, scoring on the main task was a weighted sum of performance on each of the three component subtasks. Individually, the factoid subtask was scored in terms of accuracy.

The TREC 2003 list subtask was notable because questions no longer asked for a number; implicitly, the correct answer to the question was to return all instances in the corpus. List questions were scored by taking the F measure, which is a weighted average of precision and recall in which the weight can be tuned to favor one or the

[4]See `http://www.ldc.upenn.edu`

other [33]. The list question scoring algorithm equally weighted precision and recall of list instances retrieved per question, and averaged that across all list questions. The newest type of question was the definition question, which is an open-ended question such as, "Who is Andrew Carnegie?" Systems were charged to retrieve "nuggets" of information, each an element of the total answer to the question. TREC assessors judged a subset of the nuggets as vital nuggets, and systems were evaluated using F measure over the vital nuggets, in which recall was weighted five times as much as precision. Systems took an artificial precision penalty for returning too many non-vital nuggets, or nuggets that were past a length allowance [37].

Chapter 3

Previous Work

This chapter gives an overview of previous work done by the Infolab group and by the author in TREC-style question answering (QA). MIT has participated in the TREC QA track twice to date: in TREC 2002 and in TREC 2003. In this chapter, I describe the two QA systems whose legacy contributed to our current QA system, which forms not only the basis for the experiments outlined in this thesis, but also for the Infolab's TREC 2004 question answering effort.

3.1 Aranea

Aranea is a system developed by Lin *et al.* of the Infolab group, which was MIT's submission to the TREC 2002 QA track in August of 2002 [21]. Aranea falls into a category of systems that employ shallow understanding techniques and databases of external information, a category that was well represented at TREC 2002 [36].

3.1.1 Knowledge Annotation

Aranea seamlessly merges two powerful approaches to QA: *knowledge mining* and *knowledge annotation*. The knowledge annotation component of Aranea leverages existing natural language annotation technology [15] that is a part of START, the Infolab group's publicly available web-based question answering system.[1]

[1]See `http://www.ai.mit.edu/projects/infolab/`

START is backed by a database system called Omnibase, which is capable of providing a uniform interface to a variety of data sources distributed across the web [16, 17]. Many of these data sources comprise the so-called "invisible" web; invisible in the sense that its pages are, for example, dynamically generated out of a database in response to user requests. Pages of this type are not accessible to traditional search engines that have used some kind of web-crawling tool to generate their indices, but they can be made available through Omnibase with some minor effort in knowledge engineering.

Aranea harnessed the power of Omnibase by recognizing questions that are of known forms and translating them into Omnibase calls to generate answer candidates. As a canonical example, questions that matched the pattern "When was x born?" were mapped into Omnibase queries of the form `(biography.com` x `birthdate)`. Decoded, this query means to search under the class of symbols known to be in the database backing `biography.com`, with symbol name matching the x extracted from the question, for the value of the attribute `birthdate`. Omnibase then knows to download the dynamically-generated page corresponding to x from biography.com and to extract the birthdate from it.

The mappings from question pattern to Omnibase query constitute a many to one relation. As an example, the pattern "What is the birthdate of x?" also maps down to the same Omnibase query on `biography.com` with the attribute `birthdate`. In total, there are 28 mappings from question pattern to Omnibase query, and they cover seven different Omnibase classes.

3.1.2 Knowledge Mining

In addition to knowledge annotation, Aranea takes a parallel approach to QA called knowledge mining. Knowledge mining is a web-based technique that relies on the key insight that, as the size of the corpus scales up, the number of passages that answer a question tends to increase. As that number increases, so does the probability that an answer we are looking for is simply phrased in the terms that the original natural language question used [20]. Consider this example involving question 1527

from TREC 2002, "When did the 6-day war begin?" Figure 3-1 shows passages that answer the question extracted from two documents retrieved from the AQUAINT corpus.

- `APW19981021.0384`: "Jordanian soldiers who fought against Israel in the Six Day War that began June 5, 1967."
- `APW19990306.0071`: "Retired Navy Capt. William L. McGonagle, who received the Medal of Honor for heroism as skipper of the USS Liberty when Israel unleashed a deadly attack on the intelligence-gathering ship in 1967, died Wednesday in Palm Springs. He was 73. Thirty-four crewmen were killed and 171 were wounded when the Liberty was attacked by Israeli air force planes and torpedo boats in international waters north of Sinai during the Six Day War between Israel and its Arab neighbors."

Figure 3-1: *Two passages that answer the question, "When did the 6-day war begin?"*

From a document retrieval perspective, both documents are likely to be relevant, that is to say that, not only do they contain the answer, but they also support it. In terms of extracting an exact answer to return as the output of a QA system, the first passage is clearly more convenient than the second. Knowing that the web is more likely than the competition corpus to contain easily-extractable answers that appear as simple restatements of the question, Aranea uses it to generate candidate answers. The redundancy of data available on the web also helps Aranea have confidence in candidate answers it finds, and this confidence is often correlated with frequency. Except for a small number of incidents, the heuristic that the most frequent answer is the correct answer is successful. When it fails, the most frequent answer usually turns out to be something rather nonsensical.

Aranea mines answers from the web by executing Google queries. It generally constructs queries by restating the natural language question input in the form of an assertion, and replacing the *wh*-word with a variable. A few examples of Aranea's query formulations are shown in Figure 3-2. The expectation is that, on a corpus as large as the web is, the answer is likely to appear in the summary snippet returned by Google, close to the phrase supplied to it as input. As a backoff measure, if no suitable

answer candidates are generated using these Google queries based on reformulated questions, Aranea reverts to a bag-of-words approach, doing no worse in the extreme case than the most naive QA system.

---

1. "When did Bob Marley die?" → `Bob Marley died`
2. "What is the Keystone State?" → `is the Keystone State`
3. "Where was the first McDonalds built?" → `the first McDonalds was built`

Figure 3-2: *Several examples of Google queries generated by Aranea's knowledge mining facility.*

3.1.3 Answer Projection

Having generated a set of candidate answers from knowledge annotation and knowledge mining techniques, and having filtered, combined and scored them according to Aranea's confidence heuristics, the final step for the purposes of TREC evaluation, was to project the answer candidates onto the corpus, given that the task was not simply to answer the questions, but to answer them from the designated corpus. This involved searching the corpus for documents that not only contained the answers, but also supported them. For TREC 2002, Aranea accomplished this task by running an early version of the document retrieval and passage extraction facilities that would eventually become a part of Pauchok, described in Section 3.2. Passages were scored statistically based on keyword density, using keywords from both the question and the candidate answer, and the document containing the highest ranking passage was used to support the answer.

3.1.4 Aranea Results

Aranea performed well at TREC 2002, scoring within the top fifteen of 67 runs on the main task, but did not succeed well at the confidence ordering of its results. As a

consequence, the `aranea02a` run answered more questions correctly than each of the two next best runs, but had a lower confidence weighted score [36].

Noticeably, Aranea had trouble providing support for its answers. It turned out to be difficult to project answer candidates generated by the knowledge annotation facility on to the corpus, because the answer candidate was often expressed in a complete and unambiguous form that was not always abundant in the corpus, such as "Verdi, Giuseppe (Fortunino Francesco)." Furthermore, the somewhat crude statistical measures used by Aranea to score passages could be led astray by documents that contained keywords from both the question and the answer, but that did not answer the question.

Being lenient with unsupported answers, Aranea found correct answers for about 37% of the TREC 2002 questions. Knowledge annotation provided 15% of the answers marked correct by TREC evaluators, that is to say, not only correct, but also appropriately supported by a document from the AQUAINT corpus. This statistic illustrates one of the primary contributions of Aranea; it showed that knowledge engineering efforts can provide a tangible performance enhancement if judiciously applied.

Both answer projection and confidence ordering were required to participate in the TREC evaluation in 2002, but seemed to become stumbling blocks for Aranea. These two components of the system were viewed as somewhat artificial, and little thought was invested in clever strategies for implementing them. Here in the Infolab group, we feel that Aranea's performance was best measured with respect to the correct answers it returned, and that the TREC evaluation does not necessarily reflect the accuracy of the system. We are certain that had it not been for credit lost to poorly supported answers and issues with the confidence ordering, Aranea would have been one of the top performing systems at TREC 2002.

3.2 Pauchok

Infolab's TREC 2003 submission was built around a QA infrastructure package called Pauchok, much of the early development of which was done by Tellex *et al.* [30]. Pauchok is a powerful framework for building and evaluating question answering systems. The package provides modules, such as question analyzers, document retrievers, passage retrievers and answer extractors, from which a QA pipeline can be assembled.

There are strong boundaries between components in Pauchok, such that several different implementations of each module type can coexist, and can be substituted for each other should need arise. This has led to interesting work in evaluating the performance of a variety of published passage retrieval algorithms within the Pauchok framework, which encapsulates a general pipelined approach to the QA task [31]. Unlike some other systems that attempt to dynamically decide which, for example, document retriever to use to answer a particular question [25], no attempt is made to adjust the QA pipeline on the fly in the Pauchok system.

For more information about Pauchok, see [30], especially Chapter 3, which details the architecture of the system as originally developed prior to work on TREC in the summer of 2003. For information about the system as of August 2003 when the TREC 2003 runs were being compiled, please see [18], which I co-authored.

3.2.1 Question Analysis

Analyzing the natural language question provided as input to the system is the first step toward finding the answer. In Pauchok, question analysis is comprised of two related processes known as query generation and query expansion, the common goal of which is to construct one or more queries in the query syntax of the document retrieval engine. Pauchok's query generation facility is similar to a blackboard architecture in which independent knowledge sources examine the question and supply information, such as what part of speech a word belongs to, or what type of entity a phrase represents.

Annotators are the knowledge sources that power Pauchok's question analysis

and query generation modules. The primary purpose of these annotators is that they provide information that is useful for generating expanded forms of the original query. Below, I discuss the types of external knowledge that Pauchok considers when expanding queries, and the suite of annotators that provides it.

- **Tokenization**: The word and sentence annotators are the first step in analyzing a question. The word annotator tokenizes the document, inserting tags around each word in it according to some user-specified notion of a word boundary, and the sentence annotator breaks a multi-sentence document along sentence boundaries.

- **Parts-of-Speech**: The well-known part-of-speech (POS) tagger by Brill [3] is encapsulated by an annotator of the same name. The Brill POS tags serve as the foundation for some phrase-guessing heuristics, which will be discussed below.

- **Word Morphology**: Since different tenses of a verb and different pluralizations of a noun used in the question are likely to occur in relevant documents, it is useful to be able to have a set of word morphology expansions available when expanding query terms. Miller's WordNet [23] is a useful source of word inflection information. CELEX provides a comprehensive database of word morphology,[2] including breakdowns of words into their stem and affix components, which can be used to derive related word forms from query terms.

- **IDF**: The inverse document frequency (IDF) of a word is a measure of how rare a word is across the entire corpus. The IDF information is used to iterate through query terms in order from most common, or lowest IDF, term to the most rare, or highest IDF, term.

- **Question Focus**: The START Natural Language Question Answering System by Katz [14] can identify which non-stopwords in a natural language question are function words, and what the target of the question is. For example, in the

[2]See http://www.kun.nl/celex/

question, "Name 20 countries that produce coffee," the word "name" is not a stopword, but rather a function word and, as such, is not likely to co-occur with the answer and should not be included in the query.

- **Phrases**: For the purposes of query expansion, it is useful to know which sequences of words in the input question correspond to phrases that, taken as a whole, have semantics different from the combination of the semantics of the individual component words. Sources of phrases available to Pauchok include Omnibase [16], an incredibly thorough lexicon of phrases that correspond to things like titles of movies, opera and written works, names of locations such as cities, countries and natural geologic formations, units of measure, and historical events and figures; Sepia [22], a named entity recognizer used to identify names of people, locations, organizations and facilities; selected noun–noun collocations extracted from WordNet; and proper noun phrases identified by examining the Brill POS tags on the words of the input question.

3.2.2 Document Retrieval

Pauchok utilizes the Apache Jakarta Project's Lucene[3] IR engine, which is open source and available for public download, to provide document retrieval support. Lucene is a Boolean keyword and phrase search engine based on a standard *tf-idf* model, and it provides a fairly rich query syntax. Lucene supports the Boolean connectives and parenthetical nesting, phrase queries involving wildcards, fuzzy matching, and proximity searching, and term weighting.

Previous work in analyzing document retrieval systems for the purpose of supporting passage retrieval within the context of a QA system has shown that Lucene performs just as well as cutting-edge probabilistic IR engines, even though such modern methods outperform simple Boolean query systems in terms of raw document retrieval [31]. Not only can a Boolean query model provide adequate document retrieval support for QA, but it allows the system to apply performance-enhancing

[3]See `http://jakarta.apache.org/lucene`

methods such as query expansion. Confirming our intuition, other TREC competitors are using Boolean query-based IR engines in their systems, for example, those described by Hovy *et al* [11] and Moldovan *et al* [24].

3.2.3 Query Generation and Expansion

The work outlined in this thesis began as my exploration into using query expansion techniques to boost recall of relevant documents retrieved by Lucene in the Infolab's TREC 2003 competition system. We had speculated that document retrieval was not one of the strengths of our older-generation TREC-style QA systems and prototypes, and we realized that our earlier systems frequently returned no relevant documents at all for an input question. We considered this performance to be unacceptable, because there is no sense in doing any further processing on the question if no relevant documents are retrieved; whatever answers will be extracted from any remaining documents returned are sure to be wrong.

I developed two query generation and expansion algorithms that were put into use for TREC 2003. We referred to them as Method 1, a simple approach that represents an incremental improvement over bag-of-words, and Method 2, an ambitious process that involves using the annotators described in Section 3.2.1 to identify phrases with special query expansion properties [18]. Both query expansion algorithms return a list of Boolean keyword and phrase Lucene queries that are executed in order. The list of documents returned by is composed by concatenating the list of documents retrieved by the individual queries, in order, with duplicates removed, up to a limit of one thousand documents.

Method 1

Method 1 features stopwords filtering, and inflectional and derivational expansion for query terms. The algorithm does not contain any provision for recognizing phrases embedded in the natural language question, and so it rather naively considers them as their component words, expanding these words as it would any other query term

in the question. The first query is always a conjunction of all of the query terms. The second query is the same as the first except that each query term is expanded into a disjunction of the original term and all morphological variants from WordNet and CELEX. The third through last queries are generated from the second query according to a term-dropping strategy in which terms are dropped in order of increasing IDF, meaning that the term that is the most common across the entire corpus is dropped first. After all of the terms are exhausted, the term with the highest IDF, or the most rare term, is removed, and the dropping continues as before. Assuming that query terms A, B and C are arranged in order of increasing IDF, Figure 3-3 shows the queries that would be generated by Method 1, in order. Note that not all combinations of terms are tried.

$$
\begin{array}{ccccc}
A & \wedge & B & \wedge & C \\
e(A) & \wedge & e(B) & \wedge & e(C) \\
 & & e(B) & \wedge & e(C) \\
 & & & & e(C) \\
e(A) & \wedge & e(B) & & \\
 & & e(B) & & \\
e(A) & & & &
\end{array}
$$

Figure 3-3: *Queries generated by Method 1 for query terms A, B and C arranged in increasing-IDF order.*

The notation $e(x)$ in Figure 3-3 refers to the morphological expansion of term x, which equals the disjunction $x \vee inflect(x)^{\alpha} \vee derive(x)^{\beta}$. The parameters α and β represent discount factors for WordNet inflectional and CELEX derivational word forms, respectively. They were initially set to 0.75 and 0.5, respectively, for participation in TREC 2003, but I have since devised ways of tuning the parameters automatically. These are incorporated in the current Pauchok II system described in Chapter 4.

Method 2

With Method 2, I intended to improve on Method 1 by adding linguistic information in the form of phrase analysis to the query generation and expansion algorithm.

Better knowledge of which words in a natural language question constitute phrases helps a query generator keep those words together instead of breaking them into their component query terms. Method 2 relies on the annotators described in Section 3.2.1 to identify phrase candidates within the natural language question, and it uses a first-and-longest heuristic to choose among the candidates. All non-stopwords from the question that are not part of phrases are considered to be individual query terms and are expanded exactly as in Method 1.

Method 2 also features an adjustment to the term dropping strategy, because it was feared that the extended dropping strategy of Method 1 was merely increasing the number of irrelevant documents returned. Figure 3-4 shows the queries that would be generated by Method 2 for the same example three-term query in which terms A, B and C are in increasing IDF order. Again, not all combinations of terms are tried.

$$\begin{array}{ccccc} e(A) & \wedge & e(B) & \wedge & e(C) \\ & & e(B) & \wedge & e(C) \\ & & & & e(C) \end{array}$$

Figure 3-4: *Queries generated by Method 2 for query terms A, B and C which are in order of increasing IDF.*

One of the differences inherent in Method 2 is that the query terms can in fact be entire phrases. Phrase expansion is handled differently than is normal query term expansion, which, as in Method 1, refers to the disjunction of morphological forms with their according discount factors.

Expansion for phrases depends on their type, but for all types the original form is represented as a phrase query to Lucene, that is to say, a quoted string that must be matched exactly in a document. The first level of expansion for all phrases is a proximity search, again taking advantage of some convenient features of Lucene query syntax. Proximity search allows for the individual component words in a phrase to be within some number of words of each other. Unfortunately, the proximity feature does not allow for the word ordering within the window to be enforced. Proximity searches are given the discount factor of α, which was set to 0.75 for the TREC 2003 runs.

Individual types of phrases that have special expansion are as follows. Special expansion forms are given the discount factor of β, which equaled 0.5 during TREC 2003.

- **Sepia Names**: Persons' names as identified by Sepia are expanded down to the last name, since well recognized individuals are sometimes referred to by their last names only. As an example, the entity "President John F. Kennedy" would be expanded out to simply "Kennedy," but not to "John," since that is more likely to refer to someone else. I intend to expand this expansion to cover additional forms of the name that are likely to refer to the same person.

- **Noun–Noun Collocations**: The noun–noun collocations recognized by the system have been extracted from WordNet, which can supply correct pluralization information. As as example, "box car" is expanded to "box cars," but not "boxes car." A more difficult case is "attorney general," which WordNet correctly expands to "attorneys general."

Other types of phases that Method 2 recognized but that did not have a special expansion form are as follows:

- **Prequoted Phrases**: Perhaps the most obvious types of phrases available are those that appear delimited by quotation marks in the natural language question itself.

- **Omnibase Symbols**: Omnibase contains a large number of information about names of movies, books, plays and operas as well as historical figures and events, all of which are likely to be asked about by name. Because of the sheer number of symbols available in Omnibase, the algorithm only recognizes symbols with at least three words in them. Prior experimentation had shown that there are a wealth of movie titles that match phrases of one or two ordinary words in the question when in fact the question was not referring to these movies at all. Recognizing the words as part of a phrase prevents the algorithm from generating morphological and derivational variants from them, and so, to prevent the

spurious tagging of ordinary phrases as Omnibase movie titles, we agreed on setting a three-word minimum on matching Omnibase Symbols.

- **Sepia Locations and Organizations**: Sepia can recognize names of cities, states, countries and combinations of the three; names of geological formations such as lakes, rivers and mountains; and names of organizations such as corporations, foundations, and committees. It contains heuristics, whereas Omnibase, in all of its comprehensiveness, is still a lexicon. Sepia outperforms Omnibase in some cases.

- **Proper Noun Phrases**: The algorithm gets its parts of speech for the natural language question from Brill's tagger. Proper noun phrases are identified as an optional determiner followed by zero or more adjectives and one or more proper nouns.

3.2.4 Passage Retrieval and Answer Extraction

Previous work has shown IBM's passage retrieval [13, 12] algorithm to be among the best performing, especially when backed by a Boolean keyword and phrase document retriever employing recall-boosting techniques such as query expansion strategies [31]. Infolab group members extended the algorithm to make use of linguistic information, such as WordNet hyponyms, alternate word forms, and phrase annotations, compiled by the annotators in the document retrieval phase.

Passages extracted by the IBM algorithm were passed to answer extraction for generation of candidate answers. In addition to being used for the natural language questions, the suite of annotators was also used to analyze passages to find phrases of the same type as that which was asked for in the question. If no phrases of the right type were found in a passage, the system was able to fall back to statistical and pattern-based answer extraction techniques.

3.2.5 Pauchok for List and Definition Questions

Special versions of Pauchok were developed to handle the list and definition question subtasks posed by TREC 2003 [37]. For list questions, such as question 1915, "List the names of chewing gums," the strategy was identical to the Pauchok pipeline used for factoid questions except for the answer extraction module, which was designed specifically to incorporate external knowledge in the form of lists of members of certain classes of entities extracted from Omnibase [15].

The START question annotator is capable of parsing most common forms of questions and extracting the focus word or phrase. In question 2097, "Which countries were visited by first lady Hillary Clinton?", for example, START was able to tell the system that it is looking for information about countries of the world, as opposed to about Hillary Clinton. When it came time to extract answers from passages, the list question answer extraction module was able to consult its known list of the countries of the world and restrict its answers to members of that list. For questions for which there was no list of reasonable answer candidates available, the system fell back to answer extraction heuristics carried forward from Aranea [21] that were known to work with some degree of success. Bringing external knowledge to bear on the QA task was beneficial to answering list questions. Other TREC teams have recognized this benefit as well, e.g. ISI [11].

For definition questions, the task was made somewhat more difficult by the need to achieve high recall of nuggets of information. For this type of questions, a special purpose architecture made use of a pre-indexed database of nuggets created by running common surface patterns, such as appositives and parenthetical definitions, over the corpus. In parallel, the system searched for nuggets in the web-based Merriam-Webster dictionary, projecting anything useful back onto the AQUAINT corpus using techniques pioneered in our TREC 2002 system [21]. If the first two approaches yielded no nuggets, the definition question handler used a streamlined version of the factoid Pauchok pipeline to retrieve sentences from the corpus that were likely to contain nuggets.

3.2.6 Pauchok Results

We are pleased to say that Pauchok performed rather well, scoring sixth best of fifty-four runs on the main task, third on the list question subtask, eighth on the definition question subtask, and sixth overall. The `MITCSAIL03b` run performed the best on factoid questions, scoring 0.295 accuracy. The run made use of Method 1 for query generation and the original IBM algorithm for passage retrieval. For List questions, `MITCSAIL03c`, using Method 2 and the modified IBM passage retriever, was the best run with a score of 0.134. For each run, the results for definition questions were the same, but because of variations in the ways assessors scored answers, `MITCSAIL03a`, with a score of 0.309, was the best of our runs on definition questions.

Chapter 4

Pauchok II System Design

The current system that forms the testbed for the experiments carried out in this thesis, and that will become the basis for Infolab's submission to the Text REtrieval Conference (TREC) 2004 question answering (QA) track, is known as Pauchok II, and is covered in detail in this chapter.

4.1 Overview of Pauchok II

Pauchok II is a redesign of the original Pauchok system, taking into account lessons we learned by interacting with it and using it to build a competition QA system for TREC 2003. The infrastructure for Pauchok II was designed to be simpler and much more free-form when compared with the strict modularity and linear flow of information imposed by the original Pauchok system. The same types of components exist, such as question analyzers, query generators, and document retrievers, since we believe them to be fundamental to the QA task, but they can now be freely interconnected.

The question analysis facility based on a blackboard architecture has been carried forward from the original system. As before, annotators are responsible for supplying information about the question that can be useful for query expansion, such as sentence and word boundaries, word part of speech and inverse document frequency (IDF), alternate morphological forms of query terms, and the extents of any phrases

that appear in the input.

Since Pauchok II is meant to be somewhat of a playground for building and experimenting with new query expansion algorithms, we have provided an environment that enables this, based on an embeddable Scheme interpreter. Query generators can be constructed out of a variety of components we have pre-defined, and new components are easy to create. We have provided query expansion components such as control structures, which are responsible for connecting together other query expansion components; term extractors, which locate the extents of query terms within the input question according to user-defined criteria; query term expansion functions, which when given a query term suggest alternate forms of that term that could prove useful for query expansion purposes; and term-dropping strategies, which construct a series of queries that try different combinations of query terms.

4.2 Query Expansion Components

This section describes in more detail the primitive components that combine to form a query expansion algorithm in Pauchok II, and also discusses the components that were built for the experiments described in this thesis. The primitive components have been designed to be as general and as expressive as possible, and to make building query expansion algorithms easy. We aimed to minimize the extent to which the the user is constrained to work within the limits of architectural decisions that we have made when building the system.

4.2.1 Control Structures

We have provided only one control structure, but defining others is extremely straightforward in Pauchok II. The control structure is highly generalized, and provides a unified framework for a large number of query expansion algorithms that can be built from it. The provided control structure combines a term extractor, zero or more term expansion functions, and a term-dropping strategy in a simple algorithm.

When the question is passed to a query expansion algorithm built from this control

structure, it is passed first to the term extractor, which is responsible for locating query terms within the question that meet its criteria and returning a list of them. This list of terms is then passed to any term expansion functions to which the control structure has access. Each function generates alternate forms for every term passed to it. The control structure is responsible for collecting these forms, weeding out duplicates, and building disjunctive query clauses. At this point in the process, the control structure holds a disjunction for each term that was extracted from the original question. Instead of returning a simple conjunction of these clauses, the control structure uses its term-dropping strategy to create a list of queries out of various combinations of the terms' disjunctive clauses. The result is an ordered list of nested Boolean queries that can be used to retrieve documents from the corpus.

4.2.2 Term Extractors

Term extractors are that part of a query expansion algorithm responsible for analyzing the question and producing a list of the query terms that it contains. Part of the power of the Pauchok II system is that the term extractor has sole jurisdiction over whether any word or sequence of words from the question becomes a query term or not. A Pauchok II user could encode any definition of what constitutes a query term into a term extractor, and it would immediately coexist with the other components in the system.

We have pre-defined two term extractors in the Pauchok II system. The simpler of the two term extractors is called the Word Term Extractor. It promotes all non-stopwords in the question to query terms. The other term extractor is known as the Phrase Term Extractor because it identifies phrases in the input question and returns them as query terms.

The Phrase Term Extractor interacts with the blackboard question analysis system introduced in the original Pauchok system. There are several annotators known as phrase analysis annotators that identify and mark the extents of phrases of known classes in the input question. For more discussion about these annotators, see Section 3.2.1. Selection among phrase candidates identified by the blackboard system is

carried out using a first-and-longest heuristic. The result is a set of non-overlapping phrases, and some lone words that are not part of the phrases. The phrases and lone words together constitute the list of query terms returned by the Phrase Term Extractor; words that are enclosed by the phrases are not considered to be query terms.

4.2.3 Term Expansion Functions

A term expansion function has the job of generating alternate forms of a query term supplied to it. Such a function can be specifically created to handle only certain types of terms, ignoring others it encounters, or can be generally applicable to all terms. Term expansion functions can easily obtain access to the blackboard question analysis system, which may contain alternate forms of words appearing in the input question or other useful information, or they can be completely self-contained, executing algorithms of their own directly on the query terms.

We have included two term expansion functions in the Pauchok II system. The inflectional expansion function is responsible for returning inflections of the original query term, such as tense variants for verbs, pluralizations for nouns, and comparative and superlative forms for adjectives. To accomplish this, the function checks the blackboard system for annotations that contain alternate WordNet forms for the query terms in the questions.

The second term expansion function operates in much the same way as the first, but provides different content. It is called the derivational expansion function, and it finds forms that can be derived from the original term and other related forms by searching the blackboard for information originating from CELEX, a comprehensive database of word morphology. Both of these term expansion functions operate solely on query terms consisting of a single word, ignoring any phrases, or other classes of query terms yet to be defined, that they may encounter as input. Although there are not currently any term expansion functions that operate on phrases, the system is designed so that adding such a function would not be difficult.

4.2.4 Term-Dropping Strategies

A term-dropping strategy is simply than a function that returns a list of different combinations of the items provided to it as input. These items can be query terms, disjunctions of terms, or entire nested Boolean queries. The algorithm that a term-dropping strategy implements is unrestricted. It could try subsets of the terms provided as input to it, or even insert query terms of its own invention into the mix.

We currently have three term-dropping strategies pre-defined in the Pauchok II system, and they are designated by color as the Red, Green and Blue Dropping Strategies. The Red Dropping Strategy was taken directly from the original Pauchok system's query generation Method 2. In this dropping strategy, terms are dropped in order of increasing Inverse Document Frequency (IDF), which is a measure of the rarity of a term in the entire corpus of interest. The Red Dropping Strategy is intended to be an incremental improvement over the bag-of-words approach that can serve as a baseline for experiments with improved dropping strategies. Figure 4-1 shows the queries generated by the Red Dropping Strategy for three query terms, A through C, which are in order from lowest to highest IDF. Throughout this section, $e(x)$ refers to the output of a set of term expansion functions applied to the term x, which may be a single word or an entire phrase. In Section 4.2.3, there is more information about how term expansion is implemented in Pauchok II.

$$\begin{array}{ccccc} e(A) & \wedge & e(B) & \wedge & e(C) \\ & & e(B) & \wedge & e(C) \\ & & & & e(C) \end{array}$$

Figure 4-1: *Queries generated by the Red Dropping Strategy for query terms A, B and C, from lowest to highest IDF.*

Pauchok II also features the Green Dropping Strategy, which bears some resemblance to that used in Method 1 in the original Pauchok system for TREC 2003. Like the that of Method 1, this strategy drops terms in order of increasing IDF, and when all terms are exhausted, removes the highest IDF term and restarts the dropping process. The Green Dropping Strategy does not use all combinations of the query

terms, but does execute queries that cover the entire space of documents containing at least one of the query terms. Unlike the original system's Method 1, the Green Dropping Strategy does not begin with a conjunction of all terms unexpanded; such a query is generally too narrow for our purposes. See Figure 4-2 for an illustration of this strategy. Compare with Figure 3-3, on page 40, which shows the strategy used in the original system.

$$
\begin{array}{lllllll}
e(A) & \wedge & e(B) & \wedge & e(C) \\
 & & e(B) & \wedge & e(C) \\
 & & & & e(C) \\
e(A) & \wedge & e(B) & & \\
 & & e(B) & & \\
e(A) & & & &
\end{array}
$$

Figure 4-2: *Queries generated by the Green Dropping Strategy for query terms A, B and C arranged in increasing order of IDF.*

Perhaps the most interesting term-dropping strategy available is the one we designated with the color blue, shown in Figure 4-3. This strategy arose from a desire to try all combinations of n terms before trying any combinations of $n-1$ terms. The way the strategy works is somewhat complicated. To generate the combinations, one term is plucked out of the group of terms in order of increasing IDF, and this is done recursively, adding every distinct combination to a list, which is then sorted in decreasing order of the number of terms in the combination, while preserving initial order. Intuitively, what is happening is that all combinations of size 3 are tried, followed by all combinations of size 2, in order of increasing IDF for the missing term. Finally, all combinations consisting of only a single term are tried, in order of increasing IDF for all missing terms.

4.3 Visualization Tools

Pauchok II comes with a full suite of visualization tools I developed to make working with the system easier. This is part of what makes Pauchok II such an effective playground for experimenting with query expansion algorithms. The statistics com-

$$
\begin{array}{lllll}
e(A) & \wedge & e(B) & \wedge & e(C) \\
 & & e(B) & \wedge & e(C) \\
e(A) & & & \wedge & e(C) \\
e(A) & \wedge & e(B) & & \\
 & & & & e(C) \\
 & & e(B) & & \\
e(A) & & & &
\end{array}
$$

Figure 4-3: *Queries generated by the Blue Dropping Strategy for query terms A, B and C, which go from lowest to highest IDF.*

puted by the evaluation framework are useful metrics, but nothing can substitute for the intuition gained by looking through the documents retrieved by the experimental queries. The Pauchok II visualization package is designed to make this task easy.

The visualization toolkit has two primary interactive windows, a Query Viewer and a Document Viewer. The Query Viewer is a frame that contains two lists. On the left, the queries generated by the query generation and expansion algorithm are shown in the order executed. On the right hand side of the Query Viewer, the documents are listed in the order retrieved by the query that is currently selected in the left-hand list. Duplicates are removed so that each document appears only once in the Query Viewer, in the list associated with the query that first returns it. The Query Viewer has a color-coding capability such that documents known to be relevant appear in red, and documents known to be unsupported appear in orange. Documents known to be irrelevant appear in blue. Documents for which the status is not known appear in the normal black font. To assist the user, documents that have been viewed in the current session are shown in italics.

Selecting a document in the Query Viewer and then pressing the "Examine" button raises another window known as the Document Viewer. This simple frame shows the document text with the query keywords highlighted. The user can scroll through the document, letting the highlighting guide his or her eyes to the mentions of the keywords, and can make a judgment as to the document's relevance by clicking one of the buttons at the top of the frame: relevant, unsupported or irrelevant. Making a judgment in this way causes a log file entry to be written. Any text snippet the user

may have selected with the mouse is also written to the log file and stored with the document identification number and the judgment. After the judgment is recorded, the Document Viewer will load the next document in the list for judgment. Returning to the Query Viewer is accomplished by pressing a button, or by making a judgment for the last document in the list.

The visualization tools package was most useful for assisting with the manual annotation of documents for the building of the test collection that is a part of this thesis, as described in Section 5.3.1. While building the test collection, a human assessor took each question and its answer and formulated a Lucene query from them. The query was expected to retrieve as many as was possible of the total number of documents in the corpus that were relevant to the question, which means that they not only contained the answer, but also supported it. The Query Viewer was used as a part of this process, showing on the left-hand side of the frame the single query supplied by the human annotator. The documents retrieved by that query were displayed in order on the right-hand side of the frame. With the highlighting, it did not take very long to scan the documents and judge whether they were in fact relevant. The visualization tools made it easy for the annotator to iterate through the retrieved documents and efficiently make relevance judgments for them, which were automatically recorded.

Even after we had finished building the test collection, the visualization tools were still very useful to me while I was experimenting with different query expansion techniques. After retrieving documents with queries generated by an experimental query generator, I used the Query Viewer to display the documents returned. Conveniently, the frame showed me the sequence of queries in the left-hand list. The right-hand list displayed the documents returned by the corresponding query selected in the list, and the documents were color-coded according to the relevance judgments that were available for them. With a glance, it was easy to see which relevant or unsupported documents were being returned by which query. It was also immediately clear when the first query generated was too narrow and did not return any relevant documents. I used the Document Viewer to display the retrieved documents with highlighting

of the query terms. For the documents that had not been judged, I used the controls on the Document Viewer to record whether they were relevant, unsupported or irrelevant.

Chapter 5

Building a Test Collection

This chapter describes the development of a test collection to support comparative evaluation of various query expansion techniques of my own design. I discuss evaluation of document retrieval systems in general, and the role I the test collection plays in this thesis. I describe the composition and format of the test collection, the procedure I and my collaborator, Leah Oats, used to build it and guidelines for manual annotation that we developed for it.

5.1 Deciding on an Evaluation Metric

Proper evaluation of my work on query expansion techniques and document retrieval for question answering requires a task that approximates as closely as is possible the reality of how my system will be used in practice. The task must incorporate a performance metric and a clear definition of what constitutes 'good' performance with respect to that metric.

For studies of document retrieval, an evaluation metric suggests itself immediately. I am interested in retrieving as many documents as possible that are relevant to the query and as few documents as possible that are not relevant to the query. In the standard context of performance evaluation of information retrieval systems, I can consider the precision and recall of relevant documents. Precision is defined as the ratio of the number of relevant documents retrieved to the total number of retrieved

documents. Recall is defined as the ratio of the number of relevant documents retrieved to the number of relevant documents, total, that exist for a given question. For the questions in the test collection, the expectation is that all the relevant documents are known.

Researchers evaluating retrieval systems tend to try to maximize precision and recall simultaneously, using F-measure [33], to avoid the trivial case, for example, in which a system attains 100% recall by retrieving the entire corpus, or 100% precision by retrieving no documents at all, in response to a query. Here, I take a different approach that I feel is mandated given the nature of the problem I am trying to solve.

For the purposes of document retrieval for question answering (QA), it is not necessary to focus heavily on improving precision; that is to say, the document retriever does not have go to extraordinary lengths to filter out documents that are not relevant to the query. The slack will be picked up by downstream passage retrieval and answer extraction modules, which compute a statistical measure based on keyword and phrase density and ordering, and search the document for entity references matching an expected type. Documents that are not relevant are extremely likely to receive very poor passage retrieval scores, or to not contain matching entity references. In this way, the post-document retrieval stages of a QA system serve to filter out documents that are not relevant, releasing the document retrieval engine from having to be overly strict about precision.

While precision is not the focus of this evaluation, recall is of utmost interest here. Without relevant documents, downstream modules in a QA system have no hope of generating reasonable answer candidates. In this thesis, I am concentrating on improving the recall of relevant documents of my document retrieval module, so I choose to evaluate solely on the basis of recall. Given that we have assembled a test collection in which all of the relevant documents are known and have been examined by hand, this evaluation will be meaningful. I intend to continually improve my system throughout the summer as the Infolab group prepares for TREC 2004, periodically checking against my evaluation metric to see how the system's performance has increased.

5.2 The Need for a Test Collection

The evaluation metric stated above depends on the fact that there exists a set of questions for each of which all of the relevant documents are known. For question sets used in past runnings of the TREC QA track, relevance judgments are available to the research community.[1] These judgments do not make a suitable test collection for researchers developing QA systems because they are not complete, and are sometimes wrong.

As described in the TREC QA track overview literature, document relevance judgments consider only relevant documents returned by participating systems as support for their answers. We also know that these document candidates are judged only binarily, when in fact there are three categories into which a document can fall: relevant, irrelevant, or unsupported [35].

In the relevance judgments provided for TREC 2002, no single question has more than four relevant documents identified for it, and the average number of relevant documents identified per question is 1.95. It would be wrong for us as researchers to evaluate our systems under the assumption that documents not named by the assessors are irrelevant documents. The TREC assessors do not tell us which documents they reviewed and judged to be irrelevant. The assessors also do not attempt to distinguish unsupported documents.

There are also clear errors in the TREC-provided relevance judgments. Take, for example, question 1440, "Who was the lead singer for the Commodores?" The answer is "Lionel Richie." Document `APW19980827.1319`, marked relevant by the TREC assessors, states that Richie is a singer who has had several solo hits, and that he was an original member of The Commodores, but not that he was the lead singer of the group. According to the definition of relevance we used while creating this test collection, which will be explained below, this document does not answer the question. We marked it unsupported.

In another example, one that does not depend on differing definitions of relevance,

[1] See `http://trec.nist.gov/data.html`

consider question 1443, "When did Bob Marley die?". The answer to this question is May 11, 1981, when the artist lost his battle against cancer. The provided relevant document, `NYT19991217.0112`, is an article about several different newly released albums, with a paragraph about each one. The last paragraph in the article, which discusses Marley's "Songs of Freedom", does not mention his death, but the penultimate paragraph tells readers that Harry Chapin died in a car accident in 1981. It is clear that this is just a mistake that any person could make when asked to read and annotate thousands of documents, but this document is the only relevant document identified for this question; it is in fact listed twice in the TREC 2002 relevance judgments file.

It seems clear to me that a test collection such as the one we are building would be of enormous use in furthering QA research, since what relevance judgments the community currently has are terribly incomplete. I intend for this test collection to provide much broader coverage of the space of relevant documents for each question, such that if a retrieved document is not in the provided set, it is much more likely to be irrelevant. I intend to provide markings for unsupported and irrelevant documents.

In the course of the work outlined in this thesis, I have constructed such a test collection, with the help of another student, Leah Oats. In the following sections, I describe our test collection, and the process that we used to construct it. The actual test collection in its entirety does not appear in this thesis because it is not yet ready for distribution.

5.3 Choosing Questions

When building this test collection, we selected 120 questions from the TREC 2002 question set. Almost all questions were acceptable for inclusion in the test collection, but we did reject several questions. Questions such as 1496, "What country is Berlin in?", were rejected on the basis that there were 5088 documents in the corpus that potentially discussed the answer, more than were practical to read and judge.

Other questions were rejected because it was not clear what the question was

asking. In the case of question 1422, "What two European countries are connected by the St. Gotthard Tunnel?", multiple reasonable answers suggest themselves. Several documents in the corpus explain that the tunnel is the essential route connecting Italy and Germany, two EU countries. It has also been claimed that the tunnel links Italy and Switzerland. The reality of the situation is that both endpoints of the tunnel, and its entire extent, lie within Switzerland, which is not an EU member state. The southern entry of the tunnel opens in Airolo, which is 100 km from Chiasso, the border town between Italy and Switzerland. Since there was no clear answer to question 1422, we excluded it from our test collection. Difficult questions such as these constitute only a handful of those in the TREC 2002 question set, so there were plenty of suitable questions to choose from when assembling the test collection.

5.3.1 Finding Documents in the Corpus

The test collection pairs TREC 2002 questions with sets of documents known to be relevant, unsupported or not relevant. To generate sets of documents for us to judge, we formulated Boolean queries with the aim of making them as narrow as possible while retrieving all documents that were likely to be relevant to a particular question. These queries generally took the form of conjunctions of all keywords from the question that we felt would most certainly co-occur with the answer in the corpus and the answer itself, which we found by searching the web. As these queries were formulated manually, we did not unilaterally apply any standardized expansion techniques such as word morphology or synonymy. Any terms that might need to be expanded were dropped. If the resulting query returned more documents than was practical to read, we then added terms to restrict the scope of the query, being sensitive to word morphology and synonymy issues.

Occasionally, it was necessary to add terms that came neither from the question nor from the answer to narrow the query sufficiently such that the number of documents returned was small enough to read. Consider question 1501, "How much of U.S. power is from nuclear energy?" Given that the answer is approximately 20%, the query that immediately suggests itself is `nuclear AND power AND 20 AND ( US`

`OR America OR American OR ( United AND States ) )`, which returns 1294 documents, too many to read. Adding the term `electricity` narrows the query sufficiently, returning 170 documents and making it possible to annotate this question.

Sometimes we were not able to locate a single answer to include in our query. Question 1423, for example, asks, "What is a peninsula in the Philippines?" Answers we located in the corpus included the Bataan, Zamboanga and Bicol peninsulas, but the most narrow query were able to use was `peninsula AND Philippines`, without knowing *a priori* which peninsulas in the Philippines would be mentioned in the corpus. We read 142 documents, most of which discussed the Korean Peninsula and the Philippines separately, to ensure that we found as many of the relevant documents as was possible.

There were also questions for which the answer we found on the web was not suitable for querying. Question 1513, "What is the current population in Bombay, India?" is an example of this. As of the time of this writing, the population of Mumbai, which is what Bombay's name was changed to in 1996, was 18.1 million people. In the corpus, we found several different opinions of the population of Bombay: 10 million, 12 to 14 million, and 15 million. The quoted figure depends on the year the article was written, and the source of the data. Answers in the form of numbers, such as this one, were particularly troublesome and we could not use them in our queries. The query we used for question 1513 was `( Mumbai OR Bombay ) AND ( population OR ( million AND people ) )`, which returned 296 documents to read.

To actually make the judgments, we used a convenient annotation tool that I developed. Features of the software were expressly designed to speed the annotation process. The annotation frame displayed the document's text, with query terms highlighted. This allowed the assessor to quickly scan the document for the concentration of highlighted terms and read the passage in which the terms appeared. Controls on the frame allowed the annotator to mark the document as relevant, unsupported or irrelevant. This judgment was written to a log file, along with the document number and any text snippet selected by the user with the mouse. This process continued

as the next document was opened automatically in the frame. For more information about the Pauchok II visualization package, see Section 4.3.

5.3.2 Guidelines for Judging Documents

For this test collection to be of general use, the judgments need to be relatively stable. By this, I mean that several different human annotators, each equally trained in the task of judging documents and each annotating the same question set, should arrive at the same judgments within some threshold percentage. One way of making sure that different assessors are applying the same set of criteria when making their judgments is to provide comprehensive guidelines, with examples, to help them better understand when a document is relevant, unsupported or irrelevant.

Leah and I carefully developed guidelines for judging documents that were reasonable and satisfactory to us both. We imagine our guidelines to be spelling out what a reasonable human reader who did not already know the answer to the question would think after reading the document. If there were a way for a human to read the answer out of the document, we intended to judge it relevant. For a document to be marked relevant, we settled on the requirements that the document contain the answer to the question, and clearly support it, regardless of inference or reference resolution that may be required.

We each arrived at the task of building this test collection with preconceived notions of what constituted relevance for a document. Initially, there was a tendency on both of our parts to mark relevant only those documents that explicitly gave the answer, or that we felt a machine would have a good chance of extracting the answer from. Naturally, we were using our conceptions of what a state of the art QA system could do in terms of today's technology. We realized that this was a very limiting way to make judgments, so one of our first guidelines was that the judgments should be independent of the present or future capabilities of machines, real or hypothesized.

Even more disparate than our preconceptions on document relevance were our initial opinions on what it meant for a document to be unsupported as opposed to irrelevant, which was surprising. We would have liked it to have been completely

obvious which documents were irrelevant, but concerns quickly arose. We defined a judgment of unsupported to mean that a document contains at least a part of the answer, and discusses it in the right context, but that the document does not clearly and completely answer the question.

We decided that, although an unsupported document will not necessarily do so, a relevant document must address all of the constraints in the question. We define a question constraint as a distinct factual unit that a relevant document must contain to support its answer to the question. This concept is best illustrated by example. For question 1411, "What Spanish explorer discovered the Mississippi River?", the answer is Hernando de Soto, and there are 27 documents that are returned by the query `mississippi AND soto`. Document `APW20000520.0126` tells us that de Soto "died while searching for gold along the Mississippi River," and also that he was a Spanish explorer. This document satisfies only three of the four constraints asked for in the question, namely that de Soto was a Spaniard, that he was an explorer, that he reached the Mississippi and that he discovered it. This document was marked unsupported because it almost completely supported the answer, but failed to mention that de Soto is actually credited with having discovered the river. As as extreme example, question 1834, "Which disciple received 30 pieces of silver for betraying Jesus?", which is not in the test collection, contains so many constraints that no documents found in the corpus are actually relevant for it.

When two terms are used interchangeably in the text, we agreed that this would constitute their equivalence, even though it was never explicitly stated in the text. Consider question 1456, "What is the Keystone State?" Almost never do the documents state clearly that the Keystone State is Pennsylvania, because such a fact is in most cases known to the readers. Writers instead use the phrase "Keystone State" interchangeably with the state's name, and we marked these documents relevant when they did so, because a reasonable human reader would be able to infer that Pennsylvania is the Keystone State by reading the document.

We had to resist the tendency to mark documents that required inference on world knowledge as unsupported. If a document clearly answered the question but required

external knowledge to do it, we marked that document relevant so as to set a high standard for QA systems developed in the future to aspire to. The best example of this phenomenon involves pairing universities with their mascots, as is necessary for document `NYT19990701.0055`, which answers question 1484, "What college did Allen Iverson attend?". The Philadelphia 76ers guard attended Georgetown University, but the document tells us simply that he was a Hoya. Since one day we expect QA systems to be able to synthesize an answer by aggregating information from multiple documents, we marked documents like this relevant.

Question 1475, "Who was the first person to reach the south pole?," *(sic.)* is interesting because it raises the issue of contemporary language usage. Document `NYT19981128.0126` refers to Roald Amundsen as "the first man to reach the South Pole", and it is a safe assumption that the writer intended to say that he was the first person to reach it; that is to say, there exists no woman who arrived at the South Pole before Amundsen did. This document was marked relevant, because a reasonable reader would understand what the writer means, based on knowledge of the culture at the time. Many readers, in fact, would not stop to think about this case. Document `XIE19981221.0153` tells us that Amundsen was "the first man who arrived at the South Pole by skiing solely." Most readers would assume that he was the first person to reach the South Pole and he happened to get there on skis after having read this passage. The way it is phrased without punctuation, however, leads one to believe that the article is crediting Amundsen with being the first person to ski to the South Pole, and that, possibly, someone could have reached the pole through other means. This document was marked unsupported.

Sometimes the correct answer has more than one part to it. Consider question 1534, "The sun is mostly made up of what two gases?", which is not a part of the test collection. The correct answer is hydrogen and helium, and a document that does not give both parts of the answer can not be marked relevant. It is debatable whether a document that says, for example, that the sun is comprised mostly of hydrogen can be unsupported, or whether it must be marked irrelevant. For this reason, the question was excluded from the test collection.

We arrived at these criteria incrementally; after each performing a blind annotation on a set of questions and analyzing where our judgments differed, Leah and I discussed and agreed on the content of these guidelines. We found that we needed to iterate through this process twice to achieve a level of agreement we were satisfied with. More details about this appear in Section 5.4. When we had finally settled on annotation guidelines that were acceptable to both of us, we each reviewed our own previous judgments to bring them in line with the latest guidelines.

5.4 Inter-annotator Agreement

When a group of human annotators are asked to produce a set of judgments, disagreement can occur even when there are agreed-upon guidelines. Multiple natural and reasonable interpretations for the question, what constitutes the answer, and what constitutes a relevant document can lead to genuine disagreement among annotators. The community relies on a measure known as *inter-annotator agreement* to indicate whether or not a group of annotators is applying the same set of criteria to its judgments.

To achieve good inter-annotator agreement, and to make sure that we were each faithfully applying the criteria outlined in Section 5.3.2, Leah and I conducted a series of blind tests on three question sets, totaling 26 questions. I developed a simple Perl script to compute the correlation on a per-document basis for the two sets of judgments, and also a measure of inter-annotator reliability called the Kappa statistic [6]. Kappa measures inter-annotator agreement adjusted for chance, and it ranges from zero to one, where $\kappa = 1.0$ refers to perfect agreement between the annotators and $\kappa = 0.0$ refers to agreement that is no better than chance. According to Carletta, $\kappa > 0.8$ show good reliability between the annotators, and $0.67 < \kappa < 0.8$ is sufficient evidence to draw preliminary conclusions.

Our first question set was drawn randomly from a range of questions that I had already annotated. The task was then for Leah to assess the same set of questions, shown in Table 5.1, without seeing my judgments or discussing the questions or

documents with me.

Number	Question
1400	When was the telegraph invented?
1401	What is the democratic party symbol? *(sic.)*
1403	When was the internal combustion engine invented?
1416	When was Wendy's founded?
1419	What year did Alaska become a state?
1429	What was Andrew Jackson's wife's name?
1438	What body of water does the Colorado River flow into?
1458	What was the name of the high school in "Grease"?
1494	Who wrote "East is east, west is west and never the twain shall meet"?
1504	Where is the Salton Sea?

Table 5.1: *First Question Set for Inter-annotator Agreement (IAA1)*

Initially, we had each agreed to use a "reasonable" human definition of whether each document we read was relevant or not. When it came to identifying unsupported documents, the agreement was that documents that almost but did not quite answer the question were to be marked unsupported. All other documents judged that did not meet either of these loosely-defined sets of criteria were to be marked irrelevant. There were questions for which Leah and I formulated different Lucene queries, leading to documents that one of us marked and the other didn't. For the purposes of computing inter-annotator agreement, those documents were marked irrelevant in the other assessor's judgment set. Looking at the results of the script on our first blind test, shown in Table 5.2, it was clear that Leah and I were working with different sets of criteria. The table is set up as a matrix; my judgments are along the top, and Leah's are along the side. Each cell shows the number of documents for which Leah and I made the corresponding judgments, and the percentages shown are the number of documents in the cell divided by the number of total documents mentioned by either assessor. Cells corresponding to agreement between the annotators appear on the diagonal, in bold face.

A careful analysis of our results showed that we had 83.71% total agreement, but that we had unacceptable levels of errors (over 5%) in which one of us marked a document irrelevant, and the other marked it relevant. This figure is misleading,

Oats Judgment	Bilotti Judgment		
	relevant	unsupported	irrelevant
relevant	**26 (9.85%)**	0	2 (0.76%)
unsupported	24 (9.09%)	**5 (1.89%)**	4 (1.52%)
irrelevant	13 (4.92%)	0	**190 (71.97%)**

Table 5.2: *Inter-annotator Agreement Results on IAA1*

though, because we had to correct for documents mentioned by only one assessor, as described above, and the result was an increase in this type of judgment error. The κ value for this first blind test was 0.5726, which leaves much to be desired. We also took the opportunity to agree on a better formulation of the criteria, to ensure we were both judging documents against the same standard.

After revising the criteria, we chose a second question set, shown in Table 5.3, and calculated our inter-annotator agreement again. This time, we standardized the Lucene queries used so that each of us would judge exactly the same set of documents. The complete results of this second calculation are shown in Table 5.4, formatted in the same way as Table 5.2. The overall agreement on this blind test was 80.18%.

Number	Question
1474	What is the lowest point on earth?
1475	Who was the first person to reach the south pole?
1480	What is the principle port in Ecuador? *(sic.)*
1483	Where is the highest point on earth?
1484	What college did Allen Iverson attend?
1488	What is the name of the professional baseball team in Myrtle Beach, South Carolina?
1490	What is the Boston Strangler's name?
1492	How old was Nolan Ryan when he retired?
1516	What does CPR stand for?
1520	What is the capital of Kentucky?

Table 5.3: *Second Question Set for Inter-annotator Agreement (IAA2)*

In this second blind test, we reduced the amount of relevant/irrelevant errors to less than 1% while maintaining above 80% in total agreement. On this test, we achieved a much improved Kappa measure of $\kappa = 0.6897$. We clarified the criteria

Oats Judgment	Bilotti Judgment		
	relevant	unsupported	irrelevant
relevant	**68 (20.42%)**	20 (6.01%)	3 (0.90%)
unsupported	21 (6.31%)	**40 (12.01%)**	17 (5.11%)
irrelevant	0	5 (1.50%)	**159 (47.74%)**

Table 5.4: *Inter-annotator Agreement Results on IAA2*

once more and tested a several extra questions, which are shown in Table 5.5. The results from this final blind test are excellent, showing almost 89% total agreement. Kappa for the third blind test decreased slightly to $\kappa = 0.6137$, which is likely attributable to the comparatively low number of total judgments made in this test. The complete results are shown in Table 5.6, which is again formatted in the same style as Table 5.2.

Number	Question
1527	When did the 6-day war begin?
1531	What does NASDAQ stand for?
1533	Who directed the film "Fail Safe"?
1536	What city is Lake Washington by?
1537	How many electoral college votes in Tennessee? *(sic.)*
1538	Who is the evil H.R. Director in "Dilbert"?
1539	What is Ronald Reagan's favorite candy?

Table 5.5: *Third Question Set for Inter-annotator Agreement (IAA3)*

Oats Judgment	Bilotti Judgment		
	relevant	unsupported	irrelevant
relevant	**212 (80.62%)**	7 (2.66%)	0
unsupported	8 (3.04%)	**13 (4.94%)**	0
irrelevant	0	14 (5.32%)	**9 (3.42%)**

Table 5.6: *Inter-annotator Agreement Results on IAA3*

Total results over all three question sets are shown in Table 5.7. Overall agreement is 83.96%. The overall Kappa value of 0.7351 shows significant improvement, now just short of the $\kappa = 0.8$ mark that indicates solid inter-annotator agreement.

In total, the test collection contains 120 questions, and of them, 65 (54.17%) were

Oats Judgment	Bilotti Judgment		
	relevant	unsupported	irrelevant
relevant	**306 (35.58%)**	27 (3.14%)	5 (0.58%)
unsupported	53 (6.16%)	**58 (6.74%)**	21 (2.44%)
irrelevant	13 (1.51%)	19 (2.21%)	**358 (41.64%)**

Table 5.7: *Inter-annotator Agreement Results on IAA1-3*

annotated by me personally, and the remaining questions were handled by Leah. There are 26 (21.67%) questions that were doubly-annotated by both of us for the purposes of calculating inter-annotator agreement. As of the publication of this thesis, not all of questions in the test collection have undergone adjudication, but as soon as the work is completed, the test collection will be in distributable form.

For those interested in summary statistics about the test collection, it contains 6009 total judgments; 1901 of which were relevant, 298 of which were unsupported, and 3810 of which were irrelevant. There are an average of 50.08 total judgments per question, and each question averages 15.84 documents judged relevant, 2.48 documents judged unsupported, and 31.75 documents judged irrelevant. In total, an estimated 229.60 person hours were invested in human assessment for this test collection.

Chapter 6

Evaluation

My primary goal throughout this thesis has been to build a better-performing document retrieval module for a question answering (QA) system by incorporating query expansion techniques. This chapter covers the experiments I performed using the Pauchok II evaluation framework to find the query expansion techniques that best improve document retrieval performance for a QA system.

6.1 Query Expansion Algorithms in Pauchok II

The Pauchok II system makes it easy to rapidly build and evaluate query expansion algorithms. The system comes pre-loaded with a control structure, and two term extractors, two term expansion functions and three term-dropping strategies to use with it.

- **Control structure:** With the control structure available to me, I can create a complete query expansion algorithm using one term extractor, zero or more term expansion functions, and one term-dropping strategy. The control structure works by first passing the question through the term extractor to produce a list of query terms present in the question, and then passing each of these terms through any term expansion functions available to the control structure. The control structure aggregates query terms and their alternate forms into disjunctive clauses. The dropping strategy is responsible for assembling conjunctive

combinations of those clauses into an ordered list of Boolean queries, which is then passed to the document retrieval engine.

- **Term extractors:** I have two term extractors at my disposal; the Word Term Extractor, which is the default, and the Phrase Term Extractor, which is used when incorporating phrase analysis into the query expansion algorithm. The Phrase Term Extractor can not detect phrases on its own; it relies on the blackboard-based question analysis system, the annotators of which are capable of marking phrases for it.

- **Term expansion functions:** Two term expansion functions currently are available, and they both operate solely on single-word query terms as opposed to phrases. The inflectional expansion function uses information provided by WordNet to generate alternate tenses for verbs, pluralizations for nouns, and superlative and comparative forms for adjectives. The derivational expansion function is backed by CELEX, and generates a wide range of words and phrases that can be derived from or are related to the query term.

- **Term-dropping strategies:** At this time, there are three term-dropping strategies, designated by the colors red, green and blue. All three rely on a heuristic related to term inverse document frequency (IDF) to guide the order in which they drop terms. The Red Dropping Strategy starts with a conjunction of all of the terms and drops them in order of increasing IDF, starting with the most common term. The Green Dropping Strategy improves on the Red Dropping Strategy by dropping the rarest term and restarting the IDF-order dropping after having exhausted all of the terms. Neither the Red nor the Green Dropping Strategies try all combinations of query terms, but the Blue Dropping Strategy does do so, trying all combinations of size n before dropping a term, and then trying all combinations of size $n - 1$.

With these components, I can build twelve different query expansion algorithms that operate only on single-word terms, using a combination of one term-dropping

strategy and zero or more term expansion functions. Another twelve can be built that are capable of phrase analysis.

6.2 Approach

The goal of these experiments is to compose the query expansion algorithm with the best performance possible, using the components available in the Pauchok II system. The evaluation metrics I use to determine what constitutes better performance of one query expansion algorithm over another are recall of relevant documents; total reciprocal rank, which is defined as the sum of the reciprocals of the ranks at which relevant documents are returned; and mean reciprocal rank, which is computed by dividing the total reciprocal rank by the number of relevant documents retrieved. For the purposes of the experiments in this chapter, I randomly chose 60 questions from the test collection to serve as a training set. Table A.1 in Appendix A shows the questions that are in the training set.

Central to this task is tuning the parameters α and β, which are the default discount factors for alternate forms of query terms generated by the inflectional and derivational term expansion functions, respectively. These parameters are interpreted by the document retrieval engine as being weights on query terms, such that a match on an expansion form in a document contributes less to the overall score of that document. Setting these parameters too low reduces the effectiveness of query expansion, since matching the expansion forms does not boost the score of a document enough to affect its rank. Setting the parameters too high can destroy performance by preferentially retrieving documents that contain an abundance of expansion forms and few or none of the original forms, and are in fact not relevant. I hypothesize the existence of an optimal value for each parameter that yields best performance.

My approach is to tune optimum values for the parameters α and β, first independently and then simultaneously. The query expansion algorithm used for each test is based on the Word Term Extractor and the Red Dropping Strategy. When tuning α alone, I use inflectional expansion only, and when tuning β alone, I use only the

derivational expansion function. When tuning both parameters, I use both forms of term expansion.

After finding the best values for α and β, I turn my attention to evaluating the performance of the various combinations of dropping strategies and expansion functions by setting up a matrix experiment and evaluating the performance of each of the twelve combinations.

6.3 Tuning Parameters for Word Morphology

As mentioned previously, for the purposes of query term expansion, alternate forms of a word are given a discount factor meant to represent the semantic difference between the new and the original word forms. Alternate word forms generated by the inflectional expansion function bear a discount factor of α, and those generated by the derivational expansion function bear a discount factor of β.

Originally, these values were set to 0.75 and 0.5, respectively, for the purposes of participating in the TREC 2003 QA track. The values were chosen to generally reflect an intuitive sense of how semantically distant the new word forms were from the originals, but one of my goals all along was to be able to tune these parameters automatically. This section recounts a set of experiments designed to identify optimum values for α and β. All of the experiments in this section use query expansion algorithms based on the Word Term Extractor and the Red Dropping Strategy.

6.3.1 Alpha: The Inflectional Expansion Discount Factor

To find the optimum value of α, I set up an experiment using inflectional term expansion as a part of my query expansion algorithm. As I varied α with a step size of 0.05, I generated queries for each question in the training set and used them to retrieve documents from the corpus, up to a certain limit. For each set of returned documents, my evaluation framework computed the recall, the mean reciprocal rank (MRR) and the total reciprocal rank (TRR), which is defined as the sum of all reciprocal ranks, over not only relevant documents but also combined relevant and

unsupported documents.

To determine how the limit on the number of documents returned by the document retriever would affect this calculation, I tried the same experiment with limit values of 100, 250, 500, 750 and 1000. Figure 6-1 shows a plot of recall for relevant documents as α varies along the abscissa. The five lines correspond to the experiments performed with limit values of 100, 250, 500, 750 and 1000, from bottom to top, respectively. Table 6.1 gives a summary of the experiments performed in this section.

Limit	Term Extractor	Term Expansion	Dropping Strategy
100	Word	inflectional	Red
250	Word	inflectional	Red
500	Word	inflectional	Red
750	Word	inflectional	Red
1000	Word	inflectional	Red

Table 6.1: *Experiments Performed while Finding Alpha (α)*

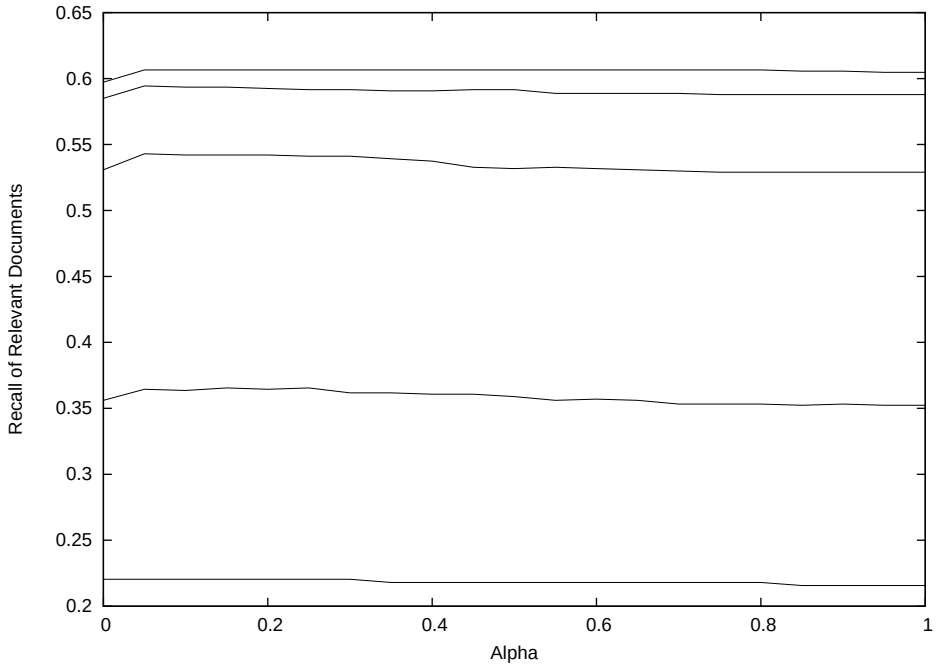

Figure 6-1: *Recall of relevant documents as α varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The results of these five experiments are not directly comparable.*

It is difficult to draw any conclusion from Figure 6-1 because the recall of relevant

documents for each of the five experiments is not directly comparable. For this reason, I normalized the recall data for each experiment, fitting each on a zero-to-one scale. Figure 6-2 shows this new data set, in which the maximum recall achieved in the experiment appears as 1 and the minimum recall appears as 0. The average over all five different limit experiments is shown as the heavy black line. By inspection, it can be seen that the best α is 0.05.

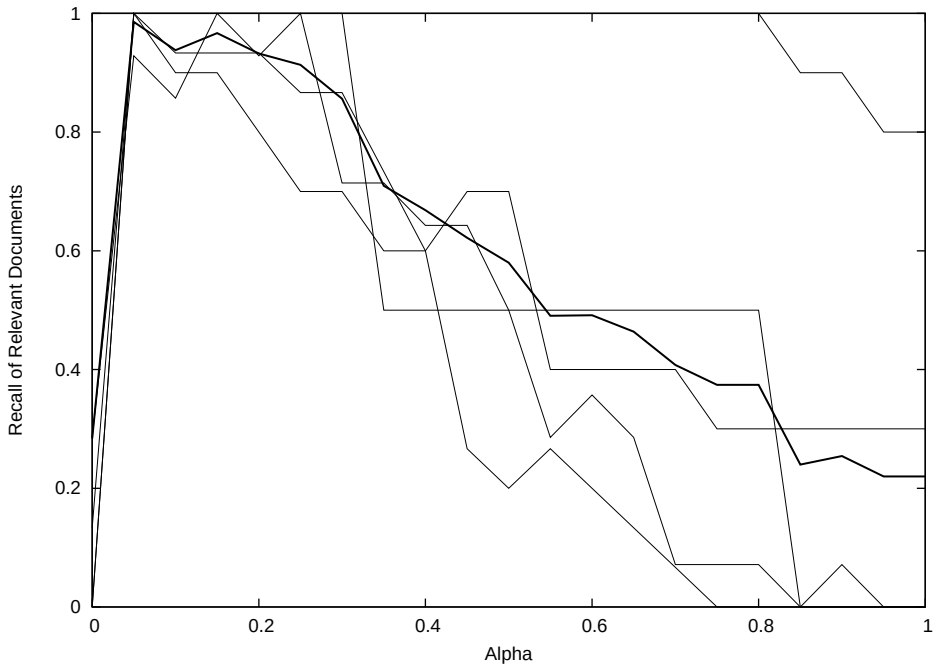

Figure 6-2: *Normalized recall analysis as α varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The heavy black line shows the average across the five experiments.*

I have prepared similar plots showing MRR and TRR averaged over the five limit experiments. Figure 6-3 shows MRR as alpha varies from 0.0 to 1.0. In the figure, the black line indicates the average across the five experiments corresponding to limit values of 100, 250, 500, 750 and 1000. Figure 6-4 shows TRR as alpha varies over the same interval, again with the average across the five experiments shown as a heavy black line. In terms of MRR and TRR, the optimal value of α is 0.35.

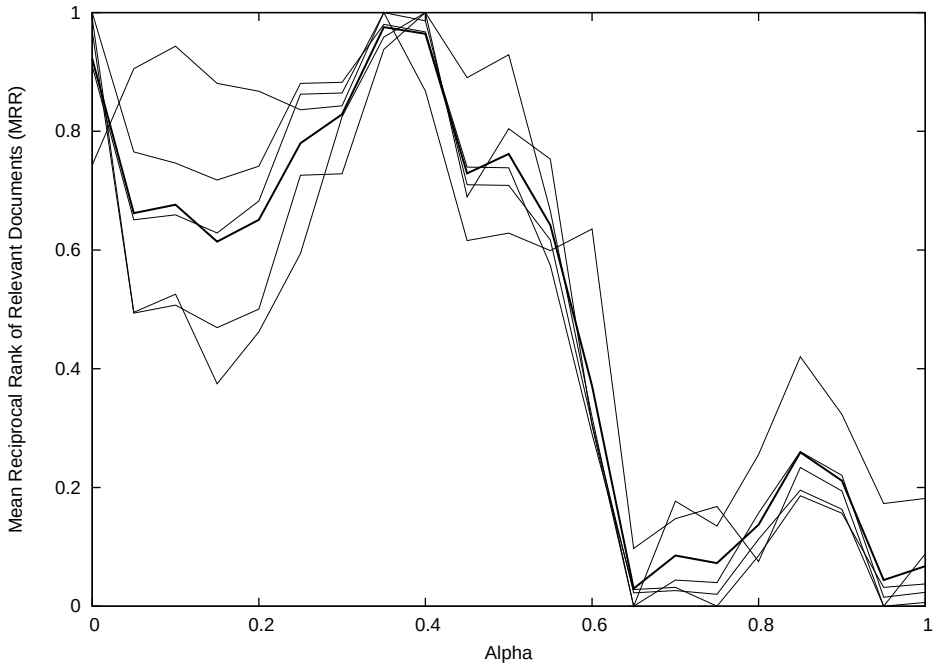

Figure 6-3: *Normalized MRR analysis as α varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The heavy black line shows the average across the five experiments.*

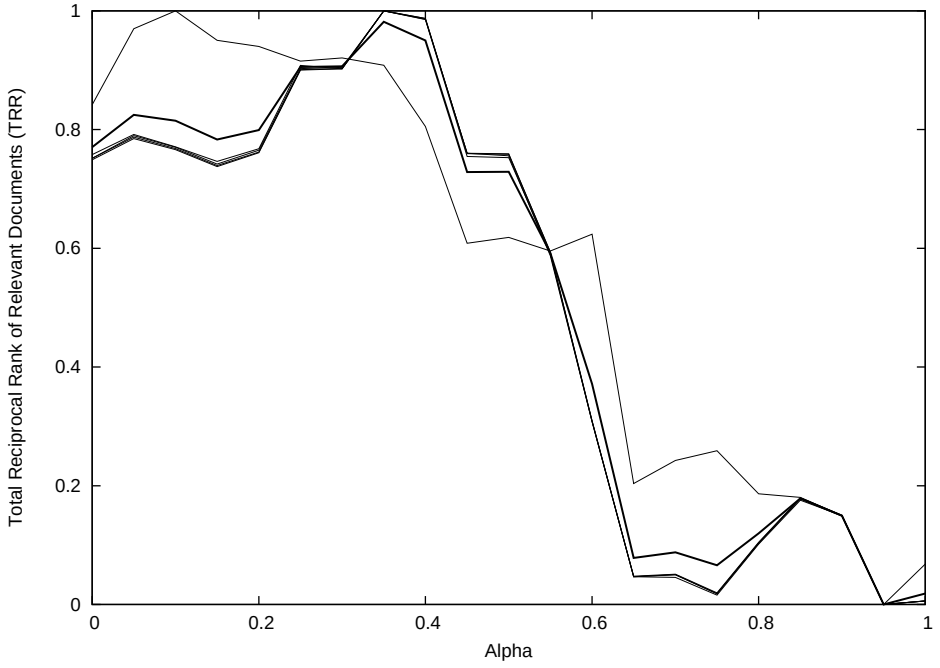

Figure 6-4: *Normalized TRR analysis as α varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The heavy black line shows the average across the five experiments.*

6.3.2 Beta: The Derivational Expansion Discount Factor

I followed a similar procedure to find the optimum value of β. I set up an experiment using only derivational expansion for expanding query terms. I again examined the behavior of the parameter over the range from 0.0 to 1.0, with a step size of 0.05. For each value of β, I generated queries and retrieved documents for the questions in the training set. I ran five experiments, each with different values for the limit on the number of documents returned, those values being 100, 250, 500, 750 and 1000. Table 6.2 shows the experiments performed in this section.

Limit	Term Extractor	Term Expansion	Dropping Strategy
100	Word	derivational	Red
250	Word	derivational	Red
500	Word	derivational	Red
750	Word	derivational	Red
1000	Word	derivational	Red

Table 6.2: *Experiments Performed while Finding Beta (β)*

From my experience with α, I knew to normalize the values provided by my evaluation framework, and then to average across the five different limit experiments. Figure 6-5 shows recall of relevant documents averaged across all five experiments. One interesting thing to note about the figure is how quickly recall falls off as β increases, when compared with how recall falls off with increased α, which is shown in Figure 6-2. The derivational morphology expansion facility usually generates at least some alternate forms that are not as closely related to the original query term as are the forms generated by inflectional morphology expansion. Increasing the weight on alternate forms that are more semantically distant from what the system is looking for causes it to preferentially retrieve documents that contain these forms, fewer of which are actually relevant. The behavior of the average recall curve shown in Figure 6-5 is consistent with this explanation, indicating that the value of β that leads to maximal recall of relevant documents is 0.15.

I was also interested in seeing how different values of β affects MRR and TRR

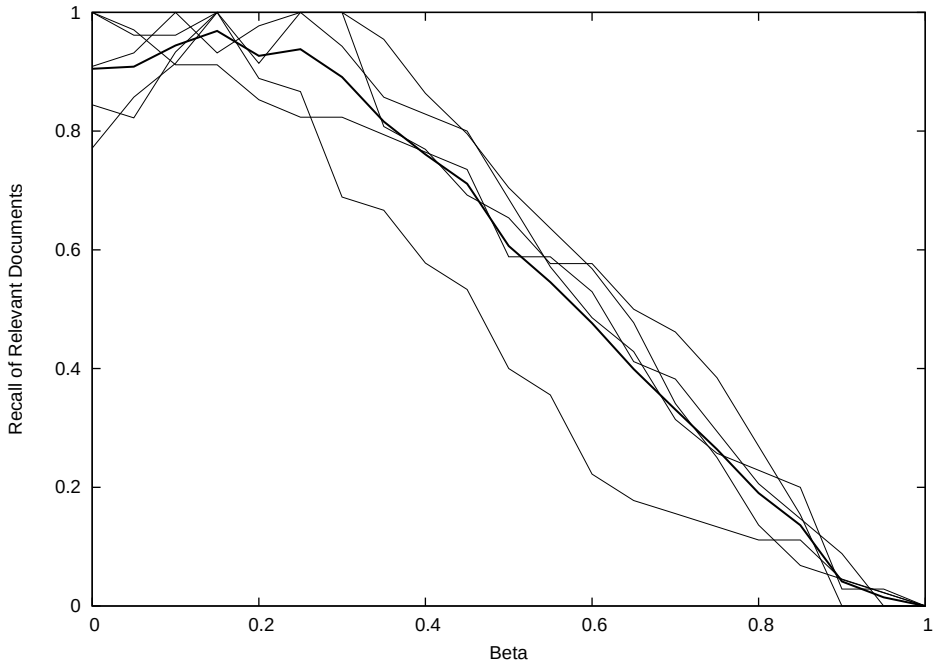

Figure 6-5: *Normalized recall analysis as β varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The heavy black line shows the average across the five experiments.*

of relevant documents. Figure 6-6 shows MRR of relevant documents as β ranges between 0.0 and 1.0 in increments of 0.05. Figure 6-7 shows the behavior of TRR as β varies over that same interval. Interestingly, after having been normalized, the variance of TRR across the five experiments is very small; the figure would have looked the same if I had plotted only the five-experiment average. Both figures indicate that optimal value of β is 0.2.

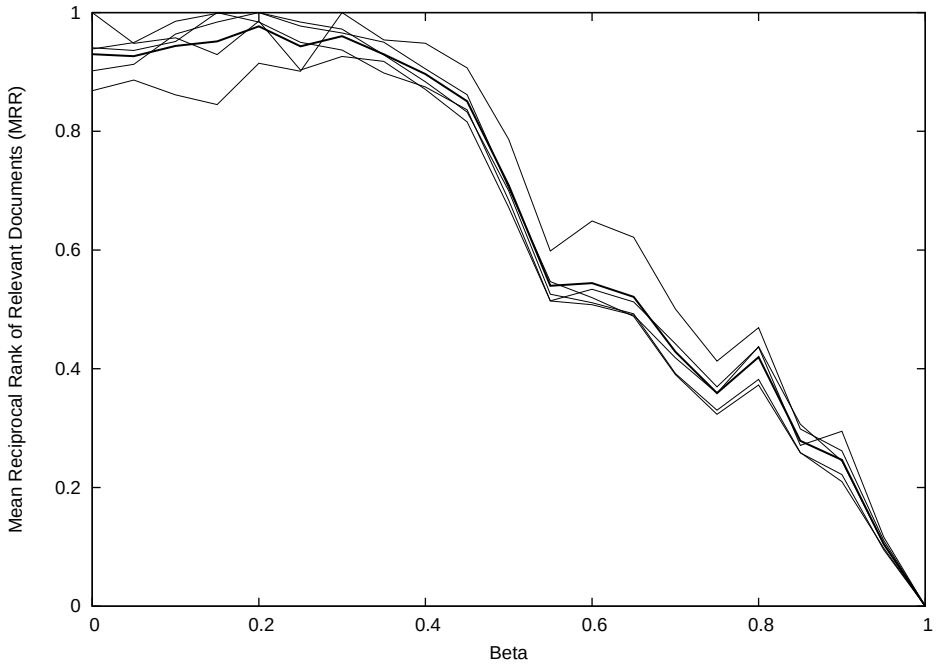

Figure 6-6: *Normalized MRR analysis as β varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The heavy black line shows the average across the five experiments.*

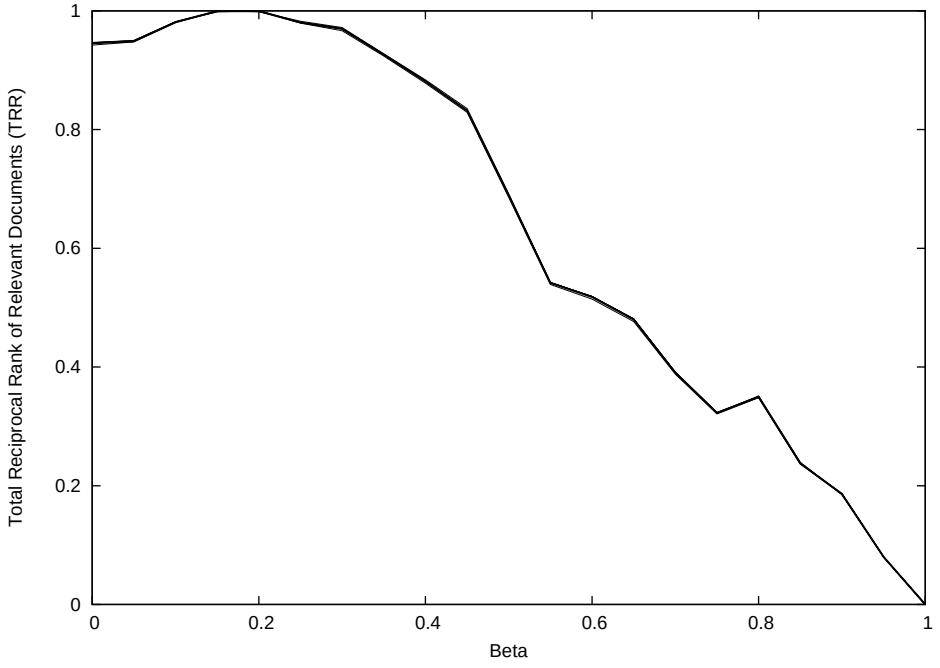

Figure 6-7: *Normalized TRR analysis as β varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top. The heavy black line shows the average across the five experiments.*

6.3.3 Interdependence of Alpha and Beta

I was concerned that the optimal values found for α and β individually might not be the best values for the parameters when both are used simultaneously. It is reasonable to expect that the inflectional and derivational expansion systems are not independent of each other because, in some cases, each return the same alternate forms for query terms. Figure 6-8 shows how recall of relevant documents varies as α and β both range from 0.0 to 1.0, with a step size of 0.05. The document retrieval limit in this case is 100 documents. It is easy to see from the contour lines that have been drawn in the x-y plane that the highest recall is achieved when $0.2 < \alpha < 0.35$ and $0.0 < \beta < 0.4$. The peak of the surface is a ridge where $\alpha = 0.3$ and $0.0 < \beta < 0.25$.

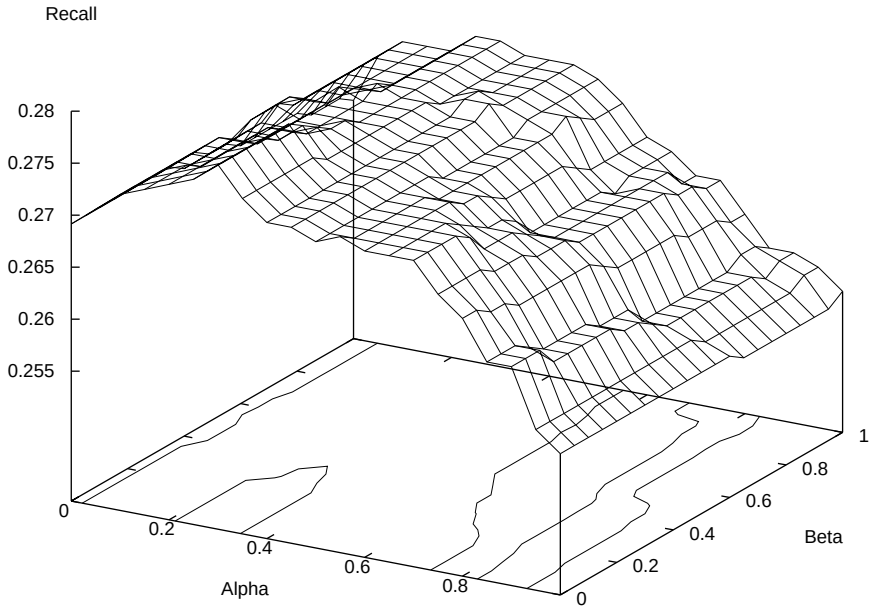

Figure 6-8: *Normalized recall analysis as β varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top.*

Figure 6-9 shows the results of the same experiment in terms of MRR. Although it is difficult to see from the figure, there is a peak of maximum MRR at $\alpha = 0.0$ and $\beta = 0.4$. The surface appears jumbled and noisy, but the picture is much clearer when looking at TRR.

Figure 6-10 shows TRR as α and β both vary in increments of 0.05 between 0.0

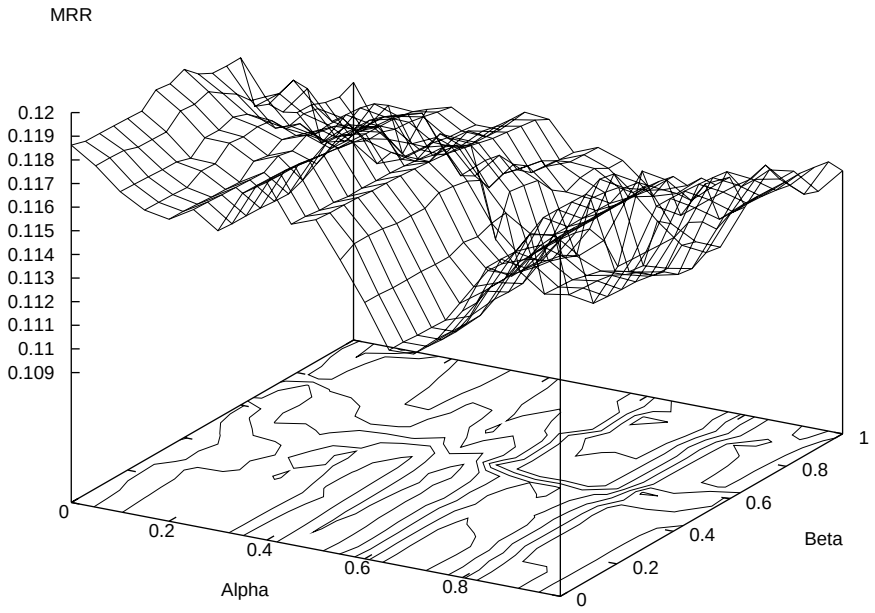

Figure 6-9: *Normalized MRR analysis as β varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top.*

and 1.0. The TRR peak is at $\alpha = 0.35$ and $\beta = 0.0$, but the highest plateau of TRR, as shown by the contour lines, is approximately where $0.0 < \alpha < 0.5$ and $0.0 < \beta < 0.5$.

To summarize, the highest-performing value for α, when considered alone, is 0.05 in terms of recall and 0.35 in terms of MRR and TRR. For β alone, the results are showing that best performance occurs at 0.15 for recall, and 0.2 for MRR and TRR. When both parameters are tuned together, it is clear from Figure 6-8 that α must be above 0.2 to achieve best recall, so the $\alpha = 0.05$ result for recall independent of β is probably the result of a dependence between the inflectional and derivational term expansion functions.

To combine the results from the independent and combined experiments for α and β, it is reasonable to average their best performing values for each experiment, because the values are close to each other. For α, best performance individually is at 0.35 and best performance in the combined experiment is at 0.3. For β, best individual performance occurs at 0.15 and 0.2 and best combined results were at

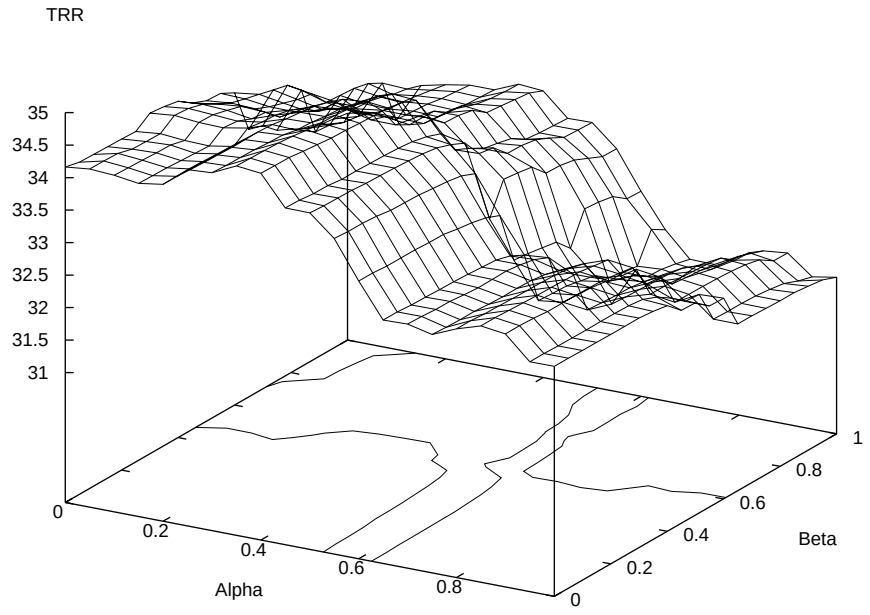

Figure 6-10: *Normalized TRR analysis as β varies in the range from 0.0 to 1.0, for five limit values of 100, 250, 500, 750 and 1000, from bottom to top.*

values of 0.0 and 0.4. All told, the results of the experiments in this section indicate that reasonable values for α and for β are 0.325 and 0.1875, respectively. These values will be used in all remaining experiments in this chapter.

6.4 Dropping strategies

Setting $\alpha = 0.325$ and $\beta = 0.1875$, I evaluated the performance of each dropping strategy, under four conditions: no term expansion, inflectional expansion only, derivational expansion only, and both. I performed this battery of experiments five times, one for each value of document limit: 100, 250, 500, 750 and 1000. In total, sixty experiments were performed, and Appendix B contains all of the raw results. In this section, I present two tables showing the percent change averaged across the document limit experiments for each combination of dropping strategy and expansion, and I discuss the results of the experiments.

In Table 6.3, I show percent change in terms of recall of relevant documents. The combination of the Blue Dropping Strategy and inflectional expansion shows the best

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	—	+3.11%	-1.57%	+2.13%
Green	+2.64%	+5.57%	+0.78%	+5.00%
Blue	+6.32%	**+6.48%**	+5.10%	+5.94%

Table 6.3: *Recall Dropping–Expansion Matrix showing percent change averaged over five values of document limit: 100, 250, 500, 750 and 1000.*

overall performance improvement according to this table. Looking at the table, it can be seen that inflectional expansion outperforms derivational expansion when a single expansion function is used. In fact, whenever derivational expansion is added to the query expansion algorithm, performance suffers relative to the original algorithm. I attribute this to the fact that the derivational expansion function often generates expansion forms that are quite distant from the original query terms in meaning.

It is also clear from the table that the term-dropping plays a significant role in improving performance. Even without any expansion, the Blue Dropping Strategy offers a commanding improvement in performance over the Red Dropping Strategy because it emphasizes querying on the original terms in all possible combinations rather than quickly broadening the query by dropping as many terms as possible.

In Section B.2, there is a matrix showing recall of relevant documents for five values of document limit: 100, 250, 500, 750 and 1000. It is interesting to examine how different dropping strategies and different term expansion methods vary in performance with respect to the document limit. The Green and Blue Dropping Strategies, for example, generate so many queries that the later ones can only be executed when the document limit is high. This is a consequence of the fact that documents returned from a sequence of queries are concatenated. Higher document limit also reduces the effect that expansion has on improving recall, because term-dropping in later queries broadens the scope of documents retrieved to include many of those that would have matched the expanded forms of the original query terms.

Table 6.4 shows percent change in terms of TRR of relevant documents for the matrix of dropping strategy and expansion combinations. TRR is a measure that

reflects not only how many relevant documents were returned, but also how highly ranked they were. It is a better measure than recall, which can be inflated as document limit is increased.

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	—	-5.18%	+1.41%	-4.45%
Green	+3.56	-1.80%	+4.87%	-1.12%
Blue	+3.26%	-3.21%	**+5.08%**	-2.40%

Table 6.4: *TRR Dropping–Expansion Matrix showing percent change averaged over five values of document limit: 100, 250, 500, 750 and 1000.*

Table 6.4 shows that, overall, the Blue Dropping Strategy performs and derivational expansion yield the best performance in terms of TRR. Regardless of dropping strategy, derivational expansion improves TRR and inflectional expansion lowers it. This is an almost certainly misleading result, because β, the weight on derivational expansion forms, is so low. If the weight were higher, derivational expansion would be causing a performance decrease just as inflection does. If the performance increase were really due to derivational expansion, the second and fourth colums of the table would not look so similar.

Performance in general suffers when term expansion is used, because of a feature of the scoring mechanism of the IR engine that Pauchok II is based on. There is a coordination value factored into the score for a document that represents the ratio of the query terms matched in that document to the total number of terms in the query. This coordination penalty explains the behavior of recall as term expansion is added to the query expansion algorithm; as the number of terms in the query increases, ranks of relevant documents are shifted downward, decreasing TRR. A certain percentage of relevant documents are shifted past the document limit, which causes recall to drop accordingly. If we had better ways of merging results from multiple queries, rather than concatenating them, we might be able to alleviate this problem somewhat.

In general, coordination seems like a reasonable component for the scoring metric of a general-purpose IR engine to include. Imagine we were looking for information

about what to serve for dessert at a dinner party, and we queried using a disjunction of terms representing different types of pastries. We would want a document containing the terms *sfogliatelle*, *zeppole* and *struffoli* to be ranked higher than a document containing a single mention of an *eclair*. For term expansion, this is not the kind of scoring we want, since expanded forms of the original term appear with it in large disjunctive clauses. Ideally, we would define a special kind of disjunction for which matches of one or more of the component terms are scored equally. If this were the case, querying on a large disjunction of morphological variants of each term would not uniformly penalize documents for coordination.

One caveat about the results presented in this section is that they are aggregated over the five different values of document limit. It is important to view the results in this context, because the optimal combinations of term-dropping strategies and term expansion methods that they suggest would be most successful if there were no single fixed document limit used in the application. If there were a specific limit on documents returned used consistently by the application, best performance would result from performing a matrix experiment, such as those in Appendix B to choose the best combination of dropping and expansion for the particular limit value needed by the application.

I believe that it would be interesting to watch how the performance of each combination of dropping strategy and term expansion behaves as α and β vary between 0.0 and 1.0. I have seen no evidence that proves to me that the optimal dropping strategy and expansion combination is independent of the parameters. Throughout the experiments in this section, I have assumed that the optimal query expansion algorithm found when α and β were set to their optimal value would be the maximally performing query expansion algorithm, but that may not be completely true. This is one of the questions that I will be investigating in my further study of query expansion algorithms.

6.5 Issues with Phrase Analysis

Although Pauchok II is capable of recognizing phrases in the input question, through the partnership of the phrase term extractor and the blackboard-based question analysis facility, no results of experimentation with query expansion algorithms that utilize phrase analysis are presented here, because of the difficulty of designing effective phrase expansion strategies.

Phrase recognition can do wonders for precision, directly targeting documents that unambiguously refer to whatever it is that the user is asking about, but it can be a hindrance as well. The more exact a phrase is, the more likely it is to be very infrequent in the corpus. A system can get into trouble using phrases to query for something, such as a person's name, which can take many forms. Without some way of generating alternate forms of phrases, a system may end up missing some percentage of relevant documents that happen to use some other way of referring to the question target. I have confirmed with experiments on the Pauchok II system that phrase analysis without expansion causes a performance penalty, not only in terms of recall but also in terms of TRR, which means that not only are fewer relevant documents retrieved when phrase analysis is in use, but also that they are returned at lower ranks. There is clearly a trade-off at work here. A system has to have some kind of knowledge of when using phrase analysis will be an advantage.

When the structure of a phrase is known, it is reasonable to use that information to generate expansion forms of that phrase, perhaps using discount factors to represent semantic distance from the original term. Consider the class of phrases that represent names of people. Certain instances of this class can give rise to a large number of expansion forms. As an example, expansion forms such as "President Franklin D. Roosevelt," "President Franklin Roosevelt," "Franklin D. Roosevelt," and even "President Roosevelt" and "Roosevelt," if there is sufficient context to make these forms unambiguous, can be generated from the phrase "President Franklin Delano Roosevelt."

Not all phrase instances can be expanded in this way. Such expansion depends on

knowing the internal construction of the phrase for the class of phrases to which the instance belongs, and knowing that the instance is a member of the class. Sometimes it is not easy to fit a phrase instance into the ontology. Consider, for example, the phrase "first person" in TREC 2002 question 1475, "Who was the first person to reach the south pole?" The phrase was mistakenly recognized by the system while it was looking to match noun–noun collocations extracted from WordNet. Clearly, the phrase "first person" is not being used in the question as a noun, but rather as an adjective–noun combination. Keeping "first person" as an indivisible phrase throughout the process for this question would be a mistake. It seems that if a system can not identify the sub-structure of a phrase, it should not be marked as such.

This thesis does not contain experiments with phrase analysis because they would not yield a great deal of insight without the application of proper phrase expansion techniques. There are a great many ways to do phrase expansion that square with intuition, but without careful experiments, there can be no telling which methods work, and which do not. Such experiments are beyond the scope of this thesis. Although I could have included phrase analysis here by designing some ad hoc phrase expansion methods, they would not have been generally applicable. Phrase expansion is instead left as a potential avenue for future query expansion research.

Chapter 7

Contributions

This thesis has been the product of approximately a year of work in document retrieval for TREC-style question answering (QA), and the desire to build a better-performing document retrieval module.

7.1 Motivation

In a pipelined QA system, one in which an upstream document retrieval stage provides a list of documents relevant to the input question to downstream passage retrieval and answer extraction stages, documents that are not retrieved early in the process can never be searched for answer candidates. In the extreme case in which no relevant documents are retrieved by the document retriever, there is no point in continuing the QA process, because answers gleaned from whatever documents were actually retrieved are unlikely to be correct. The overarching vision of this thesis is that focusing on improving recall of relevant documents at the document retrieval stage of a pipelined QA system can contribute greatly to improved accuracy of the system as a whole.

Query expansion refers to a family of recall-boosting techniques especially suited to Boolean keyword and phrase document retrieval engines. The motivating problem behind this thesis was to fully explore the potential of query expansion techniques such as inflectional and derivational word morphology, term-dropping strategies and

phrase analysis to improve recall of relevant documents for this type of document retrieval engine.

7.2 Contributions

This thesis chronicled the building of a comprehensive test collection for evaluating document retriever performance, consisting of 120 questions drawn from the TREC 2002 question set, each one associated with lists of document identification numbers of documents from the AQUAINT corpus known to be relevant, unsupported or irrelevant for that question. Only 26 (21.67%) percent of these questions were subjected to double-annotation, and subsequent adjudication and calculation of statistics regarding inter-annotator agreement, so the test collection is not yet ready for distribution. We consider this test collection to be one of the contributions of this thesis, so as soon as the work on it is completed, we intend to release it so that other research groups can benefit from it.

Another equally-important contribution of this work is the Pauchok II infrastructure itself, which is a flexible and powerful toolkit for building document retrieval modules, and evaluating them against the test collection. Because it is based on Scheme, the Pauchok II system promotes rapid prototyping and testing. Pauchok II includes pre-defined components for query expansion algorithms, such as control structures, term extractors, query term expansion functions, and term-dropping strategies. The system also makes it easy to manipulate data and understand how questions translate into queries, which then give rise to documents, by means of the Pauchok II visualization package. As part of the AQUAINT program, we intend to polish and fully document this system and release the source code to the QA research community.

Lessons learned about query expansion for QA purposes include the following. Term expansion improves recall by retrieving documents that do not contain all of the original query terms, but do contain alternate forms of those terms. Tuning the discount factors for the alternate forms can be done experimentally for a set of ques-

tions; it does not have to be ad hoc. Attention should be paid to the tuning because, if the discount factors are too high, irrelevant documents that do not contain any of the query terms, but that do contain alternate forms in abundance can overwhelm the results. If the discount factors are too small, query expansion contributes very little to the overall score of a document. One difficulty with query expansion techniques can be how the IR engine backing the system interprets the queries and scores them against documents. The big lesson in this thesis, though, is that given an appropriate test collection and evaluation framework, a variety of query expansion algorithms can be quickly built and experimented with, making finding the right query expansion technique for the task at hand easy.

7.3 Future Directions

It is my belief that high recall of relevant documents in an upstream document retrieval stage is essential to a high-quality, pipelined QA system. Query expansion techniques can contribute quite a bit to the improvement of recall, but there is a limit to what they can do. I believe that query expansion is constrained by the query interface and syntax provided by the document retrieval engine backing the QA system. If a document retrieval engine were to expose through its query syntax more fine-grained control over the document search process, there would be a much richer space of possibilities for query expansion techniques.

To fully support QA, we as a research community need to explore ways to integrate linguistic information into the indexing, querying and retrieval processes. Although they work to some degree, it is not necessarily best to index keywords for the QA task. We should be indexing named entity references, instances of classes, semantic roles, and relations among objects and concepts mentioned in the documents, and we should be providing convenient and expressive syntax for querying on the basis of these linguistic objects.

It is clear that, for the purposes of TREC evaluation and answering questions over a closed, static corpus, such thorough indexing could be done semi-automatically, or

even manually, were enough person-hours available. For this kind of IR to evolve into the mainstream, though, we need to develop ways to automatically learn how to recognize these references, induce these relations among objects, and compile these indexes. If we succeed in building IR tools that are robust, linguistically-aware and automatic, we will not only have made great advances in QA, but we will have revolutionized IR as well. Such projects are just clouds dotting the horizon as of yet, but are exciting nonetheless. I look to the future of QA with hope and expectation that we will one day be able to realize the dream, now over half a century old, of a conversational machine.

Appendix A

The Training Set

Table A.1 shows the 60-question training set used to perform the experiments described in Chapter 6. These 60 questions were drawn randomly from the test collection discussed in Chapter 5, which is comprised of 120 questions selected from the TREC 2002 question set.

No.	Question
1398	What year was Alaska purchased?
1409	Which vintage rock and roll singer was known as "The Killer"?
1410	What lays blue eggs?
1412	Who is the governor of Colorado?
1413	What river is called "China's Sorrow"?
1414	What was the length of the Wright brothers' first flight?
1417	Who was the first person to run the mile in less than four minutes?
1419	What year did Alaska become a state?
1425	What is the population of Maryland?
1429	What was Andrew Jackson's wife's name?
1431	Who starred in "The Poseidon Adventure"?
1432	Where is Devil's Tower?
1433	What is the height of the tallest redwood?
1434	What site did Lindbergh begin his flight from in 1927?
1435	What nation is home to the Kaaba?

1436 What was the name of Stonewall Jackson's horse?

1438 What body of water does the Colorado River flow into?

1439 How deep is Crater Lake?

1440 Who was the lead singer for the Commodores?

1446 How did Mahatma Gandhi die?

1453 Where was the first J.C. Penney store opened?

1456 What is the Keystone State?

1459 What is one national park in Indiana?

1460 What was the name of the dog in the Thin Man movies?

1462 Where is the oldest synagogue in the United States?

1463 What is the North Korean national anthem?

1464 Who is the detective on "Diagnosis Murder"?

1469 When did Alexandra Graham Bell invent the telephone? *(sic.)*

1470 When did president Herbert Hoover die?

1472 How do you say "house" in Spanish?

1473 When was Lyndon B. Johnson born?

1474 What is the lowest point on earth?

1475 Who was the first person to reach the south pole?

1477 What are wavelengths measured in?

1480 What is the principle port in Ecuador?

1481 What is the capital city of Algeria?

1483 Where is the highest point on earth?

1484 What college did Allen Iverson attend?

1485 What is slang for a "five dollar bill"?

1487 How much are tickets to Disney World?

1490 What is the Boston Strangler's name?

1493 When was Davy Crockett born?

1494 Who wrote "East is east, west is west and never the twain shall meet"?

1497 What was the original name before "The Star Spangled Banner"?

1501 How much of U.S. power is from nuclear energy?

1508 What was Dale Evans' horse's name?

1510 Where is Anne Frank's diary?

1512	What is the age of our solar system?
1514	What is Canada's most populous city?
1515	What was Dr. Seuss' real name?
1517	What is the state bird of Alaska?
1519	Where was Hans Christian Andersen born?
1521	What year did Ellis Island open its doors to immigrants?
1522	What are the headpieces called that the Saudi Arabians wear?
1527	When did the 6-day war begin?
1533	Who directed the film "Fail Safe"?
1536	What city is Lake Washington by?
1537	How many electoral college votes in Tennessee? *(sic.)*
1538	Who is the evil H.R. Director in "Dilbert"?
1547	What is the atomic number of uranium?

Table A.1: *Training set questions*

Appendix B

Evaluation Data

This Appendix presents in unabridged form the results of the experiments described in Section 6.4. There were two term expansion functions under consideration, the inflectional and the derivational. A query expansion algorithm could use neither, only one or both of the expansion functions. There were three term-dropping strategies available for study, designated the Red, Green and Blue Dropping Strategies. I ran experiments for five different values for the limit on the number of documents retrieved by the document retrieval engine. Those values were 100, 250, 500, 750 and 1000 documents.

In total, sixty experiments were run, one for each combination of document limit, dropping strategy and term expansion. For each experiment, I report recall, mean reciprocal rank (MRR) and total reciprocal rank (TRR), for both relevant documents and relevant or unsupported documents.

B.1 Raw Data

The five tables in this section, Tables B.1, B.2, B.3, B.4 and B.5 show the raw results of the experiments performed. There is one table per document limit value. For each combination of dropping strategy and expansion, I report recall, mean reciprocal rank (MRR) and total reciprocal rank (TRR), for both relevant documents and relevant or unsupported documents.

Dropping		Recall		MRR		TRR	
Strategy	Expansion	Rel.	Both	Rel.	Both	Rel.	Both
Red	none	0.2589	0.2451	0.1317	0.1314	36.4801	38.0919
	infl.	0.2766	0.2612	0.1157	0.1146	34.2612	35.4116
	deriv.	0.2589	0.2451	0.1339	0.1335	37.0902	38.7020
	both	0.2766	0.2612	0.1166	0.1154	34.5013	35.6518
Green	none	0.2729	0.2587	0.1313	0.1306	38.3278	39.9552
	infl.	0.2907	0.2747	0.1156	0.1142	35.9458	37.4706
	deriv.	0.2729	0.2587	0.1333	0.1326	38.9379	40.5653
	both	0.2907	0.2747	0.1164	0.1149	36.1860	37.3474
Blue	none	0.2720	0.2747	0.1304	0.1255	37.9606	40.8018
	infl.	0.2748	0.2671	0.1199	0.1186	35.2527	37.4706
	deriv.	0.2738	0.2764	0.1322	0.1272	38.7450	41.5862
	both	0.2766	0.2688	0.1205	0.1191	35.6698	37.8876

Table B.1: *Raw data for a limit of 100 documents.*

Dropping		Recall		MRR		TRR	
Strategy	Expansion	Rel.	Both	Rel.	Both	Rel.	Both
Red	none	0.3664	0.3508	0.0967	0.0953	37.8939	39.5648
	infl.	0.3710	0.3550	0.0906	0.0888	35.9689	37.2797
	deriv.	0.3383	0.3263	0.1060	0.1037	38.3651	40.0473
	both	0.3607	0.3466	0.0940	0.0917	36.2695	37.5831
Green	none	0.3738	0.3584	0.0976	0.0961	39.0557	40.7423
	infl.	0.3776	0.3618	0.0919	0.0898	37.1116	38.4334
	deriv.	0.3458	0.3339	0.1068	0.1043	39.5269	41.2148
	both	0.3673	0.3534	0.0952	0.0927	37.4122	38.7368
Blue	none	0.3850	0.3905	0.0947	0.0910	39.0276	42.0248
	infl.	0.3701	0.3727	0.0919	0.0882	37.1116	38.4334
	deriv.	0.3589	0.3668	0.1033	0.0983	39.6733	42.6573
	both	0.3617	0.3652	0.0953	0.0911	36.8713	39.3579

Table B.2: *Raw data for a limit of 250 documents.*

Dropping		Recall		MRR		TRR	
Strategy	Expansion	Rel.	Both	Rel.	Both	Rel.	Both
Red	none	0.5308	0.5038	0.0676	0.0673	38.4100	40.0955
	infl.	0.5467	0.5173	0.0624	0.0618	36.5233	37.8462
	deriv.	0.5271	0.4998	0.0690	0.0687	38.9440	40.6241
	both	0.5374	0.5089	0.0640	0.0633	36.7923	38.1138
Green	none	0.5393	0.5123	0.0686	0.0681	39.5758	41.2769
	infl.	0.5551	0.5258	0.0634	0.0627	37.6711	39.0050
	deriv.	0.5355	0.5080	0.0700	0.0696	40.1097	41.8055
	both	0.5458	0.5173	0.0650	0.0642	37.9401	39.2727
Blue	none	0.5682	0.5587	0.0651	0.0645	39.6010	42.6073
	infl.	0.5766	0.5689	0.0600	0.0588	37.0433	39.5760
	deriv.	0.5664	0.5571	0.0665	0.0657	40.3097	43.3029
	both	0.5692	0.5621	0.0616	0.0602	37.4888	40.0121

Table B.3: *Raw data for a limit of 500 documents.*

Dropping		Recall		MRR		TRR	
Strategy	Expansion	Rel.	Both	Rel.	Both	Rel.	Both
Red	none	0.5897	0.5604	0.0610	0.0606	38.5166	40.2098
	infl.	0.6028	0.5731	0.0568	0.0560	36.6256	37.9593
	deriv.	0.5888	0.5596	0.0620	0.0615	39.0548	40.7439
	both	0.6000	0.5706	0.0575	0.0567	36.9071	38.2388
Green	none	0.5981	0.5689	0.0620	0.0156	39.6824	41.3912
	infl.	0.6112	0.5816	0.0578	0.0569	37.7735	39.1182
	deriv.	0.5920	0.5680	0.0629	0.0624	40.2205	41.9253
	both	0.6084	0.5790	0.0584	0.0575	38.0549	39.3977
Blue	none	0.6280	0.6145	0.0591	0.0588	39.7121	42.7221
	infl.	0.6336	0.6221	0.0548	0.0539	37.1475	39.6940
	deriv.	0.6299	0.6162	0.0600	0.0596	40.4264	43.4233
	both	0.6336	0.6221	0.0555	0.0545	37.6069	40.1339

Table B.4: *Raw data for a limit of 750 documents.*

Dropping Strategy	Expansion	Recall		MRR		TRR	
		Rel.	Both	Rel.	Both	Rel.	Both
Red	none	0.6065	0.5790	0.0594	0.0587	38.5381	40.2356
	infl.	0.6168	0.5875	0.0555	0.0546	36.6435	37.9793
	deriv.	0.6075	0.5799	0.0601	0.0594	39.0788	40.7722
	both	0.6178	0.5866	0.0559	0.0551	36.9300	38.2618
Green	none	0.6196	0.5917	0.0599	0.0592	39.7097	41.4228
	infl.	0.6290	0.5993	0.0562	0.0552	37.7955	39.1423
	deriv.	0.6205	0.5926	0.0606	0.0599	40.2504	41.9594
	both	0.6299	0.5985	0.0565	0.0557	38.0820	39.4247
Blue	none	0.6514	0.6365	0.0570	0.0568	39.7419	42.7530
	infl.	0.6589	0.6450	0.0527	0.0520	37.1781	39.7145
	deriv.	0.6533	0.6382	0.0579	0.0576	40.4562	43.4541
	both	0.6607	0.6467	0.0532	0.0525	37.6401	40.1670

Table B.5: *Raw data for a limit of 1000 documents.*

B.2 Recall Matrices

Tables B.6, B.7, B.8, B.9 and B.10 show the experimental results arranged in a matrix, in which rows represent different dropping strategies and columns represent different types of expansion. Each cell in the matrix shows the recall of relevant documents and gives percent change with respect to the upper-left cell. The best combination, defined as the cell with the highest percent change, is in bold face. There is one matrix per document limit value.

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	0.2589	0.2766 (+6.84%)	0.2589 (+0.00%)	0.2766 (+6.84%)
Green	0.2729 (+5.41%)	**0.2907 (+12.28%)**	0.2729 (+5.41%)	**0.2907 (+12.28%)**
Blue	0.2720 (+5.06%)	0.2748 (+6.14%)	0.2738 (+5.76%)	0.2766 (+6.84%)

Table B.6: *Recall Dropping–Expansion Matrix for a limit of 100 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	0.3664	0.3710 (+1.81%)	0.3383 (-7.16%)	0.3607 (-1.02%)
Green	0.3738 (+2.58%)	0.3776 (+3.62%)	0.3458 (-5.10%)	0.3673 (-0.79%)
Blue	**0.3850 (+5.65%)**	0.3701 (+1.56%)	0.3589 (-1.51%)	0.3617 (-0.74%)

Table B.7: *Recall Dropping–Expansion Matrix for a limit of 250 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	0.5308	0.5467 (+3.00%)	0.5271 (-0.70%)	0.5374 (+1.24%)
Green	0.5393 (+1.60%)	0.5551 (+4.58%)	0.5355 (+0.89%)	0.5458 (+2.83%)
Blue	0.5682 (+7.05%)	**0.5766 (+8.63%)**	0.5664 (+6.71%)	0.5692 (+7.23%)

Table B.8: *Recall Dropping–Expansion Matrix for a limit of 500 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	0.5897	0.6028 (+2.22%)	0.5888 (-0.15%)	0.6000 (+1.75%)
Green	0.5981 (+1.42%)	0.6112 (+3.65%)	0.5920 (+0.39%)	0.6299 (+6.82%)
Blue	0.6280 (+6.50%)	**0.6336 (+7.44%)**	0.6299 (+6.82%)	**0.6336 (+7.44%)**

Table B.9: *Recall Dropping–Expansion Matrix for a limit of 750 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	0.6065	0.6168 (+1.70%)	0.6075 (+0.17%)	0.6178 (+1.86%)
Green	0.6196 (+2.16%)	0.6290 (+3.71%)	0.6205 (+2.31%)	0.6299 (+3.86%)
Blue	0.6514 (+7.40%)	0.6589 (+8.64%)	0.6533 (+7.70%)	**0.6607 (+8.94%)**

Table B.10: *Recall Dropping–Expansion Matrix for a limit of 1000 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

B.3 TRR Matrices

Tables B.11, B.12, B.13, B.14 and B.15 are similar to those in the last section, except that each cell shows TRR of relevant documents instead of recall. Again, percent change is shown with respect to the upper-left cell, and the cell showing the greatest percent increase in TRR is given in bold face. There is a matrix for each value of the document limit.

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	36.4801	34.2612 (-6.08%)	37.0902 (+1.67%)	34.5013 (-5.42%)
Green	38.3278 (+5.06%)	35.9458 (-1.46%)	**38.9379 (+6.74%)**	36.1860 (-0.81%)
Blue	37.9606 (+4.06%)	35.2527 (-3.36%)	38.7450 (+6.21%)	35.6698 (-2.22%)

Table B.11: *TRR Dropping–Expansion Matrix for a limit of 100 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	37.8939	35.9689 (-5.08%)	38.3651 (+1.24%)	36.2695 (-4.29%)
Green	39.0557 (+3.07%)	37.1116 (-2.06%)	39.5269 (+4.31%)	37.4122 (-1.27%)
Blue	39.0276 (+2.99%)	37.1116 (-2.06%)	**39.6733 (+4.70%)**	36.8713 (-2.70%)

Table B.12: *TRR Dropping–Expansion Matrix for a limit of 250 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	38.4100	36.5233 (-4.91%)	38.9440 (+1.39%)	36.7923 (-4.21%)
Green	39.5758 (+3.04%)	37.6711 (-1.92%)	40.1097 (+4.43%)	37.9401 (-1.22%)
Blue	39.6010 (+3.10%)	37.0433 (-3.56%)	**40.3097 (+4.95%)**	37.4888 (-2.40%)

Table B.13: *TRR Dropping–Expansion Matrix for a limit of 500 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	38.5166	36.6256 (-4.91%)	39.0548 (+1.37%)	36.9071 (-4.18%)
Green	39.6824 (+3.03%)	37.7735 (-1.93%)	40.2205 (+4.42%)	38.0820 (-1.13%)
Blue	39.7121 (+3.10%)	37.1475 (-3.56%)	**40.4264 (+4.54%)**	37.6069 (-2.36%)

Table B.14: *TRR Dropping–Expansion Matrix for a limit of 750 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Dropping Strategy	Term Expansion			
	None	Inflection	Derivation	Both
Red	38.5381	36.6435 (-4.92%)	39.0788 (+1.40%)	36.9300 (-4.17%)
Green	39.7097 (+3.04%)	37.7955 (-1.93%)	40.2504 (+4.44%)	38.0820 (-1.18%)
Blue	39.7121 (+3.05%)	37.1781 (-3.53%)	**40.4562 (+5.00%)**	37.6401 (-2.33%)

Table B.15: *TRR Dropping–Expansion Matrix for a limit of 1000 documents. Percent change is computed with respect to the Red Dropping Strategy with no expansion.*

Bibliography

[1] Daniel M. Bikel, Richard L. Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1–3):211–231, 1999.

[2] Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. Gus, a frame-driven dialog system. *Artificial Intelligence*, 8(2):155–173, 1977.

[3] Eric Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL 1993)*, 1993.

[4] J. Burger, C. Cardie, V. Chaudhri, R. Gaiz aus kas, S. Ha raba giu, D. Israel, C. Jac que min, C. Y. Lin, S. Mai oȓano, G. Miller, D. Mol do van, B. Ogden, J. Prager, E. Riloff, A. Singhal, R. Shrihari, T. Strzalkowski, E. Voorhees, and R. Weishedel. Issues, tasks and program structures to roadmap research in question & answering (q&a). http://www-nlpir.nist.gov/projects/duc/papers/qa.Roadmap-paper_v2.doc, 2001.

[5] James P. Callan, W. Bruce Croft, and Stephen M. Hardin. The INQUERY retrieval system. In *Proceedings of the 3rd International Conference on Database and Expert System Applications*, 1992.

[6] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37–46, 1960.

[7] Michael G. Dyer. *In-depth understanding*. MIT Press, Cambridge, MA, 1983.

[8] L. E. S. Green, Berkeley E. C, and C. Gotlieb. Conversation with a computer. *Computers and Automation*, 8(10):9–11, 1959.

[9] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question answerer. In E. A. Figenbaum and J. Feldman, editors, *Computers and Thought*, pages 207–216. McGraw-Hill, New York, 1963.

[10] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: The view from here. *Journal of Natural Language Engineering, Special Issue on Question Answering*, Fall–Winter 2001.

[11] Eduard Hovy, Ulf Hermjakob, and Chin-Yew Lin. The use of external knowledge in factoid QA. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, 2001.

[12] Abraham Ittycheriah, Martin Franz, and Salim Roukos. IBM's statistical question answering system—TREC-10. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, 2001.

[13] Abraham Ittycheriah, Martin Franz, Wei-Jing Zhu, and Adwait Ratnaparkhi. IBM's statistical question answering system. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*, 2000.

[14] Boris Katz. Using english for indexing and retrieving. AI Memo 1096, MIT Artificial Intelligence Laboratory, 1988.

[15] Boris Katz. Annotating the World Wide Web using natural language. In *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO 1997)*, 1997.

[16] Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton Jerome McFarland, and Baris Temelkuran. Omnibase: Uniform access to heterogeneous data for question answering. In *Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems (NLDB 2002)*, 2002.

[17] Boris Katz and Jimmy Lin. Annotating the Semantic Web using natural language. In *Proceedings of the 2nd Workshop on NLP and XML (NLPXML 2002) at COLING 2002*, 2002.

[18] Boris Katz, Jimmy Lin, Daniel Loreto, Wesley Hildebrandt, Matthew Bilotti, Sue Felshin, Aaron Fernandes, Gregory Marton, and Federico Mora. Integrating Web-based and corpus-based techniques for question answering. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2003.

[19] Wendy G. Lehnert, Michael G. Dyer, Peter N. Johnson, C. J. Yang, and Steve Harley. Boris–an experiment in in-depth understanding of narratives. *Artificial Intelligence*, 20(1):15–62, 1983.

[20] Jimmy Lin. The Web as a resource for question answering: Perspectives and challenges. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*, 2002.

[21] Jimmy Lin, Aaron Fernandes, Boris Katz, Gregory Marton, and Stefanie Tellex. Extracting answers from the Web using knowledge annotation and knowledge mining techniques. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, 2002.

[22] Gregory A. Marton. Sepia: Semantic parsing for named entities. Master's thesis, Massachusetts Institute of Technology, 2003.

[23] George Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):49–51, 1995.

[24] Dan Moldovan, Marius Paşca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, 2002.

[25] E. Nyberg, T. Mitamura, J. Carbonell, J. Callan, K. Collins-Thompson, K. Czuba, M. Duggan, L. Hiyakumoto, N. Hu, Y. Huang, J. Ko, L. Lita,

S. Murtagh, V. Pedro, and D. Svoboda. The javelin question-answering system at trec 2003: A multi-strategy approach with dynamic planning. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2003.

[26] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[27] Roger C. Schank, Neil M. Goldman, Charles J. Rieger, III, and Christopher K. Riesbeck. Inference and paraphrase by computer. *J. ACM*, 22(3):309–328, 1975.

[28] Robert F. Simmons. Answering english questions by computer: a survey. *Commun. ACM*, 8(1):53–70, 1965.

[29] Robert F. Simmons. Natural language question-answering systems: 1969. *Commun. ACM*, 13(1):15–30, 1970.

[30] Stefanie Tellex. Pauchok: A modular framework for question answering. Master's thesis, Massachusetts Institute of Technology, 2003.

[31] Stefanie Tellex, Boris Katz, Jimmy Lin, Gregory Marton, and Aaron Fernandes. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2003)*, 2003.

[32] A. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[33] C. J. van Rijsbergen. *Information Retireval*. Butterworths, London, 1979.

[34] Ellen M. Voorhees. Overview of the TREC 2001 question answering track. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*, 2001.

[35] Ellen M. Voorhees. The evaluation of question answering systems: Lessons learned from the TREC QA track. In *Proceedings of the Question Answering: Strategy and Resources Workshop at LREC-2002*, 2002.

[36] Ellen M. Voorhees. Overview of the TREC 2002 question answering track. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, 2002.

[37] Ellen M. Voorhees. Overview of the TREC 2003 question answering track. In *Proceedings of the Twelfth Text REtrieval Conference (TREC 2003)*, 2003.

[38] Ellen M. Voorhees and Dawn M. Tice. Overview of the TREC-9 question answering track. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*, 2000.

[39] Joseph Weizenbaum. Eliza: A computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, 1966.

[40] Terry Winograd. *Understanding Natural Language*. Academic Press, New York, New York, 1972.

[41] William A. Woods, R. M. Kaplan, and B. L. Nash-Webber. The lunar sciences natural language information system. Final Report 2378, BBN, Cambridge, MA, 1972.