IMPROVING INFORMATION RETRIEVAL WITH NATURAL LANGUAGE PROCESSING

BY

BENJAMIN E. LAMBERT

B.S., University of Massachusetts Amherst, 2003

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

The goal of this research is to transform a text corpus into a database rich in linguistic information and robust with multifarious applications. Computational natural language processing liberates the syntax and semantics of natural language to convert a plain text corpus into rich database that stores these implicit linguistic components. Information retrieval techniques index into the otherwise implicit data that is encoded by the language and its syntax. We use Lemur and the Indri query language to index this linguistic information. The resulting index supports numerous applications that can utilize the additional linguistic information. For one application, searching a document corpus with natural language queries, we show syntactic and semantic information that improve retrieval performance.

Dedicated to Yehudah Thunket

Acknowledgments

Thanks to Richard Goldwater, ChengXiang Zhai, and Andrew McCallum for their support. Thanks also to Trevor Strohman and ChengXiang Zhai for their help with the Indri Information Retrieval Engine.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

IR.....Information Retrieval

NLP ..... Natural Language Processing

POS ..... Part-of-speech

NE .....Named entity

Chapter 1

Introduction

The complexities of human language make searching for information encoded in natural language far from simple. The meaning, whether subtle or obvious to the reader, may be implicit in the language. Keyword search limits users to locating information specified explicitly at the surface level. Processing natural language text corpora helps to liberate the implicit syntax and semantics. Liberating this linguistic information allows one to transform a plain text corpus into a database rich with linguistic information. This database allows searchers to explicitly search for information that is encoded implicitly in the language. NLP can be further used to process queries and use information implicit in natural language *queries* to locate relevant documents in a natural language database.

Most contemporary search systems only support searching for words or phrases explicitly contained in a text (and possibly explicit metadata about the text such as subject headings), without respect to the syntax, semantics, or pragmatics of the terms in the original language. Searchers may become aware of the limitations of keyword search when their original query fails and they begin to search for synonyms of their original search terms or look to their own intuitions about how linguistic characteristics manifest themselves in relevant documents (for example a profession researchers such as librarians might ask themselves, "would the author be writing this query term in present or past tense?").

The problem is that the user and computer have different representations of the information. The user may have some notion of what the desired information will look like and project this notion into query terms that the user believes may be present in relevant documents. However, the user may not be

able to estimate this correctly; the relevant documents may be written in a very different voice from that which the searcher uses. The user's search terms are estimated by the user and so are subjectively influenced by the user's point-of-view and experience.

Another user searching for the same information (who perhaps has more knowledge about the field) may have a better idea of the context in which the information will appear and consequently be better able to choose search terms (for example field-specific terminology). The two users may search be searching for semantically equivalent information but on the surface the two form very different queries neither of which may match the actual relevant documents.

Techniques like latent semantic indexing (LSI) may be used to find relevant documents that do not explicitly use the query terms. However, while this may improve retrieval performance in some cases by generalizing the query, at the same time it loses the ability to make very precise queries. Ideally, we would like to model the semantics of the documents and queries so that we can generalize to other terms but not lose the original semantics of the query.

Most information retrieval and search engines do not model the linguistic semantics of documents. Most also do not model the syntax of the language. On the other hand, if they do model the syntax it is limited to the simplest syntax: word order.

The farthest extreme of modeling semantics would be to re-encode a corpus into a machine-interpretable form (such as logic) and store the corpus in a knowledge base (KB). One difficulty with this solution is that the natural language sentences are often not easily translated to a logical form. Another difficulty with this is that it almost necessitates a structured query language, because the user must be able to match predicates and relations in the KB.

This research does not attempt to dig so deeply into the semantics, but attempts to use some of the linguistic syntax and semantics to aid with information retrieval. Rather than construct an omniscient KB, we use existing statistical NLP tools to extract syntactic and semantic information from text.

If the goal of information retrieval is to match the semantics of a user query to the semantics of a text documents, keyword search falls well short of the goal. Keyword search incorporates neither syntax nor semantics into the search heuristics. This research attempts to make progress toward a more linguistically and semantically driven search engine. This research combines syntactic and semantic analysis with information retrieval.

This research incorporates systematic natural language processing (NLP) with information retrieval via existing inference network technologies. This takes the information retrieval to the point where the semantics of the natural language corpus can be accessed by a query language as though it were a database. The value of accessing information from a natural language corpus in this way is demonstrated by its generality, allowing multifarious applications. However, it does not require an average user to learn and use a complex query language. In fact, we show how effective structured queries can be automatically formulated from natural language queries and how these queries perform better than the same queries without linguistic annotations.

Chapter 2

Information Retrieval and Natural Language Processing

In this chapter, we introduce information retrieval and natural language processing to give the reader some background on these two subjects which are central to this research.

2.1 Information Retrieval

The information retrieval (IR) model we use is an example of *ad hoc* retrieval. This model is called *ad hoc* because users have information needs and to satisfy an information need they form a query *ad hoc*. Once the information need is satisfied, they throw out the query and do not use it again. The query is formulated *ad hoc* to help satisfy the information need. This is opposed to *text filtering* IR techniques where the IR model retains a representation of the information need and over time select documents to present to a user. However, the techniques presented in this research are sufficiently general that it could be applied to either *ad hoc* or *text filtering* IR.

In the standard IR model both the information need and the documents must first be translated into some alternate representation. Figure #1 shows the first step of the basic IR process where documents and information needs are translated into a new representation according to Prof. Jamie Callan's lecture notes from his Information Retrieval class taught at Carnegie Mellon University (*Retrieval Models* 2006).

In most cases, the information need is translated by a user into a *query*. Queries are often keyword queries sometimes with Boolean or phrase operators.

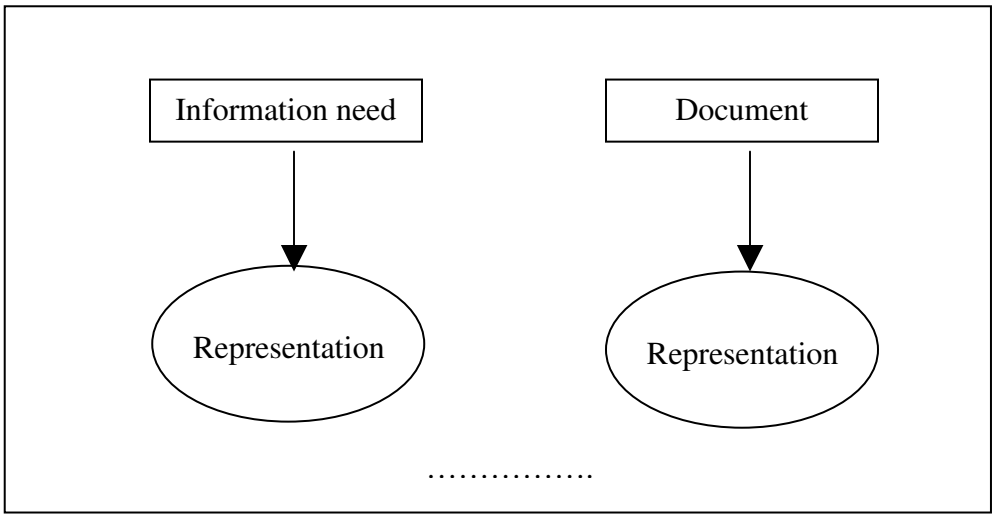

Figure 1 - The first steps of the Basic IR process (Callan 2006)

The document is often represented as a "bag of words" or an "ordered bag of words" for IR with phrase operators. Some more expressive document representations include support for structured documents that may have fields such as *title*, *abstract*, *body*, etc.

Some IR models extend the representation of the information need beyond the original keywords. This can be done by augmenting the original query terms with additional terms added found in a variety of ways such as query expansion, relevance feedback, or machine-readable thesauri (see Baeza-Yates 1999 Ch. 5 for more details). The representation used in this research retains the original query terms but augments them with linguistic tags. The text representation is augmented similarly.

New methods of text representation often change the way documents and terms are weighted and how to map document terms into new vector spaces. For example, latent semantic indexing performs a change of basis in the vector space where the original dimensions correspond to documents and the terms in them. In this research, we treat the text representation as the original text (with some stopping and stemming) augmented with linguistic tags.

This research differs from other approaches to IR in how we represent documents and queries. The information need is represented as a *natural language* query. It is often a one-sentence description of the information need.

We leverage the fact that the information need is described in natural language so that the order and structure of the description can be used to aid retrieval. What is different about our representation is that we explicitly represent the linguistic data that is automatically extracted from the natural language query. So, our information need is expressed as a natural language sentence with explicit linguistic information (such as parts-of-speech, named-entity types, etc.)

Similar to the query representation, the document representation explicitly includes automatically extracted linguistic information. Transforming a plain text document into our representation means passing it through automatic natural language processing components and annotating the documents with the linguistic information. The resulting representation is far more than an ordered bag of words. The result includes linguistic tags on individual words as well as small windows of linguistic spans such as *sentence*, *phrase*, *named-entity* that can be used in the retrieval model. An example of what this representation buys us is the ability to search for the terms *cheap* and *jewelry* in the same NP (noun phrase). This will allow us to search for sentences where *cheap* modifies *jewelry* but not other parts of the sentence. This query would match both "cheap jewelry" and "cheap sterling silver jewelry" but not "…expensive jewelry given to a cheap person…" The distinction between these two cannot be represented by proximity operators alone since proximity operators can easily cross phrase boundaries.

Some members of the IR community would argue that this is a wasteful text representation. For one thing, it requires significant additional storage to store the linguistic information. Another drawback to this text representation is that it is time-consuming to process a corpus and extract all that linguistic information. It is true that the NLP is time-consuming because every sentence in every document must be processed by several annotators. Some annotators, such as syntactic parsers, could take approximately one second per sentence. If the indexer only indexes the raw terms, it may be able to process up to 40 gigabytes per hour. Annotating every sentence requires much more time. On the other hand, data only needs to be processed once, so the time to index may not be an important factor if the corpus is relatively static. The argument in favor of

spending the time to annotate each sentence is that we get better retrieval performance and can generate queries that are more expressive.

One could argue whether as to whether we are justified in indexing all this additional linguistic data. The argument could be made that most of the necessary information is on the surface. However, the field of linguistics studies the intricacies involved in translating from some mental representation to a surface representation. Humans when reading written language need time to read and understand the text, so it may be necessary for the computer to slow down to "read" the text to gain a better "understanding" of what the text really says.

2.2 Natural Language Processing

Much of the work that has been done in natural language processing (NLP) has not been applied to IR. The extent that most search engines use NLP is to match morphological variants of query terms. In this section, we discuss some of the NLP techniques we apply to IR. These include syntactic parsing, part-of-speech (POS) tagging, named entity recognition, and others.

NLP algorithms automatically extract information contained implicitly in the text. Before discussing the application of these NLP techniques to information retrieval, it is important to understand the state-of-the-art NLP research and how much information we can hope to extract automatically from language in text format.

Much NLP research has been to identify syntactic features in language. Parsing, chunking, POS tagging, lexical cohesion are all examples of syntactic analysis. Semantic analysis, such as semantic role labeling, has generally not achieved the same degree of success as syntactic analysis. Since it seems that humans need background knowledge to comprehend the meaning of a sentence, it should not be surprising. Some researchers in the field see syntactic NLP as "low hanging fruit" which has been the primary focus of NLP researchers at the expense of research in semantic NLP.

This section provides an overview of the NLP techniques that have been used in this research. These include both syntactic NLP such as parsing, shallow

parsing, part-of-speech tagging, as well as semantic NLP such as named-entity tagging and semantic role labeling.

2.2.1 Syntactic NLP

Part-of-Speech Tagging

Part-of-speech may be one of the most basic forms of syntax. All languages have parts-of-speech, although the word classes vary from language to language (Shopen 1985). The reader is likely familiar with broadest part-of-speech word classes such as *verb*, *adjective*, *noun*, etc. These word classes can be further broken down many times over (such as into plural nouns, infinitive verbs, etc.). In English, there is no canonical set of word classes. Linguistic typologists generally formulate tests to judge what part-of-speech a particular word is. However, even these linguistic tests can discover conflicting evidence for several parts-of-speech. For example, the English word *many* shows the behavior of *determiner, predeterminer,* and *adjective*.

A rather large set of word classes is used by the Penn Treebank project (shown in Figure #2). These include non-alphabetic classes to encompass all tokens including punctuation. The Penn Treebank project provides an extensive manual of these parts-of-speech and how manual annotators are to determine a POS in context (Treebank 2002).

In this research, a POS classifier is used to tag the corpus with POS tags. This POS tagger is based on the SNoW learning architecture (Roth 1998).

# Pound sign	NNP Proper singular noun
$ Dollar sign	NNPS Proper plural noun
" Close double quote	PDT Predeterminer
`` Open double quote	POS Possessive ending
' Close single quote	PRP Personal pronoun
` Open single quote	PP$ Possessive pronoun
, Comma	RB Adverb
. Final punctuation	RBR Comparative adverb
: Colon, semi-colon	RBS Superlative Adverb
LBR Left bracket	RP Particle
RBR Right bracket	SYM Symbol
CC Coordinating conjunction	TO to
CD Cardinal number	UH Interjection
DT Determiner	VB Verb, base form
EX Existential there	VBD Verb, past tense
FW Foreign word	VBG Verb, gerund/present participle
IN Preposition	VBN Verb, past participle
JJ Adjective	VBP Verb, non 3rd ps. sing. present
JJR Comparative adjective	VBZ Verb, 3rd ps. sing. present
JJS Superlative adjective	WDT wh-determiner
LS List Item Marker	WP wh-pronoun
MD Modal	WP$ Possessive wh-pronoun
NN Singular noun	WRB wh-adverb
NNS Plural noun	

Figure 2 - Penn Treebank parts-of-speech

Syntactic parsing

Whereas parts-of-speech may be the most basic level of syntax, a full syntactic parse contains the most syntactic information. The high information content in a syntactic parse tree may be overwhelming for many computer applications, including information retrieval. In a full-parse, each word is a member of numerous constituent structures which are not readily collapsible into a single compact description of that constituent structure. Figure #3 shows one way of describing syntax and grammatical roles in a compact form. These are syntactic parse tags used in PhraseNet (Tu 2003).

NOFUNC	NP	NPSBJ	VPS	PP
ADVP	VP	ADJPPRD	NPPRD	VPSSBAR
NPTMP	ADVPTMP	VPSTPC	VPSNOM	NPLGS
SBARADV	ADJP	NPADV	VPSADV	VPSINV
VPSPRP	ADVPMNR	SBARTMP	PPPRP	
PPLOCCLR	SBARPRP		PPPRD	ADVPCLR
VPSPRN	VPSCLR	NPLOC	ADVPLOC	ADVPDIR
PPDTV	ADVPPRD	WHNP		CONJP
NPHLN	VPSQ	VPSNOMSBJ	SBARPRD	VPSPRD
NPCLR	PPPUT	NPTTL	ADJPPRDS	NPTMPCLR
	INTJ		PPTMPCLR	PPCLR

Figure 3 - Syntactic parse tags

A word in a parse tree can be described by the depth, the words in the same constituent structure, the head word of that structure, the type of constituent structure, and other information about the structure that it modifies or attaches to. Thus, for information retrieval it is important that we choose only features that can be used effectively.

One way of using all the information in a parse tree for information retrieval would be to treat each constituent structure as a section of the document. This would allow the user to search for terms that occur in any specified type of constituent (e.g. nouns phrase or s-bar) and search for terms that co-occur in the same constituent structures. One could imagine a document scoring function that weights more highly query terms that co-occur in smaller constituent structures (this would be similar to a scoring function based on term proximity but would be proximity within the parse tree). For example, given the parse:

$((\text{Jack and Jill})_{NP}\ (\text{went}\ (\text{up}\ (\text{the hill})_{NP})_{PP})_{VP})_{S}$.

We could treat each phrase as a section of the document. Picture the sentence as a structured document with section and subsection. Then section *S* is the entire document, the first *NP* is section 1, *VP* section 2, *PP* section 2.1 and so on. In this way, we can directly apply methods for structured retrieval to linguistic retrieval.

Shallow parsing

We do not attempt to index an entire syntactic parse of each sentence as discussed in the previous section. Rather, as a simplification of the syntactic parse, we only consider the main constituent structures of a sentence as "chunks" of the sentence. This is in effect a "shallow" parse. Thus, a sentence may be partitioned off into a noun phrase (NP) subject, followed by a verb phrase (VP), and perhaps ending with a prepositional phrase (PP). This type of NLP may be particularly useful for IR since it breaks a sentence down into a few manageable chunks without greatly increasing the amount of data. One use of a shallow parse in IR would be to use a document scoring function to give a higher score to documents that use some of the query terms in the same phrase both in the query and in the document. The shallow parser used in this research was developed by the Cognitive Computation Group (Munoz et al. 1999).

Lexical Cohesion

Cohesive lexical units are multi-word terms that together function as a member of the lexicon. An example of this is *fire hydrant* in which the meaning is not obviously inferred from the words in the phrase alone. A certain amount of background knowledge is needed to understand the phrase. For many of these phrases it can be assumed that speakers of the language have a lexical entry for that term.

This is an aspect of NLP where it is not known precisely how NLP can help IR. Jamie Callan asks in his Information Retrieval course at Carnegie Mellon University "Does phrase indexing help?" There is no definitive answer, "it works well in some situations … it ought to work in a wider range of situations" (*Text Representation* 2006).

In previous unpublished research this author showed that using a variety of information theoretical methods to identify cohesive multi-word lexical units improves retrieval performance on the same test suite described in chapter 8 (Lambert 2004). By automatically identifying multiword lexical units in both the

corpus and queries, we increase precision by ranking higher documents that contain the multiword search terms as a unit rather than just matching documents that contain both words. The lexical units identified were not used in the experiments described in chapter 8.

2.2.2 Semantic Language Processing

Named Entity Recognition

Many NLP tasks are semantic in nature. Named-entity recognition (NER) is perhaps one of simplest methods of semantic NLP. NER is semantic in the sense that the tagger must attempt to identify the type of the referent (e.g. *person*, *organization*, or *location*). A NER may use syntactic clues to determine the entity type (e.g. nearby words such as "Mr."), but ultimately is determining some semantic information about the words. We can tell that this is semantic information because named-entities are generally nouns and any named-entity could be replaced by any other NE (that agrees in number and gender) and the sentence would remain syntactically correct. However, the sentence would probably only make sense semantically if the named-entity were replaced by a NE of the same type.

The NER used in this research was developed by the Cognitive Computation Group (*NER* 2006). See figure #3 for the named-entity tags used in this research.

Number	Medical	Food
People	NumberUnit	Religion
LocationCountry	OrganizationPoliticalBody	Journal
Date	LocationState	LocationStructure
ProfTitle	Sport	NumberZipCode
OrganizationCorp	OrganizationTeam	Book
Organization	Animal	Plant
Location	Event	LocationMountain
LangRace	DayFestival	OrgCGroup
LocationCity	OrganizationUniv	LocationRiver
Money	Color	Perform
NumberPercent	NumberTime	Art

Figure 4 - Named-entity tags

Semantic role labeling

Another type of annotator that attempts to extract semantic information directly is a semantic role labeler. Semantic role labelers identify how the various noun phrases relate to the verb in a sentence. For example, in the sentence "John threw the ball to Mary," *John* is the agent who did the *throwing*. In this example, John is also the subject of the sentence. Therefore, if we want to search for documents that talk about John throwing things, we could use our syntactic features to search for documents that contain a sentence with *John* as the *subject* and *throw* as the verb. However, in the case of passive sentences such as "The ball was thrown to Mary by John" the subject is not the agent of the verb. In this example sentence, *the ball* is the subject. By labeling the semantic roles of each sentence, we can locate passive sentences that talk about John throwing by searching for sentences that contain *John* as *agent* with the verb *throw*.

Generating structured queries from a user's input is discussed further in the chapter on results, but semantic role labels go hand-in-hand with interpreting natural language queries. For example, for "when" questions we should search for sentences that have a temporal modifier (*arg-tmp* in semantic role labeling parlance). Questions with the subject "who" (e.g. "who threw the ball?" should search for sentences with an agent (*arg0*) that have a *person* named-entity tag. Questions using the word "whom" (e.g. "to whom was the ball thrown?") should search for sentences with a patient (*arg1*) role filled by a *person* named-entity.

Semantic role labels are not used in the experiments described in chapter 5 as the corpus had not been tagged with semantic role labels at the time of this writing. The obstacle to such an evaluation is that SRL taggers are relatively slow compared to other NLP taggers and would take a long time to process an entire corpus. Questions were SRL tagged with the Cognitive Computation Group's 2005 CoNLL semantic role labeling shared task submission (Punyakanok). These question could be readily transformed into the appropriate queries were the corpus SRL tagged.

Chapter 3

Related Research

Some research with the goal improving information retrieval performance has focused on exploiting information gathered that is *meta* to the text itself. For example, citations or links to a document (Page 1998) are data gathered from external sources. Latent semantic indexing and query expansion also fall into this category because they rely on other documents in the computation.

A deeper linguistic analysis is often only performed for specialized implementations of information retrieval. Question answering systems generally rely on language processing heavily for question classification, named entity tagging, and answer extraction. Some systems such as the linguist's search engine (Resnick 2003) are specifically designed to search for linguistic phenomena. It seems that there has been limited research on using deep linguistic analysis for general information retrieval. Many researchers prefer to develop more sophisticated algorithms to work with the same surface-level data.

See chapters 2 and 3 for a discussion more broad research related IR and NLP respectively. This chapter describes research aimed at using NLP in close coordination with IR.

3.1 Linguist's search engine

The Linguist's Search Engine (Resnick 2003) developed at University of Maryland is a powerful search engine specifically designed to search for syntactic structures. Users enter English sentences that are parsed into syntactic parse trees and displayed graphically and textually. The textual parse representation is shown as the original words annotated with parts-of-speech and constituent

structures designated by embedded parentheses. This text can then be edited to form a query.

The parse of the sentence "Jack and Jill went up the hill" is:

```
(S1 (S (NP (NNP jack) (CC and) (NNP jill))
(VP (VBD went) (PRT (RP up)) (NP (DT the)
(NN hill)))))
```

If we want to search for other sentences that contain the verb *went* followed by the particle *up*. We remove the entire parse except the part designating the phrasal verb:

```
(VP (VBD went) (PRT (RP up)))
```

Similarly, we can search for the verb *look* followed by the particle *up* with the following query:

```
(VP (VBD look) (PRT (RP up)))
```

The Linguist's Search Engine is designed to search for linguistic patterns in a corpus and provides support for processing and searching portions of the Web. The Linguist's Search Engine is similar in its implementational goal to this research (to search for linguistic structures in corpora).

The Linguist's Search Engine differs from this research in that it searches only for structures in a full syntactic parse. It is also designed primarily to be used by linguists to search for individual sentences rather than entire documents. The Linguist's Search Engine parses sentences in real time while crawling a given subdomain of the Web.

3.2 Dekang Lin's Research

Dekang Lin's research achieves some of the same functionality as described here. Lin's demos (Lin 2005) are not intended to search for documents, but rather to mine corpora for linguistic dependencies and relations. Lin's software will find other words that a given word is used with in many different grammatical contexts (e.g. adjectives used with a given noun).

In one demo, entering the term *water* shows that the most frequent noun occurring immediately after *water* in the corpus is *district* (as in *water district*), the second most frequent noun is *use* (as in *water use*) and so on. We also see that the most frequent noun that comes before *water* is *_SPEC*. When *water* is used as the first conjunct of two nouns the most frequent second conjunct is *food* (as in *water and food*). When used as the second conjunct of two nouns the most frequent first conjunct is *electricity* (as in *water and electricity*). Lin's research is similar to the research described in this paper in that we a querying a corpus for linguistic phenomena, it differs in that we extract as the results (co-occurring words vs. relevant documents).

3.3 The Sketch Engine

The Sketch Engine (Kilgarriff), developed by Adam Kilgarriff and Pavel Rychly at Lexical Computing Ltd., is a "new Corpus Query System incorporating *word sketches, grammatical relations*, and a *distributional thesaurus*. A word sketch is a one-page, automatic, corpus-derived summary of a word's grammatical and collocational behaviour." (Kilgarriff) For example, entering the noun *food* into the Sketch Engine we are given a "sketch" of the noun which tells us among other things: verbs it occurs as the object of (the most common being *eat*), verbs it occurs as the subject of (*contain*), nouns that modify it (*junk*), nouns it modifies (*poisoning*), and many others. The Sketch engine is very similar to the Dekang Lin's demos, except that it identifies different word relations (such as *subject of*) than Lin and uses a more interesting statistical metric to rank the words it finds.

3.4 PhraseNet

PhraseNet, developed by Yuancheng Tu, Xin Li, and Dan Roth at University of Illinois, is an example of a "context-sensitive lexical semantic knowledge system" (Tu 2006). PhraseNet disambiguates WordNet word senses based on the context in which they are found. The context in this case could consist of both the syntactic structure of the sentence (e.g. Subject-Verb as in "he ran" or Subject-Verb-Object-PrepPhr as in "he gave it to me") and words in the sentence. This

can be used to disambiguate the word sense for *fork* in "they ate a cake with a fork" since in that context the object of the preposition is usually either a utensil (e.g. spoon or fork) or a food (as in "cake with strawberries"). Thus, we know that the word sense for *fork* is *utensil* not as in *fork in the road*. In this example, disambiguating the word sense also tells you what the PP attaches to (if it is a utensil it attaches to the verb, if it is food is attaches to the object *cake*). PhraseNet is different from previous approaches in its novel use of WordNet to do word sense disambiguation.

3.5 Metadata-based Search

There is much research that shares the goal of this research of improving information retrieval. Some approaches attempt to improve retrieval performance algorithmically or with techniques like automatic query expansion. Some more recent approaches attempt to use external metadata about the documents. This is the case for the "semantic web". These systems operate by searching within the metadata associated with the documents. For these systems the documents generally must be manually annotated with metadata. The metadata is a often a concise and precise description of the content in the document. The metadata description of a document may use words that are not present in the document. Thus, a searcher who knows the correct topic but not words present in a document will be better able to find the right sources.

This research does not utilize externally provided metadata, rather it uses implicit semantic data in the language. This research could be extended to also search for metadata. Some work done by Marti Hearst shows how this metadata could be used to aide users in navigating an "information space" (e.g. a corpus). They show how users can find information more quickly and are less likely to get lost of overloaded by navigating a very large information space (English 2002; Hearst 2002; Hearst 2000).

Chapter 4

Software Architecture and Applications

4.1 Software Architecture

This software derives from the retrieval architecture of Lemur's Indri (successor of the popular Lemur IR engine) (Indri 2006). Indri is a powerful new extension to the Lemur information retrieval engine which incorporates a structured query language and inference network into the Lemur framework. Indri provides support for indexing a number of documents types including plain text, HTML, PDF, and XML. When indexing XML documents, the query language allows a user to specify in which sections of an XML document query terms should occur.

The research implementation described in this paper builds upon the components of Indri which allow tag-based searching of XML documents. While the XML format is convenient for some purposes, especially for annotating text, there is much redundant information. This redundancy is especially pronounced when XML is used for the numerous and short-spanning tags generated by NLP annotators. Instead of XML, we use a very compact *column* format. This format allows us to store very large corpora in nearly the minimum space required without recoding terms and tags. Many natural language processed corpora are represented in this *column* format as well so data is readily available.

A typical *column* format corpus has one *word* or *token* per line. Accompanying each word on the line are a number of "tags" for that word separated by tab characters and forming columns. For example, one column may contain the part-of-speech of the word in that row; another column may have the grammatical role of the word (examples are shown in chapter five).

Column-format files are often used as a standard file format. This could be because of the ease in parsing and compact representation. The Brown corpus which may be the first compiled machine-readable English corpus was altered to incorporate part-of-speech tags in 1979. This new version called "form c" represented the corpus in a column format with the words in the first column, the part of speech in the seconds column, and the location in the corpus in the third column (Francis 1979). This format continues to be used; the Conference on Natural Language Learning (CoNLL) has used this format to supply the data to teams in its yearly "shared task" (Conference 2006).

The software artifact resulting from this research, allows one to specify the meaning of each column of a column-format corpus in a corpus schema then index the corpus with the inclusion of the column information. The Indri structured query language can be used without modification to search for terms subject to any specified constraints on what the value of each column should be. For example, if one column is the word's part-of-speech, a simple Indri structured query can be formulated to search for words when the part-of-speech is a noun or a verb.

Details on the usage of this software are given in Appendix B. Further details on the implementation are given in Appendix A.

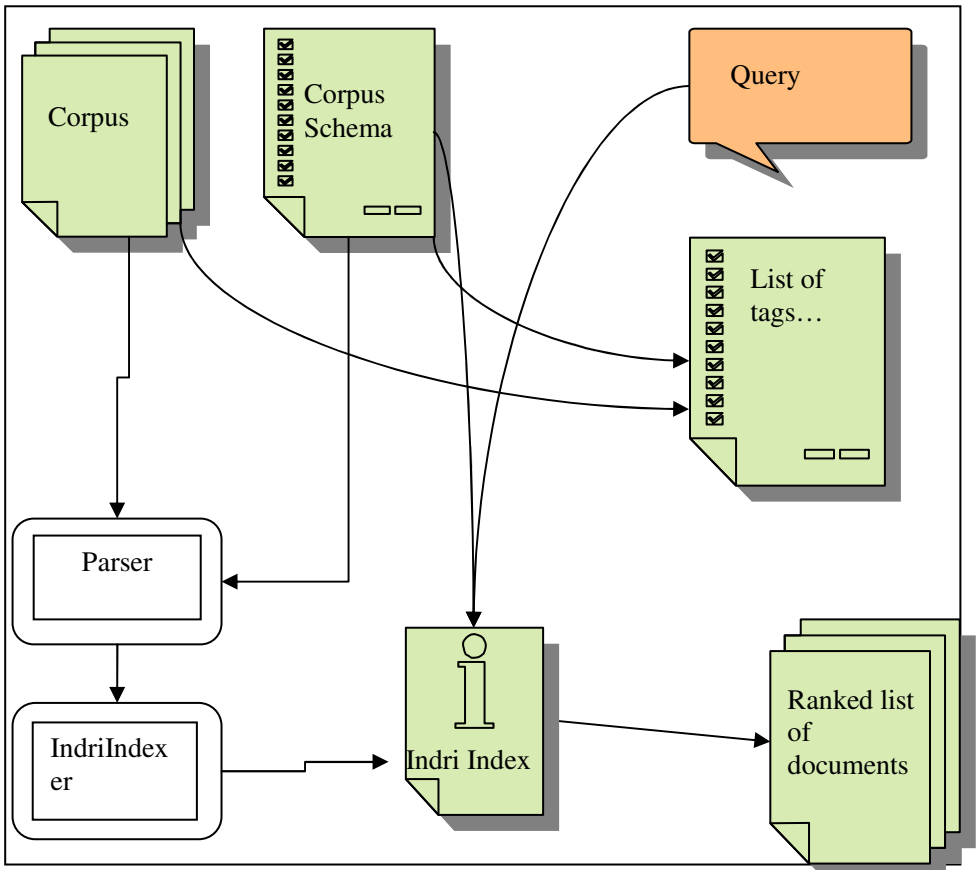

Figure 5 - Software Architecture

4.2 Applications

One of the values of this system is that it may be used as the underlying technology for many other applications. Each of the applications described in this chapter could be implemented in a straightforward manner with this system.

4.2.1 PhraseNet

PhraseNet, as described in Chapter 3, is currently implemented with a relational database. The database backend is easy to use, but does not scale well to large corpora. Indri's inference network allows for much faster searching. Whereas join operations in a relational database are very slow, the inference engine can make complex queries quickly. PhraseNet data is stored in column-format, so it would be simple to deploy. The only modifications necessary would be for the

Web forms to generate the appropriate Indri queries instead of SQL queries and some post-processing of the results to obtain the desired output.

4.2.2 iTrack

iTrack is another application that could be built on top of the research system described in this paper. iTrack is a system developed by the Cognitive Computation Group at University of Illinois (a.k.a. the MIRROR project). iTrack is intended to track named entities and to which other names entities they are associated. This can be used for entity resolution to determine whether the string "Clinton" refers to Hillary or Bill. It does this by tracking with which other named entities an entity is closely associated. We can find other person named-entities that co-occur in the same sentences as Bill Clinton with the following Indri query:

```
#combine[sentence](“Bill Clinton”.nePerson
#any:nePerson)
```

This query retrieves all documents that contain the string "Bill Clinton" that has been NE tagged as a person in the same sentence as another string that has been tagged as a person named-entity. These documents can be further processed to retrieve which people co-occur with "Bill Clinton" most frequently.

Presently, iTrack used other means to track these named entities. However, it would be straightforward to achieve this functionality with the system described in this paper.

4.2.3 Question Answering Systems

Many question answering (QA) systems may already employ some advanced retrieval functionality. QA systems that use question classification to determine the type of entity that the answer is especially benefit from being able to search for named entities. For example, if the question is a *who* question, the retrieval component of the QA system can search for documents that contain the query terms as well as other terms that are tagged as entities of type *person*.

4.2.4 Active machine learners

Active learning is a paradigm within machine learning in which the learning algorithm is able to select training examples such that it can converge to a good hypothesis more quickly. For many NLP tasks, text from reputable edited sources such as newspapers is considered positive training examples since they are "correct" sentences in the language. Such systems may benefit from an advanced language query system, as they can quickly check hypothesis sentences or sentence fragments. For example, a language generation system with low confidence as to the preposition for a particular prepositional phrase can query for all examples of that preposition used with the sentence's verb. The results (or lack thereof) may help to choose the preposition or reformulate the sentence. This use case of this system is not currently deployed but could be easily.

4.2.5 Advanced linguistic search

This application is similar to the Linguist's Search Engine (Resnick 2003), allowing a user to specify query terms and linguistic properties about them. This would allow a user to specify the POS, grammatical role, named-entity, or any other indexed linguistic tags for each term. This application would utilize the same types of queries as are used in chapter 5 except that the queries would not be constructed automatically but the linguistic properties would be specified explicitly by the user.

4.2.6 Toward a semantics-based search engine

Only a few semantic tags have been discussed thus far. The methods used in this research can be extended to include potentially even more useful tags. For example, we can label words in the corpus with their synonyms or with unambiguous concept tags (after the word-sense or entity reference has been disambiguated). It may also be possible to index relations among the concepts in the corpus and search for concepts involved in those relations.

Chapter 5

Experiments and Results

Up to this point, we have discussed the system, some NLP and IR used in the system, and possible applications for this framework. Next, we will turn to a specific application and some experimental results for this application.

We apply the framework to natural language queries for the TREC QA task. What makes this application different from other QA systems is that we use a natural language query and automatically generated NLP tags to formulate a structured query. After retrieving documents, we do not attempt to perform answer extraction from the retrieved documents. The experimental results show how retrieval performance is affected by augmenting queries with various combinations of NLP tags. The results show that part-of-speech tags improve performance more than named-entity tags.

5.1 Experimental Setup

The goal of these experiments is to determine whether transparently adding automatic NLP to corpora and queries improves retrieval performance. We tag both queries and the corpus with several types of linguistic data. The experiments compare various combinations of linguistic features used in the experiments.

For the indexing and retrieval, we use Indri 2.2 (Indri 2005). The Indri retrieval engine includes support for searching within specific fields in a document. The fields could be arbitrary spans of a document. Perhaps originally intended to describe larger sections of a document such as *title, abstract, body, sections, etc.*, they may be used to specify arbitrary regions of the document. For this research, the fields indexed are linguistic tags.

Let us begin with a simple example of how linguistic tags may be used as the fields of a document. We could take the process of shallow parsing to break sentences into smaller chunks (clauses and phrases). Chunking the sentences in a document sections it off into many small sections (several per sentence). This is useful for retrieval because the same Indri function that retrieves documents that contain certain words only in the *abstract* could be used to retrieve documents that contain certain words but only in *prepositional phrases*.

The primary linguistic tags used in these experiments are part-of-speech tags (POS) and named-entity (NE) tags. The POS tags are used to generate indexed sections of the document that are a single word long. NE tags are used to generate document sections that are one or more words long.

Indexing these linguistic fields allows us to generate queries that match certain linguistic properties of the documents in the corpus. For example, one could search for the term "state" but only when it is POS tagged as *verb*. If we know we want the verb *state* but we do not specify it, many documents that contain the noun *state* will be retrieved as well.

While the user could write structured queries constraining each term to a particular POS, this creates work for the user and the human-computer interaction becomes less seamless. However, if the user generates a query in written natural language we may be able to automatically tag the same linguistic features and use those to generate a better query. We leave manual query construction to domain experts such as linguists who want to have very precise control over the query.

Before describing the specific configurations for each experiment. Let us see an example of how the queries and corpora are represented and how those are used by Indri.

5.1.1 Document and query representation

Since we would like to do document retrieval over very large corpora and we need to annotate the corpus with additional linguistic tags, it is important that we keep the corpus representation compact. To achieve this we store the corpus in a column format. Additionally a corpus schema is used to denote what the value in

each column of the corpus corresponds to (e.g. part-of-speech tags, etc.) (see figure #6). This avoids storing redundant information and using any more syntax than necessary (e.g. as opposed to XML tags which are duplicated and surrounded by additional syntax).

```
0    0       07/04/1998 NN      O       B-NP    B-Num
0    1       07         CD      O       I-NP    B-Num
0    2       :          :       O       O       I-Num
0    3       50         CD      O       B-NP    I-Num
0    4       :          :       O       O       O
0    5       00         CD      O       B-NP    B-Num
0    6       UBS        NNP     O       I-NP    B-OrgCorp
0    7       chairman   NN      O       I-NP    O
0    8       :          :       O       O       O
0    9       pressured  JJ      O       B-NP    O
0    10      banks      NNS     O       I-NP    O
0    11      must       MD      O       B-VP    O
0    12      carefully  RB      O       I-VP    O
0    13      consider   VB      O       I-VP    O
0    14      options    NNS     O       B-NP    O
```

Figure 6 - Example sentence from the AQUAINT corpus

Figure #6 shows a sample of a document from the corpus in column format. Here we see the document terms in the column three, POS in column four, shallow parse in column five, and NE tag in column six. Columns with all zeros may be utilized for other purposes but are not used in this corpus.

We must specify how each of these columns is to be indexed. To do this, we write a schema that describes each column. We also include all of the parameters that Indri needs so that we can automatically generate the build parameter files for Indri. This schema provides the parser with all the information it needs to parse the columns and the provides Indri with all the information it needs to begin indexing. See figure #6 for an example schema file; the first six lines are parameters for Indri and the last four lines are parameter for the column parser.

```
index                /home/belamber/indicies/aquaint_index
corpus.path          /home/DATA/TREC02/Column
corpus.class         column
memory               500m
collection.field     docno
stemmer.name         krovetz
column  Word         3
column  POS          4
column  chunk        6       BIO
column  NE           7       BIO
```

Figure 7 - Example corpus schema file

This schema shown in figure #7 shows the type of information in each of columns 3, 4, 6, and 7. It also shows that columns 6 and 7 are in BIO (beginning-inside-outside) format. We use BIO tags to specify that the label spans more than one word. Since values in BIO columns are prefixed with a 'B-' or an 'I-' we must tell the parser that the prefix is not part of the value of the tag. The schema also specifies parameters to Indri such as the location of the corpus, the location to build the index, the maximum amount of memory to use when indexing, the stemmer, etc.

Queries are represented in a similar format to documents in the corpus. This example query (see figure #8) uses a different schema than the documents. This query also contains more linguistic information than the documents in the corpus (e.g., the query has dependency parse tags and semantic role labels).

```
LEAF/1  B-PER   0   B-NP/MOD_SP/1   NNP Tom     -     MOD_A1/1
NP/2    I-PER   1   I-NP/HEAD_SP    NNP Cruise  -     ARG1/3
VP/2    O       2   B-VP/MOD_SP/3   VBZ is      -     0
VP/2    O       3   I-VP/HEAD_SP    VBN married marry 0
PP/3    O       4   B-PP/HEAD_SP    TO  to      -     0
NP/4    O       5   O               NNP X       -     ARG2/3
```

Figure 8 - Sample column-format query from the 2002 TREC QA task

5.1.2 Indri's Structured Query Language

Indri's structured query language supports querying for terms and putting constraints on what portions of the document terms occur within as well as

"wildcard" terms that match any term with a given tag. The full query language is described at the Lemur project website (http://www.lemurproject.org/lemur/IndriQueryLanguage.html). The queries generated here only use some of the features of Indri structured query language.

To specify that a query term must be tagged with a particular tag, the query term is appended with a period and the tag name. Part-of-speech tags for this research are of the form "posdt" where the "pos" prefix specifies that the tag is a POS tag and the "dt" suffix specifies that the POS is *determiner*. Thus, the query "state.posnn" will search for the term state when it is tagged with "NN" signifying a singular noun.

We can also combine multiple field restrictions for a single term. For example, to search for term "cruise" when it is tagged as both *proper noun* and as a *person named-entity*, we separate the two terms with a comma, as in "cruise.posnnp,nepeop".

The Indri structured query language can also search for particular fields without specifying what term is tagged with that field. To search for any named-entity of type *location* we use "#any:neloc".

Indri supports a number of ways to combine search terms (based on the InQuery query language), which include taking the maximum, taking a weighted sum, and several others. All queries generated for this research use the query operator "#combine"

5.1.3 Generating Indri queries from natural language

Each natural language query is transformed automatically into a structured Indri query. Each query is automatically tagged with the same annotations that the corpus documents have been tagged with. Each query term tag may be used as a restriction on the field value for that query term. Therefore, if a query term is tagged as a noun, a location, and part of a verb phrase we can search for that term with any combination of those tags in the corpus.

The experiments in the following section investigate which combination of these tags is most effective for retrieval performance. First, we will provide one

complete example of the transformation from natural language to Indri query language. For this example, we will use the first question from the TREC 2002 QA task, "In what country did the game of croquet originate?"

First, the question is rephrased as an answer and becomes "the game of croquet originated in XXX" where the "XXX" is a placeholder for a *location* named entity. With the question in answer form, we can run all of our NLP annotators on it. Running annotators for POS, shallow parse, NE, SRL, and dependency parse, we get the result shown in figure #9.

```
LEAF/1 O       0   B-NP/MOD_SP/1 DT   the      -   MOD_A1/1
NP/4   O       1   I-NP/HEAD_SP  NN   game     -   ARG1/4
PP/1   O       2   B-PP/HEAD_SP  IN   of       -   MOD_A1/1
NP/2   B-Sport 3   B-NP/HEAD_SP  NN   croquet  -   MOD_A1/1
VP/4   O       4   B-VP/HEAD_SP  VBN  originated  originate  0
PP/4   O       5   B-PP/HEAD_SP  IN   in       -   0
NP/5   B-Loc   6   B-NP/HEAD_SP  NNP  XXX      -   ARG0/4
```

Figure 9 - TREC annotated natural language query

We cannot use all of these annotations on the corpus since the corpus was not tagged with all of these annotators, but we can use the POS tagger and the named entity tagger.

The baseline query which does not use any of the NLP annotations is:

```
#combine( the game of croquet originated in )
```

If we wish to include the POS tags in the query, we can restrict each of the query terms to only match corpus terms that have the same POS with the query:

```
#combine( the.posdt game.posnn of.posin
croquet.posnn originated.posvbn in.posin )
```

If we wish to include the NE tags in the query we can restrict each of the query terms to only match corpus terms that have the same NE tag with the query:

```
#combine( the game of croquet.nesport originated in)
```

In this case, only one of the query terms has a NE tag so only one term is different from the original baseline query. To restrict each query to match in both POS and NE tag we generate the query:

```
#combine( the.posdt game.posnn of.posin
croquet.nesport,posnn originated.posvbn in.posin )
```

This query could be too restrictive. Perhaps some of the search terms never occur in the corpus with exactly the same tags as in the query. If that's the case we can add other less restrictive terms to ensure that possibly relevant documents are not ignored entirely just because the they are tagged with the wrong tags. One way to do this is to generate two query terms for each of the terms in the original query where one has the tag restrictions and the other is just a plain search term without any tags specified. This technique is shown for this example below:

```
#combine(the the.posdt game game.posnn of of.posin
croquet croquet.posnn originated originated.posvbn
in in.posin)
```

Finally, we can use a placeholder in the rephrased question where the answer would normally appear if we know the named-entity tag. For this example we know the answer is a location so we can add "#any:neloc" to the query. With all of these features, we end up the query:

```
#combine( the the.posdt game game.posnn of of.posin
croquet croquet.nesport,posnn originated
originated.posvbn in in.posin #any:neloc )
```

5.2 Results

In this section, we begin by describing each of the retrieval parameters, and then look at the results for all combinations of these retrieval options. With an overall perspective on how some retrieval options affect retrieval performance we will further compare some of the combinations of retrieval options.

5.2.1 Retrieval options

Part-of-Speech (POS)

This option specifies that the tagged POS of the search term must match the tagged POS of documents containing that term. This POS is one of the 50 part-of-speech tags as shown in. For example, the search term "crowd" in the query

"the largest crowd to ever come see Michael Jordan" is tagged with the POS "NN." This query term will only match occurrences in the corpus of "crowd" that have been tagged "NN" and will not, for example, match the verb "crowd."

Simplified-POS (SPOS)

Since the 50 POS classes may be too restrictive in some cases, we also have a simplified POS feature. For example, in some queries we may not want to distinguish between noun and plural noun. Perhaps in these cases we would just like to differentiate among nouns, verbs, adjectives, and adverbs. The simplified part-of-speech retrieval option does exactly this. These simplified POS tags do not take into account the other POS tags that do not fall under noun, adjective, verb, and adverb (such as determiners, etc.).

Named Entity Tag (NE)

This retrieval option specifies that if the query term is tagged with a named entity tag that the corpus term must also be tagged with the same named entity tag. For example, in the query "tom cruise is married to," the query terms "tom" and "cruise" are tagged as named entity "person." Each of these query terms will only match corpus terms that have also been tagged as "person." Thus, corpus occurrences of "cruise" used as a verb or a noun in the "cruise missile" sense will not match this query term.

Raw search term (RAW-TERM)

Each of the three retrieval options above requires that the search term occur in the corpus with the exact same tags. This retrieval option allows us to add back the raw query term to the query. This will allow query terms to match corpus terms that do not contain the same tags as the query term. However, by using this in conjunction with tagged terms, the tagged term's weight will cause documents containing appropriately tagged terms to be ranked higher.

Empty named entity slot (NE-SLOT)

For queries where the answer type has already been determined, we can add a wildcard that matches any term that has the same named entity tag. For example, the query "the game of croquet originated in X" and X has been tagged as a location we add the Indri wildcard operator "#any:neLOC" which will match any term that has been tagged as the named entity of a location.

5.2.2 Results

There are 32 combinations of these five features. Each set of retrieval options is compared with four metrics: average precision, precision at 10 documents, precision at 20 documents, and precision at 30 documents.

Table 1 shows the set average (non-interpolated) precision over all 500 topics in the query set. Entries in bold show where the average precision has increased.

	-	with raw term	with NE slot	with both
no fields	0.1991	-	0.1973	0.1984
POS	0.1883	**0.2063**	0.1892	**0.2066**
SPOS	0.1867	**0.2053**	0.1876	**0.2070**
NE	0.1370	0.1656	0.1361	0.1660
POS, SPOS	0.1716	**0.2013**	0.1717	**0.2014**
POS, NE	0.1256	0.1675	0.1247	0.1694
SPOS, NE	0.1255	0.1670	0.1237	0.1670
POS, SPOS, NE	0.1071	0.1578	0.1039	0.1580

Table 1 - Average precision for each combination

Table 2 shows the precision at 10, 20, and 30 documents. (The three numbers in each table entry are precision at 10 documents, precision at 20 documents, and precision at 30 documents). Again increases in precision from the baseline are shown in bold.

	-	with raw term	with NE slot	with both
no fields	0.1310, 0.0912, 0.0706	-	0.1284, 0.0892, 0.0699	**0.1310**, 0.0900, 0.0701
POS	0.1199, 0.0823, 0.0645	**0.1317**, 0.0909, **0.0709**	0.1205, 0.0816, 0.0651	**0.1313**, 0.0910, 0.0704
SPOS	0.1223, 0.0840, 0.0646	**0.1319**, 0.0895, 0.0689	0.1217, 0.0828, 0.0640	**0.1319**, 0.0900, 0.0689
NE	0.0935, 0.0600, 0.0458	0.1062, 0.0751, 0.0587	0.0928, 0.0597, 0.0455	0.1069, 0.0749, 0.0593
POS, SPOS	0.1111, 0.0762, 0.0598	0.1302, 0.0882, 0.0695	0.1096, 0.0748, 0.0604	0.1308, 0.0888, 0.0689
POS, NE	0.0823, 0.0552, 0.0433	0.1054, 0.0756, 0.0597	0.0832, 0.0548, 0.0431	0.1053, 0.0752, 0.0598
SPOS, NE	0.0794, 0.0536, 0.0413	0.1053, 0.0723, 0.0574	0.0800, 0.0540, 0.0408	0.1053, 0.0723, 0.0574
POS, SPOS, NE	0.0702, 0.0472, 0.0368	0.1022, 0.0721, 0.0571	0.0702, 0.0469, 0.0372	0.1004, 0.0721, 0.0575

Table 2 - Precision at 10, 20, and 30 documents

Four of the combinations showed an increase in average precision. The relative precision increase for each of the four combinations is shown in Table 3 below.

	AvgPrec	Prec@10	Prec@20	Prec@30
POS w/ term	+3.62	+0.53	-0.33	+0.42
SPOS w/term	+3.11	+0.69	-1.86	-2.41
POS w/ both	+3.77	+0.23	-0.22	-0.28
SPOS w/both	+3.97	+0.69	-0.13	-2.41
POS&SPOS w/ term	+1.10	-0.61	-3.29	-1.56
POS&SPOS w/ both	+1.16	-0.15	-2.63	-2.41

Table 3 - Percentage increase for POS augmented queries

5.2.3 Discussion of Results

These results may be somewhat surprising. Perhaps one might think that semantic information like the named entities tags would make a greater difference, but in fact, it is the syntactic POS tags that give us an improvement. There may be many reasons that the results are such including inaccuracies of automatic taggers and typical distributions of some of the search terms with various tags.

5.2.3.1 Raw term and NE placeholder

Using the raw term

One consistent pattern in the average precisions over all topics is that the more specific and restrictive the queries are, the worse the average precision is. This means that any gains we get from the more precise query are offset by the relevant documents that do not match the more precise query. Some of this decrease in performance can be attributed to imperfect annotators.

This results means that is necessary to include the original term without restricting it to certain NLP taggings. By including search terms for both the raw original term and the term in the context with the specified tags, we get the advantages of specifying the NLP tags in that matching documents will be higher ranked but we do not get the disadvantage that relevant documents without the exact tags as the query are not matched. Thus by specifying both the raw original term and a term with NLP tags specified (combining with the Indri #combine operator) we are essentially re-ranking the results. Documents that match the terms and their respective NLP tags will be ranked higher.

Using a NE placeholder

The next observation is that adding a named-entity slot when the answer type is known sometimes helps but very little. Perhaps the first question to ask is why adding this feature does not decrease performance. If the NE tagger is correct, then any relevant document will have the correct tags somewhere in it. However, even if the NE tagger is imperfect it is likely that most documents contain one of the more broad NE tags (such as *person*, *organization*, and *location*). This may also be the reason that this feature does not help very much, because very many documents contain named entities of that type. This type of feature may have a greater impact if used with a proximity operator.

5.2.3.2 Field restriction features

Having discussed using the raw term and an NE placeholder, we see that it is necessary to include the raw term and that the NE placeholder does not have a

large effect. The remaining discussion will assume that we are using the raw term with each query and that results are not greatly impacted by whether or not we use the NE placeholder.

Why do POS features work better than NE features?

POS and SPOS features improve the precision, but NE features do not. It is not immediately clear why this is.

One reason may be that in many cases NE tags do not buy very much. For example, the first query in the set asks where the sport *croquet* originated. Since the word "croquet" should always be tagged as a sport we gain nothing. If the NE tagger does not consistently tag "croquet" as a sport, then this can only harm us. Many other NE types may demonstrate the same behavior, such as proper nouns that are not also common nouns (e.g. most last names).

The other question about NE features is: when *can* they help? The answer seems to be that they help if the named entity is also a common noun (e.g. Cruise as in Tom Cruise vs. cruise as in cruise ship) or if there are two named-entities with similar names but different types (e.g. Washington the person vs. Washington the location). However, both of these cases are likely to harder for the NE tagger to correctly tag automatically.

So, if NE tags are not helping us right now, why do POS tags help? One reason may be that searching for terms with respect to their part-of-speech does a kind of "poor man's" word sense disambiguation. For example, the two parts-of-speech that the word *state* can be, noun or verb, also correspond to the two broadest word senses for that word.

A curious result that we see in the POS results is that POS does not do very much differently than SPOS. One might have hypothesized that with 50 parts-of-speech, matching them exactly is over-fitting. On the other hand, one could also reason that the more specific parts-of-speech are more precise. For example, if the query used a verb in past tense then it could be good to only match that verb in the past tense (e.g. *state* vs. *stated* which might otherwise get stemmed to the

same term). Therefore, it is not clear why the difference in not more pronounced, a more detailed analysis of the individual queries and results would be needed.

One final question may be, why do POS tags help the amount they do for these experiments (~3% increase in average precision)? One reason could be that many of the query terms in this set of queries have a much-skewed POS distribution (i.e. they overwhelmingly occur in one POS). Alternatively, it is possible that for some queries we do not want restrict the document terms to a particular POS (perhaps "cruise" is used as both a noun and a verb in relevant documents).

Chapter 6

Conclusions

In this research, we have discussed some methods of natural language processing and information retrieval and how they may be used together. We have shown an efficient way of augmenting corpora with NLP tags, some applications in which it makes sense to do this, and a specific application where this method improves retrieval performance.

This framework can be used for a variety of applications, from linguistic analysis tools to general text search tools. Although indexing the additional linguistic data is more time and space hungry, the resulting data structures are robust with many applications and provide additional power over similar applications that do not use linguistic information in this way.

In the specific application, natural language queries for documents in a corpus, we saw that POS tags improve retrieval performance but that NE tags did not make a large difference. We also saw that it is necessary to temper the results by searching for both the original query terms and the query terms with linguistic annotations.

The increase in retrieval performance is modest but shows that there are gains to be made by using more linguistic information for information retrieval. The results open some new questions about how named-entity tags can best be used to improve retrieval performance and how much tagging accuracy affects retrieval performance.

In conclusion, indexing additional linguistic tags makes sense because it enables a large number of applications to reside on top of a general retrieval engine. As we have seen, the performance increases depend on the quality of the linguistic annotations but there is room for improvement.

References

Baeza-Yates, Ricardo, and Berthier Ribeiro-Neto, Modern Information Retrieval, New York: ACM Press, 1999. Chapter 5—pp117-140.

Callan, Jamie. "Text Representation." Carnegie Mellon University. Pittsburgh, PA. 2 February 2006.

Callan, Jamie. "Retrieval models: Boolean, vector space." Carnegie Mellon University. Pittsburgh, PA. 19 January 2006.

Cognitive Computation Group. "Named Entity Recognizer" 17 March 2006. University of Illinois Urbana-Champaign. http://l2r.cs.uiuc.edu/~cogcomp/demo.php?dkey=NER

Conference on Natural Language Learning. 15 April 2006. http://ilps.science.uva.nl/~erikt/signll/conll/

English, Jennifer, Marti Hearst, Rashmi Sinha, Kirsten Swearingen, and Ping Yee. Hierarchical Faceted Metadata in Site Search Interfaces. CHI 2002 Conference Companion.

Francis, W. N., H. Kucera, Brown Corpus Manual. Providence, Rhode Island, Department of Linguistics, Brown University. 1979.

Hearst, Marti, Jennifer English, Rashmi Sinha, Kirsten Swearingen, and Ping Yee. Finding the Flow in Web Site Search. Communications of the ACM, 45 (9), September 2002, pp.42-49.

Hearst, Marti. Next Generation Web Search: Setting Our Sites, IEEE Data Engineering Bulletin, Special issue on Next Generation Web Search, Luis Gravano (Ed.), September 2000.

Indri. lemurproject.org 2006

iTrack. 2005 http://l2r.cs.uiuc.edu/~cogcomp/demo.php?dkey=MIRROR

Kilgarriff, Adam, http://www.sketchengine.co.uk/

Lambert, Benjamin. Statistical Identification of Collocations in Large Corpora for Information Retrieval. University of Illinois Urbana-Champaign. 24 May 2004.

Lin, Dekang, 2005 <http://www.cs.ualberta.ca/~lindek/>

Munoz, Marcia, Vasin Punyakanok, Dan Roth, and Dav Zimak. A Learning Approach to Shallow Parsing. Proc. of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. 1999.

Page, Larry, Sergey Brin, R. Motwani, and T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Library Technologies Project. 1998.

Punyakanok, Vasin, Dan Roth, and Wen-tau Yih. Generalized Inference with Multiple Semantic Role Labeling Systems Shared Task Paper. Proc. of the Annual Conference on Computational Natural Language Learning 2005.

Resnik, Philip, and Aaron Elkiss. The Linguist's Search Engine: Getting Started Guide. Technical Report: LAMP-TR-108/CS-TR-4541/UMIACS-TR-2003-109, University of Maryland, College Park, November 2003.

Roth, Dan, and Dmitry Zelenko, Part of Speech Tagging Using a Network of Linear Separators. Proc. of Conference on Computational Linguistics (1998) pp. 1136--114

Shopen, Timothy, ed. Language Typology and Syntactic Description Volume I, Cambridge University Press, 1985. Chapter 1 -- Parts-of-speech systems.

Treebank Part-of-Speech Tag Guide, 2002, <ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz>

Tu, Yuancheng, Xin Li, Dan Roth, PhraseNet: towards context sensitive lexical semantics. Proc. of the Annual Conference on Computational Natural Language Learning, 2003.

Tu, Yuancheng. <http://www.linguistics.uiuc.edu/ytu/pn/ >. 2006

Appendix A

User Manual

1. BUILDING A INDEX

A. The Corpus Schema File

The first step in using this system is to describe the corpus by creating a schema file. A schema file is a text file. Following this example is a description of each parameter.

```
<Index      apw_index
corpus.path    data/apw
corpus.class   column
memory      500m
collection.field docno
stemmer.name   krovetz
usewn       false
column      Word       3
column      POS      4
column      chunk    6    BIO
column      NE       7    BIO
copy_index_to     /usr/www/cgi-bin/belamber
```

Index Indri will compile the Indri index in this location.

Corpus.path This is the location of the corpus. Indri will do a recursive decent in to the folder and index all files in the folder and recursive subfolders.

Corpus.class This is the class of the file to be indexed. This should be set to 'column'. Setting an alternative value will cause Indri not to use the column indexing system.

Memory The maximum amount of main memory that Indri will use while indexing the corpus.

Collection.field This should be set to 'docno'. Refer to the Indri documentation for more information about this parameter.

Stemmer.name The name of the stemmer used by Indri while indexing. 'krovetz' is the recommended stemmer. Refer to the Indri documentation for alternatives

Usewn Set to 'true' or 'false'. This controls whether WordNet will be used during indexing such that hypernyms and hyponyms can be searched for. This option is not available in the most recent release.

Column Column_Name Column_Number [BIO]

To specify the role of each column the line should start with the keyword 'column' followed by the column name followed by the column number. After the number, the user may optionally specify if the column is in BIO-style. See below for more information about column names and BIO-styled columns.

Copy_index_to Optionally, the index can be copied to another directory upon completion.

B. Building the supporting parameter files

Once the schema file is set up, run the Perl script `generate_build_files.pl` to generate all the supporting files. This script creates three files with names based on the file name of the schema file. For example, if the schema file is names `schema_file` then the following three files are created:

1. `schema_file.values`
2. `schema_file.build`
3. `schema_file.retr`

The script `generate_build_files.pl` may take a long time to run the first time. It is necessary for Indri to have all column names and all possible values for each column. This list of columns and values is used to create parameter files for Indri and create another list that can be later used to assist with querying.

The first time this script runs, it reads the entire corpus to obtain the possible values for each column. It then creates the three files:

1. *schema_file.values* contains the possible values for each column. Each line of the file corresponds to one column. The first value in each line is the name of the column as specified in the schema file. The remainder of the line is all of the possible values for that column, space-separated, in order of deceasing frequency.

```
POS    NNP      NN       IN       DT       JJ       CD …
chunk  NP       VP       PP
NE     Num      Peop     LocCoun  Date     ProfTitle …
```

2. *schema_file.build* is the parameter file used by Indri directly to build the index. This file contains information indri uses to construct the corpus. This includes crucial information such as the location of the corpus and the location to put the index.

```
<parameters>
   <index>apw_index</index>
   <corpus>
      <path>data/apw</path>
      <class>column</class>
   </corpus>
   <memory>500m</memory>
   <collection>
       <field>docno</field>
   </collection>
   <stemmer>
      <name>krovetz</name>
   </stemmer>
   <columns>
<column><name>Word</name><number>3</number></column>
<column><name>POS</name><number>4</number></column>
<column><name>chunk</name><number>6</number>
     <bio>true</bio></column>
<column><name>NE</name><number>7</number>
     <bio>true</bio></column>
   <usewn>false</usewn>
   </columns>
   <field><name>POSA</name></field>
   <field><name>POSN</name></field>
   <field><name>POSR</name></field>
   <field><name>POSV</name></field>
   ...
```

3. *schema_file.retr* contains basic information necessary to query the Indri index. The Indri `runquery` command needs two files: this retrieval parameter file and a file containing the queries.

```
<parameters>
   <index>apw_index</index>
   <count>1000</count>
   <rule>method:dirichlet,mu:2500</rule>
   <trecFormat>true</trecFormat>
   <runID>Exp</runID>
</parameters>
```

2. USING AN INDEX

Using the resulting index is no different than using any index constructed by Indri:

```
./IndriRunQuery index.lemur.retr query_file_name
```

Where index.lemur.retr is the retrieval parameter file and query_file_name is a file containing one or more queries.

Appendix B

Maintenance Guide

1. Files

This modified version of Lemur-Indri has the following additional files:

columnbuildindex/ColumnEnvironment.cpp
columnbuildindex/include/ColumnEnvironment.hpp
src/ColumnParser.cpp
include/column/ColumnParser.hpp
columnbuildindex/examples/generate_build_files.pl

Additionally, this file has been modified:

src/ParserFactory.cpp

Also included with this version are:

columnbuildindex/columnbuildindex.cpp
columnbuildindex/Makefile
column_query/column_query.cpp
column_query/Makefile

Two files must be modified to compile:

MakeDefns (Which must be modified by adding "wn" to the line beginning "DEPENDENCIES".)
Makefile (which must be modified by adding the line "$(MAKE) -C columnbuildindex)

2. Classes

The column-format corpus support is contained within the following classes:

columnbuildindex/columnbuildindex.cpp
columnbuildindex/ColumnEnvironment.cpp
src/ColumnParser.cpp
src/ParserFactory.cpp

A. columnbuildindex.cpp This class largely sets up all the data structures and classes before indexing begins. This class is very similar to the class buildindex.cpp that comes with Indri.

B. ColumnEnvironment.cpp This class is also mainly a supporting class and defines the "environment" while indexing. This class is very similar to the IndexEnvironment.cpp class.

C. ColumnParser.cpp Most of the work regarding the processing of columns is defined in this class. This class parses the column-format documents and uses the data to construct a ParsedDocument object.

D. ParserFactory.cpp This class native to Indri is slightly modified to instantiate ColumnParser when that format is specified in the configuration. The "name" of the ColumnParser is "column". Aliases for this can be hard-coded into this class.

E. Other classes Are used within the main classes. TermExtent, ParsedDocument, etc. Refer to the Indri documentation for more details about those classes.

3. The ColumnParser class

Since the ColumnParser class is the workhorse of this IR implementation, pseudocode for the algorithm is provided here:

```
For each line
   Split the line into tokens, separated by a space or a tab
   For each column specified in the build parameters:
      If column name is "Word"
         Add term to the term vector
         Add the position to the positions vector
      If column name is "POS"
         Add raw POS to the tags vector
         Simplify the raw POS to (Adj, Noun, Verb, and Adv)
         Add simplifies POS to the tags vector
      Otherwise:
         Remove all punctuation from the tag name
         If "BIO" column, remove the leading "B-" or "I-"
         Add string "column_name"+"column value" to the tags vector

   If Use_WordNet is true
      Query WordNet for Synonyms, hypernyms, and hyponyms
      For each hypernym, add to term vector "HYPERNYMOF" + word
      For each hyponym, add to term "HYPONYMOF" + word
      For each synonym, add to term "SYNONYMOF" + word
```