2008-09-01

Research: A Requirements Search Engine: Progress Report 2

Craig Martell

NPS-AM-08-146

ACQUISITION RESEARCH SPONSORED REPORT SERIES

ReSEARCH: A Requirements Search Engine: Progress Report 2

September 2008

by

Paige Adams
(phadams@nps.edu)

Pranav Anand
(panand@ucsc.edu)

Grant Gehrke
(gtgehrke@nps.edu)

Ralucca Gera
(rgera@nps.edu)

Marco Draeger
(mdraeger@nps.edu)

Craig Martell
(cmartell@nps.edu)

Kevin Squire
(kmsquire@nps.edu)

Prepared for: Naval Postgraduate School, Monterey, California 93943

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

The research presented in this report was supported by the Acquisition Chair of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request Defense Acquisition Research or to become a research sponsor, please contact:

NPS Acquisition Research Program
Attn: James B. Greene, RADM, USN, (Ret)
Acquisition Chair
Graduate School of Business and Public Policy
Naval Postgraduate School
555 Dyer Road, Room 332
Monterey, CA 93943-5103
Tel: (831) 656-2092
Fax: (831) 656-2253
email: jbgreene@nps.edu

Copies of the Acquisition Sponsored Research Reports may be printed from our website www.acquisitionresearch.org

Abstract

This research addresses three closely related problems. (1) Most current search technology is based on a popularity metric (e.g., PageRank or ExpertRank), but not on the semantic content of the searched document. (2) When building components in a service-oriented architecture (SOA), developers must investigate whether components that meet certain requirements already exist. (3) There is no easy way for writers of requirements documents to formally specify the meaning and domain of their requirements. Our goal in the research presented here is to address these concerns by designing a search engine that searches over the “meanings" of requirements documents. In this paper, we present the current state of the ReSEARCH project.

THIS PAGE INTENTIONALLY LEFT BLANK

NPS-AM-08-146

ACQUISITION RESEARCH SPONSORED REPORT SERIES

ReSEARCH: A Requirements Search Engine: Progress Report 2

October 24, 2008

by

Paige Adams
(phadams@nps.edu)

Pranav Anand
(panand@ucsc.edu)

Grant Gehrke
(gtgehrke@nps.edu)

Ralucca Gera
(rgera@nps.edu)

Marco Draeger
(mdraeger@nps.edu)

Craig Martell
(cmartell@nps.edu)

Kevin Squire
(kmsquire@nps.edu)

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

I. Motivation

While modern computing has made it possible to access enormous amounts of information with little effort, much of that information comes without any indexing, making manual search of it all but impossible. The science of *information retrieval* (IR) attempts to correct for this by extracting information from a collection of documents based upon a search request, or *query*. While the field of IR has focused a great deal of attention on how the form, or *syntax*, of a query, and the documents in the collection can aid the process of extracting information, it has paid far less attention to the meanings, or *semantics*, of those forms.

Semantic analysis can be computationally intensive, and for certain domains, sensitivity to meaning may not provide a system with sufficient improvement to justify the greater computational cost incurred. However, there are at least two conditions in which a semantically sensitive search can lead to improvements over keyword-based approaches: a) when the document collection is composed of human-generated free-text, and b) when the document collection is in a specialized domain, with non-standard terminology and assumptions (where the standard for most IR is the general content of the World Wide Web).

The Software, Hardware Asset Reuse Enterprise (SHARE) repository card catalog is in the intersection of both conditions mentioned above. The SHARE card catalog should ideally allow a user to search for an asset based upon both free-text overviews generated during asset submission, as well as additional structured metadata (Johnson & Blais, 2008). Because this overview is written in free-text, the syntactic form in which the information is expressed by the overview cannot be guaranteed in advance, making search over it quite difficult. In addition, the elements being searched over are descriptions of military assets. So, the document collection for this IR task is in a specialized domain, and the search process should be sensitive to the semantic connections that are particular to this domain.

In order to appreciate the challenges posed by IR over free-text and in specialized

domains, we now turn to the complications that each condition brings to the task.

A. Challenges of Free-text Search

Human language in general has several properties that make information retrieval taxing. Formally speaking, any language, human or man-made, can be expressed as a relation between *form* (syntax) and *meaning* (semantics); thus, fluency in a domain consists of knowing the relation between the forms of the language and their corresponding meanings. Man-made languages often aim to make this relation as straightforward as possible. For instance, in the mathematical language of arithmetic, the syntactic symbol "+" stands for the semantic concept of numerical addition. However, note that the symbol "−" can stand for two different semantic concepts: numerical subtraction or the marking of negative numbers. Thus, "−" has multiple meanings, and we say that it is *polysemous*. In arithmetic, only "−" is polysemous, but in human languages polysemy is pervasive. The word *tank*, for example, has multiple meanings (or, *senses*)—it may refer to weaponry or to a water tank. In the information-retrieval context, polysemy renders a query (and sentences in the document set) ambiguous: if the user is searching for tank specifications, are they asking about water tanks or weaponry?

Polysemy complicates the form-meaning relation by having multiple possible meanings for a given word. In addition, human language routinely has multiple words attached to a given meaning. We call these *synonyms*. For example, the verb *consume* has many synonyms, e.g., *devour, ingest, eat*. If a user enters the query "What type of fuel does an F-22 consume?," without an understanding of synonymy, the system will not be able to return to the user, for example, a document containing the answer "The F-22A Raptor uses JP-8." Hence, synonymy complicates the information-retrieval task by creating the (quite likely) possibility that the meaning requested by the user is expressed in a different form than the one the user searched in the query. Synonymy may occur at all levels of linguistic form; for example, the sentences, "The F-22A uses JP-8" and "JP-8 is the fuel type for the F-22A Raptor" convey the same semantic information despite their rather different forms. In par-

ticular, the first sentence lacks a synonym for *fuel*, meaning that sentence-level synonymy cannot simply be the product of word-level synonymy.

One final complication of searching over human language is that the relationships between semantic entities are not necessarily represented in the syntactic forms of the entities. For instance, the semantic entities *mother* and *daughter* are connected by a parental relation. In order to determine the sentential synonymy of *Mary is Jane's mother* and *Jane is Mary's daughter*, the system must understand the relationship between *mother* and *daughter*. This is a rather challenging task if we are simply looking at linguistic form, as there is nothing in the words *mother* and *daughter* that indicates they are connected. Such information is accessible only once we have some representation of the meanings of the words (or larger elements) and some way of deriving inferences between them.

B. Challenges of Domain-dependent Search

As detailed in (Johnson & Blais, 2008), the SHARE repository asset library currently consists of combat systems software and supporting artifacts, but will become more diverse (e.g., through the incorporation of hardware components). The card catalog will thus contain information about the specification and function of such artifacts. As Johnson and Blais note, there is (and will continue to be) a high level of similarity between the SHARE artifacts, given that they are all specifiable under the Surface Navy OA Warfare Systems Architecture Element Level Decomposition. Hence, their overviews will share many characteristics atypical of documents found on the Web, making Web-based tools sub-optimal.

However, this specialty of SHARE's domain could prove an advantage for a semantically sensitive search. For instance, it could allow for a reasonably robust polysemy control–i.e. in the SHARE context, a query involving *consume* is more likely to refer to fuel usage than to eating. Similarly, the domain could aid semantic inferencing (of the sort exemplified by the pair *mother-daughter*) based both on terms in the free-text overview and the larger functional context of a particular asset. Hence, based on facts regarding the

objects under discussion within SHARE, the system could conclude that there is a relation between *ballistics* and *shell-size*, allowing searches regarding one to consider documents containing the latter. Additionally, building on the product lines that assets play in the Navy enterprise, the system could infer that a given asset possesses certain properties that may be useful the user.

C. Domain-independent Learning for Domain-dependent Rules

Given that the polysemy and inferencing subsystems we are building are particular to the specialized domain of SHARE, one natural question is how such subsystems will be developed? One possibility is to build a handwritten set of rules and program the IR system to look to those rules when performing inferencing, such as that implemented in the Wordnet project.[1] While such strategies are undoubtedly useful, they typically: a) are time-consuming, b) lack empirical coverage (human error may cause a rule to go unnoted), and c) require constant supervision for a dynamic document collection. All three pitfalls are of concern with regard to SHARE; the most troubling is probably the requirement for constant maintenance, given that SHARE is an evolving repository and a potential model for similarly constrained repositories over different kinds of assets.

Given such problems, we propose that the domain-dependent components of Re-SEARCH be generated not by human input but by machine learning over the document collection of SHARE and, in the initial stages, additional informational resources. The goal of ReSEARCH is to develop a system of tools for determining domain-dependent resources to address the issues surrounding polysemy, synonymy and inference.

The remainder of this paper details the problems of contemporary approaches to IR and our investigations of approaches to integrate semantically sensitive tools. In the following section, we present an overview of common approaches to IR and explain why

[1]Wordnet is an ongoing project directed by George Miller at Princeton University's Cognitive Science Laboratory to encode relations between semantic entities. It may be accessed at http://wordnet.princeton.edu.

they will fail in using collections such as SHARE. We then discuss two approaches we are using to address the problem of semantics. Section III. discusses automatic inference-rule discovery, and Section IV. introduces the idea of automatic topic detection from text. We then explore graph theoretic approaches for disambiguating word meanings in Section V. . The final two sections address implementation issues and conclusions.

THIS PAGE INTENTIONALLY LEFT BLANK

II. Prior and Current Information Retrieval Strategies

The Web is a tremendously useful repository of information. Unfortunately, this information is unstructured, and there is no canonical "Table of Contents" or "Index," making web search one of the most challenging of today's Internet problems. Two attempts were made to address this challenge: (1) hand-classified directories (as originally used by Yahoo, for example), and (2) query-based search engines (for example, AltaVista and, eventually, Google). This second class is what concerns us here. For more details on Web Search Engines see Schwartz (1998).

A. Early Search Engines

Search engines employ a centralized architecture in which so-called "spiders" collect website information, and an indexer makes an index of these pages to ease the search. In the early 1990s, the first phase of web search was simply keyword search. In keyword-based search, all pages containing requested keywords are returned, ranked according to the strength of match (e.g., the number of times a word appears in a document, if it is in the title, etc.).

AltaVista used this strategy originally. In 1995, it was the first company to fully index the visible pages on the World Wide Web. Over time, it evolved different search modes: basic search, advanced search, and power search (Notess, n.d.). An example of an advanced feature that AltaVista and other search engines added was stemming (Sapp, 2000), which ensures that words with plurals and suffixes (e.g., -ed, -ing, -er) are always treated as being in their stem form (Hersh, 2003, p. 178).

B. Ranking Search Results

The second phase in Web search was the development of techniques that used the connection between pages to create a ranking of the websites for a more accurate search. The indexing problem was changed into finding the most appropriate way to "rank" each website. One easy solution was to make the rank proportional only to the number of

other pages linking to the page in question. However, this ranking method turned out to be inaccurate for a variety of reasons. In particular, it did not take into account the source of the links, allowing someone to easily boost the rank of a page by increasing the number of incoming links, thus subverting the indexing mechanism (Langville & Meyer, 2006).

In 1998, Google introduced a novel ranking method called PageRank (Brin & Page, 1998). In PageRank, the rank of each page in a search depends not only on the number of pages pointing to it, but also on the rank and the number of outgoing links of these pages. To further determine the rank of all web pages, Google simulates the behavior of virtual surfers randomly surfing the web. A page's rank is then updated based on how frequently the random surfers visit that page. This pre-existing rank of each individual website is assigned independently of any query. As a result of this ranking, the pages are ranked in order of sociological importance: the more links with higher weight there are to a page, the more important it is in the "society" of pages. Additionally, hubs—pages that have many links pointing to them—are given greater authority. In other words, the importance of a link is determined by both the rank of the linking page and the number of outgoing links from that page. One pitfall of this scheme (which Google attempts to correct) is that communities of websites can trap random surfers, which in turn, increases the rank of those websites.

Ask.com, formerly known as "Ask Jeeves," is another search site offering state-of-the-art search this time based on technology called "ExpertRank" (Ask.com, n.d.). In addition to examining the number of links entering a site, ExpertRank also attempts to identify topic clusters related to a search, as well as experts within these topics, and use all of this information to rank search results.

C. The State of Online Search Using Natural Language Processing

Since the "Semantic Web" has become a buzzword in the Internet community and in business at large, several organizations have emerged to provide "Semantic Search." Many promising companies and research projects have built search systems that crawl the

web for annotated data over which to search, such as web sites with RDF data. This search strategy, however, does not scour documents that do not have rich, hand-built, metadata. In particular, the vast majority of documents online, written in natural human language, are not searched. A small subset of these search engines, however, have begun tackling the problem of searching documents consisting only of written language, extracting semantic meaning.

Powerset Labs (www.Powerset.com), a San Francisco-based startup, has positioned itself as a forerunner in this field by attempting to leverage natural language processing in its search system. Currently honing its search algorithm, Powerset indexes and searches Wikipedia for question-answering tasks. The documents in this database are written in plain text and, for the purposes of search, do not contain extended metadata. Instead, the Powerset indexing algorithm identifies linguistic features such as *named entities* and *parts of speech* to improve search results.

Being a private, for-profit company, the Powerset search algorithm is not public, but some important functionality can be extracted from it for public demonstrations. The Powerset lab's website currently contains two methods of searching Wikipedia. The first is a general search of the document index, which encourages queries to be phrased as questions. Queries such as “When did earthquakes hit San Francisco?” and “politicians from Virginia” are among the suggested queries. Results of these queries return results that demonstrate term-matching on a higher level than keyword search. For example, the Powerset system uses “When” as a wildcard to match dates and times that appear in phrases describing earthquakes in San Francisco. “From” is used, in the second example, to search for phrases that indicate some named entity is “from” Virginia. This improves results significantly over a search with just the keywords “politicians” and “Virginia,” as are used in standard search engines.

The search “politicians from Virginia” also reveals that “politicians” matches terms such as “governor” and “senator,” indicating that an ontology is used to match the term

"politician" with its hyponym, "governor." The search "What do zombies eat?" reveals that the Powerset algorithm also searches over synonyms by returning results containing the synonymous verb "devour." This system does not perform rich disambiguation, however, as evidenced by the result ". . . zombie finishes college," in which "finishes" is considered a synonym of "eat."

Finally, results from the Powerset search "What do zombies eat" include phrases in which the information about what zombies eat is encoded in more complex sentence structures. Correct results such as "granddaughter eaten by zombies," "zombies . . . where they are brought back from the dead by supernatural or scientific means, eat the flesh or brains of the living," and "His corpse is thrown over the fence to be devoured by the zombies," all reveal that powerful parsing of the sentences is performed in the indexing process rather than strictly requiring matching phrases such as "zombies eat *." Though its indexing structure is not known, the "PowerMouse" demonstration allows the user to search the fact index more directly, confirming that these relationships exist in the index for fast searching, eliminating the need for computationally expensive parsing with every search query.

Powerset is thus building capabilities for a semantically sensitive search similar to those of ReSEARCH. However, it is not clear that Powerset's approach is designed to handle the domain-specificity of collections like SHARE; in other words it is not clear Powerset's technology can be leveraged to construct novel inferencing mechanisms in particular domains. In 2008, Powerset was purchased by Microsoft; we can, therefore, expect to see semantic techniques finding their way into Microsoft Live Search.

III. Automated Inference-rule Discovery

Recall that natural languages, unlike formal taxonomic structures, contain inherent ambiguity of both form and meaning. It is this ambiguity that presents a challenge for natural language applications such as information retrieval or question answering. Two questions arise: 1) which meaning of a word or phrase in a search term does the requester intend, and 2) how do we return results that are related to the search query, even if the search term does not contain the exact word or words? The first question is related to the problem of *word sense disambiguation* and is, itself, a well-studied area. We will discuss an approach to this later in Section V. . For now, we will turn our attention to the second problem: *inference*.

A. Semantic Similarity from Distributional Similarity

In 1992, Hearst explored using one kind of inference rules to generate others, given a body of text. Specifically, she considered how use of synonymy relations could be used to learn the relation of *hypernymy*, or subtype classification. For concreteness, consider the pair *vehicle-Humvee*. As *Humvee* is a subtype of *vehicle*, the latter is a hypernym of the former. How could a machine learn the hypernymy relation of *vehicle-Humvee* automatically? Hearst's approach exploited the fact that the co-occurrence of words in patterns of the type *X such as Y*, as well as its synonyms *X, including Y,* and *Y and other X*, implies a hypernymic relationship between X and Y. As she demonstrated, if a system were seeded with various synonyms for forms that demonstrate hypernymy, the system could induce hypernymic connections from the text provided.

While Hearst's method is useful for learning various inferences, it relies upon human-generated synonyms for expression of hypernymy (or the relation in question). More desirable would be a system that learns the synonyms themselves from the text, especially given the possibility that such synonyms could be domain-dependent. In their 2001 study, "DIRT—Discovery of Inference Rules from Text", Lin and Pantel outline an unsupervised

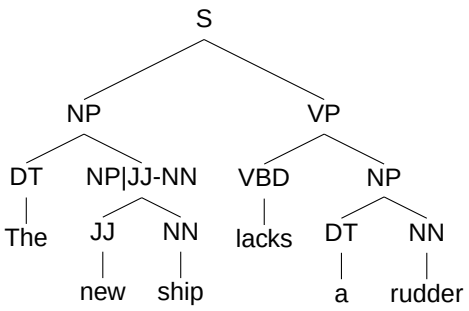

Figure 1: Parse Tree from NLTK Demo

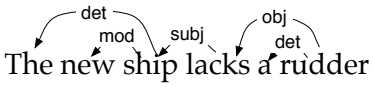

Figure 2: Dependency Tree of Same Sentence as in Figure 1

method of discovering inference rules from text, based on the idea that semantic similarity is generally correlated with syntactic similarity. We turn to this next.

B. Dependency Trees and Paths

A *dependency* relationship is an asymmetric binary relationship between two words: a **head** and a **modifier**. One can observe the structure of a sentence by examining the tree formed of the dependency relationships contained therein. The tree structure arises from the characteristic that a given word may have more than one modifier, but each word may modify a maximum of one word. Note that a dependency tree differs from a parse tree, which is concerned with the *syntactic* relationship between words. A comparison of the two are shown in Figures 1 and 2.

Dependency graphs are constructed by using Lin's MINIPAR, a broad-coverage English language dependency parser (Lin, 2008). Links in the graph represent indirect *semantic* relationships between two words. A dependency path is constructed by joining the

words and their link dependency relationships, excluding the two end words. For instance, in our example sentence, the dependency path between the words *ship* and *rudder* would be represented by the path N:subj:V←lacks→:V:obj:N. The words *ship* and *rudder* fill the slots in the path at either end. Non-slot dependency relations are called internal relations. In this manner, one can construct the paths of all word pairs in a given corpus of text.

Lin and Pantel (2001) imposed a set of constraints on the paths to be extracted:

- The "slot fillers" must be nouns, since these are variables that will be instantiated by entities.
- Dependency relations that do not connect the two content words (e.g., in the case of determiners or modifiers), will be excluded from the path.
- There will be a lower limit (threshold) on the frequency count of an internal relation.

To accumulate the frequency counts of paths in a corpus, a **triple database** was used. A triple is comprised of $(p, Slot, word)$ for two words w_1 and w_2. Correspondingly, each such pair of words has two corresponding triples: $(p, SlotX, w_1)$ and $(p, SlotY, w_2)$. $SlotX, SlotY$ and w_1, w_2 are *features* of path p.

C. Path Similarity

As alluded to above, Lin and Pantel's approach makes an assumption based on Harris's *Distributional Hypothesis* (Harris, 1954), which assumes that two words will have a similar meaning if they appear in similar contexts. Instead of words, Lin and Pantel assume that the hypothesis also holds for paths between words; i.e., if multiple dependency tree paths link the same set of words, then the meanings of the paths are likely similar. They termed this the *Extended Distributional Hypothesis*.

Computation of similarity between two paths first takes into account the **mutual information** between a path slot and its filler. The approach is similar to calculating a *tf* · *idf* (*term frequency* × *inverse document frequency*) measurement and is performed for a similar reason: to discount high-frequency words that may not have the same importance

as less-frequent words. Pantel and Lin's formula leverages the similarity measurement proposed in (Lin, 1998), but is modified to take paths into account:

$$mi(p, Slot, w) = \log \frac{|p, Slot, w| \times |*, Slot, *|}{|p, Slot, *| \times |*, Slot, w|}.$$

The mutual information thus defined, the similarity between a pair of slots is defined as:

$$sim(slot_1, slot_2) = \frac{\sum_{w \in T(p_1,s) \cap T(p_2,s)} (mi(p_1, s, w) + mi(p_2, s, w))}{\sum_{w \in T(p_1,s)} mi(p_1, s, w) + \sum_{w \in T(p_2,s)} mi(p_2, s, w)}.$$

In this formula, p_1 and p_2 are paths, s is a slot, and $T(p_i, s)$ is the set of all words that fill the s slot of path p_i. Finally, the similarity of two paths p_1 and p_2 is defined by the geometric average of the similarities of their $SlotX$ and $SlotY$ slots:

$$S(p_1, p_2) = \sqrt{sim(SlotX_1, SlotX_2) \times sim(SlotY_1, SlotY_2)}.$$

Comparison of paths in a corpus is accomplished via pairwise comparison of each path using the preceding formulae. Since comparison of all paths is computationally expensive, Lin and Pantel use a filtering algorithm that only compares paths if a candidate path's shared features with an input path p exceed a fixed percentage. This procedure ultimately produces a list of paths in descending order of their similarity to p.

D. Results

Lin and Pantel (2001) used MINIPAR to parse approximately 1GB of newspaper text from the *AP Newswire*, *San Jose Mercury-News*, and *The Wall Street Journal*. From this, they extracted seven million paths, 231,000 of them unique, which were then stored in a triple database. For evaluation, they used the first six questions of the TREC-8 Question-Answering Track, extracted the paths from the questions, and generated a Top-40 Most

Q#	PATHS	MAN.	DIRT	INT.	ACC.
Q_1	X is author of Y	7	21	2	52.5%
Q_2	X is monetary value of Y	6	0	0	N/A
Q_3	X manufactures Y	13	37	4	92.5%
Q_4	X spend Y	7	16	2	40.0%
	spend X on Y	8	15	3	37.5%
Q_5	X is managing director of Y	5	14	1	35.0%
Q_6	X asks Y	2	23	0	57.5%
	asks X for Y	2	14	0	35.0%
	X asks for Y	3	21	3	52.5%

Table 1: A Summary of Lin and Pantel's DIRT Algorithm Results on TREC-8 Questions.

Similar list using their algorithm to determine if the generated paths might contain the answer to the questions posed. This output was also compared to a set of publicly available, manually generated paraphrases of the TREC questions. In the evaluation, a path was deemed to be correct if it was likely that the path could generate the correct response to the question, given that the answer could be found in some corpus. An example used by Lin and Pantel (2001) was the path *"X manufactures Y"* generated from the TREC question, *"What does the Peugeot company manufacture?"* One of the Top-40 most similar paths is *"X's Y factory."* Since *"Peugeot's car factory"* is a likely phrase in some corpus, this generated path is classified as correct.

The DIRT algorithm performance varied widely for different paths. It was noted that paths with verb roots tended to perform better than verbs with noun roots since noun root paths tend to occur less often. Lin and Pantel (2001) also found that, even with high-scoring correct paths, there was little overlap between these automatically generated paths and the manually generated paraphrases, suggesting the difficulty for humans in the paraphrase-generation task. As noted earlier in studies of manual inference-rule generation, completeness errors exist due to the difficulty of paraphrase recall for humans. In this capacity, the DIRT algorithm shows promise in augmenting a manual-generation workflow.

A summary of DIRT results on the TREC data is in Table 1. The column labeled "Man." indicated the number of manual paraphrases generated for the question. The

next column shows the number of paths found by the DIRT algorithm. The intersection of those two is in the fifth column. The final column shows the evaluated accuracy of the automatically generated paths.

E. Related work

Snow, Jurafsky and Ng (2005) leverage a similar method of automated inference-rule discovery using dependency paths in a continuation of the hypernym discovery method pioneered by Hearst. This method involved using the dependency paths in a feature count vector and conducting a binary classification of hypernymy for word pairs based on vector-distance measurement. The results obtained represented a 16% F-score improvement over previous models, and a 40% improvement when augmented with **coordinate terms** (i.e., terms that share a common hypernym ancestor).

IV. Topic Detection and Extraction

A nascent research area trying to address the problem of semantics in documents is that of topic modeling. Topic models are probabilistic models which attempt to automatically identify topics shared among different documents. The *topics* in these models are not labels, but rather refer to groups of words that are shared among multiple documents. While a few different techniques for modeling topics exist, here we will focus on a particular topic model called Latent Dirichlet Allocation (LDA) (Blej, Ng & Jordan, 2003).

A. Latent Dirichlet Allocation (LDA)

LDA is a document model that employs an unsupervised algorithm for automatic topic discovery. The model assumes a particular random process by which a document is generated, and then tries to discover the set of parameters for that process that best explain how a given set of documents could have been generated by the model. This is (hopefully) a simplification of how the documents were generated in the first place, but is useful in that it allows us to extract related groups of words in an automatic and unsupervised manner. We will refer to these groups of related words as *topics*.

The particular random process is defined as follows. First, we assume that a document is a bag-of-words. That is, for the purpose of topic detection, the actual order of words in a document is unimportant and even if rearranged, still yields the same topics (if not an understandable document). Second, we assume the existence of a set of topics, and that each topic is defined by a probability distribution over our vocabulary (such that the most common words in that topic have the highest probability). Finally, we assume that the words in each document were drawn from multiple topics, and we can define a probability distribution over topics for each document by frequency counts of how often words from each topic appear in the document.

Given these assumptions, we assume a document is produced by the following procedure:

1. For a particular document, choose a distribution over the set of topics.
2. Choose a number N to be the number of slots in the document (to be filled with words).
3. For each slot,
   (a) choose a topic by sampling from the distribution over topics.
   (b) choose a word by sampling from the distribution over words for the previously chosen topic.

While documents are obviously not generated in this manner, this model has the benefit of allowing us to extract related groups of words, as mentioned above.

A graphical representation of the model can be seen in Figure 3. The parameters have the following meanings:

- α is the parameter for the Dirichlet distribution used as a prior for the topic distributions.
- $\theta \sim Dirichlet(\alpha)$ is the probability distribution over topics for a given document (also called a multinomial distribution).
- M represents the number of documents in the corpus.
- $N \sim Poisson(\xi)$ is a random number representing the number of words in a given document.
- $z \sim Multinomial(\theta)$ represents the topic of a particular slot in the document.
- β is the set of distributions over words, with one distribution for each topic.
- w is the word chosen for a particular slot in a particular document, determined by z and β.

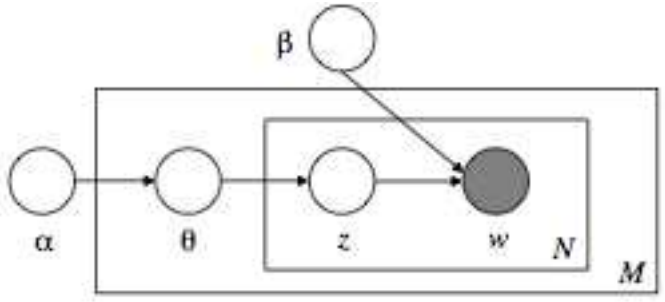

Figure 3: Graphical Model of LDA as Presented in Blei et al (2003). Plates indicate model components that can be replicated as needed. Arrows indicate dependencies among variables. Documents are represented by the outer plate, and topics and words in the document are represented by the inner plate. The variables themselves are defined in the text above.

A technique called Gibbs sampling provides a convenient way to estimate the topic distributions $p(\mathbf{w}|\mathbf{z})$. For a discussion of Gibbs sampling see Gamerman (1997).

According to the model, the posterior probability $p(\mathbf{w}|\mathbf{z}) = \left(\frac{\Gamma(W\beta)}{\Gamma(\beta)^W}\right)^T \prod_{j=1}^T \frac{\prod_w \Gamma(n_j^{(w)}+\beta)}{\Gamma(n_j^{(\cdot)}+W\beta)}$, in which $n_j^{(w)}$ represents the number of times, the topic j has been chosen for word w, and W is the number of distinct words in the corpus (Griffiths & Steyuvers, 2004) These probabilities represent the probability of the corpus given a particular distribution of topics. Since the Gamma function grows as fast as the factorial function, the logarithm of this expression is used for all computations: $\log p(\mathbf{w}|\mathbf{z}) = T(\log\Gamma(W\beta) - W\log\Gamma(\beta)) + \sum_{j=1}^T \left(\sum_w \log\left(\Gamma\left(n_j^{(w)}+\beta\right)\right) - \log\Gamma\left(n_j^{(\cdot)}+W\beta\right)\right)$. This is convenient, because $\log\Gamma(\cdot)$ can be evaluated directly as the `gammaln()` function in Matlab or Octave.

B. Experiments

For our own experiments with LDA, we used the Matlab topic modeling toolbox developed by Mark Steyvers (Griffiths & Steyvers, 2004) The model used in this toolbox is a slight variant of the model described above, where β represents a parameter of a second Dirichlet distribution used as a prior distribution for the set of topic distributions. The topic modeling toolbox provides a variety of tools to estimate topic distributions given model parameters α, β, and the number of topics T. All results were compared between three different corpora: *NIPS, Psychological Review, and Wikipedia*. NIPS and Psychological Review are corpora used and provided by Mark Steyvers in the Matlab topic

modeling toolbox. The Wikipedia subcorpus is built from a stratified sample of Wikipedia and consists of only 1000 documents. Table 2 provides a short overview of the different corpora.

	Psychological Review	NIPS	Wikipedia
Number of documents	1,281	1,740	1,000
Number of words in the vocabulary	9,244	13,649	69,729
Mean number of words per document	66.379	1,322.6	866.105
Number of words	85,031	2,301,375	866,105

Table 2: Short Overview over the Explored Corpora

Our experiments with the LDA algorithm covered three different directions. First, since we do not really know the number of topics that exist in each corpus, we estimated an optimal number of topics for each corpus for a given parameter β. Second, in a slight variation, we estimated an optimal parameter combination for each corpus. Finally, we used observed knowledge from the corpus to improve the LDA algorithm.

Finding the optimal number of topics is an optimization problem where; here, we attempt to maximize the probability of the observed corpus given the fitted model. The objective function $p(\mathbf{w}|T)$ turns out to be a strictly concave function over T, if all other model parameters are fixed. However, the function itself is not differentiable, and, therefore, the optimization problem cannot be solved analytically. Griffiths and Steyvers, (2004) used the harmonic mean over several samples from different Markov chains to estimate $p(\mathbf{w}|T) \approx N/\sum_{j=1}^{N} \frac{1}{p(\mathbf{w}|\mathbf{z}_j)}$, (in which N is the number of samples drawn,) for a different number of topics, keeping track of the max. Since the objective function is concave, there is a unique global maximum that can be found by narrowing down the interval of the maximum value. Applying this algorithm to the different corpora *Wikipedia, NIPS, and Psychological Review* data, we can draw the results in Table 3.

In our second set of experiments, to find an optimal parameter combination for the model, we manipulated the model parameters β and T one after the other. Unfortunately, from our experiments, we determined that there is no useful optimum obtained from manip-

$\log \beta$	Psychological Review	NIPS	Wikipedia	SHARE?
-10.0	20	20	20	
-10.5	20	40	20	
-11.0	20	80	20	
-11.5	20	120	20	
-12.0	20	220	20	
-12.5	40	300	20	
-13.0	60	300	20	
-13.5	100	300	20	
-14.0	220	300	20	
-14.5	340	300	40	
-15.0	580	300	60	
-15.5	800	300	100	
-16.0	800	300	180	
-16.5	780	300	300	
-17.0	unbounded	unbounded	480	

Table 3: Optimal Number of Topics per Corpus and Diversity Parameter β. For very small values of β, the number of topics increases dramatically

ulating both parameters, as the most probable combination is one in which $T = \#words$ and β is very small (See Figure 4).

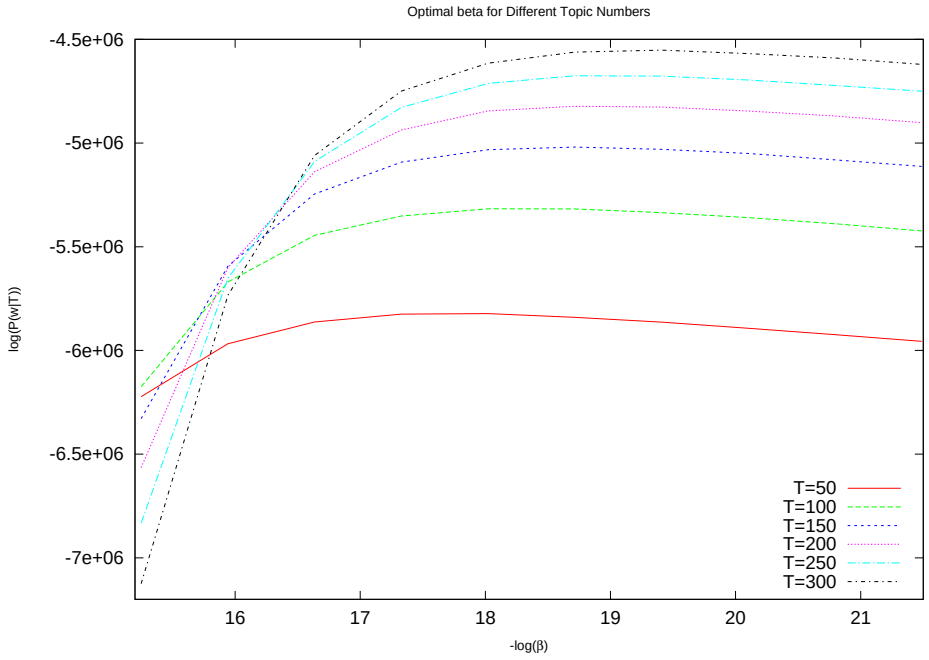

Figure 4: Finding an Optimal β/T Combination. The figure shows the unboundedness of this problem. Most likely, the probability will be maximized for $T = \#words$ and β very small. This is not useful for any practical purposes.

For the final set of experiments, we used known word frequencies in the corpus as replacement for $beta$ in the Dirichlet distribution used to choose topic distributions. The frequency vector was obtained by counting the number of occurrences for each word in the corpus and dividing by the total number of words. Smoothing of this vector was not necessary because the minimal count of words is set to five. This procedure complies with that of Griffiths and Steyvers (2004). We found that this substitution significantly increases the observed probabilities of the corpus given the model (i.e., the model is more likely to produce the documents in the corpus after this change).

V. Graph Theory for Word-sense Disambiguation

In this section, we present a brief overview of our on-going research on using graph-theoretic methods for word-sense disambiguation.

In Widdows (2004) the researcher presents a construction of a graph from a corpus. A graph is obtained from a corpus of nouns in which each vertex is a noun in the corpus, and an edge is present between two nouns if the two nouns are observed in a coordination pattern. The coordination pattern can be defined looser or tighter – some examples being that the two words are present in the same sentence, or in the same paragraph of a text. The extraction of this type of coordination was first introduced by Riloff and Shepherd (1997) and independently by Roark and Charniak (1998).

In the current paper all graphs $G = (V(G), E(G))$ are simple graphs (no multiple edges), with vertex set $V(G)$ and edge set $E(G)$. For vertices u and v in a connected graph G, the *distance between* u *and* v *in* G, $d_G(u, v)$, is the length of a shortest $u - v$ path in G. The k*-neighborhood* of u, $N_k(v)$, is the set $N_k(v) = \{u \in V(G) : d(u, v) \leq k\}$.

The detection problem in finding whether a graph is connected or not is tractable, and a search algorithm can be used. The connectivity of a graph is one way to measure how connected a graph is. That is, the *vertex-connectivity of* G, $\kappa(G)$ is the minimum number of vertices that need to be removed in order for the graph to become disconnected. Note that if the graph is a tree T, then $\kappa(T) = 1$, and the removal of any non-leaf vertex will disconnect the tree into a forest. A similar approach applies to non-trees and also to disconnected graphs. A *cut-vertex* of G is a vertex in $V(G)$ whose removal increases the number of components of G. Note that a graph G has a cut-vertex if and only if $\kappa(G) = 1$. Not every graph has a cut-vertex, and in this case we consider a vertex cut. A *vertex cut* of G is a subset of vertices whose removal increases the number of components of G.

Vertex cuts have been used in the literature to determine malicious vertices in congested ad hoc routing networks, particularly the ones with dynamic topologies. Detecting the faulty vertices in an open ad hoc network presents challenges that traditional wired net-

works do not. In wired networks, the faulty vertices can be checked at routers, gateways and switches, but the ad hoc ones cannot (Karygiannis, Antonakakis & Apostolopoulos, 2006)

Menger's theorem characterizes the connectivity of a graph G, giving results on the existence of a vertex cut of a graph as well. Two *internally disjoint* $u - v$ *paths* are two different $u - v$ paths in G that do not share any common internal vertices (i.e. they may share the end points of the paths). Menger's theorem asserts the following:

Theorem V. .1. *Let* a *and* b *be distinct nonadjacent vertices in* G*. There is a vertex cut of size* k *that disconnects* a *from* b *if and only if there are at most* k*-internally disjoint* $a - b$*-paths.*

And so, as a quick corollary, we have that if G is k-connected, then there are at most k-internally disjoint $x - y$-paths, $\forall x, y \in V(G)$.

An $O(nm)$ algorithm for finding a cut-vertex in a graph is the following (Wikipedia, n.d.):

a = number of components in G (found using DepthFirstSearch/BreadthFirstSearch)
for each $v \in V(G)$ with incident edges
    Remove v from $V(G)$ to obtain $G - v$
    b = number of components of $G - v$
    if $b > a$
    v is a cut vertex
    restore v.

An algorithm with a running time $O(n + m)$ is known using depth-first search.

The vertex-cut and cut vertex ideas extend to directed graphs, the problem being known as the maximum-flow-min-vertex cut problem. That is, in a flow network, the maxi-

mum amount of flow allowed is equal to the capacity of the vertex cut. One can think of the vertex cut in this case as a bottleneck of the networks that dictates the flow in the directed graph. And so the vertex cut gives the weakest set of links in the network. The associated decision problem is tractable: there is a low-order polynomial-time algorithm to solve it. For some multicomodity flow problems, there is an algorithm with an $O(\log n)$ factor of the upper bound given by the vertex cut (Leighton & Rao, 1999).

For other graph theory terminology we refer the reader to Chartrand and Zhang (2004).

We now introduce a diffrent type of extension of the cut vertex that will help us identify ambiguous words in a text/corpora. We define $v \in V(G)$ to be a *local cut-vertex* if $\forall u, w \in N_k(v)$ in G, we have that in $G - v$ the distance between u and w increases, i.e., $d_{G-v}(u, w) > k$, $(k \geq 1)$. In Widdows (2004) the author presents the following conjecture with a variety of examples, which is re-warded in terms of the terminology we just introduced:

Conjecture: *Let G be a graph. If $v \in V(G)$ is a local cut vertex, then v represents an ambiguous word in the corpus.*

We will consider this conjecture in the current research paper, together with a possible connection/variation of Meger's theorem. Notice that the cycles in a graph connect words with the same or similar meaning, and the meaning within a cycle may change if another polysemous word is used.

We now present an example of the conjecture. Note that the vertices "apple" and "mouse" are the polysemous words, and their removal will give two separate components of the graph, each of the clusters being grouped by one of the two meanings of the words "apple" and "mouse."

We will produce graphs from corpora and study the vertex cut and local vertex cuts on these graphs.

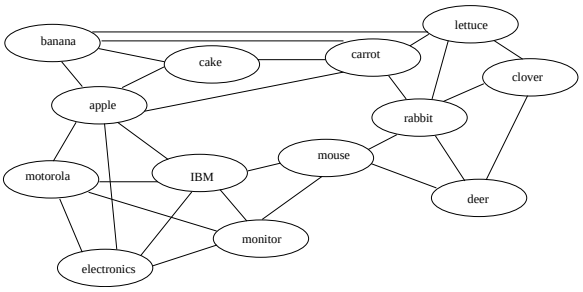

Figure 5: Example Graph Derived From a Corpus.

VI. Implementation issues

Lucene Java is an Open Source project, available under the Apache License, which provides an accessible API for the development of search applications. Lucene provides plenty of opportunities to construct a semantic search engine. A good overview and documentation is available from the Apache Lucene website (*Lucene-java Wiki*, 2008).

A search application developed with Lucene consists of the same two major components mentioned in Section 2: an indexer and a searcher. The indexer builds an index of the given documents; the structure and content of this index depends on the implementation of the indexer application. Typical contents would be the title of a document, its path, a URL, or the actual text content. Content can be stored in different ways, depending on if it has to be searchable or not. The search application typically converts a search string given by the user into a query and then searches the index for matching items. Later in this section, two short examples will demonstrate these processes.

A. Interesting Features of the Lucene API

One remarkable property of Lucene is its flexibility. By overriding the stemming and analyzing algorithms a developer can change its behavior into something completely new, particularly from a keyword search engine into a semantic search engine similar to Powerset; however, a very useful property of Lucene is its accessibility from different environments– e.g., Python.

PyLucene is a Python extension for accessing Java Lucene. This extension allows developers to implement some functionality of the desired application using NLTK, a widely used Python-based project for natural language processing. Documentation and implementation samples for PyLucene can be found in Vajda (2005).

Another very helpful feature is a package for indexing and query expansion based on WordNet synonyms. Using the WordNet application, this package creates a synonym index of words and converts search strings into queries which can be used by Lucene.

For our first tests, we built an index of synonyms from WordNet and used it to expand and convert search strings into Lucene-compatible queries.

B. The Wikipedia Corpus

For our experiments, we decided to use downloadable Wikipedia content (http://download.wikimedia.org/enwiki/20080312/enwiki-20080312-pages-articles.xml.bz2).

The size of this file is about 60 GB. This size requires an event-based parser such as SAX. For the first experiments, only about 160 MB (more than 12,000 articles) from a partial download were used.

The structure of the XML file is as follows: every article is stored in a `<page>` node, which has several child nodes. From these child nodes we used the `<title>` and `<text>` fields. The special syntax of a Wikipedia page was ignored at first, meaning that all the content of an article was given the same priority–particularly, we did not distinguish between headings, links or normal text.

Parsing and indexing 12,738 articles took about four minutes on a Windows Vista PC with an AMD64 CPU and 1 GB memory under non-benchmark conditions.

C. Sample Implementations

Two sample implementations will be introduced: a Wikipedia indexer and a small search application.

The indexer follows a sample given in Schmidt (2005). The original version had to be changed in order to obtain compatibility with the current version of Lucene. Only the main concepts will be considered at this point; for further explanations of the different classes involved, see Lucene-java Wiki (2008) and Lucene's Javadoc.

The main part of an indexing application is the index writer. It writes the index into a file system and also optimizes its structure for faster access. Logically, the written index consists of documents; in our case, every Wikipedia article is treated as a separate document. A document is then split into different fields; for the sample application, these

fields were “title” and “text.” The indexer determines whether and how a field is stored in the index. The choices are: (1) not to store at all, (2) to store, but not to index, (3) to store and to index it without first analyzing it, and (4) to store it and to index it using an analyzer. An analyzer implements a certain policy for extracting index terms from text. Lucene already implements various analyzers; we used a StandardAnalyzer for our tests. Since an analyzer determines how the content of a document is represented in the index, it provides an opportunity for the developer to implement semantic strategies for building and searching the index. The Wikpedia indexer first parses the xml file, which contains the articles and extracts all <page>-nodes, from which the <title> and the <text> nodes are extracted. Then, for every article, two new fields are generated: “title” and “text.” These fields are added to a new document, which is then passed to the IndexWriter-object. After this process, the index content is optimized by the writer, concluding the indexing process.

The searcher was implemented in Python using PyLucene, using a StandardAnalyzer. To be able to search for an article, the user’s search string has to be converted into a query. This conversion is done by a Lucene class called QueryParser, which is generated using the name of the field that contains the actual content and the analyzer. The query is then passed to the searcher, which returns an object called “hits.” This object holds a list of all matching documents with an assigned score. For our purposes, the searcher application just prints the titles of the matching articles, followed by their score.

The score is assigned by an object which extends the Scorer class in the API. The scorer itself uses a similarity implementation which is based on the cosine-distance between document and query vectors in a Vector Space Model of Information Retrieval.

D. WordNet Query Expansion Sample

Figure 8 shows the output of a standard keyword search using only a single word in the search string versus the output of the same search after expanding the search with the WordNet interface. Only results with a score higher than .5 were printed. This very simple sample shows how using synonyms can improve a search significantly. Note, the

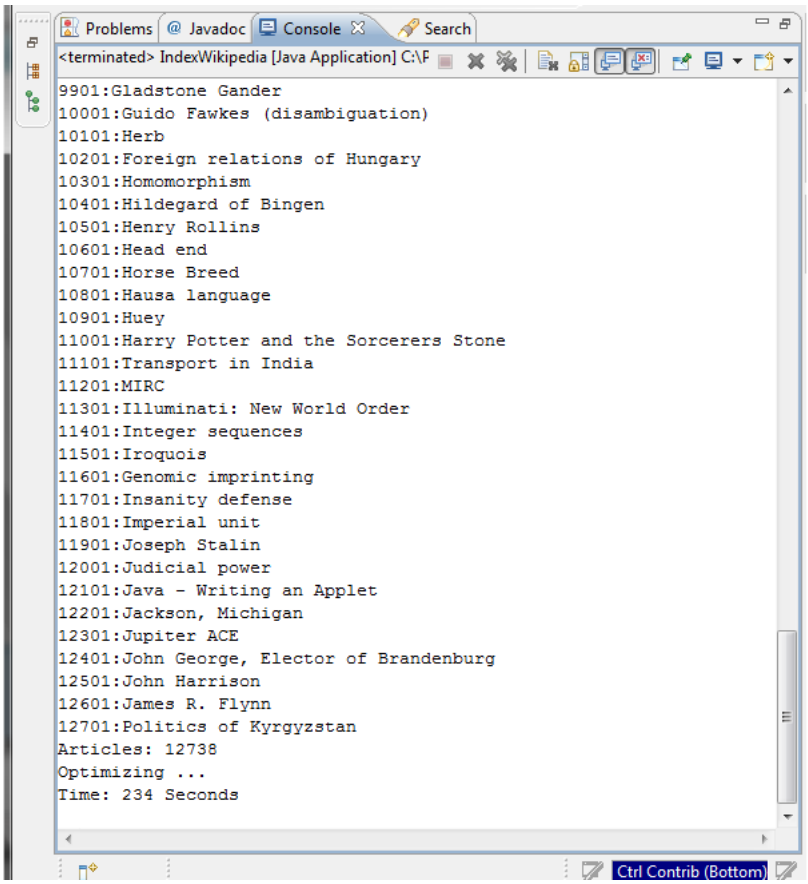

Figure 6: Output of the Wikipedia Indexer

```
>>> def searchWikipedia(queryString):
        query = parser.parse(queryString)
        hits = searcher.search(query)
        print "Hits: ",hits.length()
        for i in range(0, hits.length()):
                doc = hits.doc(i)
                title = doc.get("title")
                print i,": ",title, "score: ", hits.score(i)


>>> searchWikipedia("scream AND munch")
Hits:  6
0 :  Edvard Munch score:  0.999999940395
1 :  Afterglow score:  0.4743026793
2 :  Angst score:  0.23715133965
3 :  Fear score:  0.142290815711
4 :  August 31 score:  0.118575669825
5 :  August 22 score:  0.117710016668
>>>
```

Figure 7: Searcher Output

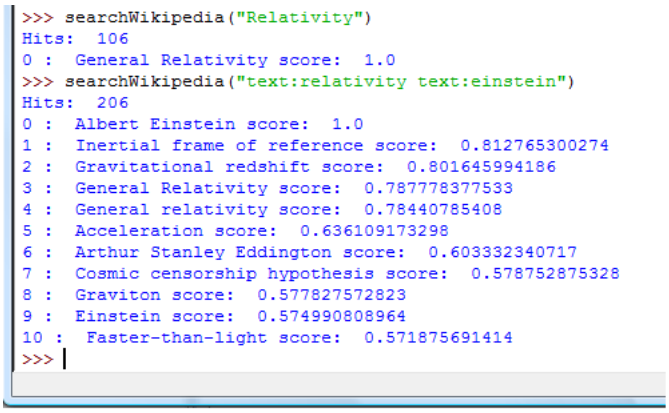

Figure 8: Comparison of Results Using a Standard Versus an Augmented Search String

Wikipedia article “Relativity” does not appear, although it should do so with a score of 1.0. The explanation for this phenomenon is quite simple: the article is not in the corpus– all experiments were applied on only $12,000$ articles, which is less than 1.7% of the actual corpus. To get a perfect hit by a single keyword is, therefore, very unlikely.

E. Open Issues

The Lucene API provides several access points through which it can be extended to a semantic search engine. Future work will determine how a document has to be represented in an index to enable a semantic search. This will involve implementation of an analyzer representing the policy for extracting index terms from the corpus. In order to match queries against documents, the analyzer will need to transform search strings into a representation compatible with that of the documents in the index. Additionally, in order to rank documents matching the query according to a scale of relevance, we will need to implement a semantically sensitive scorer.

A final issue of research is how to use WordNet for query expansion beyond the addition of synonyms. One relation between words that is worth considering is certainly the hypernym-hyponym relation. WordNet already provides a definition for this relation as a Prolog file. Therefore, a parser for the different WordNet files should be included in the implementation.

F. Topic Modeling Questions

As of today, there has been no work done to develop a search application based on probabilistic topic models. In order to utilize the model, the following issues have to be addressed:

- Compare results from Gibbs sampling with the variational inference algorithm described in Blei et al., (2003).
- Determine how model parameters can be estimated in an optimal way to reduce the time until the indexing algorithm converges and to obtain the best-fitting model for a given corpus.
- Determine how a document can be added to the corpus without reprocessing the whole database. Rebuilding the probability distributions is computationally expensive.
- Determine, how a topic distribution can be obtained for a query string that consists of a single keyword up to a whole paragraph.

The conducted experiments revealed that parameter estimation is an open issue. In particular, it is difficult to determine the optimal number of topics. Hierarchical Dirichlet Processes (Teh, Jordan, Beal & Blei, 2006) an algorithm to estimate this number directly from the corpus. Advantages and disadvantages of this procedure should be explored.

G. Necessary Steps for Implementing a Topics-based Search Engine

In order to implement a search engine that takes advantage of the topic structure of the corpus, the following steps are necessary:

1. Extend the indexing algorithm in Lucene to generate a topic-aware index.
2. Find a data model for the index and store it in a database.
3. Extend Lucene's searching algorithm to utilize the topic index.

4. Implement a web-based application that runs Lucene as underlying API and provides a convenient user interface for the search process. Ideally, the web application would be based on Tomcat, a free Java Servlet and JSP container. This would allow the search in one single application environment.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. Conclusion

The ReSEARCH project has made excellent progress so far. We have made great strides in identifying the fundamental issues involved in semantic search and how we will need to deal with them in the context of SHARE. Our next steps are to start experimentation with proxy data–the Wikipedia data referred to above–and to plan how to move towards *live* SHARE data as it becomes available. Another important aspect of the project that must be handled next is to decide what the *summary* field of the SHARE card catalog must contain.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

Ask.com. (n.d.). *http://about.ask.com/en/docs/about/ask_technology.shtml.* Retrieved 2 April 2008, from `http://about.ask.com/en/docs/about/ask_technology.shtml`

Brin, S., & Page, L. (1998, April). The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international world wide web conference* (Vol. 30, p. 107-117).

Harris, Z. S. (1954). Distributional Structure. *Word*, *10*, 146-162.

Hersh, W. R. (2003). *Information retrieval: A health and biomedical perspective*. Springer.

Johnson, J., & Blais, C. (2008). Share repository framework: Component specific and ontology. In *Nps fifth annual acquisitions research symposium.*

Langville, A. N., & Meyer, C. D. (2006). *Google's pagerank and beyond: The science of search engine rankings*. Princeton University Press. Hardcover.

Lin, D. (1998). Extracting collocations from text corpora. In *Workshop on computational terminology* (pp. 57–63). Montreal, Canada.

Lin, D. (2008, March). *MINIPAR.* `http://www.cs.ualberta.ca/~lindek/minipar.htm`. Retrieved 29 March 2008, from `http://www.cs.ualberta.ca/~lindek/minipar.htm`

Lucene-java wiki. (2008). Available from `http://wiki.apache.org/lucene-java/FrontPage?action=show&redirect=FrontPageEN` ([Online; accessed 23-March-2008])

Notess, G. R. (n.d.). *Review of altavista.*

Sapp, G. (2000). Altavista proffers updated search technology to online businesses. *InfoWorld*.

Vajda, A. (2005). *Pulling java lucene into python: Pylucene.* Available from `http://chandlerproject.org/PyLucene/Paper` ([Online; accessed 23-March-2008])

THIS PAGE INTENTIONALLY LEFT BLANK

2003 – 2008 Sponsored Research Topics

Acquisition Management

- Software Requirements for OA
- Managing Services Supply Chain
- Acquiring Combat Capability via Public-Private Partnerships (PPPs)
- Knowledge Value Added (KVA) + Real Options (RO) Applied to Shipyard Planning Processes
- Portfolio Optimization via KVA + RO
- MOSA Contracting Implications
- Strategy for Defense Acquisition Research
- Spiral Development
- BCA: Contractor vs. Organic Growth

Contract Management

- USAF IT Commodity Council
- Contractors in 21st Century Combat Zone
- Joint Contingency Contracting
- Navy Contract Writing Guide
- Commodity Sourcing Strategies
- Past Performance in Source Selection
- USMC Contingency Contracting
- Transforming DoD Contract Closeout
- Model for Optimizing Contingency Contracting Planning and Execution

Financial Management

- PPPs and Government Financing
- Energy Saving Contracts/DoD Mobile Assets
- Capital Budgeting or DoD
- Financing DoD Budget via PPPs
- ROI of Information Warfare Systems
- Acquisitions via leasing: MPS case
- Special Termination Liability in MDAPs

Human Resources

- Learning Management Systems
- Tuition Assistance
- Retention
- Indefinite Reenlistment
- Individual Augmentation

Logistics Management

- R-TOC Aegis Microwave Power Tubes
- Privatization-NOSL/NAWCI
- Army LOG MOD
- PBL (4)
- Contractors Supporting Military Operations
- RFID (4)
- Strategic Sourcing
- ASDS Product Support Analysis
- Analysis of LAV Depot Maintenance
- Diffusion/Variability on Vendor Performance Evaluation
- Optimizing CIWS Lifecycle Support (LCS)

Program Management

- Building Collaborative Capacity
- Knowledge, Responsibilities and Decision Rights in MDAPs
- KVA Applied to Aegis and SSDS
- Business Process Reengineering (BPR) for LCS Mission Module Acquisition
- Terminating Your Own Program
- Collaborative IT Tools Leveraging Competence

A complete listing of electronic copies of published research are available on our website: www.acquisitionresearch.org