J. Nelson Rushton
A Search Engine for a Hypertext Encyclopedia
(Under the direction of Michael Covington)

This thesis describes the design and implementation of OKRA, a search engine for the U.S. Forest Service's *Encyclopedia of Southern Appalachian Oak Ecosystems.* Chapter One provides background on the encyclopedia itself, and on the WordNet lexical database, which is used to find candidates for additional terms the user can add to his or her query. Chapter Two describes a module for interactively constructing and refining queries in the OKRA system. Chapter Three tells how connection strengths are determined between documents and individual query terms. Chapter Four describes how these connection strengths are combined to estimate the relevance of documents in response to a user's query, including a novel system for scoring documents releative to Boolean queries. Chapter Five describes the results of tests of the system in actual use.

Index words: Text retrieval, Document retrieval, Wordnet, Query refinement, Boolean querying

A Search Engine for a Hypertext Encyclopedia

by

J. Nelson Rushton

B.A., Auburn University, 1990

Ph.D., University of Georgia, 1997

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2001

A Search Engine for a Hypertext Encyclopedia

by

J. Nelson Rushton

Approved:

Major Professor: Michael Covington

Committee: Donald Nute
Bruce Britton

Electronic Version Approved:

Gordhan L. Patel
Dean of the Graduate School
The University of Georgia
December 2001

Acknowledgments

This work was funded by the USDA Forest Service as part of the *Intelligent Encyclopedia of Oak Cover Type Ecosystems in Southern Appalachia* (grant SRS 00-CA-1130134-090; Donald Nute, principal investigator). Special thanks to Mike Rauscher of the Forest Service for suggesting an intelligent search engine as part of the projct. Thanks to Don Nute, Michael Covington, and Bruce Britton for sitting on my thesis committee. Thanks to David Boucugnani for porting this project to LPA ProWeb so that it actually does something useful (besides being my thesis). Lastly, special thanks to Michael Covington for preparing a LaTeX style sheet for University of Georgia theses and dissertations. This was a lot of work for him, done in order to save a lot of work for other people.

Table of Contents

Chapter 1

Introduction

This thesis documents the design and implementation of OKRA (OaK Retrieval Aid), a search engine which retrieves web pages from the U.S. Forest Service's *Encyclopedia of Southern Appalachian Oak Ecosystems.* OKRA has several features which distinguish it from ordinary search engines. First, it helps users interactively refine their query by suggesting additional query terms for possible inclusion. Candidates for new query terms are chosen from the encyclopedia itself, based on the user's original query and using the WordNet electronic thesaurus. Second, the engine incorporates a novel system for scoring documents relative to key-word queries, which takes account of which words in the query are selected as synonyms of other query terms.

Finally, OKRA contains a system for assigning numerical relevance scores relative to Boolean queries. In response to a Boolean query, search engines typically retrieve an unsorted list of documents satisfying the query in a strict classical sense. For large corpora such results "usually are either empty or huge and unwieldy"(Manning and Schütze 2000, p. 530). For this reason Boolean querying has been largely ignored in recent information retrieval research, giving way to systems which assign a numerical score to each document. However, the OKRA system combines the expressive power of Boolean operators with the pliability and fine-grainedness of a numerical ranking system.

The use of WordNet for query refinement is described in Chapter 2. Chapter 3 treats the assignment of connection strengths between documents and individual

query terms, which form the atomic constituents of key-word and Boolean queries. Chapter 4 begins by describing the numerical scoring system for key-word lists augmented by synonyms (either entered as such or extracted from natural language queries). The system is then extended in a natural way to handle Boolean queries involving the operators AND and OR, and finally those which incorporate negation. Chapter 5 describes the results of tests of the OKRA system in actual use at the Bent Creek experimental forest. The remainder of this chapter gives background on the projects which OKRA ties into: WordNet and the *Encyclopedia of Southern Appalachian Oak Ecosystems.*

1.1 Encyclopedia of Southern Appalachian Oak Ecosystems

In January 2001 the U.S. Forest service began compiling a hypertext encyclopedia of forest science for the Southern Appalachians. The motivation for this project was the observation, due mainly to Dr. Mike Rauscher, that "there is a huge gap between what scientists know and what the forest management community is able to apply on the land. This gap between science and application arises because our forest science knowledge is not easily accessible nor readily usable" (Rauscher 2001).

The goal of the encyclopedia is to compile and summarize the last 80 years of forestry research, in a form which is widely and cheaply accessible and at the same time easy to use. The natural way to meet these demands was to publish the encyclopedia in the form of an online hypertext, complete with intelligent tools for navigation and search. One of the main goals of the project was, rather than simply render a traditional encyclopedia in electronic form, to design the encyclopedia as a hypertext document from the ground up. Reading with a mouse in hand, users will tend to zip from one part of the encyclopedia to another, quickly and in a highly nonlinear fashion. Thus information must be presented in such a way that users can

tell at a glance what information is contained on a page, where they can go from that page, and where the page falls in the overall structure of the system.

Additionally, users with specific information needs should be able to locate information in the hypertext by direct search, as opposed to associatively browsing from one topic to another. This is where OKRA comes in. It is hoped that OKRA will provide users with a tool for interactively refining the statement of their information need, and then locating documents in the encyclopedia which address that need.

1.2 WordNet

WordNet is a lexical database, manually compiled over the last 15 years or so by hundreds of researchers and students at Princeton University. WordNet takes the form of a semantic network, whose most important node-type is the *synset* node. In WordNet, a synset is an atomic concept referred to by one or more synonymous word senses. (Word senses, not words, are synonymous; for example $ship_1$ may be synonymous with *boat* while $ship_2$ may be synonymous with *deliver*.) Other node types include *word string*, *word sense*, *part of speech*, and *gloss* (a brief text description of a concept or synset).

Lexical knowledge is represented in WordNet by labeled links between nodes. For example, a *denotes* link between a word sense and a concept (or synset) means that the word sense denotes the concept. The boundary between lexical and "world" or "encyclopedic" knowledge can be blurry at times; thus, like a text dictionary, WordNet contains world knowledge about basic relations between concepts such as *part-of* and *kind-of*. Links denoting such encyclopedic knowledge involve only concept or synset nodes, rather than nodes for words, word senses, or other syntactic entities. For example, concepts may be linked by a meronym ("part of") relationship,

a hyponym ("is-a") relationship, an antonym relationship, or a cause-of relationship (for concepts denoted by verbs).

Literature on WordNet

The standard primer on WordNet consists of the "Five Papers" on WordNet, now available at www.cogsci.princeton.edu/~wn/papers (Miller, Gross, Fellbaum, Beckwith, and Miller 1993). The most comprehensive introduction to WordNet is *WordNet: A Lexical Database* (Fellbaum 1998), which contains an updated and expanded version of the standard "Five Papers," as well as a compliation of research articles using WordNet in various domains, such as word sense disambiguation (Leacock and Chodorow 1998), text retrieval (Voorhees 1998), and detection of malapropisms (Hirst and St-Orange 1998).

WordNet has been used in a wide variety of research projects, and an excellent bibliography is available online at the Southern Methodist University web site: www.seas.smu.edu/∼rada/wnb/. The bibliography was originally compiled by Joseph Rosenzweig of the University of Pennsylvania, but is now maintained by Rada Mihalcea of Southern Methodist University. The bibliography includes links to papers wherever possible.

One notable and frequently cited article on WordNet is Leacock, Miller, and Chodorow (1998). Here the authors have implemented a system for adaptively learning word sense disambiguation. WordNet is used to automatically locate training examples for the system in a large corpus.

Another landmark article is (Burke et al. 1997). This article overviews FAQ FINDER, a system that incorporates WordNet into an information retrieval agent. The agent takes queries in natural language and attempts to find answers by retrieving answers to similar questions from frequently asked question (FAQ) files.

Is WordNet Brittle, 1960's-Style AI?

In addition to our knowledge of syntax, we use a great deal of "world" knowledge in parsing natural language. For example, the following sentences are both ambiguous from a strictly syntactic point of view:

I saw the girl in the blue dress.

I saw the girl in the mirror.

However, we disambiguate them effortlessly based on what we know not about the English language, but about the goings on of the world: mirrors reflect images but dresses don't; girls wear dresses but not mirrors. At one time it was hoped that natural language understanding could be achieved by compiling databases of such facts. In 1970, Minsky and Papert wrote, "[The list] is not endless. It is only large, and one needs a large set of concepts to organize it. After a while one will find it getting harder to add new concepts, and the new ones will begin to seem less indispensable" (Minsky and Papert 1970, as cited in Dreyfus 1972). In the light of thirty years of research it now appears more likely that, as Hubert Dreyfus (1972) argued, the list is endless. This raises the question of whether research on WordNet, a large compilation of raw data for use in natural language processing, is perhaps a waste of time. Even after fifteen years of hard work, the WordNet is far from complete, particularly for specialized domains that are not part of plain speech – where much if not most information retrieval takes place. This is not to mention hard, ongoing controversies within the WordNet community over ontology.

However, I feel that WordNet and similar projects are worthwhile. The key distinction between a Minsky-esque database and WordNet is that the latter restricts itself to cataloguing a handful of basic relations between concepts referred to by *individual words*, avoiding the explosion in complexity which arises when we consider relations among concepts expressed by phrases or sentences. Thus WordNet

can never reflect the totality of our world knowledge used in processing language – but at the same time it aspires to a task which is both highly useful and plausibly tractable. A dictionary is not an encyclopedia, but it is still very handy.

An enlightening analogy may be made between WordNet – the first major attempt at electronic lexicography – and the early attempts at text lexicography. At one time, it seemed to many an intractable and only marginally useful project to compile a text dictionary. When Ambrose Phillips published *Proposals for Printing an English Dictionary,* the project met with resistance and was eventually dropped. Not until 1746 was Samuel Johnson commissioned to compile a dictionary of the English language, a task which took over nine years, and whose results were shoddy by any modern standard. The French counterpart *Le dictionnaire de l'Académie française* (1694), took over fifty years to complete, drawing on the combined resources of the French Royal Academy.

By comparison, WordNet has only scratched the surface in terms of effort. Yet in the emerging age of the internet and world wide web, lexical knowledge is more important than ever, as well as more convenient to collect, store, and share. Experience has shown, contrary to some early opinions, that text dictionaries are worth compiling, despite the vast efforts involved (especially in the early years), despite their perennial incompleteness, and despite hard questions and ongoing controversies about their contents. This is because lexical knowledge is among the most basic and important knowledge we possess (or do not possess) – so much so that even an expensive and incomplete compilation of it is well worth having. I feel this will turn out to be even more so in the electronic information age, and that further research will prove the worth of WordNet and similar projects.

Chapter 2

Constructing Queries in OKRA

2.1 Entering a Query

Users of OKRA may enter a query either as a list of keywords, a natural language query, or a Boolean expression delimited by AND's, OR's, NOT's, and parentheses. Queries are parsed using a DCG parser written by David Boucugnani at the University of Georgia's Artificial Intelligence Center. Natural language expressions are typically tokenized into a list of key-words which includes one exemplar of each word type used in the query (except stop words), while Boolean expressions are parsed using DCG rules. Natural language queries which conform to certain templates can be parsed using shallow NLP to yield a list of key words or a Boolean query. For example, the query:

```
Tell me about fire, water, and air.
```

would yield the same results as the following query:

```
fire AND water AND air
```

2.2 Query Refinement

In a landmark article on information retrieval, Croft and Thompson (1986) set forth the basis for their I3R information retrieval system in a single sentence, which they termed the *quality-in quality-out principle:*

> "A query that more accurately reflects the user's information need will produce better results."

Though it is intuitively obvious, Croft and Thomson note that this principle succinctly summarizes a wide range of experimental results on information retrieval. They go on to note that information retrieval can be enhanced in essentially only two ways: improve the inference process or refine the user's query.

OKRA provides a module for interactively refining the user's query by adding synonyms of query terms. The idea is that if a user includes a word, say *leaves* in the query, then documents that do not contain *leaves* but do contain its synonyms, such as *foliage,* should be considered relevant – especially if the document contains other query terms (or their synonyms).

Naively, one could automatically include all synonyms of each word in the query. Perhaps surprisingly, this approach has been found ineffective (Voorhees 2000 p. 300). The culprit here is polysemy, in which a single word string has more than one meaning. In information retrieval, polysemy is a problem in its own right, as well as confounding possible solutions to the problem of synonymy. In particular, it is impossible to determine whether two word tokens are synonyms merely by comparing strings to a database. For example, the tokens of *lecturer* and *speaker* are synonymous in the following two sentences:

> *We need to find a guest lecturer for Friday's seminar.*
>
> *The speaker for today's seminar knows no English.*

but not in the following:

> *Woodrow Wilson was an outstanding lecturer.*
>
> *The speaker in my PC is rated at 50 watts.*

In an early version of OKRA which used automatic synonym inclusion, the word *smack* was automatically added to a query containing the word *horse* – because both are street names for heroin. Clearly, synonymy is domain dependent, due to the existence of polysemy.

Polysemy poses a deceptively difficult problem for information retrieval. This is because, while it is not uncommon even in plain language, the occurrence of polysemy explodes when we consider the jargons of various specialized or semi-specialized domains – where much if not most information retrieval takes place. Experts and hobbyists in various domains are interested in subjects and objects which other people are generally not interested in. But usually, rather than inventing new words to describe these objects, they borrow words or phrases from plain language, and use them in a way idiomatic to their domain of expertise. The word *triangle*, for example, can denote a union of line segments in geometry, a percussion instrument in symphonic music, a configuration of stones in the game of go, a choke hold in judo, or a drawing instrument in architecture, just for starters (incidentally, none of the top 10 hits for the search string "triangle" on Google relate to any of the above senses). Once again, these domain-dependent word senses are not a tangential issue. Many if not most users of document retrieval systems are looking for documents in a specialized area of knowledge.

An alternative to adding synonyms automatically is to suggest synonyms for user approval. This combines the strengths of the machine (comprehensive recall) with those of the human user (precision), at the cost of demanding more time from the user to enter and refine his query. Such systems have been shown to enhance retrieval for small, homogeneous corpora such as our hypertext (Salton and Lesk 1971; Wang, Vandenthorpe, and Evens 1985). This is the approach taken in OKRA. After a user enters a query, he or she may choose to either submit the query or to refine it using the interactive synonym addition process. If the user chooses to refine the query, she

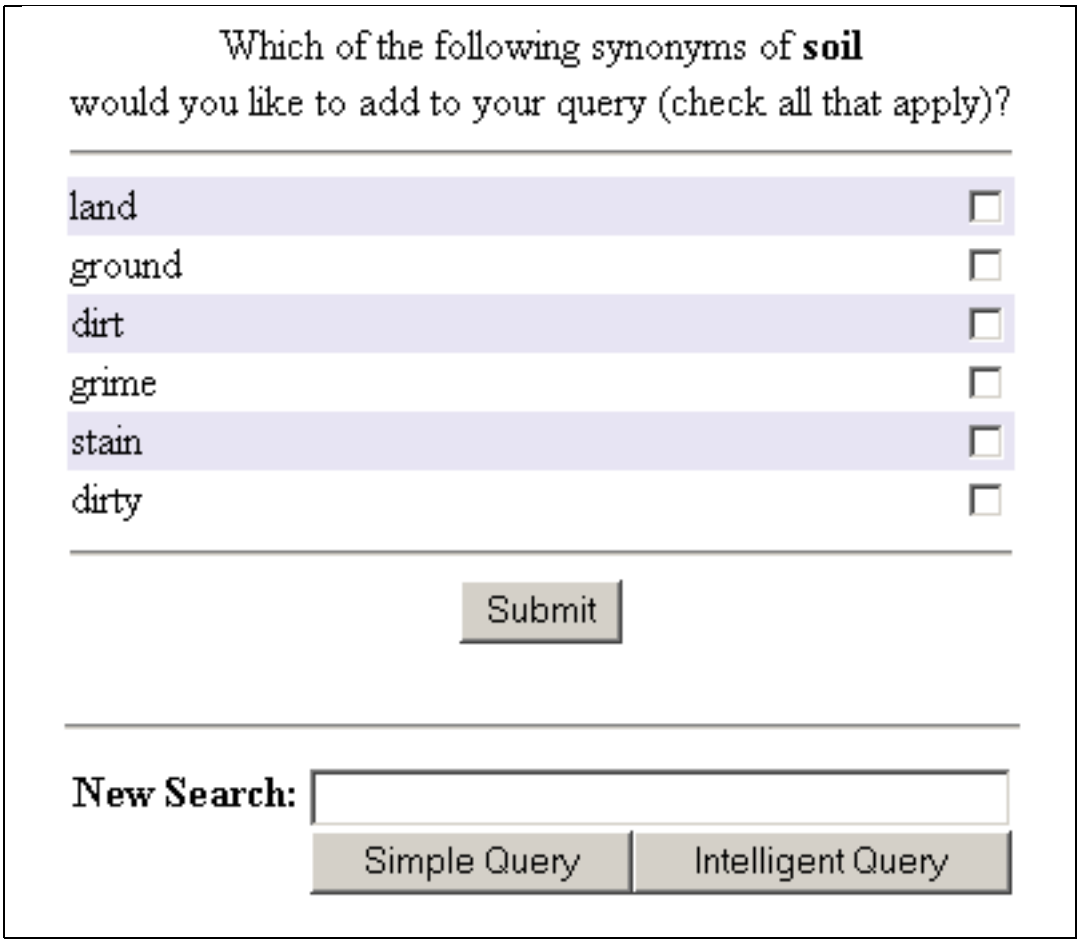

Figure 2.1: Sample synonym selection menu

is presented with a list of synonyms, if any, for each word in the query, and adds selected terms by clicking check boxes, as in figure 2.2.

In trials carried out at the Bent Creek experimental forest, users of OKRA reported that the terms suggested by the system helped them better express their information needs. Similar user responses were reported by Chen and Lynch (1992).

The synonym candidates are chosen from the WordNet thesaurus, with the constraint that only words which actually appear in the corpus are shown for possible selection. This cuts out about two thirds of potential synonyms, on average, and in some cases cuts the user's work drastically. (The encyclopedia currently contains

only about 11,500 distinct word types, compared with about 40,000 distinct word strings available in WordNet.) The synonym selection system was implemented in LPA ProWeb by David Boucugnani.

Chapter 3

Calculating Word-Document Connection Strengths

A query entered into OKRA is internally converted into a set of clauses, or disjunctions of query terms (and possibly their negations), where clause boundaries are determined by the logical structure of the query and by synonymy relations among query terms. For example, the query

```
fire AND (soil OR water)
```

where (say) *flame* and *flaming* are selected as synonyms of *fire*, and *dirt* and *earth* are selected as synonyms of *soil,* would be internally represented as a list of lists, with each nested list representing a logical clause:

```
[[fire, flame, flaming], [soil, dirt, earth, water]]
```

The estimated relevance of a document is a function of its relevance scores for the individual clauses, and the score for a clause is a function of the scores for the terms within it. The particulars of these functions are discussed in Chapter 4; but at any rate, the estimated relevance of a document is found by combining the connection strengths between the document and individual query terms, in a manner sensitive to logical and lexical context. This chapter details how those connection strengths are determined.

Word-document connection strengths do not depend on factors input at the time of retrieval, and so are pre-computed offline and held in a database on the hypertext server. Section 1 of this chapter describes a program which actually finds the

chunks of hypertext on server's hard disk. Periodically running this "file spider" eliminates the need to manually update the list of files which must be indexed as the encyclopedia changes. Section 2 describes a system for parsing HTML tags, which is necessary in order to assign appropriate weights to terms that appear in, say, boldface or italics, or which appear in the title or other META information not necessarily displayed by web browsers. Section 3 details how the corpus is rendered into a set of *word occurrence facts,* which essentially recast the entire encyclopedia in a form amenable to analysis and computation. Section 4 tells how word-document connection strengths are computed from these facts. There is a standard body of methods for doing this – most of which were not used. The reasons for this are explained in Section 5.

3.1 Finding the Documents

3.1.1 Design Issues

The documents that need to be indexed for retrieval are individual "chunks" of hypertext, each of which is stored in a separate file. These files lie in a nested directory structure which contains a great deal of additional material, including files automatically generated by Microsoft Front Page which hold data about the file and link structure of the hypertext. The result is that in order to index the chunks of the hypertext, it is first necessary to find them.

Confusingly, each hypertext chunk has a corresponding auxiliary file of the same name, including the `htm` file extension, which holds data about the chunk. The auxiliary file for a given chunk is located in an automatically generated folder which lies in the same directory as the chunk itself. For example, in the `Fire` directory is a chunk called `evaluating.htm`, as well as a folder called `_vti_cnf` which contains

another file also called `evaluation.htm`, which contains coded information about the aforementioned chunk. The latter file is not one which should be indexed

The root file directory of the hypertext contains several chunks of global significance, such as the table of content and acknowledgments, some of which need to be indexed. However, this directory also contains many other files which are not hypertext chunks, including components of the search engine which are periodically used to automatically re-index the corpus, and which, for ease of use and maintenance, are best kept in the root directory of the encyclopedia.

So the question is, in this nested structure of folders and files which makes up the encyclopedia, how does an indexing program tell *chunk* from *junk* during the indexing process? In recent stages of development, it has been suggested by Deborah Kennard and J. B. Jordin that the hypertext be reorganized so that all chunks are included in a single folder which contains nothing else. This will facilitate the maintenance of the site in many ways, including the ongoing upkeep of the search engine. However, a working search engine was needed before this suggestion could be implemented, and so a module was developed which used heuristics to locate the chunks of the encyclopedia within the file structure.

Since there were initially no hard and fast rules as to what went into the file structure of the encyclopedia, there were no hard and fast rules about how to locate the chunks. However, after browsing the folders and documents, a mostly-true heuristic was developed to single out the files containing hypertext chunks: Any file with a `htm` extension is a hypertext chunk, provided it does not occur within a folder whose name contains an underscore, or in any nested subfolder of such a folder.

This rule had only a handful of exceptions. The exceptions did not fall into any useful pattern, and, more importantly, the pattern of future exceptions could not have been forecast. Therefore, the corpus was altered ad hoc to fit the rule rather than vice versa. This merely required renaming a few folders and files. The rule

was then incorporated in the short term as a principle of authoring and arranging hypertext chunks.

3.1.2 Implementation

The directory structure is traversed depth first, using LPA Prolog's built in file handling predicate `dir/3` to obtain a list of the contents of each folder as it is expanded. As each folder is expanded, files with `htm` extensions are kept track of on a blackboard, and child folders are asserted to the unexpanded list. The program begins by asserting the root directory of the encyclopedia to the unexpanded list, and calling the following recursive Prolog predicate:

```
expand_folders:-
   retract( unexpanded_folder(Folder) ),
   assert(folder(Folder)), % keep track of the expanded folder.

   % create a file template for the
   %first argument of dir/3:
   atom_chars(Folder,FolderChars),
   atom_chars('\*.htm',S),
   append(FolderChars,S,PChars),
   atom_chars(Template,PChars),

   % Instantiate HtmlFiles with the files
   % of the current directory, as a list:
   dir(Template,0,HtmlFiles),

   % List the .htm files on the blackboard:
   store_html_facts(HtmlFiles),
   % Instantiate NewFolders with a list of
   % all folders in the current directory
   % that need to be expanded.
   atom_chars('\*',SlashAst),
   append(FolderChars,SlashAst,Template2Chars),
   atom_chars(Template2,Template2Chars),
   dir(Template2,0,List),
   List = [_,_|NewFolders], % ignore folders '.' and '..'
   store_folder_facts(NewFolders),
```

```
   % recurse:
   expand_folders.
```

Importantly, the virtual folders '.' and '..', which always appear first in the list returned by `dir/3`, are never expanded. Otherwise the loop fails to terminate. The virtual folder '..' is actually a pointer to the parent of the current folder, which, if expanded, would yield a duplicate copy of the current folder, which would itself be expanded, along with its parent, and so on *ad infinitum.* Similarly, '.' is a pointer to the current folder, which would lead to an infinite recursion if expanded.

As the code runs, a dynamic blackboard is created which includes `html_file`/1, `folder`/1, and `unexpanded_folder`/1 facts. The following is an example of what the blackboard might look like after one pass through `expand_folders`:

```
html_file( 'C:\encyclopedia\regeneration\shade_tolerance.htm' ).
html_file( 'C:\encyclopedia\regeneration\advance_regen.htm' ).
html_file( 'C:\encyclopedia\regeneration\pioneer_species.htm' ).

folder( 'C:\encyclopedia\ecosystems\regeneration').
folder( 'C:\encyclopedia\management\timber').

unexpanded_folder( 'C:\encyclopedia\management\timber\tables').
unexpanded_folder('C:\encyclopedia\management\timber\oak').
```

The program terminates when no folders remain on the unexpanded list, leaving in memory a complete list of files and folders which make up the encyclopedia.

Folders with underscores in the title, which are automatically created by Microsoft Front Page and contain no information relevant for indexing, are thrown away by the predicate `store_folder_facts`/3:

```
store_folder_facts([]):- !.
store_folder_facts([H|T]):-
   atom_chars(H,String),
   member(95,String),% If the folder name has an underscore,
```

```
   !,                   % do nothing.
   store_folder_facts(T).
store_folder_facts([H|T]):-
   % no need to check for underscores,
   % because of the cut above
   assert(unexpanded_folder(H)), % Otherwise add the folder to
   store_folder_facts(T).        % the unexpanded list.
```

The predicate `store_html_facts/1` simply accepts a list of `htm` files and asserts their existence to program space:

```
store_html_facts([]).
store_html_facts([H|T]):-
   assert(html_file(H)),
   store_html_facts(T).
```

3.2 Parsing HTML

Once the documents of the hypertext are located, each document is read in as a string of ASCII codes, which is tokenized using a DCG based tokenizer written by Michael Covington at the University of Georgia. For this project, the tokenizer was extended to recognize HTML tags and parse them for content. This was necessary in order for the search engine to take account of the font features and META features with which terms occur in a document, as detailed in Section 4. For example, the ASCII codes for the file `author_credits.htm`,

```
[60,104,116,109,108,62,13,10,13,10,60,104,101... ]
```

are converted to a list of atoms (ellipses indicate left out list items):

```
['<html>','<head>',...
'Agricultural','Handbook',583,(','),91,pp,'.','</font>', ...]
```

Notice that the exact white space structure in the original file is not preserved in the tokenized version. In the above example the opening `<html>` tag makes up the first six characters, and the subsequent `<head>` tag begins after four white space characters `[13,10,13,10]` which correspond to two carriage returns in the source file. These carriage returns are not reflected in the tokenized version.

HTML tags are parsed using DCG rules with features. The topmost node in the parse tree is handled by the rule

```
tag(Features) --> [60], name(Sem), separator, attributes(Sem2),
                  separator, [62],  {append([Sem],Sem2,Features)}.
```

60 is the ASCII code for the left angle bracket which begins the tag, a name is an HTML tag name such as `b`, `it`, or `p` (respectively boldface, italics, and paragraph boundary). The variable `Sem` is to be unified with the features signaled by the token name, such as bold, italics, or new-paragraph. A separator is white space or null. Note that no white space is allowed following the opening angle bracket, while white space is allowed before the closing bracket.

Attributes include 0 or more additional features, each denoted by a feature name, followed by an '=' sign (ASCII code 61), followed by a value enclosed in quotation marks (ASCII code 34). For example, the target page of a hyperlink is indicated by such an attribute. White space is allowed but not required on either side of the '='. Thus we have

```
attributes([])    --> [].
attributes([Sem]) --> attribute(Sem).
attributes([H|T]) --> attribute(H), whitespace, attributes(T).

attribute(Var(Value)) --> attribute_variable(Var), separator,[61],
                          separator,[34],string(Str),[34],
                          {name(Value,Str)}.
```

As the tag is parsed, the variable `Features` is unified with a list of features including the basic feature denoted by the tag name and any additional features given by attributes.

Many HTML tags have corresponding end tags which signal the browser to turn off a given text feature. End tags have the same names as their opening counterparts, preceded immediately by a front slash (ASCII code 47). Hence, all end tags can be handled using the single rule

```
name(end(Tag)) --> [47], name(Tag).
```

The remaining rules simply give names of various HTML tags as lists of ASCII codes, for example:

```
name(title) --> [116,105,116,108,101].
% These are the ASCII codes for 't', 'i', 't', 'l',
% and 'e', respectively.
```

3.3 Creating Word Occurrence Facts

Before assigning connection strengths between query terms and documents, the corpus is translated into a set of word occurrence facts. Each word token in the corpus yields a Prolog fact which keeps track of the word string, the chunk in which it appears, its paragraph and position, and any text features with which it is displayed. This collection of facts essentially represents the entire corpus in a form amenable to computation. A sample fact is as follows:

```
word_fact(fire,`Effects of Fire`,paragraph(2),position(39),[body]).
```

The above denotes that the term *fire* occurs as the 39^{th} token in the document *Effects of Fire*, in the second paragraph. The `[body]` feature list indicates that the token appears in the main body of the text, as opposed to in the title, header, or in

a META tag, and occurs in plain text as opposed to, say, italics or bold. Currently there are about 280,000 such facts in the OKRA database, one for each token in the corpus. This is one half to one third of the projected size of the final corpus.

In order to create these facts, each file is tokenized and converted to a list of atoms, each of which is either a word string, numeral, whitespace, or an HTML tag. The token list is then processed recursively by the predicate `make_word_facts(+Filename,+Atoms)`, whose source code appears in Appendix A. Features such as italics and boldface, paragraph and position number, and the name of the current file, are stored on a dynamic blackboard and updated as each token is processed. For each word token, a word occurrence fact is created and then the blackboard is updated. Other tokens, such as whitespace and HTML tags, may change the state of the blackboard but do not create new word occurrence facts.

3.4 Assigning Connection Strengths

3.4.1 Design

One advantage of searching hypertext documents, as opposed to plain text, is that each word token may appear with one or more text features such as boldface or italics. With HTML documents the advantage is even greater because HTML features are semantically rather than display oriented. For example, HTML code may indicate that a certain text string is a section heading, a list item, or emphasized – and leave it to the browser how the feature is displayed. Thus in indexing HTML documents for retrieval, we can exploit not only their natural language content, but the semantic indications of the HTML code as well.

Ideally, rules assigning connection strengths are particular to a corpus, reflecting the idiosyncratic use of HTML features by different authors or communities of authors. Hence OKRA alters connection strengths only for those features which

appear commonly and with consistent denotation in the hypertext. Though OKRA contains machinery to recognize and respond to a wide variety of features, only a handful were found which meet this criterion. These are as follows: Each appearance in the title of a document (i.e. between `<TITLE>` and `</TITLE>` tags) scores 3 points. (The document title is meta-information about the document, which is contained in the HTML file but is not necessarily displayed by the browser.) Each occurrence in the document body in bold or italics scores 2 points. Each additional occurrence in the body scores 1 point. Additional occurrences, such as in the head outside the title, or in HTML comment lines, score 0 points.

The connection strengths between words and documents were assigned using a bounded summation procedure, similar to that used by the Google search engine (Brin and Page 1998). To determine the connection strength between a term and a document, the scores for all occurrences of the term in the document are summed together, up to a maximum relevance score of 3, after which additional points are discarded. That is, a term cannot have a greater connection strength with a document than if it appears in the document's title. This bound is used in order to reduce noise from documents which are highly relevant to some but not all terms in a query. For example, in a query about using controlled burning to regenerate pine trees, a document with many prominent occurrences of *pine* and *trees*, but not other query terms (or their synonyms) would probably be less relevant than one which contained one or two occurrences of each word in the query.

3.4.2 Implementation

Connection strengths are stored as `con`/3 facts. For example, the Prolog fact

```
con(fire,`Effects of Fire on Wildlife`,3)
```

denotes that the word *fire* is connected with a strength of 3 to the document *Effects of Fire on Wildlife*. Currently there are about 120,000 such facts in the OKRA database. Word tokens occur as the first argument of `con`/3 in order to take advantage of Prolog's first argument indexing at retrieval time. Because of first argument indexing, only facts concerning words which actually occur in the query are examined at time of retrieval.

In order to compile the connection strengths, the roughly 280,000 word occurrence facts are first loaded into program space. They are then retracted one by one, and as each word fact is retracted an appropriate connection strength is created (for the first instance encountered of a given term in a given document) or modified (for subsequent occurrences). Connection strengths are held in program space and modified by asserts and retracts, rather than being stored in a list or other stack space. Though large numbers of asserts and retracts are normally to be avoided in Prolog programming, there are two reasons for this. First, a large amount of data (on the order of 50 megabytes) must be processed, which is more conveniently held in program space than on the stack. Second, as each word occurrence fact is processed the appropriate connection strength must be found in memory and updated. If the connection strengths were stored in a list and updated by a recursive procedure, this long list would have to be traversed many times in order to update the appropriate values. However, with connection strengths stored in program space we can take advantage of first argument indexing to speed up the process. Code to carry out this procedure is listed in Appendix B.

For now, the paragraph and position information in the word occurrence database are not used in computing connection strengths. However, this information is included in the word occurrence database to make the system more easily modifiable. The word occurrence database contains enough information that the corpus could practically be reconstituted from it (except only for numerals, images, and

exact distribution of white space). Thus any foreseeable change in the computation of connection strengths could be implemented without modifying the word occurrence data or how it is created.

3.4.3 Some Standard Techniques Which Were Not Used

Because of the nature of the corpus, some of the standard techniques in forming word-document connection strengths were left out of OKRA. For example, when judging relevance by counting words in a document, one common practice is to scale down the relevance for long documents, often simply dividing by the number of words in the document. The reasoning here is that token counts favor long documents rather than documents with concentrated information on a desired topic. For example, the Encarta encyclopedia entry for *wheel* contains 9 occurrences of the word *wheel,* while the King James Bible contains 48.

However, one constraint in creating the Oak Hypertext is that chunks are kept to a relatively constant length. Moreover, all the text in the encyclopedia is expertly and professionally authored (in contrast with, say, the world wide web); and so long documents are consistently expected to contain more information, and hence answer more queries, than short ones. For these reasons, document length was not taken into consideration.

Another commonly used technique is to scale down the relevance score for commonly occurring words, typically dividing by the total number of tokens of a given word in the corpus (see, e.g., Manning and Schütze 2000, p. 543). The idea here is that the most indicative document terms are those which most rarely occur outside the document. This is a specific manifestation of a general tenet of information theory: the information value of a message varies inversely with the probability of receiving it (Shannon 1948).

This technique is suitable for searching large, heterogeneous corpora, where the number of word tokens is large enough that the distribution of tokens in a corpus tends to reflect the actual frequency of use outside the corpus. However, in browsing word-occurrence data for the Oak Hypertext Corpus, it seems that many words of high frequency also have high information value, and many words which are rare in the corpus have low information value. For example in the current corpus, the content word *fire* occurs more often than the stop word *therefore*. Essentially, the 790 chunks of hypertext are too small and homogeneous a sample for stop words to be detected by frequency. However, a list of stop words was manually selected, with the aid of an automatically generated list of token frequencies. The stop word list comes into play when the word-document connection strengths are being computed, in that stop words are always assigned a connection strength of zero. Since `con`/3 facts are asserted only for nonzero connection strengths, this is implemented by simply not assigning connection strengths for stop words.

Chapter 4

Estimating Document Relevance

4.1 Key Word Queries With Synonyms

Estimating relevance is a tricky business. All else being equal, a document should be considered more relevant to a query if it contains more and more prominent occurrences of a single query term. Also, all else being equal, a document should be considered more relevant if it contains occurrences of a greater number of distinct query terms. But when these two heuristics are used together, all else ceases to be equal; and therein lies the rub.

For engines which consider only the occurrence versus non-occurrence of terms in a document, one approach is to conjoin (i.e. AND) all query terms by default. The intuition here is that an ideal document would be relevant to all of the concepts in the query. In engines which assign numerical connection strengths between terms and documents, this generalizes to scoring a document according to the minimum connection strength it has with any query term. The Google search engine uses an approach in this vein.

Naturally, a document which contains all query terms should tend to be ranked higher than one which does not. However, strict AND'ing or minimizing gives rise to brittleness. In particular, estimated relevance of a document does not degrade smoothly as the number of query terms within it decreases. This has the effect that long queries – which usually best describe the user's information need (Perez-Carballo and Strzalkowski 2000) – often return few or no documents, as frequently

happens with the Google engine, or the search function for the *Web of Science* database. Intuitively, if no document is found that contains, say, nine out of nine terms from the query, then documents with eight out of nine ought to be displayed, followed by those with seven, etc.

This suggests counting the number of terms present, or more generally adding the estimated numerical relevance from each query term as in the engine of Voorhees (1994). This approach preserves the advantages of conjunction (viz., documents which containing all query terms tend to be ranked higher) with a modicum of graceful degradation. However, there is a fly in the ointment here as well. Recall that the connection strength of a document to a single term is bounded, to keep frequent occurrences of a single term from drowning out evidence provided from other terms. (For example a document with 5 occurrences of the word *forest* and 0 occurrences of the word *fire* should not have a high estimated relevance to the query `forest fire`.) But if a query has been augmented by the inclusion of synonyms, the same drowning out effect can come into play at the semantic level. Scores from multiple terms denoting a single query concept (or a small subset of concepts) can combine to drown out the contribution of other concepts. For example, an article containing the words *forest* and *woods* (but not *fire*) might score as high as one containing *forest* and *fire*, even while the contribution from each query term is bounded.

A solution here is bound the contribution of each individual term *along with its added synonyms*. The bounding effect is thereby transferred from the syntactic to the semantic level. This allows the advantages of conjunction, while escaping the disadvantages of brittleness and over-sensitivity to individual words or concepts. In OKRA, occurrences of synonyms are scored as if they were multiple occurrences of the same word. Thus, the contribution of any term to the relevance score, together with its synonyms, is given a fixed bound. Moreover, rather than increasing linearly up to a cutoff point, successive occurrences of terms denoting a given concept yield

a *diminishing return* in their contribution to the estimated relevance score. The idea here is that once evidence is found relating a document to a given query concept, additional relevance to that same concept gives less and less support for relevance to the query as a whole (Manning and Schütze 2000, p. 542).

In information retrieval systems, such diminishing returns are often implemented by taking the square root or logarithm of the number of occurrences. This time-expensive floating point arithmetic is fine when connection strengths are computed offline. However, if diminishing returns need to be implemented at retrieval time it is preferable to rely on integer arithmetic, since relevance scores must be computed for a large number of documents (potentially the entire collection) in as short a time as possible. OKRA needs diminishing returns at retrieval time because (1) occurrences of synonyms are treated as occurrences of the same word, and (2) different sets of terms may be selected as synonyms in different queries. Therefore a function is needed here which yields diminishing returns and can be computed very quickly. A method was chosen which is essentially the simplest possible function of connection strength that yields diminishing returns: A combined connection strength of 1 for a term and its synonyms yields a relevance score of 3; combined connection strengths of 2 yield a relevance score of 5; and combined connection strengths of 3 or more yield a connection strength of 6. The ceiling of 3 was chosen for combined connection strengths (correspondingly, 6 for relevance scores) because it is the bound for word-document connection strength for a single term. That is, occurrences of synonymous terms are treated as multiple occurrences of the same item – which, for semantic purposes, they are.

Here is an example of the relation between connection strengths and estimated relevance of a document in the OKRA system. Suppose the user enters the query "harmful effects of smoke on forests," and selects *detrimental* as a synonym of *harmful*, and *results* and *consequences* as synonyms of *effects*. After removal of the

harmful	1
detrimental	1
effects	3
results	0
consequences	2
smoke	0
forests	1

Table 4.1: Connection strengths for a sample document

stop words *of* and *on*, the query would be represented as a clause set, denoted by a list of lists:

```
[[harmful, detrimental], [effects, results, consequences],
[smoke], [forests]]
```

Now consider a document with connection strengths to the individual query terms as given in table 4.1. The connection strengths for the first clause add up to 2, yielding a score of 5 for the clause. The connection strengths for the next clause total 5, for a clause score of 6. The last two clauses score 0 and 3 respectively. The score for this document is the sum of the clause scores, or 14. Note the effect of diminishing returns per clause: a document containing a given number of query terms will score highest if those terms are widely distributed among different clauses. For example, a single occurrence of the word *smoke* in this document in plain text would raise the score by 3 since it would be the first occurrence of any term within its clause. A single extra occurrence of *forests* would raise the score by 2. An extra occurrence of *harmful* or *detrimental* would only raise the score by 1; and additional occurrences of *effects*, *results*, or *consequences* would not change the relevance score, because the maximum score has already been reached for that clause.

4.2 Handling Boolean Expressions

4.2.1 Motivation

In the field of information retrieval, queries containing Boolean operators have traditionally been interpreted as classical Boolean functions of the occurrence of document terms. For example the query

```
burning OR (fire AND NOT(gun))
```

would be viewed as a request for a list of all documents which either contain the word *burning* or contain the word *fire* but not the word *gun*. Such a system provides only a yes or no for each document, with no method of ranking the retrieved documents or detecting near misses. Additionally, such systems perform poorly for large document collections. For these reasons, Boolean queries have been largely abandoned in recent research on information retrieval (Manning and Schütze 2000, p. 530).

For OKRA, I wished to build a system which would fuse the expressive power of Boolean queries with the robustness of numerical scoring systems. This required abandoning the classical view of the semantics of Boolean operators in favor of a more pliable and sensitive semantics, as they seem to have in human communication. In practice, a Boolean expression is often merely the user's best attempt at expressing a subtle system of preferences in a small space; and the savvy auditor (or engine) realizes there is some play in the semantics of the operators when compared with the classical interpretation. For example, if a job description calls for a Ph.D. *or* five years' experience, a candidate with a master's degree and four years' experience might reasonably apply – while a candidate with a Ph.D. *and* five years' experience would probably be preferable to one having only one of these features. Such subtleties are not accounted for in the classical interpretation of the OR operator.

An example of the desirability of "play" in the interpretation of Boolean operators for searching has already been discussed in the interpretation of conjunctions. Given a request for documents which satisfy a large number of criteria, it would be a mistake for the system to ignore documents which strongly satisfy most but not all of the criteria, especially if no documents satisfied all of them. Bear in mind that there is no commonly used operator or language construct which (classically) means *strongly satisfies most of the following*, or *satisfies at least one but preferably a handful of the following, preferably strongly.* Thus even if such subtle and quasi-quantitative operators play a role in the user's information need, as I suspect they often do, he must make do with AND and OR in expressing his query. OKRA aims to cut the user some slack in this department, while remaining reasonably faithful to the classical semantics of the operators.

4.2.2 Design

Such a system might seem difficult to build, and especially to fuse smoothly with OKRA's carefully designed engine for scoring key-word queries with added synonyms, accounting for HTML features, etc. Happily, it turns out that not only is the basic scoring system robust enough to *allow* the incorporation Boolean operators – it had automatically handles some of the subtleties involved.

The key ideas of the basic scoring system are that

1. a relevant document should address all, or as many as possible, of the concepts expressed in the query, and

2. the relevance of a document to a concept can be evinced by the appearance of any of several synonymous terms.

Roughly speaking, such a query asks that a document satisfy *at least one* from *each* of several sets of criteria. Thus in handling such queries, the basic system handles

a certain type of Boolean construct: a conjunction of disjunctions, or *clause set*. Moreover, it handles the clause set in a way that allows some slack in the semantics of the operators, and assigns numerical scores for goodness of fit rather than a simple yes or no. But the really good news is that *any* Boolean expression can be converted to a logically equivalent clause set. Thus with modifications that will be described shortly, the basic scoring system provides an apparatus for processing arbitrary Boolean queries. Not only do the Boolean features mesh with the existing scoring system – they rely on the system closely.

Boolean queries constructed using AND and OR are handled by converting to clausal form and then scoring in the OKRA system, treating terms within a clause as if they were synonymous. For example, the query

```
(forest AND fire) OR (prescribed AND burning)
```

in clausal form, becomes

```
(forest OR prescribed) AND
(forest OR burning) AND
(fire OR prescribed) AND
(fire OR burning)
```

Consider scoring for a document based on the above query. A document with occurrence of, say, *fire* (and no other query terms) scores twice, once in the third and once in the fourth clause, for a minimum score of 6 and maximum score of 12 (depending on the number and prominence of occurrences). Next, consider a document with occurrences of *fire* and *burning*. This document could score in three different clauses, but the scores from the final clause (fire OR burning) yield a diminishing return. In any case the document will score between 11 and 18. Finally, a document with occurrences of *forest* and *fire* – which satisfies the query from a classical standpoint – could score in all four clauses for a maximum score of 24 (the maximum possible for any document), and would receive a minimum score of 12.

In this example, as in all cases using the OKRA system, a document can receive the maximum score only if it satisfies the Boolean query as classically interpreted. This maximum will be approached or met in case the occurrences of appropriate query terms are frequent, prominent, and/or reinforced by multiple synonyms (which appear in the same clause as their parent terms) – that is, in case of nearly ideal evidence. Thus in the case of nearly ideal evidence, the semantics of of the logical operators will be essentially classical.

On the other hand in case the evidence is sparse, the diminishing returns per clause will have smaller effect, and the semantics of AND and OR will tend to blur together. In the worst case the engine will simply count query terms present, without respect to logical structure of the query. However, this is no disaster: counting query terms in a document is a viable retrieval strategy in its own right (see, e.g., Voorhees 1994). Recall that Boolean queries have been neglected in recent years in information retrieval research, partly because a strict adherence to logical structure too often excludes potentially relevant documents. Now if there simply are no documents relevant to the user's information need, then of course no relevant documents will be retrieved. The danger is that there are documents which address the concepts of the query in different terms than the user. When this happens the best we can hope for is that the relevant document will match a query term here and there – and this is precisely what will be detected and measured by OKRA's approach to Boolean queries in the worst case.

4.2.3 Handling Negation

The remaining question is how to handle negation. An answer to this question again comes from a careful analysis of how synonyms are handled. In the basic scoring system (key-words-with-synonyms), each clause is identified with a concept that the retrieved document should address. Occurrences of a word denoting a concept

is considered evidence *leaning toward sufficiency* that a document addresses that concept; while addressing an individual query concept is considered evidence *leaning toward necessity* that a document is relevant to the query. When we consider clausal forms of arbitrary Boolean queries, the difference is that a clause set may contain terms which are not synonyms, and may contain negated terms. For example, the clausal form of the query

```
burning OR (fire AND NOT(gun))
```

would be

```
(burning OR fire) AND (burning OR NOT(gun))
```

The two terms of the first clause are conceptually related; but the terms of the second clause, one of which is negated, seem to have simply been thrown together in the process of converting to clausal form.

However this does not prevent us from associating each clause with a concept, provided we allow the consideration of *disjunctive concepts*. A disjunctive concept is one marked by any of several sufficient conditions, which may vary in surface appearance, and which may incorporate negation in their description, but which nonetheless plays some well defined functional role in an appropriate context. An example of a disjunctive concept is a *strike*, as in "Strike three, you're out!" A pitch is considered a strike if it is in the strike zone and not swung at, swung at and missed, or hit foul and not the third strike in a single at bat. Viewed from the standpoint of their surface features, these various sufficient conditions for "strikeness" bear little resemblance to one another. But the point is that when we consider the peculiarities of the game of baseball, each condition can play the functional role of demonstrating the pitcher's outperforming the batter to a certain extent. At the same time, this functional role can only be satisfied once: If a pitch is in the strike zone *and* swung at and missed, it does not count as two strikes!

Similarly, when we consider the peculiarities of the user's information need, as expressed by his or her use of Boolean operators, the disjuncts of a clause in OKRA play the functional role of evincing the relevance of a retrieved document to a certain extent, in such a way that multiple conditions within a clause yielding a diminishing return. This is the real point of the scoring system for synonym lists, and this point can hold in essentially the same manner whether the diminishing returns are due to synonymy or to the logical structure of a query – and whether the individual items of evidence are positive or negative. Thus to straightforwardly handle negated terms in a clause set, the *absence* of a negated term is considered evidence for a document's relevance to a query, and is scored in basically the same way as the presence of a non-negated term. For example, given the query

```
burning OR (fire AND NOT(gun))
```

whose clausal form is

```
(burning OR fire) AND (burning OR NOT(gun))
```

a document could score in both clauses only by either containing the term *burning,* or by containing the term *fire* but not *gun.* One effect of this system is that all documents which do not contain the word *gun* – presumably most documents of the corpus – are scored as being at least slightly relevant. However, since documents compete only against each other, documents containing other, non-negated query terms still rise to the top of the list. (In actual implementation, documents are considered and scored only if they contain at least one query term which is not negated in the clausal form of the query.)

This brings up one question. Connection strengths of a term are weighted to account for the text features, such as boldface and italics, with which they are displayed. But non-displayed terms do not fit neatly into this scheme. Based on

experimentation, it was decided to give the non-occurrence of negated query terms a connection strength of 2 (yielding a minimum relevance score of 5 for clauses in which they appear). This parameter can be changed by the system administrator if the strength of the NOT operator needs to be adjusted. The higher the parameter, the less likely it is that the system will retrieve documents containing negated query terms. By making the parameter sufficiently large, it can be guaranteed that documents containing negated query terms will never be retrieved.

4.3 Implementation of the Scoring System

The OKRA user interface, written by David Boucugnani, parses Boolean queries and renders them as Prolog structures such as:

```
'OR'(foobar, 'AND'(foo,bar))
```

Each term in the query is then replaced by a disjunction of itself with all its selected synonyms. Next, the query is passed to the core search engine for evaluation.

The first task is to convert the expression to clausal form, which can be accomplished for any expression by repeated application of De Morgan's laws:

$$-(p \vee q) \Leftrightarrow (-p \wedge -q)$$

$$-(p \wedge q) \Leftrightarrow (-p \vee -q)$$

followed by repeated application of the distributive law for Boolean expressions:

$$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$$

A recursive Prolog program handles the tasks elegantly. For example, the process of traversing the tree structure of a Boolean expression and applying the distributive law as many times as necessary can be implemented in only seven lines:

```
distribute('OR'(X,'AND'(Y,Z)), 'AND'('OR'(X,Y),'OR'(X,Z)) ).
distribute('OR'('AND'(Y,Z),X), 'AND'('OR'(X,Y),'OR'(X,Z)) ).

simplify(X,Y) :- distribute(X,Y).
simplify(F(X,Y),F(XX,Y)) :- simplify(X,XX).
simplify(F(X,Y),F(X,YY)) :- simplify(Y,YY).

simplest_form(X,X) :- \+ simplify(X,_).
simplest_form(X,Y) :- simplify(X,Z), simplest_form(Z,Y).
```

The main predicate is `simplest_form`/2,which succeeds only when the distributive law no longer applies to the expression or to any nested constituent of it.

After conversion of the query to clausal form, a numerical score is calculated for each document in the corpus which contains at least one non-negated query term. This is done using the algorithm described in previous sections of this chapter, and was coded in LPA ProWeb by David Boucugnani.

Chapter 5

Performance and Evaluation

The U.S. Forest Service conducted tests of the OKRA search engine at Bent Creek experimental forest, located in Asheville, North Carolina, on July 24, 2001. Thirty-two potential users of the encyclopedia were divided into eight teams of four, and each team was given a set of four questions randomly selected from the following list:

1. What are some important criteria for selecting wildlife crop trees?
2. List 7 threats to the health of oak forests.
3. What are the effects of fire on acorns?
4. Name five hardwood species that regenerate from sprouts or suckers (advance regenerating).
5. What herbicide treatments can be used to control grapevines?
6. What are some management strategies for low-quality hardwood stands?
7. Where is old growth located in the southern Appalachians?
8. What are indicator plants for poor sites?
9. What are some important non-native invasive plant species in the Southern Appalachians besides kudzu?

10. What are some benefits of old growth to wildlife?

11. What are some of the cankers that affect oak?

12. How much forestland in the southern Appalachians is publicly held? How much of this land is unsuitable for timber?

13. What are some important things to consider when establishing artificial regeneration of yellow-poplar?

14. Is dogwood anthracnose in your county? When did it arrive?

15. What are some options for the reconstruction or restoration of problem roads?

16. Name 10 desirable timber species for moist sites in the Southern Appalachians?

The questions were prepared by Deborah Kennard, and were chosen so that the answer to each question appears somewhere in the hypertext. Each question was used at least once in the evaluation.

Each group was given between 12 and 15 minutes to answer four questions. All eight groups had 100% success, finding the answers to four randomly selected questions within the time allotted. Though not carried out under scientific conditions, this initial evaluation suggests that the OKRA search engine is quite effective in helping average users find the answers to typical questions in the encyclopedia.

Users were also asked to submit comments on the various aspects of the hypertext, including the search engine. Overall responses regarding the search engine were favorable. Responses to the synonym selection process, however, were mixed. Some users commented that the synonym utility was useful. However, others thought that the synonym selection took too much time – though no one suggested it should be completely gotten rid of. In response to this situation, an option has been added to the user interface which allows users to bypass the synonym selection process.

Bibliography

[1] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh Annual World Wide Web Conference.* See www7.scu.edu.au/programme/fullpapers/1921/com1921.htm.

[2] Burke, R. D.; Hammond, K. J.; Kulyukin, V.; Lytinen, S.L.; Tomuro, N.; and Schoenber,g S. (1997).Question answering from frequently asked question files – experiences with the FAQ FINDER system. *AI Magazine* 18 (2): 57-66.

[3] Chen, H. and Lynch, K. J. (1992). Automatic construction of networks of concepts characterizing document databases. *IEEE Transactions on Systems, Man, and Cybernetics* 22 (5): 885-902.

[4] Croft, W. B. and Thompson, R. H. (1987). I3R: A new approach to the design of document retrieval systems. *Journal of the American Society for Information Science* 38 (6): 389-404.

[5] Dreyfus, H. L. (1972). *What Computers Can't Do: A Critique of Artificial Intelligence.* New York: Harper and Row.

[6] Fellbaum, C., ed. (1998). *WordNet: An Electronic Lexical Database.* Cambridge, Mass.: MIT Press.

[7] Hirst, G. and St-Orange, D. (1998). Lexical chains as representations of context for the detection and correction of malaproposims. In Fellbaum (1998), pp. 305-332.

[8] Leacock, C. and Chodorow, M. (1998). Combining local context and WordNet similarity for word sense identification. In Fellbaum (1998), pp. 265-284.

[9] Leacock, C.; Miller, G. A.; and Chodorow, M. (1998). Using corpus statistics and WordNet relations for sense identification. *Computational Linguistics* 24 (1): 147-165.

[10] Manning, C. D. and Schütze, H. (2000). *Foundations of Statistical Natural Language Processing.* Cambridge, Mass.: MIT Press.

[11] Miller, G. A.; Beckwith, R.; Fellbaum, C.; Gross, D.; and Miller, K. (1993). Five papers on WordNet. Available at www.cogsci.princeton.edu/~wn/papers.

[12] Minsky, M. and Papert, S. (1970). Draft of a proposal to ARPA for research on artificial intelligence at MIT. As cited in Dreyfus (1972).

[13] Perez-Carballo, J. and Strzalkowski, T. (2000). Natural language information retrieval: a progress report. *Information Processing and Management* 36: 155-178.

[14] Rauscher, H. M. (2001). An Internet Based Encyclopedia of Southern Forest Science: Ready Access to the Right Information in the Right Form. Internal memo of the U.S. Forest Service.

[15] Salton, G. and Lesk, M. E. (1971). Computer evaluation of indexing and text processing. In G. Salton (Ed.), *The SMART retrieval system: experiments in automatic document processing,* 115-142. Englewood Cliffs, N.J.: Prentice Hall.

[16] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal* 27: 379-423, 623-656.

[17] Voorhees, E. M. (1994). Query expansion using lexical-semantic relations. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 171-180. New York: ACM Press.

[18] Voorhees, E. M. (1998). Using WordNet for text retrieval. In Fellbaum (1998), pp. 285-304.

[19] Wang, Y. C.; Vandenthorpe, J.; and Evens, M. (1985). Relational thesauri in information retrieval. *Journal of the American Society for Information Science* 36: 15-27.

Appendix A

Code for Creating Word Occurrence Facts

```
% make_word_facts(+Filename,+Atoms).
%
% cdr's through a list of Atoms to create word facts.
% Atoms should be a list of tokens for the file, provided
% by the tokenizer. This predicate assumes the blackboard
% has been cleared in preparation for a new file.


% if all tokens have been handled, succeed.
make_word_facts(_,[]):- !.

% if a token is a word, create a fact for it.
make_word_facts(Filename,[H|T]):-
    name(H,K),
    letter_token(K,K,[]),  % the token is a word
    lowercase_string(K,L), % convert it to lower case
    name(Atom,L),
    increment(position_in_text), % update the blackboard.
    current_paragraph(P),      % read the blackboard...
    position_in_text(Pos),
    current_file(CurFile),
    truncate_full_path(CurFile,RelFile),
    current_features(Feat),  % read the current text
            % features from the blackboard.

    % create a fact and recurse.

    writeq(word_fact(Atom,RelFile,paragraph(P),position(Pos),Feat)),
    write('.'), nl,
    !,
    make_word_facts(Filename,T).
```

```
% if an html tag ends a text feature which is currently
% active, retract it from the blackboard.

make_word_facts(Filename,[H|T]):-
    name(H,K),
    lowercase_string(K,L),
    tag([end(Feature)],L,[]), % the tag turns off a featue
    text_feature(Feature),
    Feature,
    retractall(Feature),
    !,
    make_word_facts(Filename,T).

% if a token is an html tag which begins a text feature,
% update blackboard to account for it. Note that in order
% for the this clause to succeed, the tag must begin,
% rather than end a text feature, because otherwise the
% previous clause would have succeeded and cut.

make_word_facts(Filename,[H|T]):-
    name(H,K),
    lowercase_string(K,L),
    tag([Meaning],L,[]),% the token is an HTML tag.
    text_feature(Meaning),
     assert(Meaning),
    !,
    make_word_facts(Filename,T).

% if an html tag begins a new paragraph, increment the
% paragraph counter.

make_word_facts(Filename,[H|T]):-
    name(H,K),
    lowercase_string(K,L),
    tag([paragraph],L,[]),
increment(current_paragraph),
    !,
    make_word_facts(Filename,T).

% failing all of the above, skip the token.
make_word_facts(F,[_|T]):-
    make_word_facts(F,T).
```

Appendix B

Code for Creating Connection Strengths from Word Occurrence Facts

```
% create_connection_strengths/0.
% Load word data into memory and create
% connection strengths.

create_connection_strengths:-
   consult('word_data.pl'),
   write('word data consulted'),nl,
   create_connection_strengths_aux.

% create_connection_strengths_aux/0.
% Recursively retract word facts and update
% connection strentghts. The second clause
% simply causes the predicate to succeed
% rather than fail when it runs out of
% facts to process.

create_connection_strengths_aux:-
   retract(word_fact(Word,File,_,_,Features)),
   score(Features,Score),
   increment_connection_strength(Word,File,Score),
   create_connection_strengths_aux.

create_connection_strengths_aux.

% increment_connection_strength(+W,+F,+S)
% If there is no existing connection strength,
% create one. If there is already a connection
% strength, augment it.

increment_connection_strength(Word,File,Score):-
   \+ con(Word,File,_),
```

```
   assert(con(Word,File,Score)).

increment_connection_strength(Word,File,Score):-
   retract( con(Word,File,OldScore)),
   NewScore is min(3, OldScore + Score),
   assert( con(Word,File,NewScore)).


% score(+Features,-Score)
% Returns the score for a given feature list.
% Clauses are listed in decreasing order by
% value returned, so that the value for a
% word is that of its highest ranked feature.

score(Features,3):-
   member(title,Features), !.

score(Features,2):-
   member(bold,Features), !.

score(Features,2):-
   member(italics,Features), !.

score(Features,1):
   member (body,Features).
```