INFORMATION RETRIEVAL USING LUCENE AND WORDNET

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Jhon Whissel

December, 2009
ABSTRACT

This thesis outlines the use of Apache Lucene, its subproject Nutch, and WordNet as tools for information retrieval, a science of searching through data in order to obtain knowledge that has become increasingly relevant in the Information Age. Lucene is a software library released by the Apache Software Foundation which provides information retrieval capabilities to programmers. Nutch, based on Lucene, adds Internet search functionality. WordNet is a lexical database which groups similar words into sets of synonyms, or synsets, and tracks their semantic relationships. This creates a combination of a dictionary and a thesaurus which may be browsed as such or used in software applications. Lucene and Nutch are released under the Apache Software License and are free and open source. WordNet is released under a similar license. The availability of free and open source tools such as Lucene, Nutch, and WordNet grants software developers a base from which they may create applications that can be tailored to their specifications, while simultaneously eliminating the need to create the entire code base for their projects from scratch. This practice can result in reduced programming time and lower software development costs. The remainder of this document outlines the use of these tools and then presents the methodology for the integration of their combined capabilities into a search engine capable of online information retrieval. This discussion culminates with a demonstration of how WordNet may be employed to remove search query ambiguity.
DEDICATION

This thesis is dedicated to
My father, Fred Whissel
For inspiring creativity
My mother, Barbara Whissel
For inspiring perseverance
My aunt, Carol Decker
For her unwavering support
And to all of my family
For allowing me to stand on their shoulders
So that I might reach greater heights.
ACKNOWLEDGEMENTS

I would like to extend my most sincere gratitude to everyone who has played a role in my life and education throughout the years. The guidance, support, and inspiration of these numerous individuals has helped shaped me into who I am today.

Many thanks to Dr. Pelz, Peggy Speck, Chuck Van Tilburg, and the rest of the faculty and staff of the Department of Computer Science for both sharing their knowledge as well as lending their aid whenever it was called upon. Though I was unable to attend classes with all of you, there was not a single faculty member who did not contribute in some way to my education over the last several semesters. Thanks to the Department as well for awarding me with an assistantship, providing me with the opportunity to teach at the University of Akron and an experience that I will never forget.

Special thanks in particular to Dr. Chan for agreeing to become my mentor and aiding me with my research. The crucial guidance and direction that I received has bestowed upon me the skills needed to successfully complete this thesis.

Thanks to my Aunt Carol and the rest of my family for helping me through the challenges I have faced, and for demonstrating selfless compassion and understanding.

Finally, I would like to offer my deepest thanks to my parents for raising me, instilling me with their wisdom, setting a good example for me, molding my personality, and without whose sacrifices the life I know today would not be possible.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>xi</th>
</tr>
</thead>
</table>

## CHAPTER

### I. INTRODUCTION

1. Society in Transition ......................................................... 1
2. The Information Revolution ......................................................... 2
3. Information as a Commodity ......................................................... 3
4. Impact of the Internet ................................................................. 3
5. Information Abstractions ............................................................... 4
6. Challenges of Information Management ............................................... 5
7. Information Retrieval ................................................................. 6
8. Software Development in the Information Age ......................................... 6
9. Free and Open Source Software Movements ........................................... 7
10. Open Source Today ................................................................. 8
11. Drawbacks of Traditional Search Engines ............................................. 9
12. Open Source Information Retrieval ................................................... 10
14. WordNet ...................................................................................... 11

vi
3.20. Working with Synsets ................................................................. 50

IV. CONCLUSION ...................................................................................... 53

4.1. Benefits of Open Source Development ........................................... 53
4.2. Linus' Law ....................................................................................... 54
4.3. Justification for an Open Source Search Engine ................................ 54
4.4. Potential for WordNet in Information Retrieval ................................. 55
4.5. Simpli ............................................................................................... 56
4.6. Drawbacks of Simpli ......................................................................... 56
4.7. Future Work ..................................................................................... 57
4.8. Query Ambiguity .............................................................................. 57
4.9. Disambiguation of Queries ............................................................... 58
4.10. An Example Disambiguation ............................................................ 59
4.11. A WordNet Search Engine ............................................................... 59
4.12. Potential Pitfalls ............................................................................ 60
4.13. Performance Evaluation ................................................................. 61
4.14. Query Comparisons ....................................................................... 61
4.15. Simulating WordNet ...................................................................... 61
4.16. Testing Procedure ........................................................................ 62
4.17. Initial Test Results ........................................................................ 63
4.18. Further Testing ............................................................................... 63
4.19. Additional Test Results ................................................................. 64
4.20. Final Thoughts ............................................................................... 65

BIBLIOGRAPHY ......................................................................................... 66
APPENDICES ..................................................................................................................68

APPENDIX A. APACHE TOMCAT INTERFACE .......................................................69

APPENDIX B. APACHE TOMCAT APPLICATION MANAGER .........................70

APPENDIX C. NUTCH SEARCH INTERFACE ......................................................71

APPENDIX D. EXAMPLE NUTCH QUERY .............................................................72
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Java runtime and development resources</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Apache Tomcat distributions</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Cygwin DLL release</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>Nutch architecture</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>The crawl-urlfilter.txt configuration file</td>
<td>35</td>
</tr>
<tr>
<td>3.6</td>
<td>The nutch-site.xml configuration file</td>
<td>36</td>
</tr>
<tr>
<td>3.7</td>
<td>The list.txt file</td>
<td>38</td>
</tr>
<tr>
<td>3.8</td>
<td>The Nutch crawl command</td>
<td>39</td>
</tr>
<tr>
<td>3.9</td>
<td>Apache Tomcat deployment</td>
<td>40</td>
</tr>
<tr>
<td>3.10</td>
<td>Directory structure for the query-wordnet plugin</td>
<td>42</td>
</tr>
<tr>
<td>3.11</td>
<td>The plugin.xml file for the query-wordnet plugin</td>
<td>43</td>
</tr>
<tr>
<td>3.12</td>
<td>The build.xml file for the query-wordnet plugin</td>
<td>43</td>
</tr>
<tr>
<td>3.13</td>
<td>Ant binary distributions</td>
<td>44</td>
</tr>
<tr>
<td>3.14</td>
<td>Building a Nutch plugin with Ant</td>
<td>45</td>
</tr>
<tr>
<td>3.15</td>
<td>The base build.xml for Nutch plugins</td>
<td>46</td>
</tr>
<tr>
<td>3.16</td>
<td>The updated nutch-site.xml file</td>
<td>47</td>
</tr>
<tr>
<td>3.17</td>
<td>JAWS API releases</td>
<td>48</td>
</tr>
</tbody>
</table>
3.18 JAWS setup example ........................................................................................................49
3.19 WordNetDatabase and Synset instantiation ...............................................................50
3.20 Loops for displaying synsets ........................................................................................51
3.21 Output of the loops .......................................................................................................52
CHAPTER I

INTRODUCTION

Prior to the discussion of the procedure for integrating the open source tools Lucene and WordNet into a search engine used for information retrieval, it is beneficial to offer a background on the tools themselves as well as justification of why such an endeavor is worthwhile. The first two chapters of this thesis are reserved for this purpose. Chapter one outlines the need for powerful information retrieval tools such as search engines, along with the benefits of using open source tools during the development process. The second chapter discusses the details of Lucene and WordNet, so that the purpose of each and their desirability as part of a search engine may be understood. Once this foundation is established, the remaining chapters focus on the procedure for implementing an open source search engine which makes use of both tools, and an analysis of the results.

1.1 Society in Transition

In the middle of the 20th century, it was becoming increasingly evident that many of the first world economies which had previously been based around traditional industry were shifting their focus. The number of workers holding “blue-collar” manufacturing jobs was rapidly declining. Taking the place of these lost jobs were an assortment of clerical, administrative, and management positions in offices, schools, and retail stores.
This dramatic transition from an industrial economy to a service economy resulted in an increased need for the ability to exchange and manage the data used by scientists to conduct their research and used by businesses to make prudent financial decisions. At the same time, technology was becoming a more prominent part of everyday life as new devices such as the computer appeared and became increasingly powerful, cheaper, and easier to use. The transitional phase marked the beginning of the current period of human history, one that would come to be known as the Information Age. This era has been characterized by a free flow of vast quantities of information over globally connected communications networks.

1.2 The Information Revolution

The shift from the Industrial Age to the Information Age was part of a greater information revolution that began in the middle of the nineteenth century. This revolution is often divided into three phases, with each phase featuring the introduction of new technologies which have made a significant contribution to humanity's ability to communicate [1]. Devices such as the radio, telegraph, and telephone were of chief importance during the first phase. Notable achievements from the second phase, which began in the middle of the twentieth century, were the television, the satellite, and the computer. Though still in its infancy, the third and current phase of the revolution has seen the introduction of technologies, exemplified by the Internet and the World Wide Web, both of which have had an unparalleled impact on the creation and distribution of information.
1.3 Information as a Commodity

The new technologies of the information revolution have led to an economy where data and the useful knowledge that may be extracted from it is often as equally important and valuable as raw materials were during the Industrial Revolution. It is, in fact, reasonable to assert that data is every bit as much of a raw material in this new age as iron and steel were in the previous. As awareness of the potential value of information increased, there arose a need for the development of an efficient and practical means of manipulating data. Computer systems, which rose to prominence during the early days of the Information Age, proved to be ideally suited to this task. These machines were able to execute instructions upon data as well as provide a means for its input, output and storage. With the improvement of computer technology came an ever-growing amount of data to manage. Given the newfound value now placed on useful information, it was clear that powerful software applications would need to be created to keep up with the demands of data management. One such application, the database, provided a structured means of organizing data and proved to be a critical tool of scientists and businessmen alike.

1.4 Impact of the Internet

If the hallmark of the Information Age was the ability to freely exchange information in ways never previously possible, then this paradigm was punctuated in the early 1990s when the Internet and World Wide Web began to enter the public consciousness. A globally interconnected system of computer networks, the Internet allowed its users to transfer data between computer networks on a global scale. The
World Wide Web created a system of documents on the Internet interconnected via hyperlinks, references to external pieces of information enclosed in other online web documents. In 1988, the Internet was not widely known and connected around 100 networks. This number ballooned to over 40,000 by 1995 [2]. Today the Internet, empowered by the World Wide Web and used by over a quarter of the Earth's population, has become a critical tool and stands as one of the most significant accomplishments in human history. While such a massive global network allows for an unprecedented exchange of information, its rapid growth has meant that the ability to manage its sheer volume of data so that useful knowledge may be extracted is becoming increasingly important.

1.5 Information Abstractions

While this document has referred to data, information, and knowledge thus far, it is important to make a distinction between the terms. Information is frequently represented by three basic levels of abstraction [3]. In its simplest form, information is commonly referred to as data and represents quantitative and qualitative attributes which convert physical properties into symbols. These values are often the results of observation, experimentation, and measurement. Data is the unprocessed collection of numbers, characters, or images from which information and knowledge are derived. By itself, data has no meaning other than its symbolic representation. Computer databases have become one of the most widely used methods of storing raw data. When arranged into a related collection and organized in such a manner that knowledge may be extracted from it, data is often known as information. Information is data that has been
given a relational connection, one which may allow the data to be useful. Databases are useful tools for the storage of raw data, and databases which use a relational model to organize data into information have proved to be the most popular form currently in use. At the highest level of the information abstraction is knowledge, which is data that has been refined in such a way as to become related information and then may be applied in a useful and meaningful manner. While data is difficult for humans to work with, knowledge is information that has been processed to the point where it becomes a valuable commodity with practical applications.

1.6 Challenges of Information Management

Though relational databases have enjoyed widespread adoption and are a powerful tool for the storage of information both as raw data and in more processed forms, the extraction of useful knowledge from them remains an important and increasingly complex task. Similarly, while the Internet and the World Wide Web allow for an interconnection of information at a scale never previously possible, as the volume of data stored and the number of users grows, so too does the challenge of locating and extracting useful knowledge from web pages. As a result of these concerns, a number of disciplines have now turned their attention towards the development of techniques used to search these data sources for information, resulting in a specialized field devoted to the task. Such scrutiny has led to the development of software for the management of relational databases, as well as services such as the Google Search which provide a means of navigating the World Wide Web.
1.7 Information Retrieval

The science of extracting useful knowledge from data stored in documents, relational databases, or on the World Wide Web is known as information retrieval. Due to their data-oriented nature, computers are the tools best suited for such a task. Information retrieval systems designed specifically for the purpose of locating information stored on computer systems are called search engines. The goal of search engines is to reduce the amount of time necessary to extract useful information from a data source, and to provide an interface which hides the complex details of these engines from their users. In order to interact with a search engine, a user must specify certain criteria about their subject of interest. This is known as a query. The engine then makes its best attempt to locate the most relevant search results relating to the search query. Some of the most prominent examples of search engines are the variety of online web search engines which allow users of the World Wide Web to locate web pages, images, videos, and other information over the Internet.

1.8 Software Development in the Information Age

Since computers themselves are a relatively recent invention, information retrieval is a young field with plenty of potential for growth. It is often necessary for organizations to develop proprietary search engine software custom-tailored to meet their individualized needs. Traditionally, these groups have needed to employ a team of programmers tasked with the creation of their custom search software from the ground up. Recent developments have, however, led to a number of viable alternatives to this practice. The rise of the Internet has led to a free flow of information which prior to its
adoption was unheard of. This makes possible for the first time a widespread distribution of source code coupled with a potential for global collaboration, and provides a wealth of readily available knowledge to aid in software design. One of the most notable impacts that the Internet has had on application development is the growing popularity of what is known as the open source approach to software design.

1.9 Free and Open Source Software Movements

When software developer Richard Stallman founded the collaborative free software GNU Project in 1983, it marked the beginning of the free software movement [4]. This movement was defined by a desire to replace restrictive proprietary software solutions with free alternatives, citing the perceived immorality inherent in commercial software licenses. The philosophy popularized by Stallman's free software movement reached a critical turning point in 1998, when it was announced that the source code for Netscape's then-popular Navigator web browsing application would be made available to the public. This news helped prompt a group of like-minded software developers to conduct a meeting for the purpose of discussing the implications of the release. The group determined that the free software moniker was ambiguous and unattractive to the corporate world. Rather than “free”, which held a variety of meanings in the English language as well as anti-commercial overtones, it was determined that the more commercially-friendly term “open source” should be used instead. As a result of the meeting, Bruce Perens and Eric S. Raymond founded an organization known as the Open Source Initiative (OSI), a group dedicated to the promotion of open source software. The OSI, though sharing many of the same underlying desires of the free software movement,
was quick to distance themselves from what they felt was a confrontational attitude of the elder group. One goal of the Open Source Initiative was to present the open source software model of software development as an attractive option to businesses. The group has specified a variety of criteria intended to preserve their original intentions [5]. As a result of open source advocacy, the practice has grown immensely.

1.10 Open Source Today

Open source has now come to refer to the design, development, and distribution of software for which the application's source code is made available to the general public. This distribution may be delayed until the program has been completed, however the code of open source applications is often released during creation process itself. Advocates of open source software contend that this practice lends itself to the software's rate of evolution and overall quality, as a greater number of co-developers results in a greater amount of feedback, bug reports, documentation, and design ideas contributed during the development process. Many open source projects adopt a collaborative development software model under which publicly available source code may be reviewed and modified by decentralized programmers on a large scale. Such a design approach has been popularized and made possible largely thanks to the information sharing resources provided by the Internet, and several large online source code repositories have been created. These repositories act as a centralized location used by software development teams to store and manage their projects. One of those most successful examples of such a site, SourceForge.net, now hosts more than 230,000 projects and has over 2 million registered users as of February, 2009 [6]. The open source
software development approach is popular not only with independent developers, but has also become a viable strategy in the corporate world. By utilizing open source software as a starting point, the amount of time spent on software development as well as the associated monetary costs may be significantly reduced. A study by Black Duck Software, a global provider of products and services for the acceleration of software development, has determined that to reproduce the current pool of available open source software it would take an investment of approximately $387 billion and 2.1 million people-years [7].

1.11 Drawbacks of Traditional Search Engines

While numerous web search engines have been developed since the World Wide Web was introduced at the beginning of the 1990s, most are commercially owned and few are open source. As such, these applications suffer from the same lack of transparency that the free software foundation and the Open Source Initiative were created to counteract. The algorithms of these search engines are often not available for public perusal, preventing them from being scrutinized and improved. While this practice may appear to make sense from a business standpoint, it undeniably hinders the potential for progress made in the science of information retrieval. By limiting the number of developers who have access to a search engine's source code, a number of problems may arise. With a smaller group of testers, there is an increased probability that software bugs will slip through the design process unnoticed, causing issues for the users of the search engines. A restriction is also placed on the influx of ideas which may lead to breakthroughs in search engine technology.
1.12 Open Source Information Retrieval

Fortunately, due to the growing popularity of the open source approach to software design, it has become increasingly easy to find projects for which the code is readily available. These projects span an eclectic variety of applications, meaning that no matter what the interests of the developer are, there is a decent chance of finding an open source project which suits their needs. Frequently, such projects will be accompanied by an established community of like-minded developers willing to aid in the design process and capable of providing the means to ease a programmer into the project. A number of open source search engines which fall into this category have been developed. Among the most popular and well-supported is Apache Lucene, a high-performance information retrieval library which allows software developers to add indexing and searching capabilities to their applications. For businesses or individuals wishing to customize their own search engine, without the burden of high development costs or the time investment it takes to write an application from the ground up, Lucene presents an appealing solution.

1.13 Benefits of Open Source Development

Many open source software applications have been developed to the point where they are fully functional alternatives to proprietary products and may be used “as-is” without the need for additional modification. There is nothing wrong with using these applications in such a manner, and the majority of users with no programming ability will be perfectly satisfied with the functionality provided as well as the potential to save money over commercial alternatives. For a software developer, the potential of an open
source application grows immensely due to the availability of the code for modification. Given a mature code base to work with, programmers may add or subtract features as they deem necessary, eliminate bugs, port software to different platforms, add documentation, and often find peers willing to collaborate on a project. Businesses too may benefit from a software development process supplemented with open source code. By using an existing open source application as a starting point rather than developing proprietary software from scratch, a suitable solution may be developed at a fraction of the cost, in terms of both monetary expenses and programming time.

1.14 WordNet

WordNet, a freely available lexical database for the English language, acts as a combination of a dictionary and thesaurus. Applications of the database include search, advertising, document classification, and solving other language processing problems [8]. The intention of WordNet is to map the relationships between words in a manner similar to the way the human mind stores and uses language. As the search queries used in the information retrieval process will often be entered as the query exists in the user's consciousness, WordNet is an ideal candidate for integration into a search engine, potentially adding features beyond the scope of typical engines.

1.15 Combination of Open Source Tools

With the availability of powerful open source tools such as Lucene, it becomes possible not only to fulfill the needs of a fully-featured information retrieval system, but provides a base upon which additional functionality may be placed. WordNet is just one
example of a tool that may be incorporated into an open source search engine. The idea that two such tools may be combined, ideally with results that aid the advancement of information retrieval, is representative of the principles upon which the free software movement, the Open Source Initiative, and similar organizations were founded.

1.16 Working with Lucene and WordNet

The remainder of this thesis describes in detail the use of both the Lucene search library and the WordNet lexical database, as well as their purposes in the field of information retrieval. This is followed by a detailed overview of a process by which a Lucene-powered search may be combined with WordNet functionality for use in an open source search engine capable of online web crawling through the use of the Lucene subproject Nutch. Once the process has been explained, an evaluation of the results and related work is performed, concluding with a proposal for future work based upon the foundation this paper establishes.
CHAPTER II
APACHE LUCENE AND WORDNET

The following chapter presents a more formal introduction of both Apache Lucene and WordNet. Included is a brief review of the history of both tools, as well a more in-depth discussion which details each and provides an overview of their most notable features. The purpose of this discourse is to illustrate the potential these tools have as devices with applications in the field of information retrieval. Once completed, the remainder of the thesis focuses on building on open source search engine with the potential to make use of both Lucene's indexing capabilities as well as the information stored in WordNet's lexical database.

2.1 Introduction to Lucene

Apache Lucene is a free and open source information retrieval library with full text indexing and searching capabilities. Currently, Lucene is an Apache project maintained by the Apache Software Foundation, a non-profit organization consisting of a decentralized community of software developers. The group oversees a multitude of open source projects distributed under their own Apache License. The Lucene project was initiated by open source advocate Doug Cutting, with a three month part-time development period beginning in 1997. Lucene was added to SourceForge.net under the GNU General Public License (GPL) in 2000, marking its first open source release [9]. In
the following year, Lucene was made an Apache project under the Apache Software Foundation. In 2005, Lucene was made a top level Apache project. Lucene has enjoyed considerable success for its ability to be implemented into Internet search engines and provides the search technology behind many popular web sites and applications. One of those most notable examples of a Lucene powered website is the popular web-based multilingual encyclopedia project Wikipedia [10].

2.2 Features of Lucene

The original Lucene release was written entirely in Java, though the library has now been ported to a variety programming languages including C++, C#, Delphi, Perl, PHP, Python, and Ruby. The Lucene library features full text indexing and searching capabilities, and is useful for local, single-site searches, though its high level of performance means that it may be used in larger Internet search engines as well. Among the advanced search features supported by Lucene are boolean operators, field searches, wildcard searches, and ranges searches [11]. Some of the goals of Lucene include a simple, well-documented API, support for common search engine features, and scalability. A number of Lucene subprojects have built upon Lucene, adding additional features and extending existing ones.

2.3 Lucene Drawbacks and Nutch

Lucene provides a powerful indexing and search library which may be used as a base for online search engines, however on its own the library does not include any form of web crawling or HTML parsing abilities. These features are necessary in order to
create a fully functional online search engine. Several projects have modified Lucene with the intent of adding this missing functionality. One of the most notable of these efforts is Nutch, developed by Lucene creator Doug Cutting and Mike Cafarella. Nutch began life in 2003 as a SourceForge.net project backed by its own non-profit organization. Its creators were concerned that the lack of transparency inherent in commercial search engines and the secretive nature of their internal workings allowed for the possibility of mistakes or even worse, abuse. In addition to these issues, Cutting and Cafarella felt that the lack of commercial source code accessibility was detrimental to advances in the area of search engines. Thus it was with these concerns in mind that Nutch was written. The goal of Nutch is to provide an open source search engine capable of indexing the World Wide Web as effectively as commercial search engines [12]. Like Lucene, Nutch is free and open source, allowing anyone to download, modify, and use the search engine. The project was eventually moved from SourceForge.net and added to the Apache incubator, a move which freed the project from the overhead of its independent non-profit organization. Nutch graduated from the Apache incubator and became an official Lucene subproject in June 2005.

2.4 The Lucene Query Parser

Lucene conducts its searches by means of queries entered by the user, which are then processed by the Lucene Query Parser. The Query Parser is a lexical analyzer or lexer which interprets the user's search strings as queries which may be understood by Lucene. Queries in Lucene are made up of two primary components, terms and operators. Terms may consist of both individual terms as well as phrases which combine a group of
terms enclosed by double quotation marks. In addition to basic terms, fielded data may be
stored in a Lucene index. This feature is particularly useful in that it allows users to
search specific fields, such as a title or author.

2.5 Terms and Operators

Lucene uses boolean operators to form complex queries. Among the supported
boolean operators are AND, OR, NOT, “+” and “-”. AND, which may be be substituted
by the “&&” symbol, may be inserted between terms or phrases to indicate that both
should be located. OR may be substituted by the “||” symbol and inserted between terms
or phrases to indicate that either should be located. Inserting the NOT operator between
two terms or phrases instructs Lucene to exclude the second term or phrase from the
results of the search but include the first. The “+” operator may precede a term or phrase
to indicate that it must be included in the results, while the “-” operator in the same
location specifies that the term or phrase must be excluded. Parentheses may be used to
group terms, phrases, or field values. Due to the existence of the special characters listed
above, the “\” symbol acts as an escape character, which may precede a special operator
to treat them as a regular character to be included in the query.

2.6 Term Modifiers

Range searches, conducted by enclosing two field values in square or curly
brackets, may be used to locate documents whose ranges lie within a certain upper and
lower bound. Search terms may be further modified by a variety of means including
wildcard and fuzzy searches. Wildcard searches allow the replacement of symbols in a
query term by special wildcard characters including the “?” symbol, which may be substituted for a single character and the “*” symbol, which may be substituted for multiple characters. Fuzzy searches are conducted by applying a “~” symbol to the end of a search term. By doing so, terms similar in spelling to the original term will be found. Adding a value between 0 and 1 allows for the user to define the threshold level of similarity for terms returned by a fuzzy search. For phrases, integer values larger than 1 may follow the tilde, specifying a proximity search. This kind of search will attempt to find each term of the phrase within a word proximity specified by the integer value.

Search terms may also be boosted to increase their relevance in the query. This is done by applying a “^” symbol to the end of a term, followed by a positive value.

2.7 Lucene Indexes

Lucene stores all of the necessary information about the documents upon which its searches are conducted in files called indexes. These indexes contain sequences of documents, which themselves consist of sequences of fields. Fields are named sequences of terms, and terms are simple strings. If the same term exists in multiple fields, then each is considered to be unique. Indexes in Lucene store information about terms in order to improve search efficiency. Rather than listing which terms are contained in each document, Lucene indexes list which documents contain each term. This is known as inverted indexing. Fields in Lucene indexes may be either “stored”, in which case their text is inverted or “indexed” in which case it is not. Fields may also be tokenized and placed into the index as individual tokens or placed into the index as a single literal term.
2.8 Index Segments

Lucene indexes may be divided into segments, each of which acts as a fully independent index. Segments may be searched by themselves or together with other indexes, including other segments. Documents in Lucene indexes are each assigned a document number, starting from number zero for the first document added. Subsequent index entries each have a document number of one higher than that of the previously placed document. Contained within each segment index are a list of all field names used by documents within the index, along with the actual field values for the stored documents. A term dictionary within the index stores a list of all terms which make up the fields, along with the number of documents which contain the term as well as pointers to each term's frequency and proximity data. The frequency data stores the document numbers of each document containing the term along with the frequency with which the term appears. Proximity data stores the positions where the term appears in each document. Segment indexes also contain normalization values for each field that is a part of the calculation for the score used to rank search results, term vectors for each field which store term text and term frequency, and a list of deleted documents.

2.9 The WordNet Database

WordNet is a large lexical database, a combination of a dictionary and thesaurus for the English language. Development on the project began in 1985 when a group of psychologists and linguists under the direction of George A. Miller at Princeton University's Cognitive Science Library at began developing their own lexical database which would make use of their research into the fields of psycholinguistics and
psycholexicology [13]. The goal of such a project was to create a resource which would take into account the manner in which humans process their languages. The result of the group's effort, released in 1993, was a structured lexicon which provided a more natural alternative to traditional alphabetical dictionaries and thesauri. The WordNet database may be viewed online at its official web site,

http://wordnetweb.princeton.edu/perl/webwn and may also be downloaded for offline use. Versions exist for both UNIX-like systems and Windows, along with one compatible with ANSI Prolog. Since WordNet is distributed under a BSD style license, it may be used free of charge. Development on WordNet has continued since its inception and remains ongoing at Princeton University.

2.10 Word Relationships

In WordNet, words and their relationships to each other are organized in a hierarchical manner similar to the taxonomies which may be found in the natural sciences. Words which are closely related to each other may be found in the same branch of the hierarchy's tree. Each word belongs to a set of synonyms, also known as a synset. These synsets are the foundation upon which the WordNet database is constructed. Formally, a synset is a set of one or more synonymous words that may be substituted for each other in context without changing the overall meaning of the sentence in which they are contained. Words which have multiple meanings or “word senses” appear in more than one synset. WordNet provides a polysemy count for each word which is used to track the number of synsets which contain the word. Also included is a frequency score which uses a sample text to determine how often a word will appear in a specific sense.
WordNet has grown considerably since its development began. The database is currently at version 3.0 and contains over 155,000 words which are grouped into roughly 117,000 synsets totaling approximately 207,000 word-sense pairs [14].

2.11 Word Types

Since different word types follow different grammatical rules, WordNet makes the distinction between four of the primary word types in the English language, which include nouns, verbs, adjectives, and adverbs. The nouns category contains words which refer to entities, qualities, states, actions, or concepts, and can serve as the subject of a verb. Words classified as verbs may serve as the predicate of a sentence and describe an action, occurrence, or state of existence. Adjectives are words that may modify nouns. The final word classification stored in WordNet, the adverb, is similar to the adjective and contains words which modify word types other than nouns.

2.12 Word Types Excluded from WordNet

Other than the four primary groups, the remaining word types used in the English language are excluded from WordNet for a variety of reasons. These types include determiners, prepositions, pronouns, conjunctions, and particles. Determiners, like adjectives, are noun modifiers, however they express a reference to the noun rather than its attributes. Examples of determiners are “the”, “this”, and “that”. Prepositions are words that introduce a prepositional phrase, for example the words “of”, “to”, and “with”. A pronoun, such as “him” or “her” is a word type that may be substituted for a noun. The conjunction word type connects two words, phrases, or clauses. Words which are
classified as conjunctions include “for”, “and”, and “but”. Particles are words which do not fall into any of the other traditional word classes and include “to” in its infinitive form and “not” when used as a negator.

2.13 Relationships Between Word Types

Each of the major word types used by WordNet express linguistic concepts which are distinct from the remaining groups. Because of this, each group has its own unique set of semantic and lexical connections which link that group's members. Thus, the possible relationships shared between WordNet synsets vary depending on the type of word. Noun synsets may be connected as hypernyms, hyponyms, coordinate terms, holonyms, and meronyms. Verb synset relationships include hypernyms, troponyms, entailments, and coordinate terms. Possible adjective relationships are related nouns, similar to, and participle of verb. Adjective synsets may be related as root adjectives. Words may also be connected through lexical relations such as antonyms.

2.14 Using WordNet

WordNet provides a simple interface in both its online and offline incarnations. In each, a text entry box allows the user to input search terms. Once entered, the database's search functions check for the existence of the word. If the word is found, then it will be displayed along with each of its senses and their specific definitions, as well as example sentences and frequency counts when available. If a word has senses which belong to more than one word type, then the list of definition list is separated into multiple sense categories. For example, the word “run” may be both a verb and a noun, so searching for
the term will result in both a list of verb word senses and a list of noun word senses. Sense definitions list words which share the same meaning and may be substituted for the original word without changing the overall context.

2.15 Type Relationships in WordNet

It was previously mentioned that WordNet separates its words into one of four basic word types including nouns, verbs, adjectives, and adverbs. The database is able to track semantic relationships connecting synsets that are specific to each word type, as well as lexical relationships between individual words. These relationships warrant further examination as they are integral to the fundamental concept upon which WordNet is based, which is developing a system that takes into consideration the way that human beings process their languages. By searching for a word in WordNet, not only are synsets which contain the word in various senses returned, but the semantic and lexical relationships which are applicable for each synset may be displayed as well. This allows the user to quickly find words related to their original query. The following provides further details on these possible word relationships.

2.16 Noun Hypernyms and Hyponyms

The first of the major noun relationships stored in the WordNet database is hypernymy. A word X is a hypernym of another word Y when the semantic range of word Y lies within that of word X. An example of this relationship can be shown by the synsets containing the words "banana" and "edible fruit". "Edible fruit" is a hypernym of "banana" because all bananas are edible fruits. As such, "edible fruit" would be a
hypernym of "apple" and "orange" as well. Closely related to the concept of hypernymy is hyponymy. Hyponyms are words whose semantic range lie within that of a hypernym. Going back to the previous example, the words "banana", "apple", and "orange" would all be related to the word "fruit" by hyponymic relationships, which is to say that they are all hyponyms of "fruit". If multiple words share a hypernym, as "banana", "apple", and "orange" do, then they are said to be coordinate or sister terms, the third type of noun relationship tracked by WordNet.

2.17 Noun Holonyms and Meronyms

Another noun relationship which WordNet identifies for synsets is holonymy. A word X is a holonym of another word Y if word Y represents a part or member of word X. To provide an example of this relationship, the human body consists of arms and legs, therefore the word "body" is a valid holonym of the words "arm" and "leg". The next semantic relationship for the noun type is the direct opposite of the holonymic relationship. This relationship, which is known as meronymy, refers to two words X and Y where word Y is a part of member of word X. In this case, word Y is a meronym of word X. To use the above example once more, the words "arm" and "leg" share a meronymic relationship with the word "body", as a body is made up of arms and legs. Closely related to the holonymic and meronymic semantic relationships is the concept of the domain category. Also tracked by WordNet, domain categories refer to groups of synsets whose members represent the domain to which another synset belongs. The synsets containing "animal" and "human" are domain categories of the synset containing "body" when referring to the entire structure of an organism.
2.18 Verb Hypernyms and Hyponyms

WordNet tracks several different semantic relationships for verb synsets. The first of these, the hypernymic relationship, is similar to that shared between noun hypernyms and their hyponyms. Once again, a verb X is a hypernym of another verb Y if Y is an activity which is also a kind of activity X. As an example of this form of hypernymy, the word "move" is a hypernym of the words "walk" and "run" since walking and running are both acts of movement from one location to another. WordNet does not track hyponyms for verb synsets as it does for noun synsets, instead using a different term to describe a similar concept. The second form of verb semantic relationship is the troponymic relationship. A verb X is a troponym of another verb Y if by doing activity X, you are also doing activity Y. Thus "walk" and "run" are both troponyms of the word "move" as by walking and running you are, by definition, moving. As with nouns, if two words share a common hypernym then they are called coordinate or sister terms of each other. "Walk" and "run" are coordinate terms because they share a common hypernym in the word "move".

2.19 Verb Entailments

One of the other major semantic relationship stored in the WordNet database which verb synsets may be a part of is unique to the verb word type. This relationship, known as entailment, is possible only when certain actions are taking place. A verb X is entailed by another verb Y only if by doing Y you must also be doing X. To give an example of this, the act of breathing is entailed by the acts of inhaling or exhaling. This is due to the fact that if an act of inhalation or exhalation is taking place, then by definition
an act of breathing is taking place as well. Therefore, the words "inhale" and "exhale" are entailments of the verb "breathe".

2.20 Adjective Relationships

Adjective synsets in WordNet can share several basic types of semantic relationships. One such relationship, called "similar to" by WordNet, groups similar, though not necessarily interchangeable, synsets of adjectives. The synset containing the word "shiny" is similar to the one containing the word "bright", though the terms contained in the sets have subtle differences. There is also a "see also" relationship in WordNet for adjective synsets. This relationship will display other adjective synsets related to the original query. For example, search for the word "finished" as relating to a finished product, will also return "painted", "processed", and "smooth" as results for "see also". Domain categories are also an applicable relationship for adjectives, referring to the domain in which a particular adjective may be used. As an example, the word "up" may refer to the domain of machines, having the meaning of "operating properly" when used in that sense.

2.21 Adverb Relationships

The adverb is a part of speech or word type which modifies another word type such as a verb or adjective. Modified types exclude nouns, as they are modified primarily by adjectives. These words have a single basic type of semantic relationship that is stored in the WordNet database. This relationship is the domain usage of the adverb, and refers to the domain for which the adverb is applied. To give an example, the word "well" in the
sense that it has the meaning of providing emphasis, has the domain usage of an intensifier, a noun which means the adverb modifier has little purpose other than to intensify the meaning of the noun it modifies.

2.22 Lexical Relationships in WordNet

In addition to semantic relationships that are specific to the individual word types, WordNet also connects words by means of a number of lexical and derivational relationships. While semantic relationships in WordNet apply to groups of synsets, lexical relationships apply to groups of individual words. One of the most common forms of a lexical relationship is antonymy. This concept, which can apply to both adjectives and adverbs, refers to two words that express opposite meanings when used in a given context. The words “hot” and “cold” are antonyms of each other. Another type of lexical relationship is that shared between derivational related forms. These forms are words which belong to different syntactic categories but have the same root form and are lexically related. For example, the noun “gravity” in the sense of the force of attraction between masses, has two derivationally related forms. The verb “gravitate” refers to movement due to the pull of gravity while the adjective “gravitational” refers to something that is relating to or caused by gravitation. Adjective and adverbs may share a lexical relationship that is another variation of the synonym known as the root or stem adjective, also referred to as pertainyms. This relationship refers to the base word from which an adverb is formed. The word “extreme” is an example of a stem adjective or pertainym from which the adverb “extremely” is derived.
2.23 Practical Applications of WordNet

WordNet, having been in various stages of development since 1985, is a mature project which has grown to enjoy worldwide popularity. While the original WordNet is limited to the English language, numerous other wordnets built upon the fundamental WordNet concepts have been created in a variety of languages. WordNet is now considered to be a valuable resource for researchers in linguistics, text analysis, and artificial intelligence, among others. The database has been used for machine translation, conceptual identification, query expansion, document classification, and image retrieval. Other potential applications include determining the effective similarity between various web services [16]. The large number of tasks to which WordNet may be applied has generated considerable interest in the project.
CHAPTER III

IMPLEMENTING A WEB SEARCH ENGINE

By itself, Lucene has full text indexing and search capabilities, however it lacks features which are necessary for Internet search engines, including web crawling and HTML parsing. Therefore, for Lucene to be useful as an online search tool, this missing functionality must be added. Fortunately, there is a suitable solution available in Nutch, one of the official Lucene subprojects. Built upon Lucene Java, this open source Lucene extension provides all of the features necessary for a fully functional web search engine. This chapter outlines the use of Lucene as a web search engine by way of its Nutch subproject, as well as the procedure for creating a new plugin for Nutch which allows for the addition of WordNet functionality.

3.1 Using Nutch

The remainder of this chapter will detail each of the steps necessary to set up Nutch for use as a web search engine. The target platform for this demonstration is Win32, the 32-bit Windows API. In order to use Nutch, it must first be downloaded. All Nutch releases contain the source code as well as a compiled version of the software, and may be downloaded from the Nutch branch of the Apache Software Foundation's Lucene home page, which is located at http://lucene.apache.org/nutch/index.html.
3.2 Installing the Java Environment

Nutch is written in Java, and the first step in using the software is to ensure that the Java Runtime Environment (JRE) is installed on the machine which will be hosting the search engine. This version of Java is suitable for Nutch users with no interest in making modifications. The Java Development Kit (JDK), also available, is more appropriate for software developers. JDK 6 Updates 14 was used for the purposes of this thesis. Though Nutch will run on any version of Java starting with 1.4, the newest version of either release is recommended. The Java Virtual Machine (JVM) included with both the JRE and JDK allows programs written in Java, such as Nutch, to execute using a combination of interpretation and just-in-time compilation. The latest releases of both the JRE and JDK including JDK 6 Update 14, shown in Figure 1, can be downloaded from the Sun Microsystems Java web site at http://java.sun.com/javase/downloads/index.jsp.

![Java runtime and development resources](image)

Figure 3.1 Java runtime and development resources
3.3 Selecting a Web Interface

Once Java has been installed on the machine which will host the web search engine, the next step for setting up a functional web search engine is to choose an interface which will allow users to perform searches. One application capable of filling that role is Apache Tomcat, an open source Apache Software Foundation project. Tomcat is a servlet container and provides a Java HTTP web server environment in which Java code may run. For the purposes of this thesis, the 6.0.20 version of the core Tomcat distribution shown in Figure 3.2 was used. Apache Tomcat may be downloaded from its home page at [http://tomcat.apache.org/](http://tomcat.apache.org/).

![Figure 3.2 Apache Tomcat distributions](image-url)

**6.0.20**

Please see the README file for packaging information. It explains what every distribution contains.

**Binary Distributions**

- Core:
  - zip (pgp, md5)
  - tar.gz (pgp, md5)
  - Windows Service Installer (pgp, md5)
- Deployer:
  - zip (pgp, md5)
  - tar.gz (pgp, md5)

**Source Code Distributions**

- tar.gz (pgp, md5)
- zip (pgp, md5)
3.4 Installing a Shell Environment

The final tool necessary for setting up Nutch in a Windows environment is a shell which will be used to send commands to the web crawler. Cygwin, a Unix-like shell environment and command-line interface for the Win32 platform will be used for this role in this example. This tool was originally developed by Cygnus Solutions, now a part of Red Hat, inc. Red Hat continues to maintain the application, which is released under version 2 of the GNU General Public License. As such, the source code is available for users who wish to make modifications to the software. The DLL 1.5.25-15 release of Cygwin, shown in Figure 3.3, was used for this thesis. The newest version of Cygwin may be obtained from http://www.cygwin.com/.

Figure 3.3 Cygwin DLL release

3.5 Web Crawling with Nutch

After making sure that the necessary tools are installed, it is now possible to use Nutch/Lucene as a fully functional search engine with web crawling capabilities. Nutch provides a means for crawling both specific web domains as well as unfocused large scale web crawls. Domains are smaller subsections of the Internet consisting of web resources grouped under a common name. They are frequently used by businesses, other
organizations, as well as individuals to associate themselves with a certain part of the web. The URL “apache.org” is one such example of an Internet domain and is used to identify and access the Apache Software Foundation's web resources. Many organizations whose web domains encompass a large number of URLs will provide a search engine for the purposes of locating information within their specific domain. These smaller search engines provide a good example of those which make use of a web crawl confined to a specific domain. Conversely, the most prominent example of a search engine which instead produces results based upon large scale web crawls without a specific domain target is Google. The architecture of Nutch is shown in Figure 3.4.

Figure 3.4 Nutch architecture
3.6 Large Scale vs. Small Scale Crawls

There are many benefits to crawling a limited number of specific domains as opposed to the entire Internet, as well as several drawbacks. If a search engine user knows that the information they are looking for may be found at a specific location, then it would be unnecessary to look elsewhere. In such situations, small scale web crawls confined to a relatively low number of domains would be appropriate. Also, there is an obvious implied speed increase to be had when limiting searches to a small number of domains rather than numerous ones. As the number of URLs subjected to a web crawl grows, so too does the amount of disk space which must be allocated on to store the results. This is another reason why web crawling a large number of URLs is not always the most desirable choice. The cost of maintaining a large scale web search engine can be prohibitively high as well, depending on the desired performance [17]. Finally, due to the mercurial nature of the Internet, the longer a web crawl takes to complete, the greater the chance that the information found during the crawl has become outdated, that URLs found during the crawl no longer exist, or that more relevant URLs have been added in the mean time.

3.7 Advantages of Large Scale Crawls

In spite of the vast amount of time needed to perform a large scale web crawl, there are certain reasons to do so. If a user does not know the location of the information they are seeking, then conducting a search based upon the results of a large scale web crawl becomes more suitable. Also, given the vast nature of the Internet, heavily trafficked search engines such as Google that make use of large scale web crawls are
necessary to provide users with a means of accessing as much of the Web as possible from a single location. This practice prevents search engine “islands” whose web crawling results may be dramatically different from those of similar sites. While the specific search approaches taken by large web search engines such as Google may differ and produce unique results, large scale web crawls ensure that the list of sites being crawled by each engine to retrieve those results is relatively similar. The Open Directory Project (ODP) at http://www.dmoz.org/ provides data which acts as a directory of World Wide Web sites used by many of the largest search engines.

3.8 Preparing Nutch for Crawling

Before evaluating a fully functional web search engine, it is advisable to ensure that it works with a limited number of web sites. Thus, for the purposes of this thesis a limited domain crawl was used in favor of an unfocused large scale web crawl. In order to set up such a crawl so that a search may be conducted, the domain(s) which should be crawled must be specified. In the Nutch base directory, there is a subdirectory called “conf” which contains the search engine's configuration files. The “crawl-urlfilter.txt” file located in the directory consists of lines that filter URLs for the web crawler. Each line is a regular expression that is prefixed by either a '+', which specifies that any URLs that match the expression should be included in the crawl, or a '-', which precedes expressions whose matching URLs should be excluded. The line ‘+^http://([a-z0-9]*\.)apache.org/’ was added to the file for the purposes of testing the Nutch web crawler and allows the entire “apache.org” domain to be included in the crawl. The layout of the 'crawl-urlfilter.txt' file is displayed in Figure 3.5.
3.9 Configuring Nutch

While the configuration directory contains a number of files which specify the behavior of the crawler, only one other needs to be modified before a crawl can be conducted. This file, “nutch-site.xml” allows for the overriding of Nutch configuration properties present in the other .xml files present in the “conf” directory. The “nutch-default.xml” includes several properties identifying the crawler. Internet etiquette dictates that these properties, which include “http.agent.name”, “http.agent.description”,

Figure 3.5 The crawl-urlfilter.txt configuration file

# The url filter file used by the crawl command.

# Better for intranet crawling.
# Be sure to change MY.DOMAIN.NAME to your domain name.

# Each non-comment, non-blank line contains a regular expression
# prefixed by '+' or '-'. The first matching pattern in the file
# determines whether a URL is included or ignored. If no pattern
# matches, the URL is ignored.

# skip file:, ftp:, & mailto: urls
-^\(file|ftp|mailto)\):

# skip image and other suffixes we can't yet parse
-\.
\(gif|GIF|jpg|JPG|png|PNG|ico|ICO|css|sit|eps|wmf|zip|ppt|mpg|xls|gz
|rpm|tgz|mov|MOV|exe|jpeg|JPEG|bmp|BMP)\$

# skip URLs containing certain characters as probable queries, etc.
-[^\!@=]

# skip URLs with slash-delimited segment that repeats 3+ times, to
# break loops
-.*\([/\^]+\)\^\+1\[^/\]+\1/

# accept hosts in MY.DOMAIN.NAME
+^http://\(a-z0-9\)*\.ncbi.nlm.nih.gov/

# skip everything else
-.
“http.agent.url”, and “http.agent.email” should be added to the “nutch-site.xml” file. The name field should indicate the name of the organization associated with the crawler, description should detail the purpose of the web crawler, the url field should point to a URL offering an explanation of the crawler's purpose and behavior, and email should provide an address for contacting the web crawler's handlers. These configuration properties are shown in Figure 3.6 and Figure 3.7.

```xml
<?xml version="1.0"?>
<!DOCTYPE configuration PUBLIC "-//Apache Software Foundation//DTD Config 2.1//EN" "configuration.dtd">
<configuration>
  <property>
    <name>http.agent.name</name>
    <value>jfw13</value>
    <description>Jhon's Nutch Crawler</description>
  </property>
  <property>
    <name>http.agent.description</name>
    <value>Jhon's Nutch Crawler</value>
    <description>Jhon's Nutch Crawler</description>
  </property>
</configuration>
```

Figure 3.6 The nutch-site.xml configuration file
3.10 Creating a URL List

With the configuration of the web crawler complete, the final step before conducting the crawl itself is to list which URLs to use as starting points. A new directory should be created in the default Nutch directory. The name of the directory may be whatever the user wishes, however since it will be used as part of the shell command that will begin the crawl, shorter names are preferable. Once the directory has been created, a blank text file should be added. This file should contain the list of all base URLs which will be used in the crawl. For example, the directory may be named “urls”. The contents of the directory may include a text file called “list.txt”, which has the website of the National Center for Biotechnology information, “http://www.ncbi.nlm.nih.gov/”, as an
entry. By running a web crawl using this directory in the shell command, “http://www.apache.org/” would act as the base URL for the web crawl. The sample list.txt referred to above is shown in Figure 3.8.

Figure 3.7 The list.txt file

3.11 Performing a Nutch Crawl

To begin conducting the web crawl, a single line command must be issued from the shell environment. Cygwin was used to issue such commands for the purposes of this thesis. The command used to initiate the web crawl is “bin/nutch crawl” followed by a number of parameters, some optional. All the information gained during the crawl must be stored somewhere, and the first parameter specifies the name of the folder which contains the list of base URLs for the crawl. The “-dir” parameter should specify the name subdirectory of the root Nutch directory in which crawl information will be placed. Using the parameter “-depth N” where N is replaced by an integer value allows the user to limit the depth of the crawl. This depth is the number of levels beyond the root directory which the crawler will follow. The “-topN N” parameter indicates the number of pages which may be retrieved at each level of the crawl. Again, N should be replaced by a specific integer value, as in “-topN 100”. The line “bin/nutch crawl urls -dir crawl..."
-depth 5 -topN 100” is an example of a complete Nutch command. This would instruct Nutch to search from the base URLs listed in the “url” directory, using a depth of 5 and retrieving 100 pages per level then placing the results into the “crawl” directory. Figure 3.9 shows a command in the Cygwin interface which will run Nutch using urls as the directory containing the list.txt file, crawl as the directory where the results will be placed, and maximum crawl depth level of 10. The output from this crawl will be directed to the crawl.log file.

![Figure 3.8 The Nutch crawl command](image)

3.12 Searching the Crawl Results

Once the web crawl has been completed, its results will be placed into the directory specified by the user in command line interface. Optionally, the integrity of the crawl data may be verified using the NutchBean utility distributed with Nutch. Apache Tomcat provides a sufficient interface for the search engine but must be associated with
Nutch prior to use. The Tomcat interface will be displayed by visiting
“http://localhost:8080/” from a web browser. This interface may be seen in Appendix A.
On the left side of the initial screen is a menu labeled “Administration”. One of the
options listed is the Tomcat Manager, which shows the various web applications installed
in Tomcat. By selecting this option, a list of applications and their status will be
displayed. To add Nutch to the list, the browser button under the “WAR file to deploy”
section should be selected. This is illustrated in Figure 3.10. Navigating to the Nutch root
directory and selecting the web application archive (.war) file there will create a new
subdirectory Tomcat's “webapps” directory.

![WAR file to deploy](image)

Figure 3.9 Apache Tomcat deployment

If everything has been set up properly, Nutch will be present in the list of running
applications. Appendix B shows the Apache Tomcat Manager with "nutch-1.0" listed in
the list of running applications. The final step for using Nutch in Apache Tomcat is to
direct the Nutch search interface to the directory created by the web crawl. This may be
done by navigating to the “WEB-INF\classes” subdirectory of Tomcat's Nutch directory
then editing the “nutch-site.xml” file and configuring value of the “searcher.dir” property
to be the Nutch crawl directory. Once this has been completed, nutch may be selected in
the Tomcat Web Application Manager and the Nutch search interface will be displayed. From this point, a Lucene-powered search of all web sites visited during the crawl is possible.

3.13 Modifying Nutch

The open source nature of Nutch makes it ideal for modification. This potential for customization is further aided by the modular nature of Nutch. The Nutch search engine is built upon a basic code backbone which is augmented heavily through the use of plugins. Plugins are program extensions which are be added to a host application. The release version of Nutch contains dozens of plugins which may be added or removed as desired by changing the Nutch configuration. These plugins are responsible for the parsing of different file types during the crawl, indexing of crawl results, protocols through which the crawl can operate, and querying of indexed crawl results, among other tasks. Essentially, the majority of the primary search engine functions are performed by plugins. Therefore, modifying the search engine may be accomplished by changing the configuration of the plugins, which may include adding new plugins. In order to add WordNet-related functionality to Nutch, a plugin should be created.

3.14 Creating a Nutch Plugin

One of the Nutch plugin types is the query type, the group of plugins which runs queries against various fields of the index. Since WordNet deals with searches, this type of plugin makes a good candidate for the addition of WordNet functionality. To create a new plugin for Nutch, a new folder should be created inside of the “src/plugin” directory.
The name of the folder should describe the plugin, prefixed by the name of the plugin group the new addition will belong to, if applicable. A new query plugin for WordNet might be named “query-wordnet” for example. The contents of this new directory should include three items, a file named “plugin.xml” which includes information about the plugin for Nutch, a file named “build.xml” used by the Apache Ant program for building the plugin .jar file, and another directory named “src”. Inside of the src folder will be a directory/package tree structure. A new WordNet query plugin for the Nutch community might have the directory “java/org/apache/nutch/searcher/wordnet/” for example. All .java source files for the plugin should reside in this directory. The package for this plugin would be “org.apache.nutch.searcher.wordnet”. Figure 3.11 shows the directory structure of the "query-wordnet" plugin, while Figure 3.12 and Figure 3.13 show the contents of the plugin's build.xml and plugin.xml files.

Figure 3.10 Directory structure for the query-wordnet plugin
3.15 Building a Plugin

Once the plugin has been written, it must be built and packaged for deployment. Apache Ant is a tool which may be used for the automated building of Nutch plugins. This tool may be downloaded from “http://ant.apache.org/bindownload.cgi”. Version 1.7.1, used for this thesis, is shown in figure 3.14.
The “build.xml” file in each plugin's root directory contains build instructions for Ant. When Ant has been properly configured, building a plugin simply involves navigating to the folder containing the build file via the command prompt then inputting the “ant” command. A successful build of the plugin using Ant is illustrated in figure 3.15.
Figure 3.14 Building a Nutch plugin with Ant

To integrate the new plugin into the existing Nutch build, a line must be added to the global Nutch “build.xml” file, which is located in the root plugin directory. This line will instruct Ant to add the new plugin directory to the global build and is shown in figure 3.16.
In the root Nutch directory, there is another “build.xml” file. This is the primary build file and by running the Ant command on it, all plugins including the newest addition will be compiled and turned into .jars. The final step required before a new plugin may be used is to instruct Nutch to make use of the plugin. To do so, the “nutch-site.xml” configuration file must be modified. A property named “plugin.includes” should be added if it does not already exist. The value for this plugin should be a regular expression containing the list of plugins which will be used by Nutch. Any new plugins should be added to this value. An updated version of the "nutch-site.xml" file which contains the new query plugin is shown in Figure 3.17.

Figure 3.15 The base build.xml for Nutch plugins

```xml
<ant dir="parse-swf" target="deploy"/>
<ant dir="parse-text" target="deploy"/>
<ant dir="parse-zip" target="deploy"/>
<ant dir="query-basic" target="deploy"/>
<ant dir="query-more" target="deploy"/>
<ant dir="query-site" target="deploy"/>
  <ant dir="query-custom" target="deploy"/>
<ant dir="query-url" target="deploy"/>
<ant dir="query-wordnet" target="deploy"/>
<ant dir="response-json" target="deploy"/>
<ant dir="response-xml" target="deploy"/>
<ant dir="scoring-opic" target="deploy"/>
  <ant dir="scoring-link" target="deploy"/>
<ant dir="summary-basic" target="deploy"/>
<ant dir="subcollection" target="deploy"/>
<ant dir="summary-lucene" target="deploy"/>
<ant dir="tld" target="deploy"/>
<ant dir="urlfilter-automaton" target="deploy"/>
<ant dir="urlfilter-domain" target="deploy"/>
```
3.16 Deploying a Plugin

After modifying the “nutch-site.xml” configuration file, any plugins relating to the crawl should be ready for use. A query plugin is used after the crawl via Apache Tomcat, and thus a new deployment must be prepared. Running the command “ant war” from the same directory will create a new web application archive (.war) file containing the updated Nutch. From the apache Tomcat Application Manager, this new .war file may be deployed. If a previous version of Nutch is deployed, the new version must have a different name. Once deployed, the updated Nutch may be selected from the list of running applications. This will display the query screen, which will now make use of the new plugin. The interface used to for searching with Nutch is displayed in Appendix C. When the user enters a query, the results of the search are displayed. The example results of a successful search using the crawl index built from the command in Figure 3.9 is shown in Appendix D.
3.17 Developing with WordNet

A number of WordNet APIs are available in the related projects section of the official WordNet web site. These interfaces allow developers to make use of WordNet's capabilities in their programs. The Java API for WordNet Searching (JAWS) written by Brett Spell is one such API. This API's runtime library as well as its source code may be downloaded from [http://lyle.smu.edu/~tspell/jaws/index.html](http://lyle.smu.edu/~tspell/jaws/index.html). Version 1.3, shown in Figure 3.18, was used for this thesis. JAWS contains a variety of classes and methods which may be used to access the WordNet database in Java.

![JAWS API releases](image)

Figure 3.17 JAWS API releases

3.18 Using the JAWS API

After downloading the JAWS runtime library, the API may be used by adding its Java Archive (.jar) file to the class path of the application which will be using WordNet.

48
A statement which imports “edu.smu.tspell.wordnet” should be included in .java source files making use of JAWS. The path to the WordNet database should also be specified by setting the system property “wordnet.data.dir” to the “dict” subdirectory of the root WordNet directory. The WordNetTest.java file displayed in Figure 3.19 contains the code necessary to import the JAWS API and set the WordNet dictionary directory.

```java
/** WordNetTest.java
  * John Whissel
  * Program for testing Java API for WordNet Searching (JAWS)
  */

import edu.smu.tspell.wordnet.*;

public class WordNetTest {
    public static void main ( String args[] )
    {
        final String WORDNET_DIRECTORY = "c:\program files\WordNet\2.1\dict\";
        System.setProperty("wordnet.database.dir", WORDNET_DIRECTORY);
    }
}
```

Figure 3.18 JAWS setup example

3.19 Instantiating a WordNet Database

The WordNetDatabase class provides access to the information stored in the WordNet database and must be instantiated before use. A method, getFileInstance, returns an implementation of the class that works with the local WordNet database and may be used when creating a new instance of the WordNetDatabase class. Other than WordNetDatabase, another critical component of the JAWS API is the Synset interface. This interface represents WordNet's collections of related words, or synsets. These synsets are stored as an array of word forms. Several overloaded methods of the
WordNetDatabase class known collectively as getSynsets can be used to retrieve synsets from the WordNet database by providing a starting word in the form of a string when the getSynsets is called. When instantiating a synset, the getSynsets method is used to populate the new instance of the Synset interface with WordNet information. Figure 3.20 illustrates the instantiation of a WordNetDatabase as well Synset arrays for the noun forms, verb forms, adjective forms, adverb forms, and all forms of the word "break".

```java
WordNetDatabase database = WordNetDatabase.getFileInstance();
final String TEST_WORD = "break";

Synset[] nounTypes = database.getSynsets(TEST_WORD, SynsetType.NOUN);
Synset[] verbTypes = database.getSynsets(TEST_WORD, SynsetType.VERB);
Synset[] adjectiveTypes = database.getSynsets(TEST_WORD, SynsetType.ADJECTIVE);
Synset[] adverbTypes = database.getSynsets(TEST_WORD, SynsetType.ADVERB);
Synset[] allTypes = database.getSynsets(TEST_WORD);
```

Figure 3.19 WordNetDatabase and Synset instantiation

3.20 Working with Synsets

Methods of the Synset interface are used to retrieve a variety of information from the WordNet database pertaining to the word with which the Synset was instantiated. These methods include means of retrieving word antonyms, definitions, derivationally related forms, tag counts, synset type, usage examples, as well as word forms, which are stored in arrays of Strings. To give an example, if a new Synset instance was created for the word “throw”, the getSynsets method could be used to retrieve just the synsets for the verb form of the word. All relevant WordNet information for “throw” would be stored as as elements of the Synset array, with each element of the array representing a single...
WordNet synset. The getWordForms method may be used to retrieve the individual groups of word forms for each synset stored as an element of this array, which may themselves be stored as arrays of strings containing all words similar to the original word. An example synset for the word “throw” contains word forms which include “break”, “separate”, “split up”, “fall apart”, and “come apart”. By using the getDefinition method for the Synset, the String “become separated into pieces or fragments” is returned, identifying the meaning of the synset which unites the word forms. The code used to print out the noun synsets instantiated in Figure 3.20 is shown below, in Figure 3.21. This code prints all noun synsets for the word "break", one at a time. The synset type is displayed for each set, along with its members and the definition of the set itself. This resulting output is shown in Figure 3.22.

```java
for (int i = 0; i < nounTypes.length; i++)
{
    String[] wordForms = nounTypes[i].getWordForms();
    System.out.printf("Synset Type - %s\n", wordType[nounTypes[i].getType()].toString());
    System.out.printf("Synset Members - ");
    for (int j = 0; j < wordForms.length; j++)
        System.out.printf("%s", (j > 0 ? ", " : ""), wordForms[j]);
    System.out.printf("\nSynset Definition - %s\n", nounTypes[i].getDefinition());
}
```

Figure 3.20 Loops for displaying synsets

51
Figure 3.21 Output of the loops
CHAPTER IV
CONCLUSION

The goal of this thesis was to implement a fully open source search engine which features Lucene-powered queries and the potential to make use of WordNet functionality. Chapter three detailed the procedure for doing so, and the remainder of the paper is devoted to the evaluation of the engine which resulted from the work outlined therein. Also included is a discussion of Simpli, a previous effort to produce a search engine with the capabilities of WordNet. Finally, a proposal of potential future work that would build upon the findings of this document is made. This would aim to develop a more fully-featured WordNet search engine built using an extension of the techniques for integrating Nutch and WordNet.

4.1 Benefits of Open Source Development

As this thesis demonstrates, by making use of the wealth of open source resources available on the Internet, it becomes possible to add functionality to or adapt previously existing features of software applications such as search engines. While it is not possible for the results of such projects to always lead to significant improvements, by providing the opportunity for collaboration and development conducted by a community rather than a private team, open source makes it possible for a software application to experience growth which might otherwise be difficult to achieve. The benefits of open source
development can include reduced costs both in terms of time and monetary amounts as well as the potential for increased exposure by means of online distribution.

4.2 Linus' Law

In addition to creating a convenient means by which users may access a project and its source code as well as collaborate with its community and aid in its development, the Internet also makes it possible for users of wildly different perspectives and backgrounds to do. Global availability of a project is conducive to the elimination of software flaws, a repercussion of large scale peer review known as Linus' Law. Named after Linus Torvalds, the founder of notable open source project Linux, Linus' Law states that “given enough eyeballs, all bugs are shallow”, meaning that given a large enough community of developers, it becomes possible to quickly fix problems with a software application [18]. The benefits of large scale peer review stand as one of the greatest advantages open source projects have over their private counterparts.

4.3 Justification for an Open Source Search Engine

Lucene, and more specifically its extension Nutch, are notable examples of large open source projects which can be used to perform critical tasks such as web searches. Due to the immense and growing popularity of the Internet, these searches are critical for navigation of the World Wide Web. Google search is currently the most popular search engine on the Internet, yet it is owned and maintained by a corporation which earns revenue from its advertisements. While there is nothing fundamentally wrong with this practice, Google search is essentially a valuable commodity to its owners. With that in
mind, it is understandable that the inner workings of such a tool are held secret from the general public, lest they be copied by the competition. Projects such as Lucene and Nutch, however, are completely transparent, meaning that it is possible for anyone with interest to review the details of the source code and the algorithms which are present therein. In addition to promoting improvements in the software and the removal of errors in the code, this transparency also helps to eliminate the possibility of unethical conduct by the developers. The presence of smaller and open source alternatives to major software applications such as Google search can also serve as a way of preventing monopolies by their larger brethren. In addition to this, the presence of competition can facilitate advancements in both the open source and established products, as it is difficult for any application to enjoy success when it has a sizable disadvantage in terms of functionality.

4.4 Potential for WordNet in Information Retrieval

WordNet, even when limited as a standalone replacement for a dictionary and thesaurus, represents a powerful tool, yet the potential it has as an augmentation to a search engine or similar application is immense. This thesis has detailed the process for the use of WordNet itself as well as a way by which WordNet may be used with the Nutch search engine, yet this serves merely to scratch the surface of the possible uses for the lexical database. Google maintains a list of the 100 searches which are currently most popular with its web search users, and the vast majority of these searches contain nouns. As such, a simple example of an application for WordNet when integrated into a search engine would be to show alternative words belonging to synsets for the search query. This
technique could lead to results which are related to the users query, but may have been omitted from the original results because there was not an exact term match.

4.5 Simpli

Attempts have been made in the past to integrate the capabilities of WordNet into a search engine, perhaps the most notable of which is known as Simpli. Created in 1999 by WordNet founder George Miller along with a number of professors, post-doctoral fellows and graduate students at Princeton University, Simpli was meant to lessen the gap between the intentions of a user's search query and the results returned by the search engine. The engine was quite successful, attracting the attention of a number of larger companies. Simpli's technology was first sold to the Internet service provider Netzero in 2000 and then acquired by the pay-per-placement directory service Search123 in 2002, which itself was acquired by the online advertising company ValueClick, Inc. in 2004 [19].

4.6 Drawbacks of Simpli

Today, Simpli.com is an online shopping web site which makes use of the eponymous search engine's technology. Simpli stands as an excellent example that a WordNet powered search engine can be both useful and commercially attractive. There are, however, several notable disadvantages of the Simpli implementation of a WordNet search engine. The first is that it is privately held, and as such suffers from the same disadvantages as any such software product. Another drawback of Simpli as it exists today is that the engine has strayed somewhat from its original intentions. After years of
being purchased by various owners, Simpli is now a specialized search engine for online gift shopping. Because of this, much of the potential WordNet has as a tool for general purpose information retrieval is lost.

4.7 Future Work

In order to take fully advantage of WordNet's capabilities in a search engine, an open source project which makes use of the lexical database could be created. By making such an effort open source, a greater number of developers would have the opportunity to inject their own ideas and eliminate flaws. Given enough exposure, it should be possible for a functional search engine to form, and ideally the process will lead to new ideas which serve to advance the field of information retrieval. This thesis has outlined the use of several open source tools to create the foundation of such an engine, however the resulting plugin serves merely as an example that such a fusion is possible rather than a fully-realized implementation. A complete WordNet search engine would be a considerable undertaking, one which is beyond the scope of this document, and a number of considerations must be made when designing the resulting application.

4.8 Query Ambiguity

One of the primary problems with the typical search engine is that it is simply not currently possible for a machine to think in the same manner as a human. When the user of a search engine enters a query, they have a good idea of what they would like the results to be, but it is quite likely that the software is not capable of understanding the query well enough to be able to return those results. The reason for this is that a typical
search query is often ambiguous, having any number of potential meanings. A search engine user should know which meaning the query has, but a standard search engine will attempt to match the term itself, disregarding any multiple meanings the term may have. As a result, there is a high probability that search results will be diluted, returning numerous results that may be applicable for other meanings of the query term, but unsuitable for the one entered by the user.

4.9 Disambiguation of Queries

WordNet is built around the idea of separating words into multiple word senses, each of which represents a single meaning of the word. Groups of synonyms or synsets contain similar words which have the same word sense. These word senses remove the ambiguity associated with a word and synsets provide a variety of other words sharing the same meaning. The concepts of both word senses and synsets are the key to removing ambiguity in search queries. If a search engine user is provided with a means of specifying which sense of a word they are using in their query, then it becomes possible for the engine to produce results which are more closely related to what the user is searching for. This may be done by checking the word sense definitions or synsets for a query term and including one or more supplemental terms in the search. The results are then ranked with the supplemental terms in mind, meaning that results associated with the specified word sense should be assigned higher priority and thus displayed to the user first.
4.10 An Example Disambiguation

One well-known example of an ambiguous term is the word "java". This word has three distinct word senses in modern society, the first being an Indonesian island, the second being a synonym for coffee, and the third being an object-oriented programming language featured prominently in this document. If the term "java" was entered into a search engine by itself, there is an ambiguity problem. However, if the user was able to select their desired word sense, then a supplemental search term could help refine the search results. For example, if the user indicated that they wished to search for the island of Java, then "island" would likely be a supplemental term sufficient enough to eliminate most potential confusion on the part of the search engine. This would prevent results pertaining to programming or coffee from being displayed before more other, more relevant results. Likewise, the terms "coffee" would be a suitable supplemental term for producing results related to "java" in the sense of the beverage, and a supplemental term of "programming" would allow for the retrieval of results relating to "java" in the sense of the language.

4.11 A WordNet Search Engine

Now that the case for a WordNet search engine has been made and the procedure for creating an open source version of such an engine has been outlined, this paper concludes with a brief proposal for a more formal engine. The standard single text box for entering queries is used as part of the WordNet interface and would be sufficient for simple searches as well. Terms entered into this box would be treated as WordNet queries, allowing for a list of word senses and synset information to be retrieved. For
terms with only one sense and no other synset members, the search would proceed normally. Terms which have only one sense but multiple synset members would have the additional members added to the query, with priority given to the original term. This would allow for the retrieval of results that are related to the original query but would not match the original term. In the case of queries with multiple word senses, a list of WordNet word senses for the term would be displayed, allowing the user to select which sense applied to their term. Again, if the selected sense included a synset with multiple words, these words would be added to the query, albeit with a slightly lower priority than the original term. Also added, but with a much higher priority than synset members, should be one or more supplemental terms of the type previously described. These terms would effectively disambiguate the original query.

4.12 Potential Pitfalls

The biggest obstacle for such an engine to overcome would likely be choosing the supplemental terms of disambiguation. Ideally, a single word associated with each word sense would serve this purpose. Candidates could be drawn from other synset members for some word senses, but in the case of word senses with synsets containing a single word, another alternative would be needed. These would likely have to be drawn from the sense definitions stored in WordNet. Associating word senses with a unique term of disambiguation would prove to be a time-consuming process, yet it would be one with a number of practical applications including but not limited to search engines. This is something which could be included in future releases of WordNet or stored as part of the search engine itself.
4.13 Performance Evaluation

While it is naturally difficult to judge the performance of the proposed engine, it is possible to estimate the value of such an engine using existing methods. By using an established search engine such as Google for testing, the relevance of search results as compared to the original query may be determined. Google has an unparalleled pool of resources from which to draw its results due to its efforts to crawl as much of the World Wide Web as possible, and as such is an ideal solution for testing the benefits of applying a secondary term of disambiguation to a user's query.

4.14 Query Comparisons

In order to perform a comparison of a standard query, which contains an ambiguous search term, and a modified query from which the ambiguity has been removed, searches must be performed multiple times and the relevance of the results counted for each search. By comparing the results of the search with the intended word sense of the user's query, it becomes clear to what degree the results match the user's expectations. If a user enters an ambiguous search term and the results have nothing to do with the sense of the word which the user had in mind when performing the search, then clearly the search engine has failed to provide an ideal experience and can be improved in some way.

4.15 Simulating WordNet

As discussed at length in this document, WordNet's capabilities can be applied to search engines, with the purpose of potentially removing query ambiguity. The proposed
method of doing so involves searching WordNet for a query term, displaying a list of word senses relating to the term and allowing the search engine user to choose which sense they intend to search for. Once this information is known, a second term is applied to the query and the search engine uses both terms to return its results, which would ideally be more relevant to the user's search than the results of a similar search without any form of disambiguation. Performance evaluation of the proposed search engine was performed by comparing the number of relevant search results using the original term versus the number of resulting using the modified query. All queries were entered into the Google search engine. While the inner workings of Google as well as the size of the index from which the results are drawn may differ when compared to a typical Nutch implementation, the principal of the disambiguation is the same regardless of the engine used to test its effectiveness.

4.16 Testing Procedure

For testing, a number of ambiguous terms were chosen. Each was input into the Google search engine. The top ten results were examined to determine which sense of the term they related to. Using the WordNet database, word senses for the terms were retrieved. A single term of disambiguation which would be suitable for differentiating individual word senses was chosen for each sense. New queries were constructed by using the original term along with the added term of disambiguation. These queries were input into the Google search engine and the top ten results evaluated to determine which senses they pertained to.
4.17 Initial Test Results

The first term chosen for evaluation was the aforementioned example of an ambiguous search query, the term "java". As discussed earlier, this term can have any of three very distinct word senses and thus should provide a good example for testing the value of search term disambiguation. Given the popularity of the Java programming language, it came as little surprise that all ten of the top ten results returned by Google when "java" alone was input as a query related to programming. Coffee drinkers and fans of the Indonesian island may naturally balk at these results. For each sense of the term "java", a secondary term of disambiguation was chosen. The term "programming" was applied to the Java programming word sense to result in a combined term of "java programming". As expected, when this query was input into the Google search engine, all results once again related to the Java programming language. For the word sense of "java" which means coffee, the term "coffee" was chosen as the term of disambiguation, forming the combined query "java coffee". This time, all ten of the top search results related to coffee rather than programming. The final sense for the word "java" listed in the WordNet database is the island of Java, and thus "island" was used as the term of disambiguation for a combined query of "java island". When this query was input into Google, all ten of the top ten results related to the island of Java rather than the programming language or coffee.

4.18 Further Testing

A number of additional searches were conducted, this time with more a more specific area of query in mind to see how the practice of disambiguation handled less
clearly-defined ambiguity. A number of queries relating to bioinformatics were chosen to fill this role. Important to many in the field of bioinformatics are the four nucleotide bases, also known as nucleobases. These bases are parts of DNA and include cytosine, guanine, adenine, and thymine. Each nucleobase is abbreviated as C, G, A, and T, respectively. These terms, of course, are also letters of the alphabet, making it likely that when entered into Google results relating to the letters rather than the nucleobases will be returned. Since "A" is also a word by itself, and "C" is frequently associated with the eponymous programming language, it appeared especially likely that these terms in particular might produce undesired search results for a user wishing to study nucleobases.

4.19 Additional Test Results

When entered into Google by themselves, the terms "A", "G", and "T" each returned a single result relating to the letters of the alphabet in the top ten results, along with nine completely unrelated results per term. Entering the term "C" resulting in five results related to the programming language along with five unrelated results. Secondary search terms of "alphabet" and "nucleobase" were used to disambiguate the queries, with "programming" used for further disambiguation of the term "C". The combined terms which included "nucleobase" produced search results in which each of the top ten results relating to the corresponding abbreviated nucleobase. For "C", "G", and "T", applying the secondary term "alphabet" to the queries produced top ten results which each related to the corresponding letters, and applying "alphabet" to the term "A" produced nine out of ten results relating to the letter. The combined query "c programming" led to each of the top ten results being related to the programming language.
4.20 Final Thoughts

Additional bioinformatics terms were chosen for evaluation, including "expression", which can be applied to genetics or a saying, and recombination, which can have different meanings to both physicists and geneticists. When supplied with an appropriate secondary term of disambiguation, each sense of the word could be searched for producing no less than nine out of ten results relating to the intended query. This is a marked improvement over searching for the ambiguous term by itself. The more ambiguous the term and the greater number of potential word senses which can be applied to a single word, the more likely it is that entering the term alone into a search engine will not produce useful results. It is natural, of course, to call into question why such disambiguation is necessary when it is evident that manually entering a secondary term can produce the same results. The answer to this is that doing so is a valid method of disambiguation, yet there are clearly benefits to using WordNet's alternative approach to disambiguation. These include limiting the amount of typing which must be performed by the search engine user, the possibility that the user is unable to determine an appropriate secondary term of disambiguation, and the convenience of having multiple word senses and their definitions clearly laid out upon the screen for their perusal and selection. While such disambiguation does not necessarily have the potential to revolutionize the way search engine produce their results, the benefits of adding disambiguation capabilities by means of WordNet or another method cannot be overlooked.


[16]. Konduri, Aparna; Chan, Chien-Chung. “Predicting Similarity of Web Services Using WordNet”.


APPENDIX A

APACHE TOMCAT INTERFACE

<table>
<thead>
<tr>
<th>Administration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
</tr>
<tr>
<td>Tomcat Manager</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release Notes</td>
</tr>
<tr>
<td>Change Log</td>
</tr>
<tr>
<td>Tomcat Documentation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tomcat Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Page</td>
</tr>
<tr>
<td>FAQ</td>
</tr>
<tr>
<td>Bug Database</td>
</tr>
<tr>
<td>Open Bugs</td>
</tr>
<tr>
<td>Users Mailing List</td>
</tr>
<tr>
<td>Developers Mailing List</td>
</tr>
<tr>
<td>IRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Miscellaneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Servlets Examples</td>
</tr>
<tr>
<td>JSP Examples</td>
</tr>
<tr>
<td>Surf's Java Server Pages Site</td>
</tr>
<tr>
<td>Surf's Servlet Site</td>
</tr>
</tbody>
</table>

If you’re seeing this page via a web browser, it means you’ve setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat homepage. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the [Tomcat Documentation](http://www.apache.org) for more detailed setup and administration information than is found in the INSTALL file.

**NOTE:** For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- [users@tomcat.apache.org](mailto:users@tomcat.apache.org) for general questions related to configuring and using Tomcat
- [dev@tomcat.apache.org](mailto:dev@tomcat.apache.org) for developers working on Tomcat

Thanks for using Tomcat!
# APPENDIX B

APACHE TOMCAT APPLICATION MANAGER

## Tomcat Web Application Manager

<table>
<thead>
<tr>
<th>Path</th>
<th>Display Name</th>
<th>Running</th>
<th>Sessions</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Welcome to Tomcat</td>
<td>true</td>
<td>0</td>
<td>Start, Stop, Reload, Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle 30 minutes</td>
</tr>
<tr>
<td>/docs</td>
<td>Tomcat Documentation</td>
<td>true</td>
<td>0</td>
<td>Start, Stop, Reload, Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle 30 minutes</td>
</tr>
<tr>
<td>/examples</td>
<td>Servlet and JSP Examples</td>
<td>true</td>
<td>0</td>
<td>Start, Stop, Reload, Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle 30 minutes</td>
</tr>
<tr>
<td>/host-manager</td>
<td>Tomcat Manager Application</td>
<td>true</td>
<td>0</td>
<td>Start, Stop, Reload, Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle 30 minutes</td>
</tr>
<tr>
<td>/manager</td>
<td>Tomcat Manager Application</td>
<td>true</td>
<td>1</td>
<td>Start, Stop, Reload, Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle 30 minutes</td>
</tr>
<tr>
<td>/mutch-1.0</td>
<td></td>
<td>true</td>
<td>0</td>
<td>Start, Stop, Reload, Undeploy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Expire sessions with idle 30 minutes</td>
</tr>
</tbody>
</table>
APPENDIX C

NUTCH SEARCH INTERFACE
APPENDIX D

EXAMPLE NUTCH QUERY

*rat genome*  Search  help

Hits 1-4 (out of about 9 total matching pages):

**Rat Genome Resources**
... sequence against the rat genome sequence and explore ... RGNC Genetics, Genomics, & Maps:  Rat ...

**Rat Genome Resources**
Rat Genome Resources NCBI Home Genomic Biology ... in HomoloGene clusters including mouse/rat only 803 standard GeneIDs in ... 

**BLAST: Basic Local Alignment Search Tool**
... Choose a species genome to search, or ... genomic BLAST databases. Human Mouse Rat Arabidopsis thaliana Oryza sativa Bos ...

**BLAST: Basic Local Alignment Search Tool**
... Choose a species genome to search, or ... genomic BLAST databases. Human Mouse Rat Arabidopsis thaliana Oryza sativa Bos ...