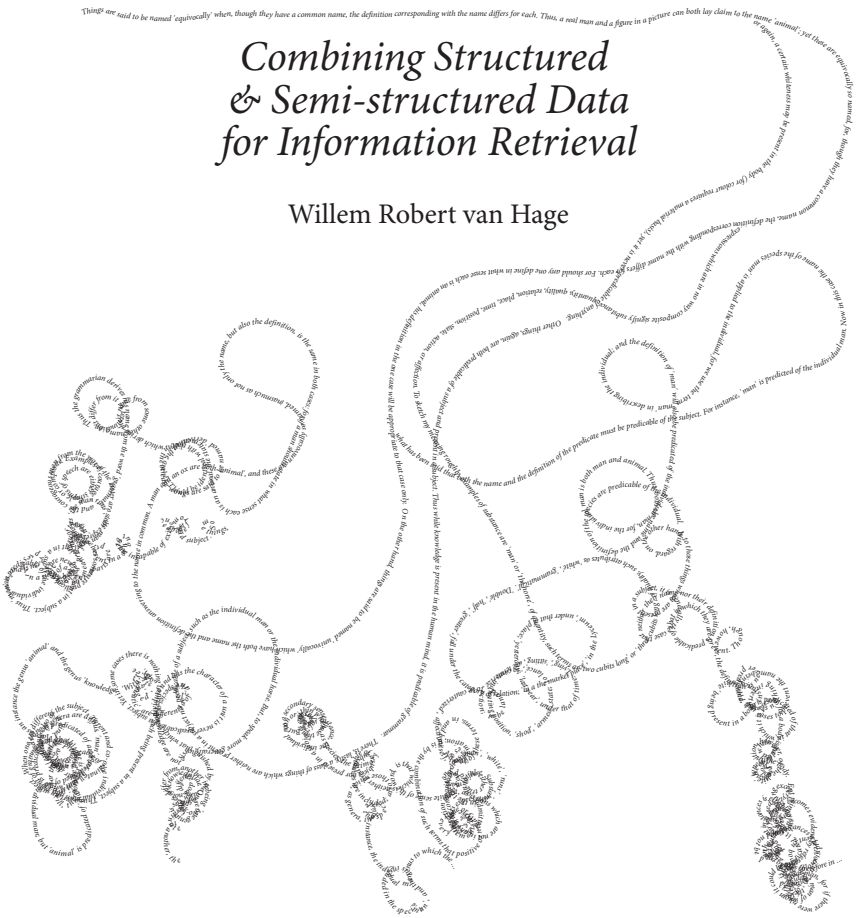
Living on the Edge



Combining Structured & Semi-structured Data for Information Retrieval

Willem Robert van Hage

Willem Robert van Hage

Living on the Edge,
Combining structured & semi-structured data for information retrieval

Master's thesis in theoretical computer science
University of Amsterdam, Language and Inference Technology Group
Supervisors: Maarten de Rijke, Maarten Marx

20 april 2004

Cover art: 'Drosophyla', algorithm by Willem van Hage and Isaac Sijaranamual, text taken from Aristotle's 'Categories'.

Preface

I started out my studies at the Univeriteit van Amsterdam with computer engineering. Amongst other things I learnt how to program microprocessors and how to use digital cirquit boards. One sunny afternoon, at the end of an eight-hour lab session, having spent two weeks trying to find out why the patch I set up on a cirquit board was not working properly, I opened the window blinds and suddently it worked. Then I closed them again and it stopped working. I was completely flabberghasted and while I stubled home in shock I decided to switch from engineering to theoretical computer science, hoping that I would never have to deal with the unpredictable nature of the outside world again. Yet here I am, graduating in theoretical computer science, doing exactly that...
I am back with a vengeance, armed with statistics!

Acknowledgements

First of all I wish to thank my supervisors, Maarten de Rijke for helping me to bring some structure into the fuzzy ideas that led to this thesis, and Maarten Marx for his guidance and for introducing me to the LIT group. Many thanks also go out to Khalil Sima'an for helping me with the theory behind detecting collocations. I also wish to thank my parents for their support, especially to my father, who helped me a lot with syntax of this thesis even though the semantics sometimes eluded him. Thank you Vera, for reminding me to live, and Isaac for helping me with the fruit fly algorithm that generated the cover art for this thesis. Finally, thanks go to all my friends at the university and in the band.

Contents

I Introduction 1
1 Motivation . . . 1
2 History . . . 2
2.1 Books . . . 2
2.2 Computers . . . 2
2.3 The Internet . . . 3
2.4 Internet Documents . . . 3
2.5 Electronic Books . . . 4
3 LoLaLi – Logic Language Links . . . 5
3.1 Handbooks . . . 5
3.2 Electronic Handbooks . . . 5
3.3 The Electronic Handbook of Logic and Language . . . 5
4 This Thesis . . . 6

II How & Why Retrieval Works 7
1 Searching in Text . . . 7
2 Preprocessing . . . 9
2.1 Document Representation . . . 9
2.2 Morphological Normalization . . . 10
2.3 Stop Words . . . 11
3 Indexing . . . 11
3.1 Inverted Indices . . . 11
3.2 Databases and B-Trees . . . 12
4 Weighting Schemes . . . 12
4.1 Cosine Similarity . . . 13
4.2 Zipf's law . . . 13
4.3 tf.idf . . . 15

4.4 Okapi 15
5 Evaluation 16
5.1 Measures 17
5.2 Significance Tests 18

III Structured Information 21
1 Searching versus Browsing 21
2 The LoLaLi Concept Hierarchy 22
3 Searching in the Concept Hierarchy 23
4 Concept Retrieval 24
4.1 Baseline 24
4.2 Short Queries and Documents 27
4.3 High Precision 28
4.4 False Matching 28
4.5 Stop Words 30
4.6 Domain-Specific Word Variations 32
4.7 Where idf & Normalization Fall Short 32
5 Noun Collocation Detection 33
5.1 Collocations in the Concept Hierarchy 34
5.2 Hypothesis Testing using the T-test 35
5.3 Collocations in the Contents of the Handbook 36
5.4 Collocations in Related Articles on the Web 37
5.5 Restraining the List of Collocations 38
6 Exploiting Collocations 39
6.1 Evaluation of Collocation Preference 41
6.2 Evaluation of Exact Match Preference 41
6.3 Conclusions 42
7 Exploiting Structure 43
7.1 Grouping Concepts from the Hierarchy 44
7.2 The Effect of Grouping 44
7.3 Evaluation of Grouping 47
7.4 Evaluation of Grouping on a Smaller Set 48
7.5 Evaluation of Grouping with Preferences 49
7.6 Conclusions 49
8 Discussion 50

IV Semi-structured Information 51
1 Automatic Hyperlinking . . . 52
1.1 Entry Points . . . 53
1.2 Overlapping Units . . . 53
1.3 Baseline . . . 53
1.4 Doing Better than the Baseline . . . 55
2 Exploiting Annotations . . . 56
3 Exploiting Word Order with Collocations . . . 57
4 Exploiting Concept Relations . . . 58
4.1 Expanding with Parents and Children . . . 59
4.2 The Response of Certain Topics to Query Expansion . . . 60
4.3 Correlations with the Reaction to Query Expansion . . . 61
5 Discussion . . . 62

V Conclusion 63
1 Concept Retrieval . . . 63
1.1 Concept Retrieval and Word Order . . . 64
1.2 Concept Retrieval and Concept Relations . . . 64
1.3 Concept Retrieval Conclusion . . . 65
2 Automatic Hyperlinking . . . 65
2.1 Automatic Hyperlinking, Annotations and Word Order . . . 66
2.2 Automatic Hyperlinking and Concept Relations . . . 66
2.3 Automatic Hyperlinking Conclusion . . . 67

A Topics 69
1 Concept Hierarchy Topics . . . 69
2 Automatic Hyperlinking Topics . . . 70

List of Tables

II.1 Examples of Porter-stemmed words. 10
II.2 Examples of *TreeTagger* lemmatized words. 10
II.3 A ranking of word frequencies in the *Handbook of Logic & Language*. 14

III.1 Baseline results for concept retrieval 26
III.2 *Incremental R-Precision* for example rankings. ○ indicates an irrelevant document and ● indicates a relevant document. 27
III.3 Average document size versus term frequency 29
III.4 Stemming and lemmatization results for concept retrieval 30
III.5 The 10 words with the highest avg. tf and cf in the concept hierarchy 31
III.6 Stoplist results for concept retrieval 32
III.7 Part of speech tag patterns for collocation filtering 34
III.8 Collocations extracted from the concept hierarchy data 35
III.9 Critical t-scores for various significance values 37
III.10 Collocations extracted from the handbook 37
III.11 Collocations extracted from the world wide web 38
III.12 Collocation preference factor results for concept retrieval 41
III.13 Collocation preference results for concept retrieval 41
III.14 Exact match preference results for concept retrieval 42
III.15 Results for concept retrieval with lemmatization, and collocation and exact match preference. 43
III.16 Grouping results for concept retrieval 48
III.17 Grouping results for concept retrieval 48
III.18 Collocation and exact match preference results for concept retrieval runs that use grouping 49
III.19 Final results for concept retrieval 50

IV.1 Automatic hyperlinking baseline results. 55
IV.2 Runs that exploit title and emphasis annotations. 57
IV.3 Runs that exploit collocations. 58
IV.4 Runs that exploit collocations. 59
IV.5 Query properties versus their reaction to query expansion. 61

V.1 Collocation and exact match preference results for concept retrieval 64
V.2 Final results for concept retrieval 65
V.3 Runs that exploit collocations. 66
V.4 Runs that exploit collocations. 67

List of Figures

I.1 The *LoLaLi Exemplar* Architecture 6

II.1 Weighting schemes assign a number to each document that indicates its similarity to a query. The documents can be ranked according to this number. 13
II.2 Zipf's Law, rank against frequency, for terms in the *Handbook of Logic & Language*. 14
II.3 Quantile plots of runs from chapter IV. Shown are a comparison between the baseline and a run exploiting emphasis annotations (left), and between a run that exploits titles and one that also exploits collocations (right). 20

III.1 An excerpt from the LoLaLi concept hierarchy 23
III.2 Information retrieval from the concept hierarchy 25
III.3 Average number of terms per web search engine query 27
III.4 The list of stop words I used 31
III.5 Removing the influence of English 39
III.6 Restraining the subject of the collocations 40
III.7 Is having important relatives better than having talent? 44
III.8 Concept grouping rules 45
III.9 Added context 46
III.10 Oedipus and modal logic 46
III.11 How a ranking is obtained after grouping 47

IV.1 An excerpt from the *Handbook of Logic & Language*. 51
IV.2 Nested annotations in LaTeX. 52
IV.3 `\key` commands refer to the smallest enclosing unit. 54

IV.4 Eliminating overlapping units from the ranking. 55
IV.5 Emphasis on a key phrase. 56
IV.6 Big units eliminate many descendants from the ranking. 60

Chapter I

Introduction

How this thesis fits in the big picture

1 Motivation

There are many ways to deal with the huge amounts of data in the world. The *Semantic Web* (Antoniou and van Harmelen, 2004) springs from the idea that underneath the apparent chaos there is order that can be modeled with logic, while information retrieval (Baeza-Yates and Ribeiro-Neto, 1999) started out by accepting that the world is a messy place that does not fit in the strict world of logic and should be accessed through statistics.

In practice these two views do not have to collide. In projects such as *Yahoo!* (Yahoo! Inc., 1995) and the *Open Directory Project* (Netscape, 1998) information retrieval and knowledge models in the form of subject hierarchies have been successfully combined, allowing users to complement their browsing with searching. In this thesis I want attempt to go one step further than just providing two separate ways to access the same data by allowing knowledge models influence retrieval directly.

I will do this in experiments within the *LoLaLi* project, which is described in section 3. Before actually getting into details I first want to provide some historical perspective of the ideas that come together in this thesis.

2 History

Informatie is de enige grondstof die groeit in het gebruik
— Johan van Benthem

Almost everybody in our society uses information on a daily basis. It pours into our houses through every connection with the outside world. Every thought that leaves our heads starts a life of its own and starts multiplying itself with the speed of its medium.

2.1 Books

Serious information growth started when people switched from the instantly vanishing spoken word, that died when there was nobody around to hear and understand it, to the longer lasting written word. As far as we know now the earliest of at least two different inventions of writing, called cuneiform, happened in ancient Sumeria around 3200 BC. Writings were scarce and people had to share. By 1700 BC the first libraries of clay tablets were established in Babylonia and in 384 BC Aristotle was born, who became the first known systematic collector of books and moreover the first to classify writings by their subject.

In 1450 AD Johannes Gutenberg invented the printing press in Mainz and in 1455 he was the first to print the *Bible*. The printing press caused the second big growth of the speed with which information multiplies itself by reducing the time it takes to make a book as substantial as the *Bible* from a lifetime to two years and by making it possible to make multiple copies of printings without additional decorations at the same time.

2.2 Computers

The next big step started with the concept of a computing machine. In 1642 Blaise Pascal created a calculating machine that could do addition and subtraction, driven by a hand operated crank. twenty-nine years later Baron von Leibnitz came up with the idea that this machine could also do multiplication and it took about a century for the mass production of similar machines for desktop use to start.

In 1837 Charles Babbage came up with the first programmable computing machine, the *Analytical Engine*. It was supposed to become a massive steam-powered brass mechanical computer that could be made to do just about anything, but it proved to be too expensive to actually build. His dream had to stay a dream until the invention of electro-mechanical computers by Konrad Zuse in 1938.

Computers continued to grow, from punch card accounting machines to enormous vacuum-tube computers used in the second world war, to the first computers that used transistors in 1958. Storage evolved from aperture card, optical coincidence cards, edge-notched cards to cassette tapes and optical storage. In the 1960s the books that once took a lifetime to copy could be copied while you were getting coffee.

The storage machinery of the 1960s was already so powerful that you could easily lose your data between the enormous amounts of other stored data. Bringing Aristotle's order to modern-day heaps of documents required something smart. In 1961 Gerard Salton started working on a system for the mechanical analysis and retrieval of text, called SMART (Salton and McGill, 1983), which in the following thirty-four years grew to become the most widely-used research tool for information retrieval.

2.3 The Internet

While Babbage was dreaming about his *Analytical Engine*, Samuel Morse was working on sending messages instantly over a long distance. In 1844 he transmitted the first electronical message, *What hath God wrought?*, taken from the Bible, from Washington to Baltimore over a telegraph wire. The telegraph took the world by storm. Eventually you could send telegraphs to the America's and even to the Himalayas.

In 1876 Elisha Gray and Alexander Graham Bell invented the telephone. Bell filed his patent only two hours before Gray tried. Bell and Gray realized that Morse's telegraph lines could be used to transmit sound. Their telephones were the first practical use of Michael Faraday's discovery that vibrations of metal could be converted to electrical pulses and back.

In 1973 TCP/IP, the *Internet and Transmission Control Protocols*, were developed by Vinton Cerf, and the internet was born. First it only connected universities and military installations, but when personal computers became affordable and small telephone modems were invented in the 1980s everybody with a phone connection could join.

2.4 Internet Documents

In 1990 Alan Emtage created *Archie*, an indexing search tool for FTP sites, because the number of internet hosts had grown larger than 100,000 and people started getting lost. One year later Timothy Berners-Lee developed the *World Wide Web*, WWW (Berners-Lee, 1994), a network of documents that everybody could add to.

To make this possible he came up with HTML, the *Hypertext Markup Language* (the World Wide Web Consortium, 1992), a document formatting standard that allowed you to point to other documents in such a way that the person (for the time being) reading the document could automagically retrieve the target document.

In 1993 Matthew Gray and Martijn Koster created two robot programs, the *World Wide Web Wanderer* and the meta-tag spider *Aliweb* respectively, that could travel this network of documents and collect data from it for their masters.

In 1994 *Yahoo!* (Yahoo! Inc., 1995) became the first hierarchical directory of the *World Wide Web*. *Yahoo!*'s trick to manage the enormous quantity of documents on the web was to appoint a caretaker, a librarian if you want, for every topic in their hierarchy. This resulted in a high quality collection of documents, but it was not a very scalable solution.

The first system that tried to index the web was *Altavista* (Altavista, 1995), which was created by the Digital Equipment Corporation's Research Lab (Olsen, 1957) in 1995. They devised a way to store nearly every word on the web in a fast searchable index. This made it possible to actually find new information on the rapidly growing web.

2.5 Electronic Books

The popularization of the internet that followed made fast and correct duplication of information possible for a great audience. The books that once took year to copy can now be downloaded in a few seconds from the web.

This recent speed-up in the multiplication of information reveals great opportunities for publishers. Publishing costs and time can be greatly reduced, since there is no need for printing. Furthermore electronic books can be transfered to readers much faster than material books.

However, there are some big problems that come with electronic books. The first being that if you sell your book to one person, you are really selling it to everybody on the internet, because it can be copied in the blink of an eye. And the second being that readers do not get the same feeling with electronical publications as with real books. Apart from subjective emotional aspects such as the smell of a book there are also objective advantages of real books. For example, books have a physical size. You can easily see how big a book is, roughly how many books there are in a library, and if you are already halfway through reading a book. Also, a physical publication gives you a much better idea of the quality of the contents before even reading it than an electronic book.

If you have to familiarize yourself with a subject you walk into the library, search for a bookcase with the right topic written on a sign, and pick out the book

that looks like an introduction or an overview. This is not possible with electronic books. They are just abstract files on a computer somewhere without any body-language.

Scientists have to familiarize themselves with subjects all the time. Electronic publishing has helped science a great deal, but electronic exploration of a new subject is still lacking. The *LoLaLi* project tackles this problem.

3 LoLaLi – Logic Language Links

3.1 Handbooks

In the world-wide scientific community people spread around their ideas by writing articles. Since the goal of researchers is to come up with new ideas and test whether or not there is some truth or money in them the output of scientific papers is enormous. Anybody new to a field of science is faced with a mountain of papers and no clue where to start reading. To help solve this problem scientists with a good view of the big picture sometimes write handbooks. These books try to connect all the research done in a certain field.

3.2 Electronic Handbooks

The aim of the LoLaLi project is to find out what scientific handbooks should look like in the age of electronic publishing. In order to do this it was decided to create a test environment based on an already existing handbook, where new ideas can be made concrete. This electronical version of the handbook will contain the same content as the original, extended with interactive tools, and will serve as a prototype for electronic handbooks in other fields of research. As an example case the *Handbook of Logic & Language* (van Benthem and ter Meulen, 1997) was chosen, so that the people involved would have some affinity with the content.

3.3 The Electronic Handbook of Logic and Language

The prototype of the electronical version of the handbook, dubbed the *LoLaLi Exemplar*, contains the original text of the *Handbook of Logic & Language* and a hierarchy of concepts that appear in the text.

The concept hierarchy provides the reader with a conceptual map of the handbook. It can give the reader an idea of the contents much faster than diagonally reading the 1200-page book. The idea is that the reader can browse through the concept hierarchy in a similar way as browsing through *Yahoo!'s* subject hierarchy

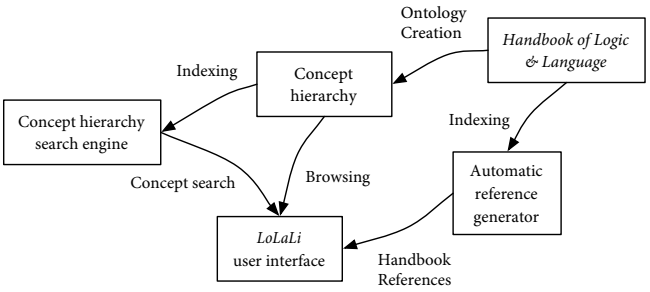


Figure I.1: The *LoLaLi Exemplar* Architecture

and when he is lost or wants to jump around in the hierarchy he can type in some keywords and continue his search at the locations provided by a search engine.

Each concept in the hierarchy contains links that refer to pieces of text in the handbook that are about the same subject. So the hierarchy can also be used to search for a subject in the book, just like an index in the back of a book. The difference being that such an index is sorted alphabetically, while the concept hierarchy is sorted by the meaning of the words.

4 This Thesis

This thesis is about two search tasks. The first task is constructing a search engine that helps the reader find concepts in the concept hierarchy. This is what happens inside the block labeled 'Concept hierarchy search engine' in figure I.1. Chapter III explains this task in detail.

The second is a providing a way to automatically generate references from the concepts to places in the handbook. In figure I.1 this corresponds to the block labeled 'Automatic reference generator'. Chapter IV shows how this can be accomplished using information retrieval methods.

Through the experiments necessary to accomplish these tasks I would like to learn whether (and how much) information retrieval can benefit from knowledge models such as thesauri and ontologies, and from the information encoded in the structure of semi-structured data.

Before I start with searching in the concept hierarchy I will first, in chapter II, explain the information retrieval tools I will use for computing similarity and evaluating results.

Chapter II

How & Why Retrieval Works

A short introduction to information retrieval

> *Zoekt en gij zult vinden, vindt gij niet dan is het zoek.*
> — Pun on Matthew 7:7

This chapter deals with trying to find something you are looking for in a large piece of text or a collection of pieces of text. Trying to find something can mean many things, for example: looking up something which you know to be somewhere in the text beforehand, *known item search*; (Mishne, 2004); looking for a passage that answers a question you have in mind, *question answering* (Monz, 2003; Monz and de Rijke, 2001; Jijkoun et al., 2003); trying to find new information about something you already know, *novelty search* (Harman, 2002; Monz et al., 2002b); and finding passages which are about a certain topic, *relevance search* (Salton and McGill, 1983). The kind of search I will discuss is of the last kind, *relevance search*. Relevance is nothing in itself. Something is relevant to something. The key thing in relevance search is that which things are relevant to, which is called an information need.

1 Searching in Text

Say your information need is finding someone's phone number. With a little experience you can quickly find names in a phonebook. You do not have to start reading the book at the beginning, but you can split the book where you guess the

name will appear and if the name is not on the page where you opened the book you can search the part of the phonebook that does contain the name in the same way. This way even a bad guesser (someone who just splits the book in half) can find the name he is looking for in a phonebook with 1000 pages in about 10 splits and a bit of reading on the page that contains the name. (Knuth, 1968)

Such search methods are only possible if: *a)* it is clear when you have found what you are looking for and *b)* the text is sorted in some way. The latter is only possible if you know in advance what the relevant property of the text is and if that property can be ordered. The part of a phonebook that is relevant to the need to look up someone's phone number is the numbers that go with the names. Every number has a name, the reader knows the name he is looking for, and names can be sorted, so the phonebook can be sorted by name. If your information need is different however you can not always sort the content you are interested in. In a novel or a newspaper for example the relevant part of the text is the meaning of the text, which can not be sorted just like that.

The solution to this problem is to make the text sortable, while not upsetting its meaning so much that it becomes impossible to see if the text is relevant to the user's information need. The first thing you have to do is to formalize the kinds of information needs you want to cater to. Instances of such information needs are called topics. Based on this formalization you can decide what pieces of the text can possibly satisfy such information needs. Given that you cut the text into these pieces that could be relevant to a possible topic. For example a newspaper could be cut into articles. These pieces are called documents. Now deciding whether you have found what you are looking for has been reduced to checking whether a document is relevant to a topic. This leaves four open questions:

1. *How do you represent documents mathematically?* I will discuss this in section 2 about preprocessing.

2. *How do you sort the documents in such a way that you can efficiently search through them?* This will be answered in section 3.

3. *How do you model the human capability to decide whether a document is relevant to a topic in such a way that it can be done by a computer?* I will discuss some solutions that I use in this thesis in section 4 about weighting schemes.

4. *How do you evaluate the quality of such a model?* In section 5 about evaluation measures and methodology I will elaborate how this is usually done in the information retrieval community.

2 PREPROCESSING

For a computer text is just a big stream of symbols. To make it clear which of those symbols belong together and what means the same thing and what does not, you must process the text. Deciding what is a viable word and what is not is not an simple task. Words can include all sorts of characters: 'bag-of-words', 'Sima'an', 'S5', '1984', etc. Sometimes white space can even be disputable as a word delimiter. The problem becomes clear when you look at the official spellings of 'phone number' and 'phonebook'. To every rule there are exceptions, so to get anything done you have to decide which rules hurt the least.

2.1 DOCUMENT REPRESENTATION

To reason about the contents of documents we need to formalize it. Documents are usually represented with multi-sets, sets of pairs of words and their frequency. An example is shown below.

$$d = \{\langle the, 204\rangle, \langle brown, 12\rangle, \ldots\}$$

This representation is called a *bag-of-words*, because the order of the words is lost hence it is just like throwing all the words in a bag. If you want to look if two words occur next to each other, or if you want to measure the distance between words like Minimal Span Weighting (Monz, 2003) does, you will need the position of the words. You can adapt the representation to include positional information however. For example by using triples instead of pairs where the third element is a set of positions in the document where the word occurs.

$$d = \{\langle the, 204, \{1, 6, 19, \ldots\}\rangle, \langle brown, 12, \{2, 20, \ldots\}\rangle, \ldots\}$$

To index an entire collection you number all the documents and add the document number to all pairs or triples in a document's representation.

$$C = \{\langle 1, the, 204\rangle, \langle 1, brown, 12\rangle, \ldots, \langle 2, the, 67\rangle\}, \ldots$$

In real world text there are many possible writings for the same word, with and without capitals for example. To identify these variations all words are given a term number and words that should be identified get the same number. The mapping from words to numbers is usually called the dictionary.

$$C = \{\langle 1, 1, 204\rangle, \langle 1, 2, 12\rangle, \ldots\}$$

$$Dictionary = \{\langle The, 1\rangle, \langle the, 1\rangle, \langle brown, 2\rangle, \ldots\}$$

Russell's	→	russel
papers	→	paper
are	→	ar
philosophical	→	philosoph

Table II.1: Examples of Porter-stemmed words.

Russell's	→	Russell
papers	→	paper
are	→	be
philosophical	→	philosophical

Table II.2: Examples of *TreeTagger* lemmatized words.

2.2 Morphological Normalization

The mapping to numbers makes two terms either equal or unequal. Some words are more equal than others however, such as conjugations of a verb, or the singular and plural form of a noun. The different forms of the same word are called morphological variations. It has been shown that mapping morphological variations to the same term improves retrieval significantly (Hollink et al., 2004). There are various ways to accomplish this: *n-grams*, *stemming*, and *lemmatizing*.

Porter's Stemmer Stemming is a simple way to reduce a word to a stem form that does look at its context. The most popular stemmer is *Porter's stemmer* (Porter, 1980), which uses a set of regular expressions on the end of the word and chops off the matching part. In table II.1 you can see a few examples of Porter-stemmed words. As you can see *Porter's stemmer* stems quite radically and the stems are not the linguistic stems of the words. This is so because it is not possible to determine the correct stem of a word if you know nothing about its context.

Schmid's TreeTagger Lemmatizers address this problem. They parse a text 'deeply' enough to determine what the function of a word is in a sentence, i.e. if a word is a noun, adverb, verb, etc. Based on this it is decided what the right stem of the word is. An example of a lemmatizer is *Schmid's TreeTagger*. (Schmid, 1994) The output of the *TreeTagger* is shown in table II.2. As you can see, the verb 'are' is now recognized as a conjugation of the verb 'to be'.

N-Grams Another way to match word stems is to look at the text through a shifting window. So that you only look at part of a word at a time and therefor enable partial matching of a word, so that 'philosophical' and 'philosopher' match,

but less so than 'philosopher' and 'philosopher'. Letter sequences of length n are called n-grams. It is possible to let them cross word boundaries or to let them stop at word boundaries. The sequence 'papers are' contains the following border-crossing 4-grams:

$$\{\texttt{pape}, \texttt{aper}, \texttt{pers}, \texttt{ers␣}, \texttt{rs␣a}, \texttt{s␣ar}, \texttt{␣are}\}$$

or the following non-border-crossing 4-grams:

$$\{\texttt{pape}, \texttt{aper}, \texttt{pers}, \texttt{are}\}$$

2.3 Stop Words

In section 4.2 I will show that in every natural text there are words that occur very frequently, for example 'the', 'of', 'is' and 'a'. If your topics and documents are large enough to need these words (try thinking up a sentence of 20 words without using any of the words listed above) these words can negatively influence retrieval scores. (Hollink et al., 2004) A solution to this problem is to create a list of words that can be disregarded. Such a list is called a stop word list or stop list.

3 Indexing

If you want an answer to your topic quickly, checking the similarity between the query and every document will take too much time. Just like reading the phone-book from the front to the back takes too much time. Since similarity is defined in terms of terms that occur both in the query and the document you only have to check documents that contain at least one query term.

3.1 Inverted Indices

In order to do that quickly, the collection will have to be sorted by term. Meaning that you can look up a term and get a list of documents that contain it. This sorting process is called inverting. The resulting index is called an inverted index. Some inverting methods minimize the necessary time, others minimize the memory or disk space. Usually time and memory space are scarce, so disk space is sacrificed. A fast, economical inverting method that minimizes inverting time and memory space is *Fastinv* (Witten et al., 1999).

3.2 DATABASES AND B-TREES

Another option is to let a database do the work for you. Internally most database systems are organized with B-Trees.

B-Trees combine a good insertion time and a good lookup time. The access time of a B-Tree is $\mathcal{O}(\log n)$. Since B-Trees are balanced trees the worst-case lookup time is close to the average-case lookup time, because the length of all the paths in the tree is almost the same. Inserting a record into a B-Tree takes $\mathcal{O}(1)$ time after the right place in the tree has been found, which takes $\mathcal{O}(\log n)$ time. So inserting an entire collection takes $\mathcal{O}(n \log n)$ time, where n is the number of terms to be indexed.

In this thesis I will use the FlexIR retrieval system (Monz et al., 2002a) on top of a *MySQL* database. (DuBois, 2003)

4 WEIGHTING SCHEMES

Humans with enough knowledge on the subject of a document can decide with reasonable certainty (Schamber, 1994) whether it is relevant to a given topic. True understanding of the meaning of a document cannot be computed. So if you want to let a computer decide which documents are relevant to a topic you have to use other, computable properties of the documents that have a strong correlation with relevance.

In information retrieval this is usually done by creating a function that takes the mathematical representation of a topic (called a query), and a document, and assigns a number to the document that indicates how similar they are. Such a function is called a weighting scheme. Then all the documents are sorted by this number, producing a ranking with the documents that the function decided to be the most similar at the top and the least similar at the bottom. This is illustrated in figure II.1. Weighting schemes assume that similar documents are relevant documents. The meaning of a text lies in its words. Words that look the same usually mean the same thing and words that look different usually mean something different. So it is not a strange assumption that two texts which contain the same words are about the same subject.

So this moved the problem from deciding whether or not a document is relevant to a topic to assigning high numbers to documents that are similar to a topic. Many different modeling strategies have come up in the past decades, the most prominent are boolean, vector and probabilistic models. (Salton and McGill, 1983) I will only go into detail about two vector models, *cosine similarity* and *tf.idf*, and one probabilistic model, *Okapi*.

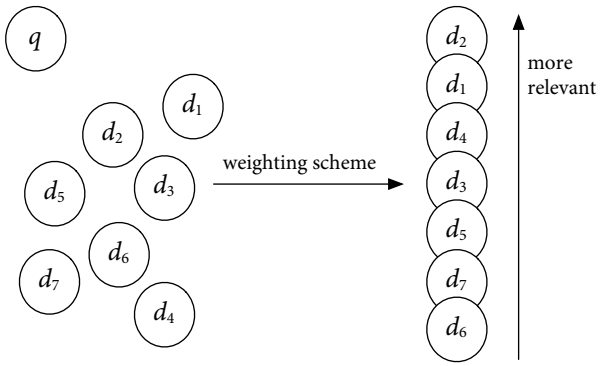


Figure 11.1: Weighting schemes assign a number to each document that indicates its similarity to a query. The documents can be ranked according to this number.

4.1 Cosine Similarity

As mentioned in 2.1, documents are represented as vectors of term frequencies. Vectors that point in the same direction have a similar term composition and orthogonal vectors have no common terms, so an obvious way to define the similarity between two document is as the angle between their vectors.

The *cosine measure* is a function of the angle between a document and a query. Its values range between zero (orthogonal), and one (identical). The definition is shown in the equation below, where d_i stands for the frequency of term i in document d.

$$Sim(d,q) = \frac{\sum_{i=1}^{n} d_i q_i}{\sqrt{\sum_{i=1}^{n}(d_i)^2 \cdot \sum_{i=1}^{n}(q_i)^2}}$$

The *cosine measure* does not take into account that some words are less meaningful than other words.

4.2 Zipf's law

All natural human language is susceptible to Zipf's law (Zipf, 1949) also called the power-law, or Pareto distribution when formulated differently. This means in the case of words that there are only a few words that occur frequently, such as 'the' and 'a', and many words that occur infrequently, such as 'articulated' and 'smoldering'. Zipf states that if you count all words in a corpus and make a list of the frequencies as shown in table 11.3, that the relation between a word's frequency,

word	frequency	rank
the	30.559	1
of	21.703	2
conjugations of 'to be'	21.640	3
a	17.168	4
and	13.354	5
...	...	...

Table II.3: A ranking of word frequencies in the *Handbook of Logic & Language*.

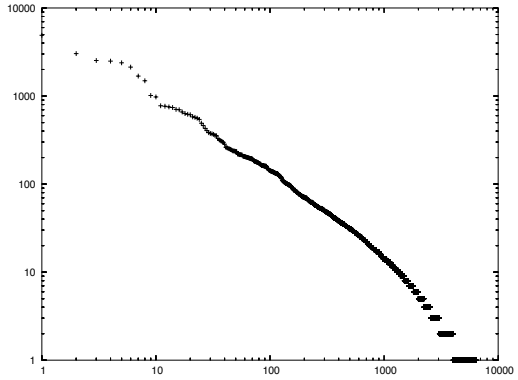


Figure II.2: Zipf's Law, rank against frequency, for terms in the *Handbook of Logic & Language*.

f, and its rank r is:

$$f = \frac{k}{r}$$

for some constant k, which depends on the corpus. That means that if you plot the frequency against the rank, both scaled logarithmically, that you get a straight line similar to the one shown in figure II.2.

Such a heavy-tailed distribution occurs whenever there is a clash between introducing new objects and using already existing objects and when there is a slight preference for using existing objects (Barabasi et al., 1999). When people want to communicate they have to use the same words more often than not, or they will not be able to understand each-other, but if they only use the same words all the time they will not be able to express themselves sufficiently. Since people tend to

use the same words as often as possible to make their message easier to understand by the receiver, their choice for a rare word implies that the word is important for the meaning of the message. Seen in the light of information theory, these rare words contain the most information about the text (Shannon, 1948; Li and Vítanyi, 1997).

Most weighting schemes exploit the fact that rare terms are likely to be important by letting them influence the similarity between a query and a document more than other terms.

4.3 TF.IDF

The *tf.idf* weighting scheme is such a weighting scheme. (Salton and McGill, 1983) The *idf* part of *tf.idf* stands for 'inverted document frequency', which is defined in the equation below, where N is the total number of documents in the collection and n_i is the number of documents that contain term i:

$$idf_i = \log \frac{N}{n_i}$$

There are a few different definitions of the *tf.idf* similarity measure. The simplest version is listed below, where q is the query and d_j the document and $tf_{i,j}$ stands for how often term t_i occurs in document.

$$tf \cdot idf_{q,d} = \sum_{t_i \in q \cap d_j} tf_{i,j} \cdot idf_i.$$

In this thesis I use a version that normalizes the term frequencies in a document. This has the effect that the *idf* factor has relatively more influence. This version is shown below, where $\max_k tf_{k,j}$ stands for the frequency of the most frequent term in document j.

$$tf \cdot idf_{q,d} = \sum_{t_i \in q \cap d_j} (\frac{1}{2} + \frac{1}{2} \frac{tf_{i,j}}{\max_k tf_{k,j}}) \cdot idf_i.$$

4.4 OKAPI

Another weighting scheme that uses a form of inverted document frequency is *Okapi* (Kwok et al., 2000; Robertson et al., 1996). *Okapi* is a probabilistic model, meaning that it estimates the probability that a document is relevant to the query, whereas vector models such as *tf.idf* just produce an ordering by estimated relevance. In addition to differentiating between common and rare terms, *Okapi* also implements document length normalization.

In this thesis I will use the BM25 variant of *Okapi*, which is described below:

$$okapi\_sim(q,d_j) = \sum_{i \in q} okapi_{i,j} \cdot npn_{i,j}$$

$$okapi_{i,j} = \frac{(k_1+1)\cdot tf_{i,j}}{K + tf_{i,j}}$$

$$npn_{i,j} = ntf_{q,i} \cdot \log \frac{N - df_j}{df_j}$$

$$ntf_{q,i} = \frac{1}{2} + \frac{1}{2} \cdot \frac{tf_{q,i}}{\max tf_q}$$

where:

- $tf_{i,j}$ is the term frequency of term j in document i
- $K = k_1 \cdot ((1-b) + b \cdot \frac{|d_i|}{avgdoclen})$
- *avgdoclen* is the average document length
- $\max tf_q$ is the maximal term frequency of any term in the query
- N is the number of documents in the collection
- df_j is the document frequency of term j in the collection

The $okapi_{i,j}$ part of the weighting scheme calculates whether term j occurs particularly often in document i, based on the estimated frequency of j given the length of document i. The $npn_{i,j}$ part calculates how rare term j is, based on how many documents contain it, compared to the total number of documents.

5 EVALUATION

The quality of a search engine depends on how well it separates documents that are relevant to the user's information need from those that are irrelevant to it. So to determine how good a search engine is it is necessary to know which documents are relevant and which are not. A big problem is that people do not always agree on whether a document is relevant or not. (Schamber, 1994) The less specific the topic, the more disagreement. This makes it difficult to make strong claims about the quality of a search engine in itself, for comparative evaluation however, they are very useful (Voorhees and Harman, 2002).

To compare the results of two search engines it is necessary to have some measure that expresses the quality of each ranking in a meaningful number. These numbers can then be compared. If there is a significant difference between these numbers it is safe to say that one method is better or worse than the other. Van Rijsbergen goes into great detail about evaluation in his classic 'Information Retrieval' (van Rijsbergen, 1979).

In the next subsection I will describe some of the evaluation measures I will use in this thesis. In subsection 5.2 I will go into detail about how to decide whether the difference between two evaluation scores can be called significant.

5.1 MEASURES

RECALL *Recall* measures how many of the relevant documents in the collection have been retrieved.[1] Let *Relevant* be the set of all documents that are relevant to the topic and *Retrieved* be the set of all retrieved documents for the query corresponding to that topic.

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|}$$

PRECISION *Precision* measures how many of the retrieved documents are relevant to the query.[2]

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$$

P@N – PRECISION AT A DCV If you want to know if the relevant documents are anywhere near the top of the ranking you can cut-off the ranking at a certain point, called a document cut-off value (DCV), usually named n. For this top-n of the ranking you can compute *Precision*, this is called *p@n*. It is defined as follows:

$$p@n = \#\{d \in R_q \mid \text{rank}(d) \leq n\}/n,$$

where R_q is the ordered list of retrieved documents for query q.

R@N – RECALL AT A DCV You can calculate *Recall* at a DCV too, as follows:

$$r@n = \#\{d \in R_q \mid \text{rank}(d) \leq n\}/\#R_q$$

[1] The whole truth,
[2] and nothing but the truth.

AVERAGE PRECISION The mean of the *Precision* scores obtained after each relevant document is retrieved, using zero as the *Precision* for relevant documents that are not retrieved.

$$AP = \sum_{d \in RR} \frac{p@\text{rank}(d)}{|RR|},$$

where $RR = |Retrieved \cap Relevant|$. The the mean of the average precision score over all queries is called the *Mean Average Precision*, MAP. (not to be confused with the abbreviation of 'maximum a priori')

$$MAP = \sum_{q \in Q} \frac{AP(q)}{|Q|},$$

where Q is the set of queries.

R-PRECISION A downside of MAP is that when the ranking is longer than the number of relevant documents in the collection, MAP does not sum up to 1, but to $\frac{|Relevant|}{|Retrieved|}$. A measure that solves this problem is called *R-Precision* which essentially is $p@n$ where $n = |Relevant|$:

$$RP = p@|Relevant|.$$

INCREMENTAL R-PRECISION *R-Precision* makes no difference between finding three out of ten relevant documents at the top of the ranking or at the bottom of the ranking. Furthermore it is a relatively unstable measure (Buckley and Voorhees, 2000), like any $p@n$, because it is a single measurement and not mean value. A solution for both these problems is to average the precision values at 1 through $|Relevant|$:

$$IRP = \sum_{n=1}^{|Relevant|} \frac{p@n}{|Relevant|}.$$

5.2 SIGNIFICANCE TESTS

There are two ways in which one search engine can perform better than another. One is that it performs well more often than the other, and the other is that when it performs better, it performs much better than in other cases.

SIGN TEST The former, frequency based, property can be measured using the *sign test*, which counts how often the result of search engine A, A_i, is better than the result of search engine B, B_i, and compares that to the number of samples n.

$$T = \frac{2 \cdot \sum I[A_i - B_i > 0] - n}{\sqrt{n}}$$

where $I[A_i - B_i > 0] = 1$ if $A_i - B_i > 0$ and 0 otherwise.

The null hypothesis of the *Sign Test* is that there is no difference between the two search engines, under the assumption that $P(A_i > B_i) = P(B_i > A_i)$ and that the number of positive and negative samples follow a normal distribution.

This method works best when there is a large number of samples, i.e. topics, available. The power of the *sign test* is that it is non-parametric, i.e. that it does not look at the size of the differences, but only at how many positive samples there are compared to the total number of samples. For the evaluation of information retrieval this means that it gives a good sense of the average case performance, because extremely good performance on only one topic does not throw it off. The downside of using the *sign test* is that it needs many samples to give meaningful results. This is no problem for large organized evaluation projects such as TREC (TREC, 2003), CLEF (CLEF, 2003) or INEX (INEX, 2004), but when the assessments have to be done by one person, it is not feasible to have more than a few dozen topics.

In this thesis I will use some techniques that only work well for a subset of the topics. Given the small number of topics that I will be able to assess on my own, these techniques are bound to produce results that are not seen as significant by the *sign test*. When combined however, a collection of techniques that achieve insignificant improvements according to the *sign test* might give good overall performance that can be significant. Therefor I will only use the *sign test* in the conclusion of every experiment I will do in this thesis. For all intermediate results that only look at individual techniques I will use a statistical test that looks at the quantitive difference between two runs, the *paired t-test*.

PAIRED T-TEST When it is important how much better one method does than the other, for example when there are not enough topics to smooth the error distribution, the *paired t-test* can be used. The *paired t-test* looks at the variance of the differences between the results A_i and B_i.

$$t = (\overline{A} - \overline{B}) \sqrt{\frac{n(n-1)}{\sum_{i=1}^{n}((A_i - \overline{A}) - (B_i - \overline{B}))^2}}$$

where n stands for the degrees of freedom, which is equal the the number of samples that are compared, and $\overline{A}$ stands for the average of A_i for all samples i. The *paired t-test* is described in detail and compared to other methods in an article by Hull (Hull, 2000).

Like the *sign test* the *paired t-test* assumes that the errors follow a normal distribution. The problem with the *paired t-test* however is that the distribution is not over the number of positive and negative samples, but over the differences in

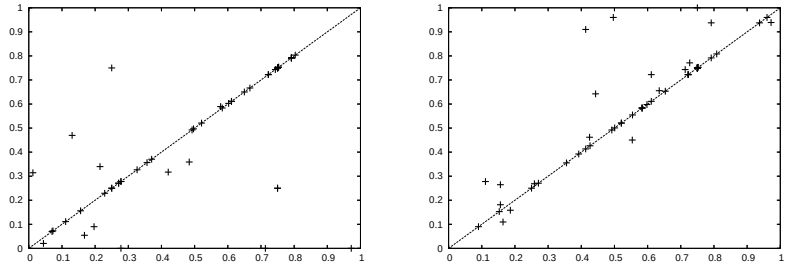


Figure II.3: Quantile plots of runs from chapter IV. Shown are a comparison between the baseline and a run exploiting emphasis annotations (left), and between a run that exploits titles and one that also exploits collocations (right).

performance per topic. These difference are usually not normally distributed in information retrieval. Furthermore, the *paired t-test* assumes that the distribution is continuous. This is clearly not the case with the retrieval measures mentioned in section 5.1. Van Rijsbergen states in his standard on information retrieval (van Rijsbergen, 1979) that this disqualifies the *paired t-test* for use in information retrieval evaluation. Hull however is less pessimistic and states in his article (Hull, 2000) that this strictly speaking van Rijsbergen might be right, but that the *paired t-test* is relatively robust to violations of its assumptions as long as the distribution is not skewed and there are few outliers. These two properties can be tested by making a quantile plot of the data. If the data tend towards normality, this should produce a straight line. Two quantile plot comparing runs from chapter IV are shown in figure II.3.

Based on these plots I make the subjective judgement that the assumptions the*paired t-test* makes of the data are safe.

In tables that show results I will use markings that show when a difference is significant (95% confidence) or material (99% confidence) (Spärck Jones, 1979). Significant differences and are marked with $\triangle$ and $\triangledown$. Material differences with $\blacktriangle$ and $\blacktriangledown$.

For the *paired t-test* the t scores that correspond to 95% and 99% confidence are respectively $t \geq 1.645$ and $t \geq 2.346$.

Chapter III

Structured Information

Searching in the concept hierarchy

Information that is stored in an orderly fashion is called 'structured information'. An example of structured information is the data stored in a database, ordered in tables, sorted by key. Another form of structured data is the ontology, an organization of things that are. The word 'ontology' derives from the Greek 'ον', which means 'thing that is' or 'being'; and 'λογος', which means 'definition', 'reason' or 'relation'. So 'ontology' means exactly what it does. It describes the relation between concepts. An example of an ontology is Linnæus' plant taxonomy.

The relations in an ontology are subsumption relations, which means that all the properties of the parent concept pass on to its child concepts. So if the concept Chordate has the property that its instances have a spine, and the concept Human is relation to Chordate by the subsumption relation subclass-of, then the fact that Carl Linnæus is a Human being also means that he has a spine.

1 Searching versus Browsing

One of the main purposes of ontologies is to function as conceptual map of a domain. Browsing through an ontology can give a user a good idea about how the concepts in the domain are related. Furthermore, ontologies can be used to quickly determine the meaning of a concept in the domain. Sometimes however browsing through an ontology can be a difficult and laborious process. The following examples illustrate cases where browsing can be difficult.

- The user does not know where to look for a certain concept. For example, he is looking for 'modal logic' and does not know whether he should search in the branch 'philosophy' or 'mathematics'.

- The user does not know the name of the concept he is looking for. This can be the case when the user is looking for the logic used to describe knowledge and belief, and he has no idea that this is called 'epistemology'.

- The ontology is so large that it simply takes forever to find what the user is looking for.

In these cases information retrieval can help by providing random access to concepts in an ontology and flexible interpretation of the user's information need.

In this chapter I will discuss the problems that come with searching in ontologies and how I tackled them while building a search engine for the *LoLaLi* concept hierarchy.

First I will describe the *LoLaLi* concept hierarchy. Then, in section 3 I will discuss the specific search task I will address. In section 4 I will first introduce the topic set I will use and then discuss the problems that come with concept retrieval and show some results that support the decisions I made. One of the things I will do to solve the problems discussed in section 4 is to extract collocations from the handbook and a web collection. In section 5 I will take a detour to describe the text extraction which was necessary to extract the collocations. Then in section 6 I will come back to the subject of concept retrieval and exploit the collocations to improve the results of concept retrieval. At the end of that section I will evaluate and discuss the results. In section 7 I will explore another way to improve the concept retrieval results, exploiting the relations in the concept hierarchy. In that same section I will evaluate and discuss the results. Finally, in section 8 I will draw conclusions from the results in the chapter.

2 The LoLaLi Concept Hierarchy

The *LoLaLi* concept hierarchy is an ontology created by Maarten de Rijke and Caterina Caracciolo (Caracciolo et al., 2002) of the concepts in the *Handbook of Logic & Language* and the relations they have with each other. Each concept has a name, consisting of a few words, e.g. 'modal logic', and most concepts have a description of a few sentences. Most concepts have instances that are hyperlinks to parts of the handbook.

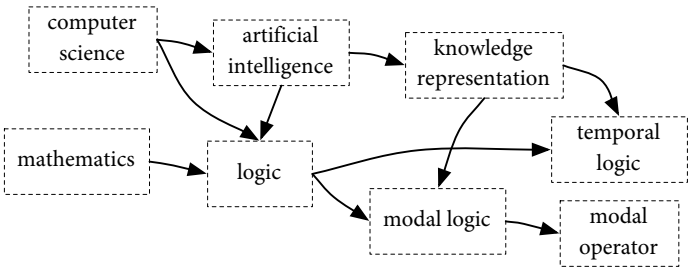


Figure III.1: An excerpt from the LoLaLi concept hierarchy

The concept hierarchy is a connected graph with three kinds of relations. Those of a subtopic-supertopic kind: is-a and part-of. Associative relations: has-notion, has-mathematical-result, etc. and thesaurus relations: antonym-of, related-to, etc.

An example of the hierarchy is shown in figure III.1, where the arrows show any kind of subtopic-supertopic relation between two concepts. Throughout this thesis arrows point in the direction of the narrower concept.

As described in the beginning of this chapter, the concept hierarchy can be used to browse through the concepts of the handbook. For example to determine the meaning of such a concept or to see whether a certain topic is included in the book. But the concept hierarchy can also be used as an index to the actual text of the handbook by following the hyperlinks at the concepts. This way the concept hierarchy can also be used as a *Yahoo!*-like index.

3 Searching in the Concept Hierarchy

The goal of the concept hierarchy search engine is to solve the browsing problems describe in the beginning of this chapter. This is done by letting the user describe the concept he is looking for with a short list of query terms, like most web search engines allow you to do. The search engine takes this query and provide the user with a few entries into the concept hierarchy that fit the user's request the best. Say the user is interested in 'epistemology', but he does not know that term, then he types in '`logic used to describe knowledge and belief`' and the search engine should return a short, ordered list of references to concepts in the concept hierarchy that have to do with epistemology, such as primarily the concept 'epistemology' itself, but possibly also 'modal logic' or even 'temporal logic'.

An illustration of the process from the user's thought to the output of the

search engine is shown in figure III.2. It shows the case where the user knows what he is looking for, but he does not know where to find it.

4 CONCEPT RETRIEVAL

This section deals with what happens inside the box in figure III.2 with the description *Retrieval of the query terms*: Finding the concepts that match the query the best; and the box *Rank the results according to content & location in the hierarchy*: Computing a list of the retrieved concepts.

To decide if a concept should end up in this list, the search engine computes the similarity between the user's question and the concept. I use information retrieval techniques to compute this similarity. The words from the user's question will be my query and the names and glosses of the concepts in the hierarchy will be my documents.

In this section I will describe the specific problems of concept retrieval:

- Dealing with the lack of data due to short queries and short documents.
- Achieving high-precision retrieval without redundancy.
- Dealing with false matching due to morphological normalization.
- Dealing with stop words in a domain-specific collection.
- How to find good discriminators in domain-specific documents.

In order to test my intuitions about how to deal with these problems I will need a topic set and a basic search engine to set a baseline. I will now introduce the baseline I will use in the rest of this section.

4.1 BASELINE

As a first baseline I use the FlexIR retrieval engine (Monz et al., 2002a) with the *tf.idf* weighting scheme as described in section 4.3. I will see the name and description of a concept as one document, but I count the terms in the name twice. This assures that a concept's name is more important than its description.

To test anything you need a topic set. This set should be large enough, heterogeneous, representative of the information needs the search engine caters to, and unbiased. In practice it should also be small enough to be assessable by hand.

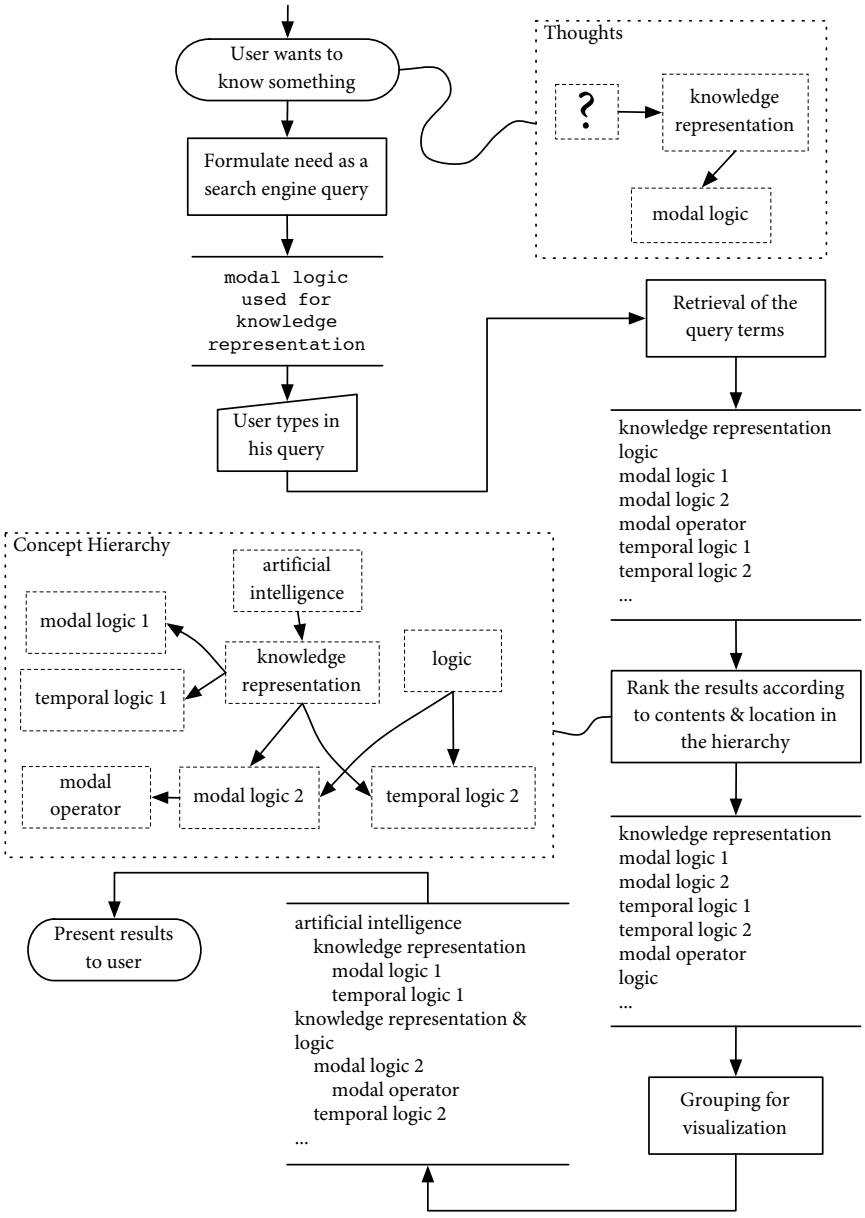


Figure III.2: Information retrieval from the concept hierarchy

The minimal size of the topic set depends on which significance level you want to get with which evaluation measure. Some measures are more stable than others. The value of stable measure does not change much when you make a small change to the list of documents it measures. *Incremental R-Precision* is quite a stable measure, because it is an average of *R* values. So if there is a small change in the ranking, the difference in the *Incremental R-Precision* is also small. This means you only need a few documents to get a reasonable error margin. 25 is seen as the minimum number of topics to produce believable results. More about measure stability can be found in papers by Voorhees and Buckley (Buckley and Voorhees, 2000; Voorhees and Buckley, 2002).

To get more topics I could simply sit down and write a whole bunch, but this is likely to produce very homogenic topics and it is certainly biased, because I am the author of the search engine and I know the contents of the concept hierarchy very well. In practice it was difficult to get others to write topics for me that were of any use, mainly because of the small size of the hierarchy. This restrains the range of possible topics a lot, and thus makes it more difficult to think up good topics. So as a middle-of-the-road solution I took the topics three people (Caterina Caracciolo, Balder ten Cate, and Yoav Signer) wrote for me, copied the good ones and adapted the bad ones a bit. Caterina Caracciolo and Stephan Stipdonk did user tests with first-year artificial intelligence students to see how well they understood the user interface. As part of this user test the students had to type in some queries. I took these queries from the webserver's log and wrote some topics in the style of the students. This brought the total number of queries to 26, written by four people, of which two (plus the students) did not know the contents of the hierarchy. Furthermore, they cover various parts of the hierarchy. The topics can be found in appendix A, section 1.

The result of the baseline is shown in table III.1

	Inc. R-Precision	*Recall*
baseline	.57	.50

Table III.1: Baseline results for concept retrieval

The results of the baseline are reasonably good. I find half of the relevant concepts and an average *Incremental R-Precision* of .57 means that for about fifty percent of the topics the highest ranked concept is relevant, or the majority of the following concepts are relevant. Examples of rankings with their *Incremental R-Precision* are shown in table III.2

ranking	*Incremental R-Precision*
●●	1.00
●●●○○○	0.80
●○	0.75
●●●●●●○○●○	0.57
○●●●○●	0.53
●○○○	0.52
●○○○○○	0.41
○●○●	0.33
○●	0.25
○○	0.00

Table III.2: *Incremental R-Precision* for example rankings. ○ indicates an irrelevant document and ● indicates a relevant document.

4.2 Short Queries and Documents

The major difference between a regular relevance search task and searching in the concept hierarchy is the lack of data. First of all, the documents are very short. The glosses are on average only 23.3 words long, The concept names are on average 1.8 words long. So the average document is about 26 words long. Regular weighting schemes are suitable for dealing with documents with thousands of words. And second, the queries are very short too. Bernard Jansen showed (Jansen, 2000)

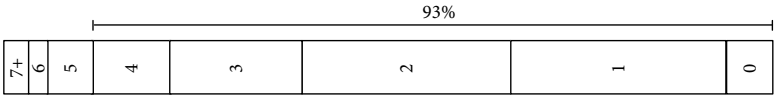


Figure III.3: Average number of terms per web search engine query

that approximately 93% of all web search engine queries contain between 0 and 4 terms, with a mean of about 2.2,[1] see figure III.3. A user study with first-year artificial intelligence students, performed by Caterina Caracciolo and Stephan Stipdonk as shown that the user interaction with the concept hierarchy search engine is very similar to a web search engine when it comes to query length. The average query lenth was even a bit below 2.

[1] N.J. Belkin et al. (Belkin, 2003) showed that providing a bigger query entry box than the one-line text entry field generally used in web search engines and actively telling users that a longer query leads to better performance can push the mean query length up to about 6 terms.

The fact that both the queries and the documents are so short rule out any techniques that work by redundancy, that would work on large collections such as the documents that *Google* or *Yahoo!* index.

4.3 High Precision

The purpose of the concept search engine is to provide the user with a few high quality concepts, like with a web search engine, where users will usually not bother to look at pages below the top-10.

Providing the user with the best top-n concepts requires a method that yields very high precision. A web search engine can achieve this by just being pedantic about the similarity, because it has access to an enormous amount of data. When you have such heaps of data using no morphological normalization works fine. In the case of the concept hierarchy it is not that simple. If you just throw away all the documents that contain 'operator', but not 'operators' when the query contains 'operators', you might throw away all of the important documents, just because there are so few documents that contain 'operator' or 'operators'. So the idea that you have to match more liberally when you have sparse data clashes with the thought that you cannot match too liberally if you want high precision. I will try to find a good middle course for the task at hand.

4.4 False Matching

Any kind of morphological normalization can introduce errors called 'false matches'. For the simple reason that a word once normalized can mean something different from its original form. When you use a 4-grams instead of words to match morphological variations of words, for example 'formal' and 'formalism', it may happen that you match a word that looks similar to the word in the query even though it does not have the same stem, like: 'formal' and 'normal'.

Say the query contains the word 'formal', then one occurrence of the term 'formal' generates 5 matches: `␣for`, `form`, `orma`, `rmal`, `mal␣`, and one occurrence of 'normal' generates 3 matches: `orma`, `rmal`, `mal␣`. This means that if document $A = \langle$'formal'$\rangle$ and $B = \langle$'normal', 'normal'$\rangle$, then the irrelevant document B matches $2 \times 3 = 6$ times, while the relevant document A only matches 5 times. This phenomenon becomes a great problem when documents are short and the collection is small.

The terms in short documents are relatively diverse. The longer documents are, the higher the average word's term frequency, tf, will be. Think of it this way: It is easy to come up with a sentence that does not use words more than once. As

a matter of fact, that very sentence did not contain duplicate words, and neither does this one. If you try to write ten sentences without using the same word more than once you will run into trouble once you run out of words like 'a', 'the' or 'is'. In figure III.3 you can see the average tf increase with the document size.

$$\text{avg. } tf = \sum_d \sum_{w \in d} \frac{\text{freq}_{d,w}}{|d|}$$

As you can see, the LoLaLi concept hierarchy has an avg. tf of almost 1. That

collection	avg. document size	avg. tf
LoLaLi concept hierarchy	25	1.043
CACM abstracts	91	1.465
Wall Street Journal 1989	414	1.782
LA Times, May 1994	536	1.835
New York Times[2]	759	1.971
IEEE T. K. 2001 articles[3]	4864	5.030

Table III.3: Average document size versus term frequency

means almost all words occur only once in every document that contains them. The exceptions being the stop words, as shown in figure III.5. If an arbitrary word in a document matches a query term, it will on average match as often as the average tf. This means that when your documents are long as there are relatively many matches per document, and since false matches generally occur less often than true matches the false matches will be overwhelmed by the true matches. When your documents are short there is no space for statistics to do its work, and every false match can make a difference. When you have a huge collection there can be so many relevant documents that they can completely fill up the list of retrieved documents, which means the documents introduced by false matches will be pushed off the bottom of the list by the relevant documents. When the collection is small however, there are usually too few relevant documents to swamp the documents introduced by false matching.

Since the documents in the concept hierarchy are so short and since there are so few of them, one false match makes a big difference. So my conclusion is that either I should not do morphological normalization at all, or it should be something cautious, not n-gramming, but plural stemming or lemmatization.

[2] New York Times news stories from the *Aquaint* 2000 corpus

[3] Articles from the IEEE *Transactions on Knowledge and Data Engineering* journals of 2001

I will compare the results of no morphological normalization to those achieved with the *Porter* stemmer (Porter, 1980) and the *TreeTagger* (Schmid, 1994) lemmatizer. Previous research has shown a few times that even though lemmatizing would seem to be the correct thing to do stemming preforms better for English (Hollink et al., 2004). Since English contains so few irregular morphological variations as opposed to for example German or Icelandic. This could well be the same in this case.

NULL HYPOTHESIS Morphological normalization does not improve concept retrieval scores.

ALTERNATIVE HYPOTHESIS Morphological normalization can improve scores.

The results are shown in table III.4.

	Inc. R-Precision	*Recall*
plain baseline	.57	.50
TreeTagger	.67 △	.60 △
Porter's stemmer	.69 △	.62 △

Table III.4: Stemming and lemmatization results for concept retrieval

There is a significant improvement from the plain baseline to both runs that use morphological normalization, both for *Incremental R-Precision* and *Recall*. This means I can accept the alternative hypothesis. My intuition proved to be true, although there is no significant difference between the stemmed and lemmatized results.

Based on this result I decide to continue experimenting using these lemmatized and stemmed runs as my new baseline.

4.5 STOP WORDS

As mentioned before most of the words in the concept hierarchy with an avg. $tf > 1$ are stop words. All words with an avg. $tf \neq 1$ are shown in the left table in figure III.5. This is not the only discriminator for stop words however. Another good indicator for a word to be a stop word is if the word occurs very frequently in the entire collection. The 10 words with the highest cf are shown in the right table in figure III.5.

Stop words occur frequently in general and are very bad discriminators for relevant documents, which means they can introduce many irrelevant documents into the ranking. So it is important to carefully select which words should and which words should not appear in the stop list. I say 'carefully', because the concept

word	avg. tf	stop word?	word	cf	stop word?
graphs	2	no	of	85	yes
or	1.437	yes	the	56	yes
noun	1.333	no	logic	51	no
a	1.245	yes	a	49	yes
computer	1.250	no	that	32	yes
and	1.230	yes	theory	31	no
of	1.197	yes	or	23	yes
the	1.143	yes	language	21	no
that	1.143	yes	to	17	yes
language	1.050	no	and	16	yes

Table III.5: The 10 words with the highest avg. tf and cf in the concept hierarchy

a, about, above, after, again, against, all, am, an, and, any, are, as, at, be, because, been, before, being, below, between, both, but, by, did, do, does, doing, down, during, each, few, for, from, further, had, has, have, having, he, her, here, hers, herself, him, himself, his, how, i, if, in, into, is, it, its, itself, me, more, most, my, myself, no, nor, not, of, off, on, once, only, or, other, our, ours, ourselves, out, over, own, same, she, so, some, such, than, that, the, their, theirs, them, themselves, then, there, these, they, this, those, through, to, too, under, until, up, very, was, we, were, what, when, where, which, while, who, whom, why, with, you, your, yours, yourself, yourselves

Figure III.4: The list of stop words I used

hierarchy is filled with text on a very specific subject, which means that some of the frequent words are not stop words. The word 'logic' for example, which is an important keyword, occurs in more documents than all stop words except 'of', 'the' and 'a(n)'. So just taking the bunch of words with the highest frequencies and calling them a stop list is no option.

I compiled a stoplist without nouns or domain-specific verbs such as 'reason' or 'compute' shown in figure III.4.

There is one other important thing to take into account: the size of the documents. A list of stop words can only do its work when the documents are large enough for the stop words to have any significant influence. Using a list of stop words is a good thing, but in the case of the concept hierarchy I expect the actual influence of such a list to be marginal. This means the benefit of a stoplist might

not outweigh the minute chance that a user will not be able to find a concept because what he is looking for is best described by a word in the stoplist.

I will now test the impact of the stoplist in figure III.4.

NULL HYPOTHESIS Using a stoplist does not improve concept retrieval scores.

ALTERNATIVE HYPOTHESIS Using a stoplist can improve concept retrieval scores.

The results are shown in table III.6.

	Inc. R-Precision	*Recall*
TreeTagger baseline	.53	.38
TreeTagger with stoplist	.54	.37
Porter's stemmer baseline	.54	.38
Porter's stemmer with stoplist	.55	.37

Table III.6: Stoplist results for concept retrieval

There is no significant difference between using a stoplist and not using a stoplist, so I have to accept the null hypothesis. This is probably the case because the topics and the documents are almost completely made up of keywords.

Based on this result I decide not to use a stoplist for the rest of the experiments.

4.6 Domain-Specific Word Variations

Since the concept hierarchy is filled with scientific content, it contains lots of fancy morphological variations of words, e.g. 'temporal', 'linear', 'recursive', etc. In the case of the example query 'logics used to describe time-related phenomena', it is necessary to match 'time-related logics' with 'temporal logic' in order to return the right concepts. These variations are not included in *TreeTagger* or Porter's stemmer. Correct lemmatization could be accomplished with a scientific lexicon, or with feedback on a much bigger corpus.

For now I decide to ignore this problem, because the people using the search engine will be either logicians or students. Both can be expected to know the scientific jargon, because if that would not be the case they would not be able to understand the text in the hierarchy in the first place.

4.7 Where idf & Normalization Fall Short

Apart from being short and full of lingo the contents of the concept hierarchy are different from the contents of a novel or an article in another way. The glosses are so concise and to the point that there are relatively few words that do not deal

with the subject of the gloss, while in prose, where you have to read in between the lines, the opposite is true. This leads to a relatively high collection frequency, *cf*, for keywords and a relatively low *cf* for usual stop words, that function as glue. As mentioned before, the word 'logic' occurs very frequently in the concept hierarchy. This makes 'logic' a bad discriminator. Meaning that it doesn't tell two concepts apart very well, which is contrary to what you would expect from an odd word like 'logic'. This becomes a problem when your query does not have any good discriminators in it, e.g. 'recursion theory', both very common words in the concept hierarchy. When all the query terms occur in the document and when 'recursive' and 'recursion' are normalized to the same term 'recursive function theory' and 'recursion theory' get the same idf score, even though they are only distantly related.

Possible solutions to the problem of having few good discriminators are the following.

COLLOCATIONS It is clear that when 'temporal' and 'logic' appear consecutively in the query that they belong together. In section 5 I will try to detect word sequences that belong together and exploit this knowledge about word order in section 6.

EXACT MATCHES A very simple, but effective rule of thumb is that when a user types in 'modal logic' he really means 'modal logic'. Whenever there exists a concept with a name that literally corresponds to the topic, it is extremely likely to be a concept the user is looking for. When such a concept does not exist you will have to use other tricks. I will combine exact match preference with collocation preference and show the results along with the results of collocations.

MEAN SPAN WEIGHTING *Minimal Span Weighting* (Monz, 2003) adds a bonus to a document's score when the query words occur close together. MSW is easy to implement and does not require any outside data, but it does not order the words, so 'logical semantics' gets the same score as 'semantical logic', when 'logic' $\sim$ 'logics' and 'semantical' $\sim$ 'semantics'. This works best with large documents, so I will not use it for the concept search engine.

5 NOUN COLLOCATION DETECTION

All concept names in the concept hierarchy with more than one word: 'Löwenhein-Skolem-Tarski theorem', 'temporal logic' etc., happen to be noun collocations. If a user types in a query that exactly matches a concept name, it is very likely that he is really looking for that concept. And if he types in something like '`semantic`

A	N		*logical study*
N	N		*computer science*
A	A	N	*floating decimal point*
A	N	N	*recursive enumerable set*
N	A	N	*card programmed calculator*
N	N	N	*program storage unit*
N	P	N	*theory of computation*

Table III.7: Part of speech tag patterns for collocation filtering

`relation`', which doesn't have its own concept in the hierarchy, but which is described in concepts such as 'antonymy', 'synonymy', and 'hyponymy', it is likely that those are the concepts he is looking for. Other concepts might contain more occurrences of the words 'semantic' or 'relation', and would be likely to end up high in the ranking of a regular tf.idf system. So in this case it is important to stress the fact that 'semantic relation' is a collocation, and not just two words occurring in the gloss.

5.1 Collocations in the Concept Hierarchy

Manning and Schütze (Manning and Schütze, 1999) describe a basic algorithm by Justeson and Katz (Justeson and Katz, 1995) for finding collocations in a piece of text. They first label all the words with a part of speech tag, and then select certain combinations of tags, shown in figure III.7. 'A' stands for antecedent, 'N' for noun, and 'P' for preposition.

The data in the concept hierarchy is so sparse that the only 11 collocations come up when it is preprocessed this way. Figure III.8 shows the collocations, how often they occur in the concept hierarchy, and how often their constituents, the words that make up the collocation, occur in the concept hierarchy. The last number in the table is the t-score of the collocation, explained later on in this section and in Manning and Schütze's book on page 163–166 (Manning and Schütze, 1999).

So as you can see the collocation 'semantic relation' was discovered, but many other collocations, which only occur once in the concept hierarchy, but which occur frequently in other literature about logic, such as 'formal description', were thrown away. The collocation 'branch of computer' is the start of the collocation 'branch of computer science', but since we don't look for collocations longer than 3 words the full collocation was not detected. Justeson and Katz's algorithm (Justeson and Katz, 1995) continues by only keeping collocations of which the constituents occur together relatively often. Which in theory should remove sequences

collocation	freq	cf of the constituents	t-score
theory of computation	2	4, 70, 2	79.672
study of language	4	12, 70, 10	41.058
branch of computer (science)	2	6, 70, 6	37.516
system of logic	2	5, 70, 18	23.677
semantic relation	3	5, 4	20.454
computer science	3	6, 4	18.645
modal logic	2	2, 18	10.040
temporal logic	2	3, 18	8.118
logical study	2	5, 12	7.676
function word	2	4, 21	6.402
content word	2	4, 21	6.402

Table III.8: Collocations extracted from the concept hierarchy data

such as 'world wide journey', but keep 'world wide web'. They do this with hypothesis testing.

5.2 Hypothesis Testing using the T-test

NULL HYPOTHESIS If you assume that the occurrences of the words w_1 and w_2 are completely independent, then $P(w_1w_2) = P(w_1)P(w_2)$. This is our null hypothesis, H_0.

As an example I will show how to compute the null hypothesis for the collocation 'modal logic' in the handbook. In practice you can approximate $P(w)$ with the collection frequency, cf_w, of w and the total number of terms, $\sum_t cf_t$, in the collection.

$$P(w) = \frac{cf_w}{\sum_t cf_t}$$

'modal' occurs 353 times in the handbook, 'logic' 2859 times, and the total number of terms is 511881 (including symbols that mark the beginning and end of files).

$$H_0 = P(\text{'modal logic'}) = P(\text{'modal'})P(\text{'logic'}) =$$

$$\frac{353}{511881} \cdot \frac{2859}{511881} \approx 3.852 \cdot 10^{-6}$$

T-SCORE To express how likely a certain combination of words is we need a statistical test. Brigitte Krenn and Stefan Evert (Krenn and Evert, 2001) have shown that for the related task of detecting PP-verb collocations the t-test works very well, so I choose the t-test. The t-test looks at the difference between the sample mean $\bar{x}$, and the mean, μ, of the distribution, divided by its variance $\frac{\sigma^2}{N}$, where N stands for the size of the data set.

$$t = \frac{\bar{x} - \mu}{\sqrt{\frac{\sigma^2}{N}}}$$

Say you are looking for a collocation $w_1, \ldots, w_n$. If the null hypothesis is true, then you can see the process of randomly generating n-grams from the words in the corpus as a Bernoulli trial if you interpret the n-grams that match the collocation as a 1 and all other n-grams as a 0. The chance p of getting a 1 will be the probability of $w_1, \ldots, w_n$ occurring at random,

$$p = P(w_1, \ldots, w_n), \;\; q = (1 - p)$$

In the case of the 'modal logic' example the mean of the Bernoulli distribution is $3.852 \cdot 10^{-6}$, because $p \approx \bar{x}$, and the variance is $\sigma^2 = p(1-p) \approx p^4$, and the sample mean

$$p \approx \bar{x} = \frac{cf_{\text{modal logic}}}{\sum_t cf_t} = \frac{99}{511881} \approx 1.934 \cdot 10^{-4}$$

So for 'modal logic' in the handbook the t-score is computed as follows:

$$t = \frac{1.934 \cdot 10^{-4} - 3.852 \cdot 10^{-6}}{\sqrt{\frac{1.934 \cdot 10^{-4}}{511881}}} \approx 69.102$$

The critical t-score for significance value $\alpha = .005$ is 2.576, as shown in figure III.9. The t-score for 'modal logic' is 69.102, which is greater than 2.576, which means we can reject H_0, which states that the constituents are independent, with a confidence much greater than 99%.

5.3 COLLOCATIONS IN THE CONTENTS OF THE HANDBOOK

The same method applied to the text in the handbook yields many more collocations. The results are shown in figure III.10. Some of them start with '`formal␣`'. One of the collocations we could not extract from the concept hierarchy, because

α	t-score	confidence
.05	1.645	95%
.025	1.960	97.5%
.01	2.326	99%
.005	2.576	99.5%
.0025	2.807	99.75%
.001	3.091	99.9%

Table III.9: Critical t-scores for various significance values

collocation	freq	cf of the constituents	t-score
formal learning theory	6	508, 18, 2334	664.807
formal language theory	11	508, 2060, 2334	113.835
formal semantic analysis	2	508, 779, 588	67.084
formal learning	6	508, 18	44.758
formal description	2	508, 264	3.395
...	...	...	...

Table III.10: Collocations extracted from the handbook

it only occurs once was 'formal description'. Using the data from the handbook it turned up.

So what about the collocations such as 'semantic network' which the data from the handbook doesn't reveal? One way to discover these collocations is to use auxiliary data, such as a related handbook, or scientific articles from the same field of science. The concept contains, after preprocessing, 511,881 words, which reveal 4238 collocations. A much bigger corpus will be needed to discover all the collocations we need.

5.4 COLLOCATIONS IN RELATED ARTICLES ON THE WEB

There is an abundance of scientific papers available on the *World Wide Web*, and some of them are easy to find using the present-day web search engines, such as *Google*[5]. The point of these papers is that other researchers understand what is in them, so they are bound to contain many common collocations.

I decided to use a very simple method to gather a data set from the web. For every concept in the concept hierarchy I construct a query consisting of the con-

[4] $p(1-p) \approx p$, because p is very small, so $1-p \approx 1$.
[5] http://www.google.com

collocation	freq	*cf* of the constituents	*t*-score
huge semantic network	3	277, 10985, 6393	468.875
semantic network retrieval	3	10985, 6393, 1332	213.807
propositional semantic network	4	4502, 10985, 6393	155.048
semantic network	155	10985, 6393	85.562
...	...	...	...
modal logic	1387	6309, 38179	413.930
mathematical logic	569	5284, 38179	183.998
philosophical logic	309	1705, 38179	177.085
predicate logic	529	12629, 38179	107.777
hybrid logic	58	727, 38179	50.272
description logic	160	11580, 38179	31.025

Table III.11: Collocations extracted from the world wide web

cept name with the switch '`filetype:pdf`' added to it. So the concept 'modal logic' is translated to the query '`filetype:pdf modal logic`'. I send all these queries to *Google*, one by one, and gather all PDF documents in the top 10 of the results.

This way I gathered 1.1GBs of PDF files, which after preprocessing yielded 358MBs of raw text. After additional preprocessing, leaving only words that could possibly be part of a sensible collocation, 128MBs of terms were left, corresponding with about 21.7 million terms, roughly 40 times the size of the handbook. Using the same extraction method I found 206,475 collocations, which is about 40 times the number of collocations I found in the handbook.

This time, as you can see in figure III.11, even 'semantic network' was extracted. Even some kinds of logic not described in the concept hierarchy were discovered, such as 'hybrid logic' and 'description logic'. This suggests the use of these collocations as suggestions for terms that could be added to the concept hierarchy. If you would like to use it for this purpose the list of discovered collocations is much too long. There are too many irrelevant terms in the list such as '19th century', 'high value', and 'final part'.

5.5 Restraining the List of Collocations

It could be possible to filter out the collocations that do not belong to the field of logic and language by removing collocations extracted from an English text on another subject. This is illustrated in figure III.5.

Another, more viable option is to throw away all the collocations that do not

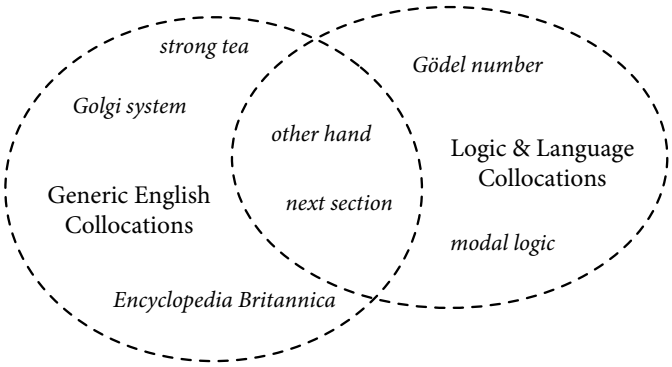


Figure III.5: Removing the influence of English

contain a constituent from a certain set. This set can be a list of words you are interested in, or even a topic. This is illustrated in figure III.6, with the topic 'the semantic relation of two words with the same meaning'. This topic filtered out all but collocations 435 from the list of 206,475. The resulting list of 435 collocations is ofcourse polluted. Collocations like 'new relation type' are hardly interesting, but this technique works a lot better than the previous one for filtering out odd collocations like 'static thing', 'norwegian salmon', and 'blue dust'.

I want to apply the collocations to improve the search results of the concept hierarchy search engine, so I can use the topics as a restraint on the list of collocations. I simply check for all pairs and triples of query words whether they form a collocation that appears in the list. This can be done in a split second, because I put all the collocations (6.5MBS) in a database, which stores them in a B-Tree.

6 EXPLOITING COLLOCATIONS

This leaves the question *How does one improve search results using collocations?* One possible way is to do 'phrase retrieval', where the collocations would be indexed as terms. (Pohlmann and Kraaij, 1997)

I use the *tf.idf* weighting scheme to determine a concept's score. The higher the score, the more relevant the concept is supposed to be. When a concept contains a collocation that appears in the topic, it is more likely to be relevant than a concept that contains the same words that do not form a collocation. So the concept with the collocation should get a higher score. I decide to multiply the score

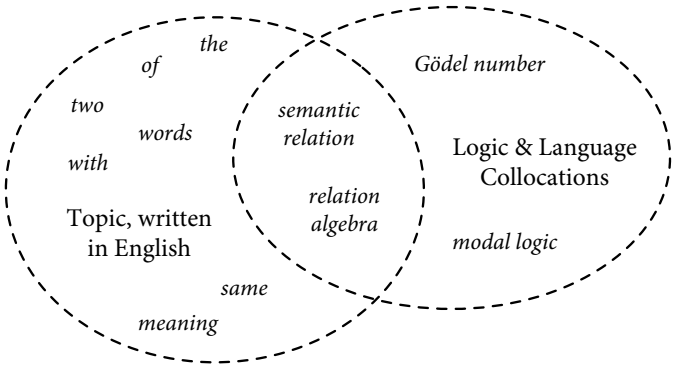


Figure III.6: Restraining the subject of the collocations

of the concept by a certain factor when it contains a collocation that appears in the topic. Multiplication makes more sense than addition, because insignificant words, i.e. words with a low *tf.idf* score, that make up a collocation should get a lower absolute bonus, but the same relative bonus as significant words.

This shifts the question to *How big should this factor be?* This depends on the number of documents in the collection, but also on how many other factors there are. I will not only give collocations a bonus, but also exact matches, and matches in the concept name versus matches in the gloss. The number of documents (i.e. the number of concept names and concept glosses in the hierarchy, which is equal to 529) lies between e^6 and e^7, so the idf score of a term lies between $\log e^6/e^6 = 0$ and $\log e^7/1 = 7$. The idea is that the factors of the collocations; the exact matches; and the word occurring in the concept name, combined are of the same magnitude as the idf score. I decided 2 would be a reasonable value for each factor, because $2^3 = 8$ and 8 is slightly larger than 7, which means that all factors combined can just outweigh any idf score. I think a higher combined bonus than 8 will not make much of a difference. A higher bonus could cause good documents that do not benefit from the bonus to be pushed out of the top-10, which could make the scores worse. To test this I will check whether giving any of the factors a higher value improves anything. The results are shown in table III.12. There is no difference in the scores when a factor of 2 or 4 is used. This supports my intuition.

	Inc. R-Precision	*Recall*
TreeTagger baseline	.67	.60
TreeTagger with collocations ×2	.69	.62
TreeTagger with collocations ×4	.69	.62

Table III.12: Collocation preference factor results for concept retrieval

6.1 EVALUATION OF COLLOCATION PREFERENCE

I tried out this way of exploiting collocations in combination with both Porter's stemmer and the *TreeTagger* lemmatizer. Previous experiments that tried to add more linguistic knowledge to retrieval have been inconclusive. I expect collocations to improve scores.

NULL HYPOTHESIS Collocations do not improve concept retrieval scores.

ALTERNATIVE HYPOTHESIS Collocations can improve concept retrieval scores.

The results are shown in table III.13.

	Inc. R-Precision	*Recall*
TreeTagger baseline	.67	.60
TreeTagger with collocations	.69	.62
Porter's stemmer baseline	.69	.62
Porter's stemmer with collocations	.69	.62

Table III.13: Collocation preference results for concept retrieval

For the stemmed run there is no difference at all in the scores when I apply collocations. For the lemmatized run there is a small difference in the score, but the difference is not large enough to be significant. So I have to accept the null hypothesis.

6.2 EVALUATION OF EXACT MATCH PREFERENCE

I tried out combinations of exact match preference and collocation preference for both types of morphological normalization. I expect exact match preference to improve precision.

NULL HYPOTHESIS Exact match preference does not improve concept retrieval scores.

ALTERNATIVE HYPOTHESIS Exact match preference can improve concept retrieval scores.

The results are shown in table III.14.

	Inc. R-Precision	*Recall*
TreeTagger baseline	.67	.60
TreeTagger with exact match	.71	.64
TreeTagger with coll. and exact.	.73 △	.65
Porter's stemmer baseline	.69	.62
Porter's stemmer with exact match	.73	.65
Porter's stemmer with coll. and exact.	.72	.65

Table III.14: Exact match preference results for concept retrieval

Exact match preference improves precision, but by itself the difference is not significant. Together with collocation preference it attributes to a significant improvement over the baseline for the lemmatized runs. For all runs except the lemmatized run with both collocation and exact match preference I have to accept the null hypothesis. For the latter run I can accept the alternative hypothesis.

6.3 CONCLUSIONS

STEMMING VERSUS LEMMATIZATION Runs that use no other techniques than morphological normalization benefit more from stemming than from lemmatization. This corresponds to previous results on the CLEF english monolingual collection (Hollink et al., 2004).

COLLOCATION BONUSSES AND LEMMATIZATION Runs that use both lemmatization and collocations perform at least as good as runs that use stemming.

EXACT MATCH BONUSSES Runs that give concepts whose name literally match the topic a bonus perform about 6% better on *Incremental R-Precision* and 8% on *Recall*. Unlike collocation bonusses, exact match bonusses do not boost the score of lemmatized runs to the same level as stemmed runs.

COLLOCATION AND EXACT MATCH PREFERENCE As for the baseline, collocation bonusses push the score of lemmatized runs up to the level of stemmed runs.

THE BEST RESULTS The greatest improvements over the baseline are made by the run that combines lemmatization with both collocation and exact match preference, shown in table III.15. This run performs 28% better on *Incremental R-Precision* and 30% better on *Precision* than the baseline. The statistical test used to

determine significance of the results shown in table III.15 is the *sign test*, as opposed to the previous results in this chapter where I used the *paired t-test*. A discussion about this decision can be found in subsection 5.2 in chapter II.

	Inc. R-Precision	*Recall*
plain baseline	.57	.50
TreeTagger baseline	.67 △	.60 △
TreeTagger with coll. and exact.	.73 ▲	.65 ▲

Table III.15: Results for concept retrieval with lemmatization, and collocation and exact match preference.

My final conclusion is that even though the improvements lemmatization, collocation preference, and exact match preference make by themselves are not staggering, sometimes even insignificant, the combination of these techniques works very well.

7 Exploiting Structure

This section deals with what happens inside the box labelled *Grouping for Visualization* in figure III.2: Grouping the concepts before presenting them to the user.

Things get a little more complicated than plain relevant document retrieval when you consider the relations concepts have with each other in the hierarchy. Concepts inherit the information from their parents and make it more specific in some way. For example, a 'modal operator' is a child of 'modal logic', and is clearly part of modal logic, but only one of the many things that make up modal logic. The opposite goes for parent concepts. They contain all the information of their children, but are less specific. Queries have to be answered as precisely as possible. The answer has to be general enough and also specific enough. When the user asks for 'modal logic', returning 'modal operator' is too specific for his needs, but returning 'logic' is too general. Often, as with 'modal logic', the concept that matches the query best is the right concept, but sometimes things are more complicated, as shown in figure III.7. The graph shows concepts that match the query `semantic relation`. Underneath the concept names you can see which terms in that concept's name or gloss match. In the case of the bottom three terms, the entire query, which happens to be the collocation 'semantic relation', matches.

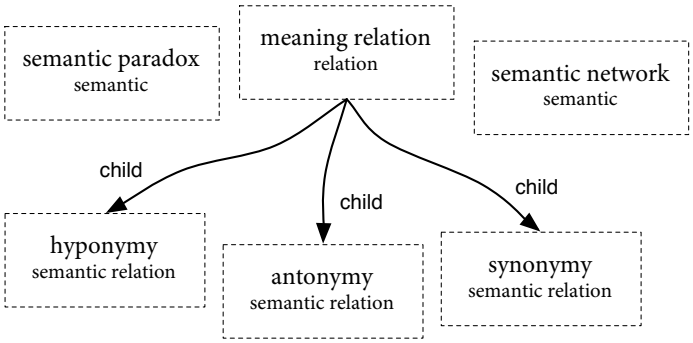


Figure III.7: Is having important relatives better than having talent?

7.1 GROUPING CONCEPTS FROM THE HIERARCHY

What is the best concept to return in this setting? 'meaning relation', or one of its children? There is no solution for this problem that works in all cases. Returning all matching concepts ordered by their retrieval score is an option, but then closely related concepts may appear far away from each other in the ranking that is presented to the user. This problem can be solved through grouping the returned concepts by the relations they have with each other.

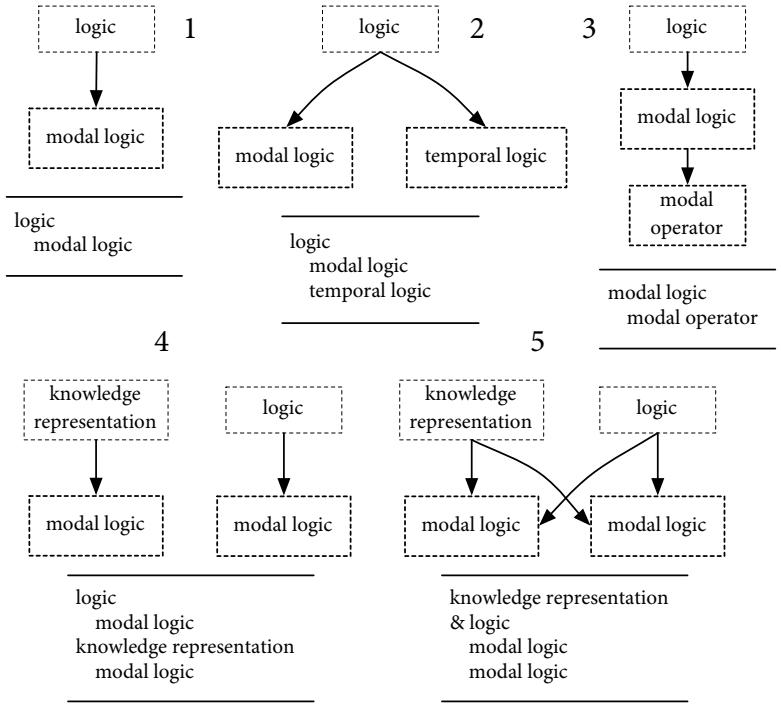
Figure III.8 shows the rules I use to cluster the concepts. Matching concepts are surrounded by a thicker box than non-matching concepts.

7.2 THE EFFECT OF GROUPING

Grouping puts closely related concepts together in the ranking and adds context concepts that might also be interesting.

In practice context nodes are always useful to show the user where the matches appear in the ontology, e.g. 'modal logic' under 'logic' and not under 'knowledge representation'. This is shown in figure III.9.

The quality of this information is very difficult to assess, because a context node does not need to be relevant to provide useful information about the location of a concept in the hierarchy. Whether a context node is interesting to the user is a subjective matter, because it depends on the user's knowledge. The only right way to assess the value of context nodes is through extensive user testing, which I will not do in this thesis. For this experiment I will show the effect of two assumptions. The first being the assumption that any parent of a relevant concept is useful and



1. Every matching concept should be clustered under its parent; this parent concept shows the context of the concept.
2. Matching concepts with the same parent should be put together under that common parent, ordered by their own score.
3. Every chain of parent-child related matching concepts should end in a non-matching concept that shows the context of the chain.
4. Unrelated clusters are joined together as a forest, ordered by the maximum score of the cluster.
5. When parents have the same children they are joined together and get the highest of the two scores.

Figure III.8: Concept grouping rules

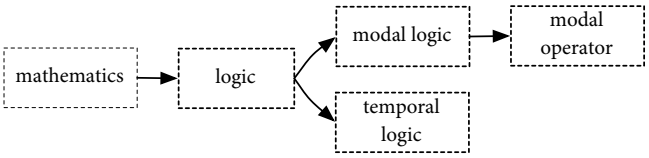


Figure III.9: Added context

the second that only relevant concepts are useful. When I show results of grouping I will show the scores under both assumptions.

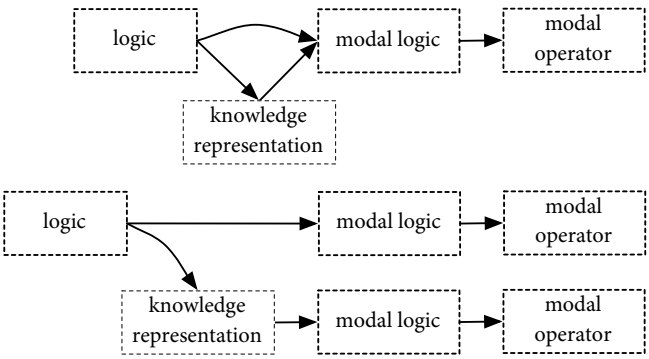


Figure III.10: Oedipus and modal logic

Ranking concepts under their parents can introduce duplicate concepts when a concept has more than one parent. Rule number 5 tries to reduce the number of duplicates by collapsing trees when they have the same children. Sometimes this goes wrong. For example in Oedipuslike situations, where a child is a sibling of its parent, shown in figure III.10. My rules essentially make a generated submodel of the hierarchy in which all worlds, concepts, either match the topic or have a child that matches. This resulting model contains no cycles, it is a tree, and that means it can contain duplicate concepts. I chose to do it this way because the search engine needs to be responsive and taking out all duplicates while preserving the structure takes EXPTIME. The only thing that I will evaluate however is whether or not the correct concepts have been found in the ranking, not in which form they are presented to the user. So for the assessments I will ignore all duplicates, because the ranking which I will evaluate is a set of concepts. This means that I will not get a higher score for retrieving a relevant concept twice.

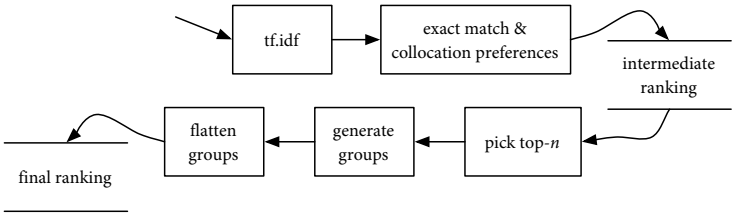


Figure III.11: How a ranking is obtained after grouping

When generating a submodel you start with a set of worlds W to generate a set of related worlds W'. You can choose these worlds to be all the concepts in the ranking, which corresponds to the top-10 in my case because I only retrieve the top-10 best concepts; or you can choose to use a smaller set, say the top-4 of the ranking. The exact procedure is shown in figure III.11. When you start with less concepts the generated trees will be smaller. Smaller trees mean that the concepts in at the top of the resulting ranking will be more diverse, since more small trees fit in the top-n than large trees. This could be a good thing. Furthermore, the average score of a smaller top-n set of concepts will be higher. That means the resulting trees will have been generated based on a set of concepts with a higher average score than when you start with more concepts. This could be beneficial to the resulting ranking.

7.3 Evaluation of Grouping

I will now test the impact grouping has on concept retrieval scores. The first thing I will examine is the effect of grouping by itself on the stemmed and the lemmatized baseline. For this first test I will use grouping on the top-10 of the ranking. I expect grouping to improve *Recall*, and in some cases, where grouping could disambiguate the query terms, to even improve *Incremental R-Precision*.

Null Hypothesis Grouping does not improve concept retrieval scores.

Alternative Hypothesis Grouping can improve concept retrieval scores.

The results are shown in table III.16.

When all context is counted as irrelevant, *Incremental R-Precision* is slightly lower and *Recall* slightly higher when grouping is applied. The effects are stronger for the lemmatized run than for the stemmed run. For this case I have to accept the null hypothesis.

	context irrelevant		context relevant	
	Inc. R-Prec.	*Recall*	*Inc. R-Prec.*	*Recall*
TreeTagger baseline	.67	.60	.53	.38
TreeTagger with grouping	.62	.64	.68 ▲	.63 ▲
Porter baseline	.69	.62	.54	.38
Porter with grouping	.65	.62	.67 △	.62 ▲

Table III.16: Grouping results for concept retrieval

When context is counted as relevant, the lemmatized run achieves a material improvement over the baseline for both *Incremental R-Precision* and *Recall*. The stemmed run is only significantly better for *Incremental R-Precision*. For this case I can accept the alternative hypothesis.

I expect that in reality, where only some context is relevant, a significant improvement can be achieved using grouping.

7.4 Evaluation of Grouping on a Smaller Set

Now I will compare grouping performed on the top-10 to grouping on the top-4 of the initial ranking. My intuition is that grouping on the top-4 might have a better *Incremental R-Precision*.

Null Hypothesis Grouping on the top-4 of the initial ranking does not perform better than grouping on the top-10.

Alternative Hypothesis Grouping on the top-4 is better than grouping on the top-10.

The results are shown in table III.17.

	context irrelevant		context relevant	
	Inc. R-Prec.	*Recall*	*Inc. R-Prec.*	*Recall*
TreeTagger, grouping on top-10	.62	.64	.68	.63
TreeTagger, grouping on top-4	.64	.48 ▼	.67	.50 ▼
Porter, grouping on top-10	.65	.62	.67	.62
Porter, grouping on top-4	.66	.48 ▼	.66	.48 ▼

Table III.17: Grouping results for concept retrieval

Grouping on a smaller set has slightly better *Incremental R-Precision*, but the difference is not significant. *Recall* however is materially worse for grouping on

the top-4 than for grouping on the top-10. So for all cases I have to accept the null hypothesis.

The price to pay for a slightly higher *Incremental R-Precision* is too high, so for the rest of the experiments I will continue with grouping on the top-10.

7.5 EVALUATION OF GROUPING WITH PREFERENCES

I will now test how collocation and exact match preference affects runs that use grouping. I expect the cumulative result of collocation and exact match preference to give a significant improvement over the run with only grouping.

NULL HYPOTHESIS Collocation and exact match preference does not improve concept retrieval scores for runs that use grouping.

ALTERNATIVE HYPOTHESIS Collocation and exact match preference can improve concept retrieval scores for runs that use grouping.

The results are shown in table III.18.

	context irrelevant		context relevant	
	Inc. R-P.	*Recall*	*Inc. R-P.*	*Recall*
TreeTagger with grouping	.62	.64	.68	.63
TreeTagger with grouping and pref's	.69 △	.66 △	.74 △	.67 △
Porter with grouping	.65	.62	.67	.62
Porter with grouping and pref's	.67	.65	.70	.65

Table III.18: Collocation and exact match preference results for concept retrieval runs that use grouping

On all aspects, the lemmatized run benefits significantly from collocation and exact match preference, so for the lemmatized run I can accept the alternative hypothesis. The stemmed run however shows no significant improvement at all, so in this case I have to accept the null hypothesis.

My intuition was only correct for the lemmatized run. The stemmed run stayed behind.

7.6 CONCLUSIONS

STEMMING VERSUS LEMMATIZATION The previous experiment revealed that collocation and exact match bonusses work better for lemmatized runs than for stemmed runs. The results of this experiment show the same behaviour. After applying grouping as a reranking method all the best runs are lemmatized runs. My guess is

that lemmatization prepares topics better for combining with other evidence than stemming.

SIZE OF THE GROUPING SET Runs that generate the groups based on the top-4 of the intermediate ranking score slightly higher on *Incremental R-Precision* and much lower on *Recall* than those based on the top-10. This can be explained by the fact that all relevant concepts that are not in the top-4 and that are also not related to a concept in the top-4 are lost.

COLLOCATION & EXACT MATCH BONUSSES AND GROUPING Grouping changes nothing about the relation between the scores of runs with only exact match bonusses and those with both bonusses. Since lemmatized runs work better with grouping than stemmed runs it was to be expected, given the results of the previous experiment, that the combination of both bonusses works best after grouping.

THE BEST RESULTS The greatest improvements over the baseline are made by the lemmatized run that uses collocation and exact match preference combined with grouping on the top-10, shown in table III.19

	context irrelevant		context relevant	
	Inc. R-P.	*Recall*	*Inc. R-P.*	*Recall*
plain baseline	.57	.50	.45	.30
TreeTagger with grouping and pref's	.69 ▲	.66 ▲	.74 ▲	.67 ▲

Table III.19: Final results for concept retrieval

8 DISCUSSION

I wanted to get high precision for a domain-specific search task with very short documents and even shorter queries while retaining a decent recall. The combination of lemmatization, collocations, exact match bonusses, and grouping worked very well at achieving that goal. Furthermore I came across the nice corollary that the collocations mined from the internet could be used to help ontology builders.

Chapter IV

Semi-structured Information

Connecting the handbook to the concept hierarchy

Between structured information such as databases and free text found in most books lies something called semi-structured information. Semi-structured information is free text with some added information about the structure or the meaning of the text, e.g. annotations that indicate chapters and sections or emphasized text. Examples of semi-structured information are XML and LaTeX. The majority of the *World Wide Web* is semi-structured information (Abiteboul et al., 1999), and so is the *Handbook of Logic & Language*, which is written in LaTeX. An excerpt from the handbook is shown in figure IV.1.

LaTeX documents have a tree-like structure, like XML, with one or more layers of annotation surrounding leaves that contain the actual (free) text. This is illustrated

```
\paragraph{Categorical combinators and CCG.}
To round off the discussion of
\keyab{Lambek calculus}{categorical combinators} the axiomatic
presentation, we present the logics {\bf NL, L, NLP, LP} with
a proof term annotation, following Lambek~(\citep{LAMBEK88}).
The proof terms -- categorical combinators -- are motivated
by Lambek's original category-theoretic interpretation of the
type logics. \key{category theory}
\keyab{Lambek calculus}{category-theoretic interpretation}
The category-theoretic connection is not further explored here,
but the combinator proof terms will be used in later sections as
compact notation for complete deductions.
```

Figure IV.1: An excerpt from the *Handbook of Logic & Language*.

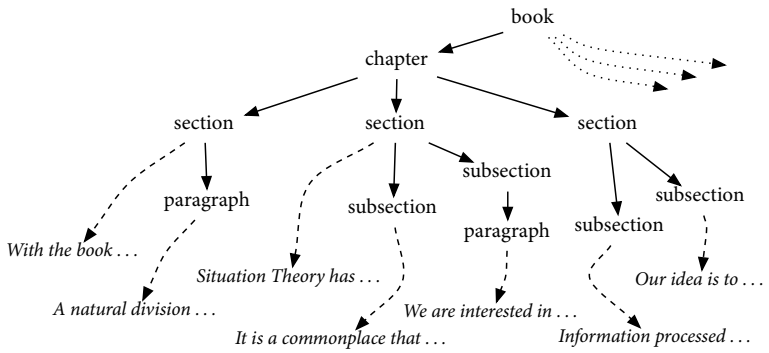


Figure IV.2: Nested annotations in LaTeX.

in figure IV.2. Commands like `\section` and `\paragraph` group text about the same subject together on different levels. The text in the example, figure IV.1, is a paragraph about Lambek calculus. It can be seen as an instance of the concept 'Lambek calculus' in the concept hierarchy.

1 Automatic Hyperlinking

Building indices for books is a difficult, laborious, and therefor costly process for publishers. Often, the process of building an index holds up the process of publishing a book in mid-production, especially when there is more than one author involved. In this chapter I will discuss the automation of index creation to assist indexers and speed up the process.

If a human wants to find the places in the handbook that deal with each concept in the hierarchy he will either have to remember all the concepts at the same time or he will have to read through the book more than once. For a small book this might be feasible, but imagine having to do this for an entire library, or for all the concepts in the subject hierarchy of *Yahoo!* and all the web pages indexed by *Yahoo!*.

Therefor I will try to do this automatically in this chapter by searching for instances of concepts in the handbook. I will do this with a similar search engine as in the previous chapter except now the concepts are the queries instead of the documents and my documents are pieces of the handbook.

In this chapter I will evaluate three techniques that could improve the retrieval performance for the automatic hyperlinking task. I will do that by setting a base-

line and measuring the improvements made by (combinations of) these three techniques. Before I describe the techniques and their results I will briefly discuss two main research problems connected to the task of automatic hyperlinking to semi-structured information.

1.1 ENTRY POINTS

The first problem is how to decide which locations you want to link to. In the case of the handbook there is already an index of where certain concepts are mentioned in the handbook, this is the index in the back of the book. This index only lists the numbers of the pages where a concept is mentioned, but it sais nothing about the focus of the concept. The entire chapter might have to do with the concept, or it might only be mentioned as a reference.

1.2 OVERLAPPING UNITS

This brings us to the second problem: Finding the focus of a certain topic in a semi-structured document. i.e. the annotation with the right size to include enough about the topic, but small enough to keep from including too much else. (Kazai et al., 2002)

If a paragraph is about Lambek calculus then the section that contains it is also about Lambek calculus. Indeed, every containing unit on the path from the text about Lambek calculus to the root of the tree, i.e. the entire handbook. So which unit should be linked to the concept? If you choose the unit too close to the root then the subject might be closer to another concept, e.g. 'Categorial Type Logics', but if you choose the unit too close to the text then you might just end up with an introductory paragraph on Lambek calculus and miss out on important information that follows it.

This problem is also encountered in the INEX XML retrieval evaluation. (Kamps et al., 2003) In XML retrieval evaluation it frustrates the evaluation measures, because the payoff for just returning all the units from the relevant text to the root of the tree is higher than the penalties for doing so. This is explained in detail in an article by Kazai, Lalmas, and de Vries. (Kazai et al., 2004)

1.3 BASELINE

I decided, based on the results of chapter III to use the *TreeTagger* lemmatizer for all my runs. For the baseline I will use FlexIR with the weighting schemes *tf.idf* and *Okapi BM25*.

```
\section{Linguistic inference: The Lambek systems \label{simple}}
\key{linguistic inference}
\key{Lambek calculus}
In the following sections we present ...

\subsection{Gentzen calculus, cut elimination and decidability}
\key{Gentzen calculus}
\keyab{Gentzen calculus}{cut elimination and decidability}
The axiomatic presentation ...
```

Figure IV.3: \key commands refer to the smallest enclosing unit.

As my topic set I take all the concepts in the hierarchy that appear literally in the index of the handbook. This automatically gives me 143 topics, with on average 1.5 words. A list of these topics can be found in appendix A, section 2.

For the assessment of the topics I will use the annotations that were used to automatically create the index. In the LaTeX code these are macro's such as \key and \keyab. Each topic has on average 3 of these references in the book. The index commands usually refer to the smallest enclosing unit, which can be of any size, e.g. a chapter, subsection, etc. I choose to see only that unit as relevant to the index key. For example in figure IV.3 only the subsection is relevant to the concept 'Genzen calculus'. I use these units as my gold standard of what to return for each topic.

Evaluating overlapping units properly is difficult. For the task of automatic hyperlinking I would like to have no overlapping units in my rankings, so I return at most one unit per path from a text leaf to the root of the tree. I choose to return the highest ranking unit. I do this with a simple algorithm that walks through a ranking from top to bottom and remember what it has seen. If it encounters units that are descendants or ancestors of a unit it has already seen they are removed. This is illustrated in figure IV.4. Since rankings are sorted by score this eliminates only lower scoring units than those already seen.

Since the number of relevant documents differs per topic, depending on how many index entries there are for the concept, I have to use an evaluation measure that takes this into account. Like in chapter III I choose to use *Incremental R-Precision* for my assessments. To give an impression of how many documents are outside the top-R I also measure *Recall*. To show how many times I chose to return the wrong unit (i.e. the section when I should have returned the subsection) I will also show *Incremental R-Precision* and *Recall* when I allow returning more than one unit per path. The results of the baseline are shown in table IV.1.

As you can see in table IV.1, the differences between *tf.idf* and *Okapi* are not

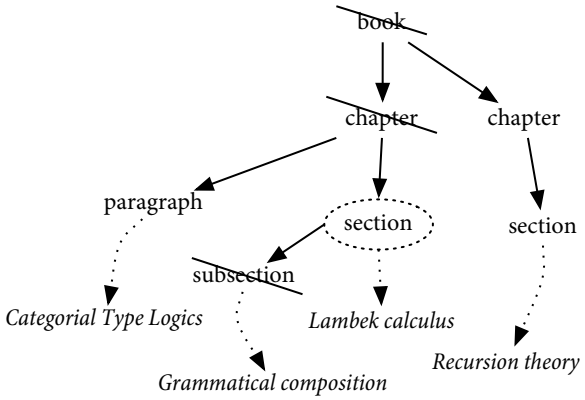


Figure IV.4: Eliminating overlapping units from the ranking.

	one per path		more per path	
	Inc. R-Precision	*Recall*	*Inc. R-Precision*	*Recall*
tf.idf	.35	.63	.34	.89
Okapi	.33	.64	.32	.86

Table IV.1: Automatic hyperlinking baseline results.

significant. In the ranking we are interested in, the 'one per path' ranking, *tf.idf* seems to do slightly better at precision, while *Okapi* does slightly better at *Recall* in the top of the ranking. Since I want high precision I continue my experiments with *tf.idf*.

1.4 Doing Better than the Baseline

In the rest of this chapter I will try three things to do better than the baseline.

1. I will try to use the fact that I am searching in semi-structured data and not free text, by exploiting LaTeX annotations. This is discussed in section 2.

2. As in chapter III, I will try if it is possible to get benefits from collocations. This is shown in section 3.

3. I will try to use concept hierarchy based query expansion to fight ambiguity. This is discussed in section 4.

```
The domain $W$ is the set of {\em linguistic resources}.
```

Figure IV.5: Emphasis on a key phrase.

2 EXPLOITING ANNOTATIONS

The first way I will try to outperform the baseline is by using the characteristics of the documents. The documents are semi-structured information. That means that there is more information in them than just the words and frequencies. Some words are more equal than others. For example words in titles of section and the like could be more representative of the subject of the section than words in the rest of the section.

Experiments with HTML tags (Cutler et al., 1997) have shown that one's intuitions cannot always be trusted on which annotations are useful. Titles in HTML documents are often something like 'Welcome!' or 'Page 2'. I suspect this might not be the case for the *Handbook of Logic & Language*, where the sections are only used to separate subjects from each other.

The HTML study also shows that emphasis is also not to be trusted. I expect that the opposite might be true for the handbook, where emphasis is frequently used to stress key phrases like in figure IV.5.

I will now test if these intuitions are true by formulating a null hypothesis and an alternative hypothesis about text in titles and emphasis and testing the validity of these hypotheses by comparing search engines that exploit titles and emphasis to the baseline.

NULL HYPOTHESIS Titles and emphasized text are just like other words in the text and do not represent the subject better than other words in the documents. Preferring documents that contain query terms in the title or in emphasis will not improve retrieval results.

ALTERNATIVE HYPOTHESIS It is possible to get better performance by exploiting title or emphasis annotations.

The way I will implement the preference for terms with a certain annotation is by increasing the score of the document: If a document starts with a title, like sections do, and that title contains the literal query I double the document's score. If it contains nothing else, i.e. if it is exactly the same as the query, I double it again. If a document contains `\em` or `\emph` commands that literally contain the query I double the score. If the emphasis is on nothing but the query I double it again.

The results of the experiments with title and emphasis preference are shown in table IV.2.

	Inc. R-Precision	*Recall*
baseline	.35	.63
title preference	.41 ▲	.67 ▲
emphasis preference	.33	.63
title & emphasis preference	.39 △	.66 △

Table IV.2: Runs that exploit title and emphasis annotations.

There is no significant difference between runs that prefer emphasized text and those that do not, but there is a material difference for those that prefer titles. So for emphasis I accept the null hypothesis and for titles I accept the alternative hypothesis.

3 EXPLOITING WORD ORDER WITH COLLOCATIONS

In the previous chapter I recognized collocations in the descriptions of the concepts and tried to use these to improve retrieval results. I mentioned that all concept names are either nouns or noun-collocations. My queries are now concept names, so every query with more than one term is a collocation. That means that for this search task finding collocations in documents is the same as finding literal appearances of the query in the documents.

The way I will implement this is the same as in the previous chapter. If a document contains all the terms of the query and they appear literally as in the topic, i.e. if the document contains the literal query, I double the score.

For searching in the hierarchy the documents were so short that I did not have to worry about what to do when they contained more than one collocation and I just doubled a document's score for every collocation I found. With the handbook that is different. Documents that contain a collocation are very likely to contain it more than once, so I choose to only double the score once per document.

As with title and emphasis preference I will formulate a null hypothesis and an alternative hypothesis and test them by comparing search engine runs.

NULL HYPOTHESIS Considering collocations does not improve retrieval results.

ALTERNATIVE HYPOTHESIS It is possible to get better performance by exploiting collocations.

The results of the experiments with collocation preference are shown in table IV.3. As you can see there is no significant difference between the baseline and

	Inc. R-Precision	*Recall*
baseline	.35	.63
collocation preference	.35	.63
title preference baseline	.41 ▲	.68 ▲
title and collocation preference	.46 ▲	.68 ▲
emphasis preference baseline	.33	.63
emphasis and collocation preference	.33	.63

Table IV.3: Runs that exploit collocations.

runs that only exploit collocations. So for collocation preference by itself I have to accept the null hypothesis.

However when I combine title preference with collocation preference there is a material difference with the baseline and even with the title preference only run. So for runs that use title and collocation preference I accept the alternative hypothesis. For combined emphasis and collocation preference I have to accept the null hypothesis.

These results can be explained as follows. For preference to work there has to be some information about the subject missing in the baseline that is corrected by the bonus. Apparently the literal occurrence of the query in a document's body or emphasized text does not say more about a document's probability of being relevant to the query than its *tf.idf* score. While the literal occurrence of the query in both a document's title and body is disproportionately strong evidence of relevance to the topic.

4 Exploiting Concept Relations

Since my queries are concepts that come from a hierarchy I know more about them then just their name. I also know their relations to other concepts in the hierarchy. Many concept names such as 'reference' and 'category' are highly ambiguous. The relations they have with other concepts should tell more about their meaning. For example 'reference' is a child of 'semantics' and 'category' is a child of 'formal language theory'. In this section I will try to use query expansion for disambiguation.

There are two opposing forces at work when you use query expansion. On the one hand precision should improve when you add more information to the query, because it disambiguates the query, but on the other hand the added terms can cause 'topic drift', which means that it dilutes the meaning of the query, which

causes bad precision. This makes query expansion a tricky practice.

4.1 Expanding with Parents and Children

I will try a few simple query expansion strategies based on the hierarchy.

1. add the parents' names to the query
2. add the parents' names to the query, but let them weigh half as strong as the concept's own name.
3. add the children's names to the query
4. add the children's names to the query, but let them weigh half as strong as the concept's own name.
5. add the parents' and children's names to the query
6. add the parents' and children's names to the query, but let them weigh half as strong as the concept's own name.

As with the previous experiments my null hypothesis will be that there is no significant benefit to be gotten from these techniques and my alternative hypothesis is that there is.

The results are shown in table iv.4. Obviously there are more topics that are hurt by this query expansion than that benefit from it. So my null hypothesis holds.

	Inc. R-Precision	*Recall*
title & coll. baseline	.46	.68
parents	.36 ▼	.52 ▼
parents half weight	.38 ▼	.59 ▼
children	.40 ▼	.60 ▼
children half weight	.40 ▼	.59 ▼
parents and children	.35 ▼	.49 ▼
parents and children half weight	.36 ▼	.53 ▼

Table iv.4: Runs that exploit collocations.

It may seem impossible that adding terms can hurt *Recall*, but that can be explained by the fact that longer queries favor larger units, e.q. chapters instead of subsections. When a large unit is picked high up in the ranking many possibly

relevant units are removed from the ranking to ensure that only one unit per path is retrieved. This is illustrated in figure IV.6.

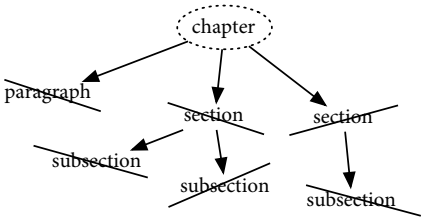


Figure IV.6: Big units eliminate many descendants from the ranking.

4.2 The Response of Certain Topics to Query Expansion

I will now show a ranking of one of the experiments and show which topics benefit from query expansion and which are hurt by it. This might help to come up with a solution to improve the results.

'quantification' △, added: 'formal semantics', 'first order logic'.

In logic 'quantification' is very ambiguous. It is clear that 'formal semantics' and 'first order logic' disambiguate it.

'context' △, added: 'knowledge representation'.

idem.

'update' △, added: 'belief revision'.

idem, 'update' is even outside of logic a very ambiguous term.

'donkey' ▼, added: 'animal'.

In semantics it is common practice that donkeys are just things that are hit by farmers. 'animal' causes topic drift in this case.

'modal logic' ▼, added: 'knowledge representation', 'symbolic logic', 'logic'.

'logic' is in the top-10 of the most common words in the handbook. That means adding 'logic' causes serious topic drift.

'ellipsis' ▼, added 'grammatical constituent'.

> The concept 'ellipsis' in itself is quite unique in the book. If you add anything to a topic that contains a good discriminator you are bound to cause topic drift.

My conclusion from these examples is that the results might improve if I apply query expansion only in certain cases. Like when the average *idf* value of the query words before expansion is low. This way queries that already contain a good discriminant will remain pure, while ambiguous queries might benefit from the disambiguating effect of the query expansion.

4.3 CORRELATIONS WITH THE REACTION TO QUERY EXPANSION

To see if this is possible at all I calculated the average *idf* of all queries that got a significant benefit from query expansion and those that were significantly hurt by it and the *idf* of those that were not influenced significantly. The results are shown in table IV.5.

Unfortunately the average *idf* of queries that react well and those that react poorly is nearly the same. This and the fact that there is about an equal number of them makes that *idf* can not pick out the right topics.

I tried the same for the average *idf* of the added query expansion terms. To see if I can detect when added terms introduce topic drift. This showed relatively low values for unaffected topics and relatively high values for topics that benefit and those that are hurt. Again the difference between both sides is not significant.

The average query length of all types of queries is the same.

benefit	average *idf*	average *idf* of expanding terms	average query length
▲	1.942	1.544	1.417
△	1.484	0.921	1.500
–	n/a	1.330	1.523
▽	2.009	n/a	n/a
▼	2.137	1.484	1.512

Table IV.5: Query properties versus their reaction to query expansion.

I also looked at the way the *tf.idf* values change from the top to the bottom of the ranking. I think the crude, untuned, way of simply giving documents that are 'good' a bonus destroys the evidence needed for proper analysis of the ranking.

Predicting when to use query expansion and when not to is very hard. Amati has shown that it is possible however with a solid probabilistic foundation (Amati et al., 2004). A good probabilistic retrieval model such as *Okapi* is more suitable for such analysis than *tf.idf*.

5 Discussion

I wanted to achieve a high precision for the task of automatically generating hyperlinks to places in the handbook. I successfully applied collocation preference in conjunction with exploiting title annotations. I attempted to use the structure of the concept hierarchy for query expansion, but this yielded only negative results. Voorhees' work on query expansion using *WordNet* (Voorhees, 1993) has shown the same results. Based on the work of Amati (Amati et al., 2004) and (Xu and Croft, 1996) however I expect that good results can be achieved with a more careful approach to query expansion.

Chapter V

Conclusion

1 Concept Retrieval

In chapter III I tackled the task of providing good random access to an ontology to solve some problems users can face when they browse through the ontology. To recapitulate, the problems I addressed are listed below.

- The user does not know where to look for a certain concept. For example, he is looking for 'modal logic' and does not know whether he should search in the branch 'philosophy' or 'mathematics'.
- The user does not know the name of the concept he is looking for. This can be the case when the user is looking for the logic used to describe knowledge and belief, and he has no idea that this is called 'epistemology'.
- The ontology is so large that it simply takes forever to find what the user is looking for.

I chose to solve these problems by creating a search engine for concept retrieval. My solution is very similar to the HAIRCUT hybrid information retriever (Shah et al., 2002b,a; Mayfield and Finin, 2003). The main difference between my search engine and HAIRCUT is that HAIRCUT uses reasoning to filter out irrelevant concepts, while I use reasoning to group concepts together to improve the quality of the resulting ranking without discarding results.

To try out which techniques to add to my search engine I set up a basic search engine and tried to improve its performance using three techniques. Two word

order based techniques: collocation preference and exact match preference; and one ontology based technique: grouping concepts by their relations. To see if these techniques improved search results I tested them on the *LoLaLi* concept hierarchy, described in section 2 of chapter III. In the following two subsections I will recapitulate the results I achieved.

1.1 Concept Retrieval and Word Order

In section 6 of chapter III I examined the influence collocation and exact match preference has on concept retrieval scores. The results are summarized in table V.1. The significance test for these results has been done using the *sign test*.

	Inc. R-Precision	*Recall*
plain baseline	.57	.50
plain with coll. and exact.	.62	.55
TreeTagger baseline	.67	.60
TreeTagger with coll. and exact.	.73 ▲	.65 ▲
Porter's stemmer baseline	.69	.62
Porter's stemmer with coll. and exact.	.72	.65 ▲

Table V.1: Collocation and exact match preference results for concept retrieval

My conclusion is that even though the improvements lemmatization, collocation preference, and exact match preference make by themselves are not significant, sometimes even insignificant, the combination of these techniques works very well. Even though stemming seemed to give the best results at first I achieved the best results using the *TreeTagger* lemmatizer. It is not clear to me why this is the case, but I got the same result in chapter IV for semi-structured data retrieval. I think it would be interesting to further investigate this phenomenon.

1.2 Concept Retrieval and Concept Relations

In section 4 of chapter III I looked at the impact of grouping the retrieved concepts by their relations on concept retrieval scores. The results were very good. Table V.2 shows an overview of the resulting scores. The statistical test used in this table is the *sign test*.

Grouping proved to significantly improve retrieval performance, even under the assumption that concepts that show the context of relevant concepts are irrelevant. This assumption is too pessimistic. In practice context is a valuable source of knowledge for users. In this thesis I did not go into detail about user interaction,

	context irrelevant		context relevant	
	Inc. R-P.	*Recall*	*Inc. R-P.*	*Recall*
plain baseline	.57	.50	.45	.30
TreeTagger with grouping and pref's	.69	.66 ▲	.74 ▲	.67 ▲

Table v.2: Final results for concept retrieval

but there is much research to be done on which context is valuable to users and which context is not. Also, I did not look at the various types of relations that exist in the concept hierarchy. Exploiting various relations in different ways might yield even better results, since much of the semantics of the concept hierarchy lies in the types of these relations.

1.3 Concept Retrieval Conclusion

I wanted to achieve high precision for a domain-specific search task with very short documents and queries while retaining a decent recall. The combination of lemmatization, collocations, exact match bonusses, and grouping worked very well at achieving that goal. I did not only achieve high precision, but also high recall. I think this is due to the fact that the techniques I use are of a 'rewarding' as opposed to a 'punishing' nature. I gave bonuses to good documents instead of removing bad documents, which preserved recall while boosting precision.

Apart from achieving my goals I also came across a few promising key phrase extraction techniques in section 5 that use collocations and relative frequencies. In this thesis I only described detecting key phrases. I did not look into single noun phrases, i.e. keywords, because what I wanted to achieve using the detected phrases is to add a sense of word order to retrieval where I thought it was necessary. Single noun keyword detection is an interesting task however and the resulting keywords might prove to be as useful as the collocations I detected in section 5 of chapter III. One possible application of keyword and key phrase extraction outside the scope of this thesis is to assist ontology builders with enumerating the concepts needed to build an ontology.

2 Automatic Hyperlinking

In chapter IV I tackled the problem of automatically populating the concept hierarchy with instances. This is similar to automatic index construction for semi-structured information.

I tried to solve this problem using information retrieval techniques, some of which I introduced in chapter III. I tried ontology based query expansion, exploiting word order with collocations and like other search engines for semi-structured information (Cutler et al., 1997; Kamps et al., 2003) I tried to treat the text with various annotations differently.

In subsection 2.1 I will discuss my findings concerning the exploitation of word order and annotations and in subsection 2.2 I will discuss the results of ontology based query expansion.

2.1 Automatic Hyperlinking, Annotations and Word Order

The results I achieved by exploiting collocations and some of the annotations of the LaTeX code are shown in table V.3. The statistical significance shown in this table was tested using the *sign test*.

	Inc. R-Precision	*Recall*
baseline	.35	.63
collocation preference	.35	.63
title preference baseline	.41	.68
title and collocation preference	.46	.68 ▲
emphasis preference baseline	.33	.63
emphasis and collocation preference	.33	.63

Table V.3: Runs that exploit collocations.

There is no significant difference between the baseline and runs that only exploit collocations or the runs that only exploit annotations. When I combine title and collocation preference however, I achieved a significant improvement in *Recall*.

2.2 Automatic Hyperlinking and Concept Relations

I was unable to get any improvements using ontology based query expansion. The results in table IV.4 in section 4 of chapter IV show many material decreases of the scores using the *paired t-test*. Using the *sign test*, this picture, shown in table V.4, looks much more optimistic. This is due to the fact that query expansion seriously hurt only a few topics, while it did absolutely nothing to the majority of the topics.

My conclusion is that I treated the task of query expansion too roughly. Query expansion needs to be tuned very carefully to yield good overall results. I think my

	Inc. R-Precision	*Recall*
title & coll. baseline	.46	.68
parents	.36	.52
parents half weight	.38	.59
children	.40	.60
children half weight	.40	.59
parents and children	.35	.49
parents and children half weight	.36	.53

Table v.4: Runs that exploit collocations.

negative results are not representative of ontology based query expansion. I expect that much better results can be achieved with a more careful approach.

2.3 AUTOMATIC HYPERLINKING CONCLUSION

I wanted to achieve a high precision for the task of automatically generating hyperlinks to places in the handbook. The best *Incremental R-Precision* I was able to get was .46, which was a great improvement over the baseline, but which, when interpreted absolutely, is kind of disappointing. The high *Recall* value of .68 shows that I choose to put the wrong concepts at the top of the ranking.

I expect the mediocre results are partly due to the fact that I used automatic assessments in the form of `\key` commands in the LaTeX code. I assumed that the only correct annotation to retrieve, i.e. the best 'focus', was the annotation that directly surrounds the command. This might have been a wild assumption. What my scores represent now is not how relevant the retrieved sections are, but how closely they match the ones that were chosen to be put in the index by the creator of the index in the back of the handbook. If the quality of the index is low, then a very good retrieval system can never get high scores. For trustworthy results human assessment should have been done.

Appendix A

Topics

1 Concept Hierarchy Topics

```
any kind of modal logic
modal logic used for knowledge representation
Kripke model
deodontic logics
logic used to describe time
what is a function word?
a term, as used in natural language processing
what is a sentence exactly?
semantic relation
ambiguity
categorial type logics
montague grammar
sentence ambiguity
part of speech
formal logical semantics
logic
logical operator
truth function
examples of paradoxes
recursion theory
turing machines
automatas
work of Cantor
aleph 0
```

2 AUTOMATIC HYPERLINKING TOPICS

```
saturation
cylindric algebra
residuation
property theory
quantification
definability
donkey
category
proof nets
algebra
modal logic
subsumption
subformula property
frame problem
proof theory
presupposition
compositionality
correspondence theory
interpolation
operator
recursively enumerable language
natural deduction
linear logic
indexicality
situation semantics
combinatory logic
epistemic logic
stability
artificial intelligence
structural rules
update
formal language theory
category theory
functional composition
ellipsis
model
ambiguity
underspecification
syntax
model theory
```

```
s4
intension
context free language
conditional logic
universal algebra
type
phrase structure grammar
prolog
completeness
unicorn
polymorphism
dynamic logic
determinacy
partiality
context
kripke semantics
extension
situation
consistency
quantifying in
relevance logic
hypothetical reasoning
deontic logic
unification
liar paradox
relative clause
structural ambiguity
discourse
knowledge representation
type shifting
ordering
anchor
lattice
domain theory
intensionality
scope
set theory
definite description
antecedent
intuitionistic logic
pragmatics
feature logic
```

```
83  counterfactual
84  implicature
85  turing machine
86  proposition
87  compactness
88  truth conditional semantics
89  frame
90  abstraction
91  partial logic
92  domain
93  discourse representation theory
94  ambda calculus
95  semantics
96  movement
97  bisimulation
98  filter
99  many-valued logic
100 denotational semantics
101 database
102 dynamic semantics
103 sense
104 undecidability
105 ptq
106 context dependence
107 categorial grammar
108 language acquisition
109 bound variable
110 belief revision
111 hierarchy
112 assignment
113 lexical semantics
114 standard translation
115 demonstrative
116 satisfaction
117 decidability
118 homomorphism
119 relation
120 montague grammar
121 belief
122 classification
123 normalization
124 extensionality
```

```
coordination
boolean algebra
higher-order logic
feature structure
reference
situation calculus
independence
type theory
context change
abstract model theory
expressive power
aspect
application
attitude
synonymy
individual concept
nonmonotonic reasoning
negation
truth
```

Bibliography

S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 1999.

Altavista. Altavista, 1995.
URL: `http://www.altavista.com`.

G. Amati, C. Carpineto, and G. Romano. Query difficulty, robustness and selective application of query expansion. In *Proceedings of the European Conference on Information Retrieval*, 2004.

G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. MIT Press, 2004.

X Baeza-Yates and N. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

A. Barabasi, R. Albert, and H. Jeong. Mean-field theory for scale-free random networks. In *Physica A 272*, 1999.

N.J. et al. Belkin. Query length in interactive information retrieval. In *Proceedings of SIGIR*, 2003.

T. Berners-Lee. the World Wide Web Consortium, 1994.
URL: `http://www.w3.org/`.

C. Buckley and E.M. Voorhees. Evaluating evaluation measure stability, 2000.

C Caracciolo, M de Rijke, and J Kircz. Towards scientific information disclosure through concept hierarchies. In *Proceedings of the 6th International ICCC/IFIP Conference on Electronic Publishing (ELPUB02)*, 2002.

CLEF. Cross-Language Evaluation Forum, 2003.
URL: `http://www.clef-campaign.org/`.

M. Cutler, Y. Shih, and W. Meng. Using the structure of HTML documents to improve retrieval. In *Proc. USENIX Symp. Internet Technologies and Systems*, 1997.

P. DuBois. *MySQL: The definitive guide to using, programming, and administering MySQL 4*. Sams Publishing, 2003.

D. Harman. Overview of the trec 2002 novelty track. In *TREC*, 2002.

V. Hollink, J. Kamps, C. Monz, and M. de Rijke. Monolingual document retrieval for european languages. *Information Retrieval*, pages 33–52, 2004.

D. Hull. Evaluating evaluation measure stability. In *Proceedings of SIGIR 2000*, 2000.

INEX. INitiative for the Evaluation of XML Retrieval, 2004.
URL: `http://inex.is.informatik.uni-duisburg.de:2004/`.

B.J. Jansen. An investigation into the use of simple queries on web ir systems. *Information Research: An Electronic Journal*, 6(1), 2000.

V. Jijkoun, J. Kamps, G. Mischne, and M. de Rijke. The university of amsterdam at trec 2003. In *TREC*, 2003.

J.S. Justeson and S.M. Katz. Technical terminology: some linguistic properties and an algorithm for identification in text., 1995.

J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. Xml retrieval: What to retrieve. In *Proceedings of SIGIR*, 2003.

G. Kazai, M. Lalmas, and A.P. de Vries. The overlap problem in content-oriented xml retrieval evaluation. In *Proceedings of SIGIR*, 2004.

G Kazai, M Lalmas, and T Roelleke. Focussed structured document retrieval. In *Proceedings of the 9th International String Processing and Information Retrieval Symposium, SPIRE*, 2002.

D.E. Knuth. *The Art of Computer Programming*. Four volumes. Addison-Wesley, 1968. Seven volumes planned.

B. Krenn and S. Evert. Can we do better than frequency? a case study on extracting pp-verb collocations. In *Proceedings of the ACL Workshop on Collocations*, 2001.

K.L. Kwok, L. Grunfeld, M. Chan, and N. Dinstl. Trec-7 ad-hoc, high precision and filtering experiments using pircs. In *Proceedings of TREC-7*, 2000.

M. Li and P. Vítanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 1997.

C.D. Manning and H. Schütze. *Foundations of statistical language processing*. The MIT Press, 1999.

J. Mayfield and T. Finin. Information retrieval on the Semantic Web: Integrating inference and retrieval. In *Proc. SIGIR 2003 Semantic Web Workshop*, 2003.

G. Mishne. Source code retrieval using conceptual graphs. Master's thesis, Universiteit van Amsterdam, 2004.

C. Monz. *From Document Retrieval to Question Answering*. PhD thesis, Universiteit van Amsterdam, 2003.

C. Monz and M. de Rijke. Tequesta: The university of amsterdam's textual question answering system. In *TREC*, 2001.

C. Monz, M. de Rijke, J. Kamps, W. van Hage, and V. Hollink. The **FlexIR** information retrieval system. Manual, Language & Inference Technology Group, U. of Amsterdam, 2002a.

C. Monz, J. Kamps, and M. de Rijke. The university of amsterdam at trec 2002. In *TREC*, 2002b.

Netscape. The Open Directory Project, dmoz, 1998.
URL: `http://dmoz.org/`.

K. Olsen. Digital Equipment Corporation, 1957.
URL: `http://en.wikipedia.org/wiki/Digital_Equipment_Corporation`.

R Pohlmann and W Kraaij. The effect of syntactic phrase indexing on retrieval performance for dutch texts, 1997.

M.F. Porter. An algorithm for suffix stripping. *Program*, pages 130–137, 1980.
URL: `http://www.tartarus.org/~martin/PorterStemmer/`.

S.E. Robertson, S Walker, S Jones, and M.M. Hancock-Beaulieu. Okapi at trec-3. In *Proceedings of TREC-3*, 1996.

G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

L. Schamber. Relevance and information behavior. *Annual Review of Information Science and Technology*, pages 3–48, 1994.

H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, 1994.

U. Shah, T. Finin, A. Joshi, R.S. Cost, and J. Mayfield. Information retrieval on the semantic web. In *Proc. CIKM 2002*, 2002a.

U Shah, T Finin, and J Mayfield. Information retrieval on the semantic web, 2002b.

C.E. Shannon. The mathematical theory of communication. *Bell System Technology Journal*, 1948.

K. Spärck Jones. Automatic indexing. *Journal of Documentation*, pages 393–432, 1979.

the World Wide Web Consortium. HTML, 1992.
URL: `http://www.w3.org/MarkUp`.

TREC. Text REtrieval Conference, 2003.
URL: `http://trec.nist.gov/`.

J. van Benthem and A. ter Meulen, editors. *Handbook of Logic and Language*. Elsevier, 1997.

C.J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

E. Voorhees. Using wordnet to disambiguate word senses for text retrieval. In *Proceedings of the 16th annual international ACM SIGIR conference on research and development in information retrieval*, 1993.

E.M. Voorhees and C. Buckley. The effect of topic set size on retrieval experiment error. In *SIGIR*, 2002.

E.M. Voorhees and D. Harman. Overview of the ninth text retrieval conference (trec-9). In *Proceedings of TREC-9*, 2002.

I.H. Witten, A. Moffat, and T.C. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann, 1999.

J. Xu and W.B. Croft. Query expansion using local and global document analysis. In *SIGIR*, 1996.

Yahoo! Inc. Yahoo! Directory, 1995.
URL: `http://www.yahoo.com/`.

G.K. Zipf. *Human Behaviour and the Principle of Least Effort: An introduction to Human Ecology*. Addison-Wesley, 1949.