Content-oriented retrieval on document-centric XML

Vom Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Diplom-Informatiker Philipp Dopichaj

Dekan des Fachbereichs Informatik:

Prof. Dr. Reinhard Gotzhein

Promotionskommission:

Vorsitzender:	Prof. Dr. Markus Nebel
Berichterstatter:	Prof. Dr. Dr. Theo Härder
	Prof. Dr. Michael M. Richter

Datum der wissenschaftlichen Aussprache:

21. Dezember 2007

D 386

Acknowledgements

"I have to thank you for a good deal," said he. "Perhaps I'll pay my debt some day."

(The Adventure of the Norwood Builder)

This thesis is the result of my work in the Databases and Information Systems group (DBIS) at the University of Kaiserslautern over the course of four years. In this time, many people supported me in my work, and I would like to take the opportunity to thank them.

First and foremost, my thanks go to my advisor, Prof. Dr. Dr. Theo Härder, and to Prof. Dr. Michael M. Richter for accepting the role of the second examiner. Prof. Härder provided me with the opportunities to explore the topic freely and was amazingly fast in providing feedback on draft versions of this thesis. Prof. Richter showed a great interest in my research, and many discussions helped to clear up several important questions; without his comments, this thesis would be quite different. I would also like to thank Dr. Ulrich Marder, who initially drew my attention to XML retrieval and later proofread draft versions of this thesis; his valuable comments helped to improve it significantly. Dr. Wolfgang Lenski provided useful input in the early stages of my work.

Furthermore, two students worked on Masters' theses in the context of this work: Benedikt Eger, who created the initial implementation of the search engine based on Apache Lucene, and Christoph R. Hartel, whose excellent work on content-and-structure search unfortunately could not be included in this thesis due to time constraints.

I would also like to thank my (former) colleagues for the nice atmosphere in the group and the daily break in the *Teeecke*. Special thanks go to Jürgen Göres, Nikolas Nehmer, and Karsten Schmidt for moral support; Andreas Bühmann for his expert advice on LaTeX problems; Christian Mathis and Dr. Michael P. Haustein for relaxing and entertaining music sessions; and Lothar Gauß and Manuela Burkart for a real-world perspective.

The quotations at the start of the chapters were taken from Sherlock Holmes stories (Doyle, 2007). Chapter 8 is in part based on an article of mine (Dopichaj, 2005) whose copyright is held by the Gesellschaft for Informatik (re-used in accordance with the copyright assignment).

Kaiserslautern, January 2008 Philipp Dopichaj

Abstract

XML is the perfect format for storing (mostly) textual documents in a digital library; its flexibility enables users to store both highly structured data (like database records) and free text in the same document. The data-centric parts can be searched using query languages like XPath and XQuery, where exact conditions on the structure can be imposed. For digital libraries, however, it is important to be able to search the free-text parts effectively. Standard information retrieval systems would return complete documents as retrieval results, which is useful for short documents such as web pages. If the document collection consists of books, however, this result granularity is too coarse. Users should be able to find the information that helps them solve their problem *without having to wade through much information that is not relevant for their problem.*

To this end, content-oriented XML retrieval can help: In content-oriented XML retrieval, documents are not considered atomic entities as they are in traditional text-based information retrieval. A retrieval result can not only contain complete documents, but also parts of documents such as chapters or paragraphs.

This thesis investigates several major aspects of content-oriented retrieval on XML documents:

- The adaptation of standard information retrieval techniques to XML. Although the adaptation is mostly straightforward, but several peculiarities of XML have to be taken into account.

- A space-efficient implementation of this base retrieval engine using customized index structures. Although the base retrieval engine uses the same concept of similarity as standard information retrieval systems, it is possible to take the XML structure into account when indexing to save space and time.

- A novel method for improving retrieval quality by making use of the document structure. In particular, section titles are exploited for finding highly relevant sections in the documents.

- A detailed evaluation of the retrieval quality for the base retrieval system and the proposed method. This evaluation is based on my own implementation of the retrieval system and a standard benchmark.

- Preliminary ideas for searching data-centric parts of the documents.

The only major part missing for a usable retrieval system is the user interface.

Contents

Nomenclature

$\mathrm{cf}(t)$	The collection frequency of term t, page 53
$\mathrm{coord}(q, d)$	The number of terms from q that also occur in d, page 28
$\mathrm{df}(t)$	The document frequency of term t, page 26
$\mathrm{ef}_A(t)$	The number of occurrences of term t in elements of type A (element frequency), page 48
$\mathrm{sim}(q, d)$	The similarity of query q to document d, page 27
$\mathrm{istitle}(e)$	A predicate that determines whether e is a title element, page 93
$\mathrm{len}(d)$	The length (in terms) of document d, page 31
$\mathrm{type}(e)$	The type of element e (that is, the element name), page 50
$\mathrm{parent}(e)$	The parent element of element e, page 72
$\mathrm{pos}(e)$	The position of element e inside its parent element, in terms., page 101
$\mathrm{tf}(d, t)$	The frequency of term t in document d, page 26
θ_T	A language model for the text T, page 49
$\mathrm{tsim}(q, e, t)$	A function to determine the new similarity of parent e of title t, page 96
N	The number of documents in the collection, page 26
$P(A)$	The probability of the occurrence of A, page 32
$P(A\|B)$	The conditional probability of the occurrence of A given B, page 32

1 Introduction

"It is evidently a case of extraordinary interest, and one which presented immense opportunities to the scientific expert."

(The Hound of the Baskervilles)

1.1 Motivation

Although a wide range of retrieval methods and search engines for text exist, these methods are not necessarily suitable for all modern uses. More and more documents are stored in semistructured format – in particular XML –, and users want to be able to search this data as effectively as possible. Naturally, it is possible to use standard search engines (designed for flat documents) on the XML documents, but without special tailoring, these search engines will not be able to exploit the structure to pinpoint the most suitable results.

For example, a user who wants to find a brief description of the differences between the two XML processing models DOM and SAX might search an electronic collection of computer books. A standard search engine will have a fixed notion of suitable retrieval results, it might, for example, index on a document or on a chapter basis, and then all retrieval results would be complete documents or chapters. Specialized search engines for semistructured retrieval can be more flexible: They recognize the structure inherent in the documents and can return anything from a single paragraph to a complete book. For the user in this scenario, it would not be very useful to get a complete book, even if its title is "DOM and SAX" – in this case, the task of searching the book itself would be delegated to the searcher, but this should really be done by the search engine. A search engine that is prepared to make use of semistructured documents can handle this task and find a single section that discusses this topic in condensed form, saving the user a lot of time. I will call the retrieval of elements chosen by the search engine *element retrieval* from now on. Larsen et al. (2006) discovered in interactive experiments that users are indeed interested in looking at smaller parts of documents (in the tests, mostly sections from journal articles), so there appears to be a real need.

Semistructured data provides further opportunities to the search engine in addition to the choice of the result granularity. A search engine for semistructured data can also exploit the structure of the documents in other ways: It can make use of semantic markup in order to "understand" the contents better. For example, if all persons' names are marked up as such, it becomes much simpler to answer a request for information about a person named John Parrot than if the surname occurs in the text exactly like the animal's name.

Traditional information retrieval for flat documents is a mature research area, but it has not been very long since research has gone into semistructured retrieval. Although it is possible to make straightforward modifications to standard IR methods, the resulting search engines will neither be efficient nor will they make good use of the document structure. Thus it is necessary to enhance the standard IR techniques with novel methods specifically designed for semistructured retrieval.

1.2 Real-world use cases

In fact, limited forms of XML element retrieval are indeed used in the real world: Online book services like O'Reilly Safari[1] and Books24x7[2], provide extensive virtual libraries of technical books (Dopichaj, 2006a). Users have several options for locating the books they want to read:

- They can use the hierarchical categorization system that the library services provide; for example, Safari places a book on DOM and SAX in "Tech books → Internet/Online → XML". A book can cover several topics, so it may well occur in several categories, and, by necessity, there is still a large number of books in each category, so it can be tedious to find a suitable one.

- As an alternative, they can perform a keyword search on all books, which results in a display of the most suitable books and sections. Instead of having to wade through all books in a category and then manually inspect the book to find the most suitable part, the user is presented with direct references to the best entry points of the most relevant books.

In these cases, element retrieval is arguably a significant improvement of the user interface.

Indeed, XML retrieval is important enough that there are several commercial vendors that produce and sell retrieval engines (Lehtonen, 2006); see table 1.1. This indicates that there is commercial interest in XML retrieval functionality. Presumably, most of the deployed XML retrieval systems are only used internally for document and information management, with the systems hand-tuned to the situation and the documents.

Trotman et al. (2007b) discuss various existing and potential use cases for XML information retrieval, each one demonstrating the need for sub-document retrieval. The examples range from maintenance personnel searching for specific information to writers of academic papers who need to find relevant sections from related work. Lehtonen et al. (2007) create a hierarchy of the use cases:

- Layout-oriented document types
  - Book search

[1] http://safari.oreilly.com
[2] http://books24x7.com

Vendor	Product
Astoria Software	Astoria XML Content Management Platform
IBM	WebSphere Information Integrator OmniFind Edition
IXIASOFT	TEXTML Server
Mark Logic Corp.	MarkLogic Server

Table 1.1: Vendors of commercial XML retrieval systems (Lehtonen, 2006).

  - Article search
  - Fragment search
- Content-oriented document types
  - Semantic search
  - Entity search
  - Data retrieval
- Process-oriented document types
  - Multimedia search
  - Feed search
  - Web Services search
  - Message search

Thus, element retrieval in general and XML retrieval in particular is a research area of practical value that warrants the development of more advanced retrieval techniques.

1.3 Aims of this thesis

In this thesis, I will present the concepts and techniques for a versatile XML retrieval system that can be used for both text-based and structure-based retrieval on (mostly) document-centric XML files without much manual tweaking. In the context of the classification from Lehtonen et al. (2007), I will focus on layout-oriented document types, in particular, book and article search.

The aim is to be as independent of the concrete document schemas as possible, even to the point of supporting collections from different sources and with different tagging conventions without special-case coding. The search engine still tries to deduce the roles of some elements from their structural relations to the other nodes so that it can recognize, for example, section titles and use this information to improve retrieval quality. On the other hand, of course, the system can still make use of further information about the schemas, if the users are willing to provide it.

Not only the administrators of the search system can help improve retrieval results, the searchers, too, can support the search engine: Instead of using simple keyword queries (as made famous by Web search engines), the users can also add structural parts to the query which provide hints to the search engine about which elements might be most likely to contain relevant information.

Obviously, the more the users help the search engine, the easier it gets to obtain good results (although it might well happen that ill-informed users provide misleading “help”), but even without further support from the user, the results should be of good quality.

1.4 Overview

Chapter 2 introduces preliminary concepts that are relevant to this thesis: XML-related technologies and information retrieval.

Chapter 3 combines XML and information retrieval to introduce XML information retrieval. The core aims of XML information retrieval are discussed in detail, and an overview of the related work in this area is given. Furthermore, a benchmarking workshop for evaluating retrieval quality of XML information retrieval systems is described; this benchmark is the basis for the quantitative evaluation in chapter 7.

The remaining chapters discuss my contribution to this research area. First, the overall design of my base retrieval system is described in chapter 4. This base retrieval engine is a minimal adaptation of standard information retrieval techniques to XML retrieval; the main difference is that the basic retrieval unit is now an element instead of a document. The concepts are generic enough to provide for variation of the similarity calculation.

The implementation of the base retrieval engine is discussed in chapter 5. The main focus of this chapter is the description of space-efficient index structures for full-text XML retrieval, based on the combination of standard information retrieval structures and XML-specific extensions that take the tree structure of the documents into account. These index structures can lead to great space savings, which – due to reduced input/output operations – also lead to shorter retrieval times.

Chapter 6 discusses an extension of the basic retrieval model from chapter 4. Whereas the base model mostly ignores the document structure, this extension makes use of semantic information that is present in XML format, but not readily available in standard information retrieval. In particular, a parameterized similarity measure that makes use of section titles is introduced.

Chapter 7 evaluates the retrieval quality of both the base retrieval engine and the extended XML-specific version on a standard benchmark. Although the benchmark does not quite match the intended usage scenario, the evaluation can still give important results about the utility of the contribution of this thesis.

So far, the similarity measures make little use of existing background knowledge about the document collections. Chapter 8 briefly describes the core idea of using existing similarity measures from other research areas in the context of XML retrieval. Unfortunately, the standard benchmark is not suitable for this method of retrieval, so no quantitative evaluation could be performed.

The thesis concludes with chapter 9, which summarizes the work and outlines future research directions.

2 Preliminaries

"Elementary," he said.

(The Adventure of the Crooked Man)

To understand XML retrieval, one must first understand what XML is and what retrieval is, before one can understand how these two can be combined. Thus, this chapter gives an overview of XML and traditional information retrieval.

2.1 Overview of XML

The Extensible Markup Language XML (Bray et al., 2006b) is a metamarkup language with a long history. The aim of markup languages in general is to enrich textual data with metadata. The metadata is contained in the markup, and the markup is linked to the text in some way – often by embedding. One example of a widely-known markup language is the Hypertext Markup Language (HTML), which is used for describing web pages. HTML markup contains instructions for rendering the web pages in a device-independent manner, like "the following text should be presented in italics" (`<i>`) or "the following text is a first-level heading" (`<h1>`).

The markup vocabulary of HTML was specifically designed for web pages, so it cannot easily be used for other types of text or data, which may require different markup languages. However, the basic syntax of embedding markup in the text – angle brackets with the markup tags – can be reused for many markup languages, so it is useful to define a metalanguage for describing markup languages. Indeed, HTML is just a specific instance of an SGML-based markup language. SGML (the Standard Generalized Markup Language (ISO, 1986)) was standardized by ISO in 1986 and provides means for describing markup languages and creating processors for working with documents written in these languages. SGML is a powerful metalanguage, and thus complex to understand and implement. It provides syntactic constructs that make hand-typing an SGML document easier, but these constructs complicate parsing considerably. It also requires the use of document schemas, which prevents ad-hoc invention of new markup elements.

XML was designed as a successor to SGML that reduces complexity to the minimum needed for the task at hand. Connolly et al. (1997) give a detailed overview of the history of XML. The remainder of this section gives a short overview of XML and related technologies, as far as relevant to this thesis.

```
 1  <?xml version='1.0' encoding='utf-8' standalone='no'?>
 2  <!DOCTYPE booklist SYSTEM 'booklist.dtd'>
 3  <?xml-stylesheet href="stylesheet.css" type="text/css" ?>
 4  <booklist>
 5      <!-- My book list -->
 6      <book key="ModernIR">
 7          <title>Modern Information Retrieval</title>
 8          <author>
 9              <first-name>Ricardo</first-name>
10              <last-name>Baeza-Yates</last-name>
11          </author>
12          <author>
13              <first-name>Berthier</first-name>
14              <last-name>Ribeiro-Neto</last-name>
15          </author>
16          <summary>
17          This book contains <emph>comprehensive</emph> coverage of information
18           retrieval techniques.
19          </summary>
20          <is-in-stock/>
21      </book>
22  </booklist>
```

Figure 2.1: An example XML document

2.1.1 XML documents

XML documents are plain-text documents that adhere to a simple structure, as figure 2.1 shows. The first line is the prolog which contains the optional XML declaration; the XML declaration can be used to declare the XML version – either 1.0 or 1.1 at the time of writing –, the character encoding of the input document (optional), and a declaration whether the document is standalone, that is, whether the parser needs any other files in order to process this document. If a document contains no XML declaration, the parser can assume it is written in XML version 1.0 and in UTF-8 character encoding, and not standalone. The second line in the example contains a reference to the document type definition (DTD) of the document, which describes the permissible structure of the document; section 2.1.3 details what this means. In line 3, even before the main contents of the document starts, a *processing instruction* occurs that points the application to a *stylesheet* that is used for displaying the XML document. Line 4 contains the *start tag* <booklist>, and the last line contains the corresponding *end tag* </booklist>. Each pair of matching start and end tags demarcates an *element*, which consists of the tags and their contents. Each XML document contains at least the root element.

Apart from elements, other types of nodes can occur in the document tree:

Comments, which are enclosed in <!-- and -->; they are intended for human readers of the XML markup, so they are ignored by the processor. Line 5 in the example document contains a comment.

- **Document** x
  - **Documenttype** x
  - **ProcessingInstruction**
    target='xml-stylesheet'
    data='href="stylesheet.css" type="text/css"'
  - **Element** name=booklist
    - **Comment** “My book list”
    - **Element** tagName='book'
      - Attr={name='key' value='ModernIR'}
      - **Element** tagName='title'
        - **Text** wholeText='Modern Information Retrieval'
    - **Element** tagName='author'

Figure 2.2: The DOM tree of the example document (incomplete). The items in **boldface** are the nodes.

Text nodes, which contain most of the data in XML documents.

Attributes, another container of data.

Preprocessing instructions, which the processor of the XML file can interpret (they may contain data like formatting details that should not be expressed in the main contents).

Since the focus of this thesis is document-centric XML, the text in an XML document is of utmost importance to us. Text includes the obvious strings like “Modern Information Retrieval”, but also the whitespace between the tags; for example, the first child of the booklist element is a text node consisting of a new-line and four space characters.

Elements may also contain *mixed content*, that is, a combination of text and sub-elements, like the summary element in lines 16 to 19. The sub-elements in mixed content – also called *inline elements* – are frequently used to provide semantic information or rendering information about the embedded text.

The start tags of elements can also have *attributes*, like key for the <book> element in line 6. Each attribute name may only occur once in each element, and the order of the attributes is not important. Attributes have values, which are usually strings, but can be restricted further in schemas (see section 2.1.3). Attributes can in principle be replaced by sub-elements of the element whose start tag has the attribute, so it is up to the schema author to decide what is appropriate in a given context. On the other hand, attributes can not replace sub-elements in all cases, because attribute values have some restrictions on their contents. Some people argue that attributes should be used for metadata about the corresponding element that should typically not be displayed when rendering, but there is no widespread consensus about this.

It can happen that two schemas from different domains use the same element names to mean completely different things, for example, an apple element could be used both for computers and fruit. This is no problem as long as the schemas are not used in the same document; in reality however, this need arises frequently. The original version of the XML specification provided no means for mixing content from different schemas. The XML specification (Bray et al., 2006a) addresses this problem by introducing namespaces as qualifiers for element and attribute names, so the element could be disambiguated as either fruit:apple or computer:apple. Using a simple alphanumeric string as the qualifier could still lead to collisions, however, so it is only used as an abbreviation for the real namespace, which is a uniform resource identifier (URI).

In practice, namespaces are usually URLs, for example, the namespace of XSLT is http://www.w3.org/1999/XSL/Transform, so an XSLT stylesheet typically begins with the following line:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

The namespace is declared with the attribute-like syntax xmlns:*prefix*="*uri*", which binds the given prefix to the URI, so that all later occurrences of this prefix below the element on which the declaration occurs refer to that namespace. Note that the prefix is arbitrary, so instead of xsl, one could just as well use xquery. Because namespaces are attached to elements, their scope is limited to their descendants, and a prefix can also be overwritten. It is also possible to specify a default namespace with xmlns="*uri*"; this namespace applies if no prefix is given.

2.1.2 Processing models

Although XML is a relatively simple markup language, writing a parser for it is still not an easy task, so software applications using XML should not include their own implementations but should use a reusable parser in a library.

SAX, the Simple API for XML[1] (Brownell, 2002), was developed in late 1997 and 1998 – soon after the adoption of the XML standard – in order to unify the interface to XML parsers. It is an industry standard that was originally developed for Java, but is now available for all major programming languages. SAX is event-based: The application program registers an event handler with the SAX parser, and the parser reads the XML document sequentially and sends events like *start element* or *text* to the event handler, which can then interpret them and take corresponding actions; see figure 2.3 for the events generated for the example document from figure 2.1. The parser drives the application, so it is not possible to navigate the XML document at will. If the application needs to access to preceding parts of the document, it must store them itself.

DOM, the Document Object Model (Hors et al., 2004), works at a higher level of abstraction than SAX: SAX is a low-level API – the application programmer is responsible for storing the necessary information from the XML document. In many circumstances, a higher level of abstraction is useful, an abstraction that

[1] http://www.saxproject.org/

start document	
start element	booklist
characters	<whitespace>
start element	book
characters	<whitespace>
start element	title
characters	Modern Information Ret
characters	rieval
end element	title
...	
end element	booklist
end document	

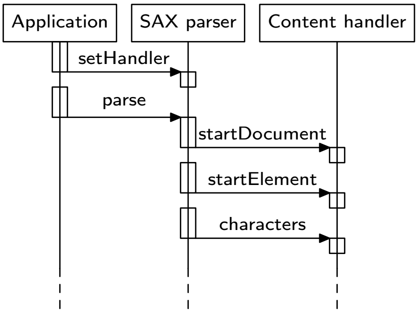


Figure 2.3: A possible sequence of SAX events generated for the XML file in figure 2.1. <whitespace> indicates that the SAX parser passes the whitespace from the document to the document handler. Note that the parser may split the character data into several events, like "Modern Information Ret" and "rieval" in the example.

deals with the XML concepts of elements, text nodes, and so on. DOM is the most popular implementation of this concept. It is a specification by the World Wide Web Consortium (W3C). DOM is based on the observation that an XML document can be interpreted as an ordered tree of nodes, and all nodes are descendants of the root element. DOM provides easy access to the XML document tree using operations like *getFirstChild*, *getParent*, *getNextSibling*, and so on. Every part of the XML document is accessible at all times, either through DOM navigation operations or through direct stored references to the nodes. DOM is language-independent, so a programmer working with several programming languages will find the same interface in all of them, at the cost of the interface not making good use of each language's features.

In typical implementations, DOM trees use a lot of main memory (about ten times as much as the size of the XML file according to Brownell (2002, pp. 6–7)). Programs using SAX parsers, however, can determine themselves how much data they want to store, so memory consumption can be reduced significantly. If random access to all nodes is required, however, DOM is preferable to SAX – if SAX is used, the programmer must manually keep track of the document structure.

2.1.3 Well-formedness and validity

An XML document must be *well-formed* so that it is not rejected by the XML parser – standard-conforming XML parsers *must* reject documents that are not well-formed. As such, it is not a means of classifying XML documents: a document that is not well-formed is simply not XML, according to the standard. To be well-formed, a document must obey the simple syntactic rules of XML, like not using invalid characters for tag names and nesting elements properly; for example,

```
<!ELEMENT booklist (book*)>
<!ELEMENT book (title, author+, summary, is-in-stock?)>
  <!ATTLIST book key ID #REQUIRED>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT emph (#PCDATA)>
<!ELEMENT author (first-name, last-name)>
<!ELEMENT is-in-stock EMPTY>
<!ELEMENT summary (#PCDATA|emph)*>
```

Figure 2.4: A DTD for the XML document from figure 2.1. A booklist element may contain a sequence of book elements (whitespace is ignored), a book element must have a title, one or more authors, a summary, and an optional is-in-stock element. #PCDATA stands for textual content; the summary element may have mixed content, that is, text as well as element children.

<a><b></a></b> is not a well-formed XML fragment, because the <b> start tag is closed after the <a> tag which precedes it. Thus, well-formedness merely checks that the document conforms to the XML metasyntax, but not that of any specific markup language.

A stricter requirement is *validity*, that is, conformance to a specific markup language as specified in a *schema*. A schema describes the syntactic structure of an XML document at a higher level, for example, which element types may contain text or what other element types they must contain in what order. For the document in figure 2.1, a schema might specify that a <book> element contains a <title> element followed by one or more <author> elements and an optional <summary>.

Less complex schema description languages, like Document Type Descriptors (DTDs), do not give much control over the textual contents of elements to the schema authors, whereas more recent languages like XML Schema and Relax NG provide support for data types like integers or dates; this allows for better parse-time checking of the data-centric parts of XML documents. If the data types are checked at parse time, the application writer is relieved of that burden and can simply assume that the data is of the expected type. Although DTDs are the most primitive of the schema languages, they still play an important role: They are part of the base XML specification (Bray et al., 2006b), so every validating XML parser must support them – support of the advanced schema languages is not universal. See figure 2.4 for a DTD for the example document.

In order to check the validity of an XML document, the program needs to use a *validating parser*, that is, a parser that is capable of handling the schema language and set to operate in validating mode. In this mode, a document not conforming to the schema is simply rejected so that it cannot be accessed by the application. A *non-validating parser*, on the other hand, ignores the schema so that even a document that does not match the schema can be parsed.

2.1.4 Types of XML documents

XML documents come in several flavors, the most important distinction being document centric versus data centric. *Data-centric XML documents* use XML to store strongly structured data – akin to typical database records – in a hierarchical form. Schemas for data-centric XML will often specify data types for the fields and a rigid structure that forbids mixed content (elements interspersed with text). One well-known example is the XML version of the DBLP collection of computer science references[2]; the document from figure 2.1 is also data-centric.

Document-centric XML documents are more free-form (and closer to the original ideas that spurred the invention of markup languages): They mostly consist of text, and XML markup is used to impose structure on the text and provide semantic or formatting instructions for the document processor. Well-known document-centric XML formats are DocBook and the Open Document Format, as produced by OpenOffice.org. The occurrence of *mixed content* – text with embedded markup elements – is a common characteristic of document-centric XML. Schemas for document-centric XML typically provide element types for the structure of the documents – chapters, sections, paragraphs, and so on –, and for semantic or visual inline markup, like italics or acronyms.

The distinction between these two extremes is not as clear as it might appear at first sight. Document-centric XML often contains strongly structured metadata like author information, and data-centric XML frequently has mixed content embedded in some fields.

In any event, it is necessary to search documents of either type effectively, and different approaches can be used for maximum effectiveness.

2.1.5 Rendering XML

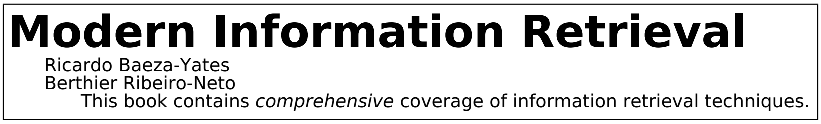
Although XML files are human-readable, they are not nice to read. Especially for document-centric XML, the desired output is usually a nicely typeset document suitable for on-screen display or printing. The important part is that the styling information is stored separate from the XML document; thus, it is possible to change the way a document is displayed without changing the document itself. This is especially useful for rendering documents differently for different output devices. A web page, for example, usually has navigation bars that are useful for in-browser display, but are merely clutter in printed output. Different style sheets can be used for this (or a single style sheet with different parameters), so that the navigation is omitted for printed output.

Effectively, web browsers can be seen as rendering engines for HTML and, in recent versions, also XML.

Cascading style sheets (CSS) (Bos et al., 2006) is one of the simpler options for rendering XML, see figure 2.5 for an example style sheet. Cascading style sheets specify how (and if) the elements in the input documents should be rendered, but it provides not means for changing the order of the elements. In the example, it is impossible to make the summary appear before the author names.

[2] http://dblp.uni-trier.de/xml/

```
 title  { font-weight: bold; font- size : 22pt; }
summary { display: block; text-indent: 4em; }
author {  display : block; text-indent: 2em; }
emph { font- style : italic ; }
```



Modern Information Retrieval

Ricardo Baeza-Yates
Berthier Ribeiro-Neto
This book contains *comprehensive* coverage of information retrieval techniques.

Figure 2.5: A cascading style sheet for the XML document in figure 2.1 and the rendered document.

The Extensible Stylesheet Language XSL (Berglund, 2006) is significantly more powerful than CSS, but it is also harder to write. Rendering an XML document is done in two steps: First, an XSLT style sheet is used to convert the input document to the intermediate format XSL-FO, and then an XSL-FO processor renders this intermediate form.

XSLT is a general-purpose XML style sheet language that can be used to transform XML into other XML dialects or other text formats. It is much more powerful than CSS, as it is a turing-complete programming language, so that it can re-arrange the input elements. In fact, it can also be used as a query language for XML, as will be shown later.

XSLT alone can only be used for transforming one dialect of XML into another; for rendering, one needs a dialect that provides formatting operations. The specification for XSL *formatting objects* (Pawson, 2002) does just that: It provides an XML vocabulary for describing rendered documents.

Figure 2.6 shows an example style sheet and the final rendered output. Note how the order of the authors' first and last names is reversed compared to the input document.

2.1.6 XML information set

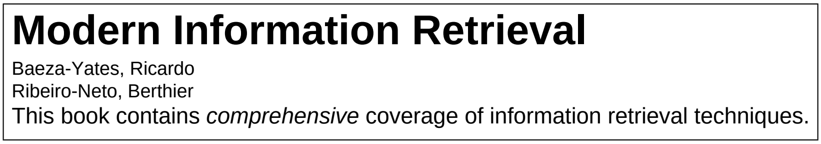
The *XML Information Set* – also called Infoset – provides a model for describing the content of well-formed XML documents that conform to the XML namespaces extension. It operates at a higher level of abstraction than the syntactic structures that the XML standard talks about. For example, `<elem></elem>` and `<elem/>` are produced by two different syntax rules, yet they both represent the concept of an empty element. The infoset is used as a basis for the specification of later versions of the XML query languages XPath, XQuery, and XSLT (see section 3.1.1).

It provides definitions for eleven types of *information items*, each of which has several characteristic properties. For this thesis, the following information items are relevant:

Document: There is exactly one document information item in an XML document, representing the document itself.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <xsl: stylesheet
 3     xmlns: xsl ="http://www.w3.org/1999/XSL/Transform" version="1.0"
 4     xmlns:fo="http://www.w3.org/1999/XSL/Format">
 5
 6     <xsl:template match="/">
 7         <fo:root>
 8             <fo:layout-master-set>
 9                 <fo:simple-page-master master-name="master">
10                     <fo:region-body region-name="body"/>
11                 </fo:simple-page-master>
12             </fo:layout-master-set>
13             <fo:page-sequence master-reference="master">
14                 <fo:flow flow-name="body">
15                     <xsl:apply-templates/>
16                 </fo:flow>
17             </fo:page-sequence>
18         </fo:root>
19     </xsl:template>
20
21     <xsl:template match="title">
22         <fo:block font-size="22pt" font-weight="bold"><xsl:apply-templates/></fo:block>
23     </xsl:template>
24
25     <xsl:template match="author">
26         <fo:block font-size="10pt">
27             <xsl:value-of select="last-name"/>, <xsl:value-of select="first-name"/>
28         </fo:block>
29     </xsl:template>
30
31     <xsl:template match="summary">
32         <fo:block><xsl:apply-templates/></fo:block>
33     </xsl:template>
34
35     <xsl:template match="emph">
36         <fo: inline font-style="italic"><xsl:apply-templates/></fo:inline>
37     </xsl:template>
38 </xsl: stylesheet >
```

(a) XSLT style sheet for converting the book list from figure 2.1 to XSL-FO. The root template in lines 6 to 19 sets up the page layout and a region for text, and the other templates format the text.



Modern Information Retrieval

Baeza-Yates, Ricardo
Ribeiro-Neto, Berthier

This book contains *comprehensive* coverage of information retrieval techniques.

(b) Output generated from the XSL-FO.

Figure 2.6: Example of XSLT and XSL-FO.

Element: Each element in the XML document is represented by an element information item.

Attribute: Attributes can only occur as children of elements. Attribute values are normalized.

Character: Each character in texts is stored in a separate entity as far as the infoset is concerned.

2.1.7 Terminology

The terminology is based on the definition in the infoset and DOM W3C recommendations.

The *element name* consists of two parts: the optional namespace name and the local name. It occurs in two places in the XML document file, in the start tag and in the end tag.

The *parent* of an element is the parent element as defined by DOM; likewise, the *child elements* (*children*) are a list of the children of type "element" in the DOM tree in document order.

The *recursive text content* of an element is the concatenation of all descendant text nodes in the DOM tree, whereas the *direct text content* of an element is the concatenation of all text nodes that are children of this element. For practical reasons related to tokenization (see section 5.2.1), all tags are replaced by a single space to ensure that element boundaries are also word boundaries. For example, given `<a>1␣<b>2</b>␣3<c/>4</a>`, the recursive text content of `a` is "`1␣␣2␣␣3␣4`", and the direct text content is "`1␣␣␣␣3␣4`" ("`␣`" denotes the space character).

2.2 Information retrieval and data retrieval

Before delving into XML retrieval, it is helpful to first obtain a working knowledge of retrieval on both unstructured text documents and highly structured data. In this section, I will give a short overview of information and data retrieval, with an emphasis on the former. Due to the nature of the topic, this section cannot cover it in detail; instead I will focus on the parts that are relevant to this thesis. Baeza-Yates and Ribeiro-Neto (1999) provide an in-depth treatment of many facets of information retrieval.

Information retrieval aims at satisfying a user's *information need* by retrieving the most relevant documents from a larger collection. In current systems, the user typically has to specify his information need in a formal language that the information retrieval system understands; this formulation is called the *query*. In information retrieval, the query is not a clear constraint, but rather a vague hint; this implies that the retrieval system must try to deduce from the documents' contents to which degree a given document satisfies the information need specified in the query. The retrieved results are then ordered by this estimation of *relevance*, which, in the ideal case, matches the user's perception of it. In the context of XML retrieval, information retrieval is most useful for document-centric XML.

Data retrieval differs from information retrieval in this respect: The query exactly specifies the constraints that a document has to satisfy in order to be retrieved; there is no vagueness. Thus, the retrieval result is a set of relevant documents, all of which are equally good. The query language SQL from the relational database world and the basic forms of XQuery and XPath (see section 3.1.1) without the full-text extensions are examples of query languages for data retrieval. They are characterized by operators with clear-cut semantics, for example "year $>$ 1945". In XML retrieval, this form of retrieval can be used for data-centric XML documents.

2.2.1 The vector-space model

In databases, the records are normally in attribute–value form, that is, the schema defines which attributes are present in a record and their data types, and each record assigns values to the attributes. The data types are mostly simple, like numbers, dates, or short texts (less than a line long). Information retrieval, on the other hand, deals with *documents*, that is, longer text with little or no inherent structure. In order to provide more than substring search or pattern matching, the search engine must first prepare the input documents. The word is the smallest unit of information in a text, so it is reasonable to split the text into a sequence of words and assume that the queries are also composed of words. Determining the relevance of a document then leads to a comparison of the words in the document and the words in the query.

For various reasons, it can be desirable to process the words before the comparison, for example, the search engine should be able to match different forms of words like "compute" and "computing". The processed form of a word is called an *index term*, and the process of going from the word in the input document to the index term is called *conflation*. There are many forms of term preprocessing:

- Case folding, so that "For" and "for" are considered the same term.
- Stemming, that is, reduction to the stem of a word; for "compute" and "computing", the stem could be "comput" for example (it is usually not the base form). Well-known stemmers for the English language are the Porter and the Lovins stemmer.

Term preprocessing serves two main purposes: It makes it easier to formulate queries, since the different word forms do not have to be taken into account, and the size of the index is reduced significantly – Witten et al. (1999, p. 147) cite reduction of 30 to 40 percent for the TREC collection. An unintended side effect is the reduction of precision, since many terms that are distinct in the documents can no longer be distinguished from the index terms alone. This may be a problem if the terms before and after conflation mean different things. Overall, the benefits outweigh the problems, so real-world search engines usually perform some form of conflation.

The simplest form of retrieval on documents is *Boolean retrieval*, for which the search engine only determines whether the query terms occur in the documents, so the result is an unordered list of documents the searcher has to inspect manually to find the ones satisfying his information need.

Obviously, this is not satisfactory. For ranked retrieval, more complex methods must be used so that the search engine can determine to what degree a document matches the query. A first step in that direction would be to count the number of query terms that are matched by a document and then rank accordingly; this hardly improves the situation because most queries are only a few words long, so that there are still many documents for each given number of matches.

One implicit assumption of the Boolean model is that either a given term is relevant for a document or it is not. In reality, however, relevance is not that coarse-grained, that is, a term might be relevant to a certain degree.

There are many approaches to this problem, but I will focus on one of the most popular models, the one that forms the basis of the retrieval engine presented in this thesis: the *vector-space model.*

In the vector-space model, each index term is presented by an axis in an n-dimensional vector space, where n is the total number of distinct index terms. Every document is mapped to a vector in this space by assigning a *term weight* to each term, denoting the importance of this term for the given document. If term t does not occur in document d_i, the weight $d_{i,t} = 0$. Queries are transformed into their representation in the same way documents are. If a query mentions a term that does not occur anywhere in the documents, it is dropped.

The term weights can be determined in different ways. A common assumption is that the more frequently a term occurs in a document, the more important it is for the document's content; on the other hand, terms that occur in (almost) all documents are less useful, for example, the fact that the term "to" occurs in a document tells us less about its contents than the occurrence of "retrieval". Thus, the weight of a term t should be the product of its *term frequency* in a given document $\mathrm{tf}(d,t)$ (that is, the number of times it occurs in that document) and its *inverse document frequency* $\mathrm{idf}(t)$ (inversely related to the proportion of all documents the term occurs in $\frac{N}{\mathrm{df}(t)}$):

$$d_t = \mathrm{tf}(d,t) \cdot \mathrm{idf}(t) \tag{2.1}$$

Inverse document frequency (IDF) is based on the observation in early retrieval experiments that query terms that occurred in almost all documents dominated the score if this was based only on the sum of the term frequencies (Spärck Jones, 2004). Originally, IDF was an ad-hoc invention to address this problem, which it certainly does well. Robertson (2004) examined *why* IDF works as well as it does by deriving it from probabilistic theory; he shows that TF-IDF can be seen as a BM25 weight (BM25 is a probabilistic retrieval model; see section 2.2.2 for details). A simple and well-proven form of IDF, given N the total number of documents and n_i the number of documents containing term t_i is as follows:

$$\mathrm{idf}(t_i) = \log \frac{N}{n_i} \tag{2.2}$$

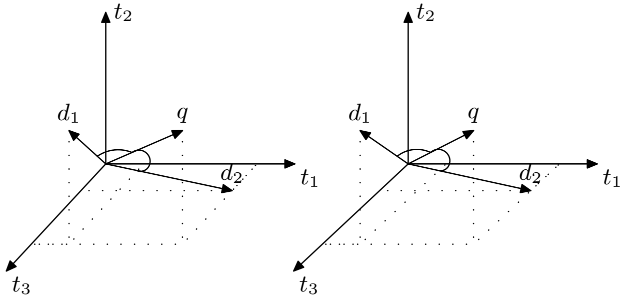


Figure 2.7: Document similarity in the vector-space model with the cosine similarity measure. The axis represent the terms t_1, t_2, t_3, and the components of the document vectors $d_1 = (1, 3, 3)$ and $d_2 = (4, 0, 1)$ and the query vector $q = (4, 3, 3)$ represent the term frequencies. The smaller the angle between the vectors is, the more similar they are; since all components are non-negative, the cosine can be used. In this case, d_2 is more similar to q than d_1: $\mathrm{sim}(q, d_2) = 0.79 > \mathrm{sim}(q, d_1) = 0.51$.

Many words in texts bear little or no semantic meaning, for example conjunctions like "and". Although IDF prevents them from dominating – these terms will occur in almost all documents, so that their IDF is close to 0 –, it is still useful to completely omit them from the index to save space.

For ranking the documents d given a query vector q, each pair of query and document (q, d) must be mapped to a numeric value denoting the search engine's estimate of the *similarity* of the document to the query, which in turn is intended to reflect the relevance of d with respect to q. A concrete similarity value is also called *retrieval status value* (RSV). Under the assumption that a document matches the query well if the query terms are as close to the index terms of the document as possible, one can use the angle between the document d and the query vector q as a measure for the similarity: the smaller the angle, the higher the similarity. For practical reasons, the cosine of the angle between q and d is used; if the vectors point in the same direction, the cosine is 1, if they are at right angles, the cosine is 0. This similarity measure is called the *cosine similarity measure* (the cosine can be obtained by dividing the scalar product of the vectors by the products of their lengths):

$$\mathrm{sim}(q, d) = \frac{q \cdot d}{|q| \cdot |d|} = \frac{\sum_t (q_t \cdot d_t)}{\sqrt{\sum_t q_t^2} \cdot \sqrt{\sum_t d_t^2}} \tag{2.3}$$

Figure 2.7 illustrates the cosine similarity measure in a three-dimensional term space.

After having calculated the similarity for each document, the search engine can then create the result list by returning the documents in decreasing order of similarity.

Obviously, it would be wasteful to match every term vector with the query: Typically, only a small fraction of the documents has any term in common with the query. This leads to the introduction of *inverted lists*, which provide an index structure that leads from a term to a list of all documents in which this term occurs, similar to the index in a book. This alone would enable us to only compute the similarity for documents with a non-zero similarity, but the search engine would still need to access the term vector for each of these documents. Most of the weights in this vector are irrelevant to the similarity; all that is really needed is the weights of the query terms and the length of the document. Thus, the inverted lists are augmented with the term weights, and the length of the documents' term vectors is stored in another index structure so that it does not need to be calculated from the term vector.

The open source search engine Lucene[3] uses a different formula:

$$\mathrm{sim}(q,d) = \mathrm{coord}(q,d) \sum_{t \in q} \left[\sqrt{\mathrm{tf}(d,t)} \left(1 + \log \left(\frac{N}{\mathrm{df}(t)+1} \right) \right) \mathrm{lnorm}(d) \right] \tag{2.4}$$

$$\mathrm{lnorm}(d) = \frac{1}{\sqrt{\mathrm{len}(d)}} \tag{2.5}$$

$$\mathrm{coord}(q,d) = |\{ t \in q : \mathrm{tf}(d,t) > 0 \}| \tag{2.6}$$

Compared to the cosine similarity measure, several things have changed:

- The *coordination factor* $\mathrm{coord}(q,d)$, which is the number of query terms in q that also occur in d, is new. The intention is to reward documents that contain more of the query terms. The result is that documents that contain all the query terms will usually end up in the first ranks in the result list, which is usually the right thing to do.

- Normalization is not done by dividing by the vector lengths, but by the square root of the number of terms in the document; this has the advantage that the normalization factor does not depend on the document frequency so that it need not be recomputed every time the document collection changes (Lee et al., 1997).

- The influence of very high term frequencies is reduced by taking the square root.

The vector-space model is rather ad-hoc, lacking a sound theoretic foundation. Although it works well in practice, it is not clear *why* it does; thus, other models were developed based on probabilistic theory.

[3] see http://lucene.apache.org

2.2.2 Probabilistic retrieval and language models

Probabilistic retrieval models aim at ranking based on the probability of relevance, rather than use some abstract value like the angle between two high-dimensional vectors. Given a query q and a document d, the probabilistic model tries to estimate the probability that d satisfies the searcher's information need. The documents in the collection can then be ranked by decreasing probability of relevance. The main difference between implementations of this basic model lies in how the probability of relevance is estimated, but the most successful methods can be seen as applications of *language models*.

Language models go back to Shannon (1951): Shannon created statistics for English texts based on n-grams (character sequences of length n): He counted the relative frequency of each n-gram and used this data to predict unknown letters from another text and found out that the information content is much lower than the amount of storage needed to encode the text in its original form. These statistics are a language model of the text corpus he analyzed; they can be seen as a means of generating new text that has the same statistical properties as this corpus.

For information retrieval, character-based language models are not useful, given that the term is the basic unit of information, so term-based models are more appropriate. For practical reasons, unigram models ($n = 1$) are the most frequently used models in this context: For each term in the corpus, the relative frequency is obtained.

Spärck Jones et al. (1998) give an exhaustive overview of the basics of *Okapi BM25* retrieval model. The two basic events that are useful for ranking are R, the event that document d is relevant, and the inverse $\bar{R}$, the event that d is not relevant. The idea is to rank based on the probability that document d is relevant, given its presentation, $P(R|d)$.

This probability cannot be calculated directly; the application of Bayes' theorem results in the following formula:

$$P(R|d) = \frac{P(d|R)P(R)}{P(d)} \tag{2.7}$$

Unfortunately, the input values cannot be computed easily, so order-preserving transformations are needed (see Spärck Jones et al. (1998) for details). Like for the vector-space model, the naïve assumption that the terms in the documents occur independently is made. Then weight function $W(A_i = a_i)$ can be defined, denoting that attribute A_i has the value a_i; for convenience, it is defined so that $W(A_i = 0) = 0$. Given this weight, the basic similarity function is:

$$W(A_i = a_i) = \log \frac{P(A_i = a_i|R)P(A_i = 0|\bar{R})}{P(A_i = a_i|\bar{R})P(A_i = 0|R)} \tag{2.8}$$

$$\mathrm{sim}(q, d) = \sum_i W(A_i = a_i) \tag{2.9}$$

The simplest approach to determining the attribute values is based on presence or absence of terms: If term t_i occurs in the document, then $a_i = 1$, otherwise

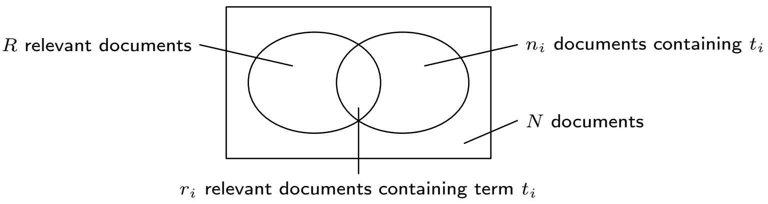

R relevant documents
n_i documents containing t_i
N documents
r_i relevant documents containing term t_i

Figure 2.8: Relevant document sets and counts for the RSJ term weight formula.

$a_i = 0$. Thus, given $p_i = P(t_i \in d|R)$ and $\bar{p_i} - P(t_i \in d|\bar{R})$, the weight function yields:

$$w_i = \log \frac{p_i(1-\bar{p_i})}{\bar{p_i}(1-p_i)} \tag{2.10}$$

Assuming the search engine has information about the document's relevance, given $n_i = \mathrm{df}(t_i)$ documents containing t_i, with r_i of those documents being relevant, N the total number of documents, and R the total number of relevant documents (see figure 2.8), the *Robertson and Spärck Jones* (RSJ) formula follows:

$$w_i = \log \frac{(r_i+0.5)(N-R-n_i+r_i+0.5)}{(R-r_i+0.5)(n_i-r_i+0.5)} \tag{2.11}$$

Unfortunately, usually no relevance information is available (thus $R = r_i = 0$), so this turns into a weighting function similar to IDF:

$$w_i = \log \frac{N-n_i+0.5}{n_i+0.5} \tag{2.12}$$

As section 2.2.1 has shown, not all terms are equally important for determining the topic of a document. In the vector-space model, the inverse document frequency is used to take that into account (the more documents contain a given term, the less useful it is for discriminating).

Okapi BM25 (Robertson and Walker, 1994) is a probabilistic retrieval model derived from the 2-Poisson probabilistic model. The term frequency is modified in order to obtain a probability of *term eliteness*, which says if a document is "about" the concept represented by the term. Term frequency information is easily available, and – as the vector-space model shows – can be helpful for improving retrieval quality, it is reasonable to use a more detailed weight function. This function can only be based on the raw term counts for the documents, since relevance information is usually not available. A certain term can occur both in documents that are about the concept that the term represents and in documents that are not.

Direct application of Poisson distributions to approximating the probabilities lead to a complex formula that is difficult to interpret, because little information is available. Robertson and Walker (1994) derived a simpler weighting formula that has similar behavior:

$$W(A_i = \mathrm{tf}(t_i)) = \frac{\mathrm{tf}(t_i) \cdot (k_1 + 1)}{k_1 + \mathrm{tf}(t_i)} w_i \tag{2.13}$$

The conversion from the plain term frequency to the term eliteness probability can be adapted with the global parameter k_1; the formula ensures that the term eliteness is 0 if the term frequency is 0, and it asymptotically approaches 1 as the term frequency increases. This implies that the first few occurrences of a term make the greatest contribution to term eliteness – the function is steep close to 0. The eliteness of term t_i for document d, using a document-length normalization constant K (see below) is defined as:

$$\mathrm{eliteness}(t_i, d) = \frac{(k_1 + 1)\,\mathrm{tf}(t_i, d)}{K + \mathrm{tf}(t_i, d)} \cdot \underbrace{\log \frac{N - \mathrm{df}(t_i) + 0.5}{\mathrm{df}(t_i) + 0.5}}_{w_i} \tag{2.14}$$

Another important enhancement of BM25 over the 2-Poisson model is the *document-length normalization.* Based on the assumption that document length is caused either by needless verbosity – this implies normalization – or a more thorough treatment of the subject – this implies no normalization –, BM25 uses partial length normalization. The degree of normalization is controlled by a global parameter b.

$$K = k_1 \left((1 - b) + b \cdot \frac{\mathrm{len}(d)}{\mathrm{avg}(\mathrm{len}(d))} \right) \tag{2.15}$$

The final similarity of document d to the query q is then accumulated as follows:

$$\mathrm{sim}(q, d) = \sum_{i=1}^{m} \mathrm{eliteness}(t_i, d) \cdot \frac{\mathrm{tf}(t_i, q)}{k_2 + \mathrm{tf}(t_i, q)} \tag{2.16}$$

For realistic queries, the term frequency in the query is usually 1, so the second multiplicand is frequently omitted ($k_2 = 0$).

The lack of relevance information for training implies that the probability of relevance can be estimated based on the query terms only. In other words, Okapi BM25 is based on estimating the probability that the *language model* of the query generates the document.

Language modeling approaches to information retrieval look at the problem the other way around: They estimate for each document the probability that the language model of the document predicts the query (Ponte and Croft, 1998; Croft, 2003). A very pragmatic reason for this approach is that the language model of a document can be estimated better, because documents are typically much longer than queries.

A statistical language model tries to capture the properties of sample text so that it becomes possible to estimate the probability of linguistic units such as sentences.

Information retrieval approaches based on language models start from the assumption that the query is in fact a distorted version of the ideal document that satisfies the searcher's information need. Thus, it is the search engine's task to find the document from the collection that has the highest probability that it generates the query. For document d, this probability can be calculated from the products of the probabilities of each query term occurring in d.

$$P(d|q) = P(q|d) \cdot P(d) \tag{2.17}$$

In this equation, $P(d)$ is the *prior probability* that the document is relevant; this probability is usually assumed to be uniform.

The *query-likelihood retrieval model* (Ponte and Croft, 1998) is a rather simple language-based model. It estimates the probability that the query is a sample from the distribution of terms found in a document.

2.2.3 Evaluating information retrieval effectiveness

The scientific method demands comparison of new strategies with old ones to show that progress has been made. For data retrieval, the quality of the retrieval results is not an issue: Either a result satisfies the constraints given in the query or it does not. Thus, comparing different retrieval approaches can only been done based on other data, most importantly space usage and retrieval time; a system that obtains the same results in a shorter time is clearly superior to its predecessor.

With information retrieval, time and space are still a concern, but now the quality of the retrieval results plays an important role, too. The relevance of a document for an information need expressed in a query cannot be determined automatically, as long as there is no true artificial intelligence. The retrieval system can only attempt to approximate relevance as well as possible. It is unreasonable to expect a system to match the ideal result – even discounting its lack of intelligence, not even people who assess the relevance of retrieval results agree anywhere close to 100 percent.

It is thus necessary to be able to compare the quality of retrieval results, and it is clear that the expected or ideal retrieval result can only be determined by people. In *Cranfield experiments* – so called because they were first done at Cranfield in the 1960s –, the retrieval results from search systems are assessed, and then a numerical measure of retrieval quality is obtained.

In 1992, the Text Retrieval Conference (*TREC*)[4] was started as a venue to compare different retrieval systems in Cranfield experiments. The conference provides several large collections of documents and, each year, new information needs and queries are formulated and used for evaluating the participating systems. Voorhees (2005) gives an overview of TREC and the evaluation methods used there.

Considering the size of the test collections – about 800,000 documents are typical –, it is infeasible to judge every document in the collection for relevance for each query. Because of this, *pooling* of results is performed: The participants submit

[4] http://trec.nist.gov/

their retrieval results for the TREC queries, and the top 100 results from each submission are collected in a pool. This pool is then assessed by the assessors, and all documents outside the pool are considered irrelevant for evaluation.

Obviously, it is necessary for scientific work to be able to perform quality evaluation outside the yearly schedule of TREC. To address this need, TREC provides an evaluation tool that can be used to perform evaluation of systems based on previous relevance assessments.

2.2.4 Retrieval of partial documents

All the retrieval methods described so far assume that the complete document is the unit of retrieval. Even before the advent of XML, there were experiments on the retrieval of parts of documents.

Salton et al. (1993) report work on passage retrieval, in which they apply a two-stage process: First, the complete documents are searched, and only the documents with a sufficient similarity are retained. Next, the retained documents are examined for highly similar passages; if there are passages with a higher similarity than the document in which they occur, they replace the document in the results. The decomposition of documents into passages is done hierarchically, that is, a document is decomposed into suitable sections, then each of these sections is further decomposed. The result of this is a tree structure similar to the one of XML documents.

Salton et al. show that it is possible to retain or even improve the quality of document retrieval, while at the same time reducing the size of results, measured in paragraphs. This is a strong argument in favor of discarding the notion of a document as an atomic unit.

Moffat et al. (1993) (see also Zobel et al., 1995; Wilkinson, 1994) executed experiments on the TREC corpus, based neither on full-document nor on passage retrieval, but on retrieval based on pages and sections (for documents marked up in SGML). The rationale they give – apart from technical reasons – is the same as for element retrieval: some of the documents are too long to be manually inspected for matches (the longest is 400,000 words long), so it is more sensible to return parts of these documents.

They propose several options for displaying the results:

- The retrieval system could retrieve complete documents and only use the parts for scoring the documents.
- The retrieval system could present the parts in the same way as the complete documents.
- The retrieval system could present the parts in the context of the whole document.

This foreshadows some of the retrieval tasks that were later introduced for the INEX workshops, which will be covered later.

For experimental evaluation, they had to augment the relevance assessments from old TREC data (Moffat et al., 1993). The TREC data only has relevance assessments for complete documents, so section- or page-based assessments had to be added. While doing this, they came across several documents that were marked relevant, but did not appear to contain any relevant sections; this is a further example that relevance assessment is not objective. In contrast to element retrieval, however, the retrieval units are not overlapping – the result list cannot contain both a section and a paragraph from that section.

Thus, although retrieval at the sub-document level has proved to be reasonably successful for the examined researchers, it is not without difficulties. One central problem is the unavailability of structural information; if the documents contain no easily parsable structure, the search system has to resort to kludges like page-based retrieval. With good XML- or SGML-structured documents, the retrieval system is relieved of that burden and can focus on choosing the best structural units as marked up by the document author.

2.3 Summary

This chapter covered the base technologies and methods that are relevant to this thesis: Starting with XML and related technologies as a suitable document format for digital libraries, going on to cover standard information retrieval basics for flat documents, this chapter has covered the major aspects of searching XML. With the focus on content-oriented retrieval, we have then taken a look at INEX, which provides an evaluation testbed to determine the quality of retrieval results.

The next chapter will give an overview of general aspects of XML retrieval and existing approaches to element retrieval.

3 XML retrieval

"I don't think I ever drove faster, but the others were there before us."

(A Scandal in Bohemia)

The previous chapter provided the foundations, but so far, the core topic of this thesis – XML retrieval – has not been discussed. This chapter discusses the combination of XML and information retrieval and describes selected approaches to searching XML.

3.1 Information retrieval on XML

Although the traditional information and data retrieval techniques could be used without modification for XML retrieval by ignoring the markup and indexing only the text of the root element, there are several reasons for developing extended retrieval methods:

- As mentioned before, it is more user-friendly to return only the elements of interest to the searcher, instead of making him wade through complete documents.
- In XML, data-centric and document-centric parts are frequently combined in one document, so the search engine should provide mechanisms for combining both kinds of constraints, as well as support constraints on the structure itself (for example, "return bibentry elements about information retrieval").

Traditional information retrieval provides no good approach to this. This section gives an overview of existing approaches for searching XML.

XML documents consist of two distinct types of data, content (text nodes) and structure (the structure of the document as reflected in the nesting of elements and attributes). These types of data are different in nature, so it is reasonable to use different types of constraints for each of them: Structural constraints specify conditions on the XML structure of the results, for example, searching for all author elements. Content constraints specify conditions on the textual contents of the elements, similar to what web search engines provide: Find all elements about relational databases. These types of constraints are (like the document types) not well-separated. For complex information needs, users may want to express both structural constraints and content constraints in the same query, for example, "in all articles written by Date, find the elements about relational databases".

Natural-language queries are not accessible to the computer, so the searcher has to specify his query in a special language with a clear and easily parsable syntax. Based on different requirements, many languages for searching XML have been specified and implemented. In the remainder of this section, I will give an overview of these languages and their semantics.

3.1.1 Data retrieval languages

XML data retrieval languages are all languages that do not provide convenient support for term-based retrieval (although they may provide substring search). First of all, the query languages specified by the World Wide Web Consortium (W3C) must be mentioned: XPath and XQuery.

XPath (Berglund et al., 2007) is a language for selecting nodes or values from the abstract tree of an XML document. As the name implies, *path expressions* play an important role in XPath; the searcher may use them to freely navigate the tree structure by using *axes* to obtain a node's parent, siblings, attributes, and so on.

The first part of the XPath query in figure 3.1a, //author/last-name, specifies which nodes to retrieve, in this case, the last names of authors. It is followed by a condition that the last-name node must satisfy in order to be retrieved; it must have a preceding sibling first-name with the given textual contents (preceding-sibling is the axis along which should be searched).

XQuery 1.0 (Boag et al., 2007) embraces XPath and supplements it with functionality for constructing XML documents containing the query results. Queries follow the basic pattern For – Let – Where – Return (FLWR), and the syntax is inspired by SQL in order to lower the barrier to entry. XQuery was designed to have more than one syntax binding, and so an XML syntax called XQueryX (Melton and Muralidhar, 2007) was derived from the grammar. It is intended mainly for programmatically created queries (it is too verbose to be written by hand).

XSLT (Kay, 1999), was developed in parallel to XQuery by a different W3C working group. Although it is primarily a transformation language used for rendering XML documents, it can also be used as a query language. XSLT was inspired by functional programming languages, so it is completely different in style compared to XQuery. It can output both XML and text files, which makes it convenient to convert XML documents to non-XML formats, for example, a data-centric document could be converted to SQL insert statements for importing the data into a relational database.

Figure 3.1 shows an example query for each of the aforementioned query languages.

3.1.2 Data and information retrieval languages

The data languages provide limited full-text operators, mostly substring search. For text documents, substring search is not sufficient. On the other hand, the need for combining data and information retrieval capabilities is so obvious that all the languages presented here not only provide facilities for searching the content, but also support structural and data-retrieval constraints to varying degrees.

```
//author/last-name[preceding-sibling :: first-name = "Berthier"]}
```

(a) XPath

```
<result>
{
for $a in doc("booklist.xml")/booklist/book/author
let $lastname := $a/last-name
where $a/first-name = "Berthier"
order by $lastname ascending
return $lastname
}
</result>
```

(b) XQuery

```
    <xqx:stepExpr>
        <xqx:filterExpr>
            <xqx:functionCallExpr>
              <xqx:functionName>doc</xqx:functionName>
              <xqx:arguments>
                  <xqx:stringConstantExpr>
                      <xqx:value>booklist.xml</xqx:value>
                  </xqx:stringConstantExpr>
              </xqx:arguments>
            </xqx:functionCallExpr>
        </xqx:filterExpr>
    </xqx:stepExpr>
```

(c) XQueryX (excerpt; the original version has 75 lines)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:template match="/booklist">
        <result>
            <xsl:apply-templates select="book/author[./first-name = 'Berthier']"/>
        </result>
    </xsl:template>
    <xsl:template match='author'>
        <last-name><xsl:value-of select="./last-name"/></last-name>
    </xsl:template>
</xsl:stylesheet>
```

(d) XSLT

Figure 3.1: Example queries in data retrieval languages. The aim is to find the last name of all authors whose first name is "Berthier" in the XML document from figure 2.1. Note that the queries are somewhat more complex than necessary in order to highlight the respective features.

Trotman and Sigurbjörnsson (2005) created *NEXI*, a query language supporting both structural constraints and typical keyword queries, for the Initiative for the Evaluation of XML Retrieval (INEX; see section 3.2 for details). NEXI is based on XPath, but reduces it to a core feature set and adds an about function for specifying keyword-based information needs. The query in figure 3.2a searches the book list from figure 2.1 for authors named Ricardo who have written books about information retrieval. The first condition about the summary specifies *support elements*, that is, elements that are not directly returned to the user, but are still relevant for the query, whereas the second condition specifies *target elements*, the elements that should occur in the result list.

NEXI is intended as the input query language for all participating search engines, so its exact semantics are unspecified. In particular, it is up to the implementors how to tokenize, index, and search the documents, no model or similarity measures are prescribed. In fact, even the structural conditions are not necessarily strict constraints.

A more detailed overview of NEXI will be given in section 3.2.5, in the context of INEX.

The W3C recognized the need for information retrieval capabilities beyond simple substring search, so XPath and XQuery are in the process of being extended with full-text retrieval functionality (Amer-Yahia et al., 2006). *XQuery Full-Text* is based on the notion of words and positions; the central new operator is ftcontains, which provides for ranked retrieval based on simple keyword queries. The query in figure 3.2b illustrates the extensions. It has roughly the same semantics as the NEXI query from above, but, in this case, the condition that the author's name is Ricardo is a strict filter, and ranking is purely based on the degree the book's summary matches information retrieval. NEXI does not give this level of control to the searcher, there is no way to specify that a condition must be satisfied.

XQuery Full-Text does not specify exactly how the input documents are to be tokenized, other than that attributes are *not* tokenized at all. Apart from splitting the input into words, the tokenizer also assigns word, sentence, and paragraph positions, but these, too, are left unspecified, so there is considerable leeway for the implementation. Scoring too is completely at the discretion of the implementation; all the specification dictates is that the scores are between zero and one, and higher scores indicate a higher level of relevance.

The query languages described so far are not meant for ad-hoc use: they have a complex syntax that is hard to master; even posing a syntactically correct query can be challenging for novice users. The question is how one can at the same time keep the expressive power and reach a user-friendliness rivaling that of web search engines. A first step in this direction is to use XML itself as the query language, that is, searchers create an XML document that represents their information need. This is a form of query by example; Carmel et al. (2003) implemented this under the name of XML fragments. Figure 3.2c gives an approximation of the example query in this format.

Although queries using XML fragments are easier to grasp than XPath or XQuery queries, they still require that the searcher knows about XML and is willing to produce a well-formed document that matches the collection schema. The next

```
//book[about(.//summary, "information retrieval")]//author[about(., Ricardo)]
```

(a) NEXI

```
for $a in /booklist/book/author[. ftcontains "Ricardo"]
let score $s := $a/summary ftcontains "information retrieval"
order by $s descending
return <result score="{$s}">{$a}</result>
```

(b) XQuery Full-Text

```
<book>
        <author>Ricardo</author>
        <summary>information retrieval</summary>
</book>
```

(c) XML fragments

We are searching for authors named Ricardo who have written books whose summary is about information retrieval.

(d) Natural language

Figure 3.2: Example queries in several data and information retrieval query languages. The aim in each case is to find authors named Ricardo who have written a book about information retrieval, but due to restrictions in the expressiveness of the languages, the exact semantics of the queries vary between the formulations.

logical step to overcome this hurdle is the use of natural language queries (Geva and Sahama, 2005). Figure 3.2d gives a conceivable example query; note that the supported syntax is completely unspecified, so it is possible that the existing parsers fail to understand this query.

This query expresses complex structural constraints without using a special query language, so it should be more suitable for high-precision ad-hoc retrieval. Internally, the natural-language query is translated into NEXI for retrieval, so it is not currently possible to express information needs that are too complex for NEXI.

Overall, it seems unavoidable that significant developments will happen on query language design; XQuery will likely be the dominant language for internal use, and if natural language queries keep their promises, they are a good alternative for end-user interaction.

3.2 The Initiative for the Evaluation of XML Retrieval

Evaluating the effectiveness of information retrieval methods has long been an important part of this research area, as witnessed by the long running TREC workshop. There was a workshop on XML and information retrieval as part of SIGIR 2000 (Carmel et al., 2000) (and this was not the first work done on element retrieval), but assessment of the methods only became possible with the start of the INEX workshops. *INEX*, the Initiative for the Evaluation of XML Retrieval, was started in 2002 (Gövert and Kazai, 2002) and funded by the DELOS Network of Excellence in Digital Libraries[1]. Since then, it has been a yearly event with a stable number of participants. According to the INEX 2002 homepage[2], the aims of the initiative are as follows:

> The aim of this initiative is to provide means, in the form of a large testbed (test collection) and appropriate scoring methods, for the evaluation of retrieval of XML documents. The test collection will consists of XML documents, tasks/queries, and relevance judgments given by (potential) users with these tasks. Participating organizations contribute to the construction of this test collection in a collaborative effort to derive the tasks/queries and relevance judgments for a large collection of XML documents. The test collection will also provide participants a means for future comparative and quantitative experiments.

Indeed, this aim was reached in the following years by providing collections of XML documents – IEEE Computer Society Journals published between 2002 and 2005 and a converted version of Wikipedia (Denoyer and Gallinari, 2006) in 2006 –, and the participants provided the topics (information needs with associated queries) and manual relevance assessments of the submitted results. Although it has not yet reached the number of participants, document collections, and queries that TREC has, it is on the way there.

In addition to the evaluation part, INEX also provides a forum for introducing and discussing all kinds of XML retrieval research at the yearly meeting and since 2005 in a mid-year workshop. Here, the participants present and discuss their retrieval approaches as well as theoretical aspects such as the discussion of the aims of information retrieval and evaluation measures.

3.2.1 INEX tracks

As of 2006, INEX has the following tracks that focus on different aspects of XML retrieval (Malik et al., 2006, 2007):

Ad-hoc track: This track focuses on the evaluation of retrieval engines for both content-only (CO) and content-and-structure (CAS) queries. This track is the main focus of this thesis.

Interactive track: This track aims at testing XML retrieval systems with real users and finding out what these users expect. In controlled experiments, the users are asked to perform certain tasks along the lines of finding information about a given topic. Their interaction with the retrieval system is recorded, and they fill in a questionnaire; this provides information about what users really want – or do not want – concerning XML retrieval systems.

Use case track: A thought experiment on how XML retrieval systems might be used in the real world.

[1] http://www.delos.info/

[2] http://qmir.dcs.qmw.ac.uk/INEX/

Multimedia track: This track combines text search on XML with search on other media, in particular images. The XML fragment including the image might provide further information about its contents, which helps supplement the limited image analysis methods.

Relevance feedback track: This track explores the use of relevance feedback in XML retrieval. It is executed in two phases: First, each participants submits an initial run, which is then manually assessed, and this feedback is distributed to the participants. In the second phase, the feedback can be used to improve retrieval results.

Document mining track: This track provides a forum for researchers attempting to extract information from XML documents. In the first executions of this track in 2005 and 2006, the task was to cluster the collection.

XML entity ranking track: The aim of this track is to find certain entities of a predefined type that match the query. For example, if the entities are actors and the query is "action", one suitable result might be "Arnold Schwarzenegger".

Natural language processing track: Instead of using the queries formulated in NEXI, the participants in this track use the free-text description from the topics (which is meant to express the same information need) and try to translate it into NEXI. The aim is to support a query language that supports structural constraints without having to resort to complex syntax.

Heterogeneous collection track: In the base collections of INEX, all documents come from the same source, so they adhere to the same schema. In the real world, documents are often from multiple sources with different schemas. This track aims at evaluating systems that can cope with this situation.

The ad-hoc track is definitely the dominant track with the highest number of participants, and the only one that has been running since the very start. Many of the other tracks suffer from a lack of participation and good test collections and, in fact, many of them were not completed in the INEX time frame.

I will only look at the ad-hoc track, because it fits my aims and has the most extensive collection of test data. Considering that I also aimed at working on heterogeneous documents with different schema, one might wonder why I am completely ignoring the heterogeneous track. Unfortunately, this track has not been active in 2005 and 2006, and before then, the test collections used were mostly based on data-centric XML documents in the form of bibliography entries. The main focus of this thesis is more on document-centric XML documents with smaller data-centric parts, so the available test data is not suitable for evaluating my methods.

3.2.2 Ad-hoc tasks

The ad-hoc track is not a single task, but it is split into several sub-tasks that simulate various possible retrieval scenarios and user interfaces.

For the *thorough* task, there are no restrictions on the submissions; in particular, it is permissible to return nested elements without penalty (a run could, for example, contain both a section and a paragraph from that section). The *thorough* task is an element-based task, that is, each result from the runs is an isolated element. It may well happen that the result list contains a mix of elements from various documents, for example, the highest-ranked result might be from document A, followed by results from other documents, and then another result from document A. The aim is to retrieve as many good elements as possible, without regard to how they are presented to the user – interactive studies have shown that users do not like to see overlapping elements (Kazai et al., 2004). An interesting aspect of this task is that most INEX participants base all their submissions for the other sub-tasks on their *thorough* results; for the *focused* task, for example, all overlapping elements must be eliminated. The *thorough* task is the only task that has been run in every instance of INEX since its inception in 2002.

In contrast to the *thorough* task, the *focused* task is meant to be a user-oriented task, that it, the runs could be presented to searchers without further postprocessing. For this reason, submissions for this task must not contain overlapping results. This implies that the search engine has to make a choice whether it should prefer a single large element or return several of its children as separate results.

In contrast to the previously-described tasks, *relevant in context* is a task where the results are grouped by document. This means that, if the first result is from document A, all other results from this document must immediately follow it, before any results from other documents occur in the result list. Within each document, the results are ordered by relevance. The intention is that the user is first and foremost directed to the most relevant documents and then gets to see which parts of the documents are the most relevant. The relevant-in-context task was introduced for INEX 2006; for INEX 2005, there was the vaguely similar *fetch-and-browse* task.

Best in context is a task that seeks to find the single best entry point for each document. This entry point should be the point at which the user should start reading the document.

Then, of course, there is the distinction between the content-only and the content-and-structure task, which differ in which of the query formulations in the topics should be used.

3.2.3 Test collections

The test collection consists of three major parts (Malik et al., 2006):

- The *document collection*, from which the results are retrieved,
- the *topics*, which specify an information need in natural language and as a NEXI query, and
- the *relevance assessments*, which are created by manual inspection of the submitted search results for determining if they satisfied the information needs from the topics.

The document collection typically remains stable for several years, whereas the topics and the corresponding relevance assessments are new for each round of INEX.

A *topic* consists of several parts: a brief *description* of the information need, the *title* and/or *castitle* that express the same information need as NEXI queries, and the *narrative*, a detailed description of the information need and notes on what is and is not relevant. The search engines use the titles – or, for natural language queries, the description – as the queries, whereas the narrative is later used by the assessor to determine whether a given element is relevant.

All submitted results for a topic are pooled, and a human assessor inspects them for relevance. For INEX 2002 to 2004, the assessment interface was based on elements, and for each element, the assessor had to determine to what degree the element satisfied the information need (*exhaustiveness*) and to what degree it only contained relevant information (*specificity*). Starting with INEX 2005, a new assessment interface was introduced where assessments are done on two phases:

1. The assessor highlights the highly specific parts of a document with a tool resembling a text marker. Now, the highlighting is not restricted to element boundaries, for example, it is possible to mark a single sentence in a long element as highly specific. Depending on the fraction of highlighted material, the specificity is automatically calculated.

2. The assessor assigns an exhaustivity value to all elements that contain any highlighted portion.

For INEX 2006, the process was even more simplified by omitting the elicitation of exhaustivity; relevance is purely based on specificity.

3.2.4 Document collections

So far, INEX has used two document collections, IEEE (from 2002 to 2005) and Wikipedia (starting in 2006). The characteristics of these collections could not be more different: The IEEE collection is a collection of IEEE journal articles, whereas the Wikipedia collection is a conversion of Wikipedia to XML format.

The IEEE collection has a clear focus on computer science, and the articles are rather similar in length (several pages).

As an encyclopedia, Wikipedia covers a broad range of topics, and the length of the articles varies a lot, from simple redirection pages telling the user little more than "the topic is ambiguous, you might mean either of these pages" to long articles on the history of a place.

For both collections, the average length of an element rapidly decreases the deeper the element is in the document. In the IEEE collection, the articles have a mean length of more than 2000 words (excluding stop words), but the average length on level four is already below 50 words (with a median below 20). Although the maximum depth is 20, the statistics imply that by far most relevant elements should occur on the first few levels; the low-level elements are simply too short. In the Wikipedia collection, for each level down from level three, 75 percent of the elements are less than 20 words long; the IEEE collection goes to level six. It is reasonable to assume that retrieval results should have a minimum length to be useful to the searcher, which implies that results can only be found on the first few levels. The levels below are (at best) useful for providing further retrieval hints or for verifying content-and-structure queries directed at these elements.

In both collections, the documents are rather homogeneous: All have the same root element name, and on the second level, there are only five (IEEE) and three (Wikipedia) different element names. It is on this level that the main text of the documents is enclosed in a single element, bdy (IEEE) and body (Wikipedia). From the third level on, the full richness of the schemas is explored in full, with element names denoting anything from inline mathematical formulas to sections. For the search engine, it is a challenge to automatically determine the best result granularity; to a human, it may be obvious that the additional metadata that is available in the article outside the body, for example the article's title and authors, may be useful even if it does not contain the query terms – but to the search engine, it will usually be pointless to include material without any matches. It may help to hard-code rules like "if a bdy element is to be returned, return the corresponding article element instead", but this is not practical for collections with diverse schemas.

The IEEE collection uses 178 distinct element names (the DTD defines a few more, but they are unused in the given collections), and the Wikipedia collection uses 1003 distinct element names, 90 percent of which are used at most ten times in the collection (most of these element names are the result of conversion errors; the Wikipedia markup has less than 100 directly expressed types of markup).

Overall, the difference between the two collections that most affects retrieval is size: On the one hand, the Wikipedia collection is larger as far as the total number of documents and the total amount of text are concerned, but the IEEE collection contains longer articles on average. Keeping in mind that the foremost aim of element retrieval is to return useful parts of documents, IEEE is arguably the more interesting collection for evaluation; many of the extremely short Wikipedia articles do not lend themselves to being subdivided. Unfortunately, even the IEEE articles are not a perfect example for performing sub-document retrieval, because the sections and paragraphs usually do not stand on their own.

3.2.5 NEXI

As mentioned before, NEXI (Trotman and Sigurbjörnsson, 2005) is the query language for the INEX topics. Its syntax is derived from XPath, but significantly simplified and extended with the information-retrieval-specific about function.

The simplest valid NEXI queries are keyword queries, resembling the queries for typical web search engines. For example, information retrieval is a valid NEXI query that instructs the search engine to retrieve results about this topic. Each query term is a sequence of alphanumeric characters (with optional hyphens and apostrophes, as long as they do not occur at the start of the word) or positive or negative numbers. Furthermore, the following hints are possible:

Phrases: A phrase is a sequence of words enclosed in double quotes ("information retrieval"); this indicates that the searcher expects those terms to occur adjacent to each other in relevant documents.

Must-have terms: Terms or phrases can be preceded by a + sign to indicate that they are expected to occur in a relevant element. For example, information +retrieval conveys the intention that the term information may or may not occur in relevant elements, whereas the term retrieval is definitely expected to occur.

Should-not terms: Terms and phrases can also be preceded by a - to signify that they are expected not to occur. For example, apple -computer shows that the searcher is not interested in computers, but rather in fruits.

In any case, none of these hints are binding for the search engine, it may choose to ignore them. For example, it might be useful to break a phrase its component terms if the phrase does not occur in any document.

Content-and-structure queries provide limited support for the XPath descendant axis only (in its abbreviated form //).

- //T[t]: Return elements with paths matching T that are about t.
- //S[s]//T: Return T descendants of S, where S is about s (this form has been deprecated since INEX 2003).
- //S[s]//T[t]: Return T descendants of S, where S is about s and T is about t.

The target path T and the support path S are paths, and t and s are filters. Paths consist of an alternating sequence of descendant axis qualifiers // and path components, where a path component is either a tag name, @ followed by an attribute name (may only occur at the end of a path), the wildcard *, which matches any tag, or a set of tags like (a|b|c).

Filters are content-only queries, arithmetic conditions, or Boolean expressions of filters that should apply only to the preceding path. They are of the form about(*relative-path, co-query)*, where relative-path is either a single period . – indicating the current node – or a period followed by a path. The query co-query is of the same form as the content-only queries from above.

Arithmetic filters are a relative path followed by one of the operators <, >, =, <=, or >=, followed by a numeric constant. For example, //article[.//fm//year > 2005] requests articles that contain a value greater than 2005 in a //fm//year element.

Boolean expressions of filters are sequences of filters linked by the Boolean operators and and or, possibly using parentheses to force a certain priority. For example, the following query requests articles that have a value greater than 2005 in a //fm//year element and contain p elements that are about ski jumping, snooker, or both:

```
//article[.//fm//year > 2005 and (about(.//p, ski jumping) or about(.//p,
snooker)]
```

Like content-only queries, the filters in content-and-structure queries are only hints, so the search engine can ignore any part of them. The structural constraints, on the other hand, may be binding, depending on whether the results are for the strict or the vague content-and-structure task.

3.3 Probabilistic and language-modeling approaches

Section 3.1.2 has shown that most XML query languages do not specify exact retrieval methods, they only specify the general intent. One reason for this is that, XML retrieval being a young area of research, it is not yet clear which retrieval method is the most suitable one for different scenarios.

3.3.1 Okapi-based

Okapi BM25 (see section 2.2.2) is one of the most successful retrieval models for traditional text-based information retrieval, so it is not surprising that several research groups based their search engines on this model.

Lu et al. (2006) use an extension to basic BM25 which applies field weighting, called BM25F (Robertson et al., 2004). This is further extended in a straightforward manner for element retrieval, yielding a variant called *BM25E*.

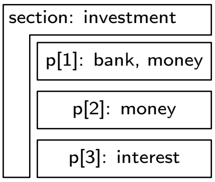
XIRQL (Fuhr and Großjohann, 2001) uses term frequency to estimate the probability of relevance for a given node; although in principle other methods could be used, the current version is based on normalized Okapi BM25 weights.

The most important extension lies in the partitioning of the XML document into non-overlapping index nodes based on element names. The choice of which element names to index also determines which element names can be returned to the user; for example, inline elements are typically not indexed, so they can never be retrieval results. Indexing is done bottom-up, and the index entries for parent elements do not include the text that occur in indexed children. For example, if chapter and section are chosen as element names to index, and a chapter element consists of a title element followed by several section elements, the index entries for the chapter only contain the text from the title (which is not itself indexed); see figure 3.3. This ensures that a single term occurrence does not contribute to two separate probabilistic events.

XIRQL uses a term's weight for estimating the probability that this term is relevant for the index node it occurs in. Based on this assumption, it is then possible to derive the probability for higher-level elements that span several index nodes by starting from the event of a term occurrence, where matches for the same term in several index nodes are linked with *or*, whereas the different term matches are linked with *and*. The events can then be turned into disjunctive normal form, and the probability of this new formulation can be calculated using the inclusion-exclusion formula:

```
<section>
    <title>Investment</title>
    <p>bank, money</p>
    <p>money</p>
    <p>interest</p>
</section>
```

(a) XML document



(b) Indexing units

Figure 3.3: Example of XIRQL indexing. In this case, section and p are the indexing units. Note that the indexing unit of the section element only contains the text that is not in other indexing units. For probability estimation, the information from all four nodes must be combined.

$$P(C_1 \vee \cdots \vee C_n) = \sum_i (-1)^{i-1} \left(\sum_{1 \leq j_1 < \cdots < j_i \leq n} P(C_{j_1} \wedge \cdots \wedge C_{j_i}) \right) \tag{3.1}$$

Assuming independence of the events, one can obtain the probability of a conjunction by multiplying the probabilities of the underlying events:

$$P(C_1 \vee \cdots \vee C_n) = \sum_i (-1)^{i-1} \left(\sum_{1 \leq j_1 < \cdots < j_i \leq n} \prod_{j_1 \leq k \leq j_i} P(C_k) \right) \tag{3.2}$$

Consider the example from figure 3.3, given a query “bank money interest” on section elements. To save space, I will refer to the indexing units of the section and p elements as S and P_1 to P_3. The search engine needs to obtain the probability of relevance for S, but S has several children, so their scores must be combined: P_1 contains the terms “bank” and “money”, P_2 contains “money”, and P_3 contains “interest”. Thus, the events are (P_1, bank), (P_1, money), (P_2, money), $(P_3, \text{interest})$, and the following overall conjunction results:

$$e = (P_1, \text{bank}) \wedge ((P_1, \text{money}) \vee (P_2, \text{money})) \wedge (P_3, \text{interest}) \tag{3.3}$$

Obtaining the probabilities of these events and rearranging the terms into disjunctive normal form, we get:

$$\begin{aligned} P(e) \quad = \quad & P((P_1, \text{bank}) \wedge (P_1, \text{money}) \wedge (P_3, \text{interest})) \vee \\ & P((P_1, \text{bank}) \wedge (P_2, \text{money}) \wedge (P_3, \text{interest})) \end{aligned} \tag{3.4}$$

Assuming independence of the events, the final probability of index node A being relevant is as follows:

$$\begin{aligned} P(e) \quad = \quad & P((P_1, \text{bank})) \cdot P((P_1, \text{money})) \cdot P((P_3, \text{interest})) + \\ & P((P_1, \text{bank})) \cdot P((P_2, \text{money})) \cdot P((P_3, \text{interest})) - \\ & P((P_1, \text{bank})) \cdot P((P_1, \text{money})) \cdot P((P_2, \text{money})) \cdot P((P_3, \text{interest})) \end{aligned} \tag{3.5}$$

Now that the probability has been broken down to the probabilities of single-term events, the probabilities (term weights) can be used to obtain the final score for the section element.

In addition, XIRQL allows the searcher to specify weights (that is, probabilities) for the query terms. To support this, further events for the query terms have to be introduced: (q, money), (q, bank), $(q, \text{interest})$. Assuming that the events are disjoint, that is, $P((q, x)) \wedge P((q, y)) = 0$ for all query terms x and y, one needs to *and* the query event's probability with the corresponding index node events.

Beyond the capabilities of NEXI, XIRQL supports vague predicates for certain data types; for example, if the searcher specifies 2000 as the value of a year element, values from 1995 to 2005 might all be acceptable results.

3.3.2 Language-modeling approaches

XML retrieval approaches based on *language models* extend the models to deal with the tree structure.

The retrieval model of TopX (Theobald, 2006; Theobald et al., 2006) is based on Okapi BM25 (Robertson and Walker, 1994), a probabilistic retrieval model derived from the 2-Poisson probabilistic model. The score is not normalized, so TopX performs an additional indexing-time normalization step (Theobald, 2006, pp. 77–78).

TopX extends Okapi BM25 to content-oriented XML retrieval by using different frequency values (Theobald et al., 2005): Term frequency on the document is replaced by the term frequency for a single element, and document frequency is replaced by element frequency. The element frequency $\text{ef}_A(t)$ of term t in elements with name A is the number of elements with name A that contain t. In principle, this is equivalent to basic Okapi BM25 with each element of type A indexed as a separate document.

This, given the query Q searching elements of name A for terms $t_1, \ldots, t_n$ – //A[about(., t1, . . . tn]) in NEXI notation –, the element frequency $\text{ef}_A(t_i)$, the total number of elements with name A, we get:

$$\sum_i \frac{(k_1 + 1)\,\text{tf}(e, t_i)}{K + \text{tf}(e, t_i)} \log\left(\frac{N_A - \text{ef}_A(t_i) + 0.5}{\text{ef}_A(t_i) + 0.5}\right) \tag{3.6}$$

$$K = k_1 \left((1 - b) + b \frac{\sum_{t \in \text{content of } e} \text{tf}(e, t)}{\text{avg}_f \left\{\sum_t \text{tf}(t, f)\right\}}\right) \tag{3.7}$$

Like for Okapi BM25, k_1 and b are constants; Theobald reports best results for $k_1 = 10.5$ (the Okapi default is 1.2) and $b = 0.75$ (this is the Okapi default). It is also conceivable to use different constants for different target element types A, but this has not yet been explored.

TopX is optimized for the efficient retrieval of the top k results.

Sigurbjörnsson and Kamps (2006) use a multinomial language model that combines the language models for the element, for the corresponding document, and for the whole collection to obtain the element's language model:

$$P(w|e) = \lambda_e P_{\mathrm{MLE}}(w, \theta_e) + \lambda_d P_{\mathrm{MLE}}(w|\theta_d) + (1 - \lambda_e - \lambda_d) P_{\mathrm{MLE}}(w) \tag{3.8}$$

The smoothing parameters λ_e and λ_d determine the influence of the different language models. For INEX 2005, they experimented with various indexing units, and the parameters differ dramatically. Furthermore, for some runs, a prior probability of an element proportional to its length is used, because element length affects relevance dramatically (Kamps et al., 2005):

$$P(e) = \frac{\mathrm{len}(e)}{\sum_f \mathrm{len}(f)} \tag{3.9}$$

The final probability of element e for the query Q can then be calculated from the prior probability of the element and the probabilities of relevance for the query terms w:

$$P(e|Q) = P(e) \prod_{w \in Q} P(w|\theta_e) \tag{3.10}$$

Ogilvie and Callan (2003, 2005) estimate the language models of leaf elements based on their text and the language models of inner elements by taking the linear interpolation of the language models of its children and the language models of its remaining text (that is, the text that occurs outside of the child elements in mixed content). They also estimate the *prior probability* that an element is relevant given its properties like type and length, without regard to the query terms. This information is obtained from old relevance assessments on the same collection, so it may not be a realistic way of improving retrieval quality. This is incorporated in the retrieval model by inverting the conditional probability and ranking based on the probability of the document given the query.

The maximum-likelihood estimator for term w given the observed text T and the language model θ_T is:

$$P_{\mathrm{MLE}}(w|\theta_T) = \frac{\mathrm{tf}(w, T)}{\mathrm{len}(T)} \tag{3.11}$$

Using only the maximum-likelihood estimator does not yield good results in all cases, because the sample of text is often too small to obtain a good probability distribution. Because of this, *smoothing* is performed by also considering external language models, for example from the complete document collection. Smoothing can be done using linear interpolation:

$$\begin{aligned} P(w|\theta) &= \sum \lambda_i P(w|\theta_i) \\ & \sum \lambda_i = 1, \lambda_i \geq 0 \end{aligned} \tag{3.12}$$

Ogilvie and Callan (2005) take the tree structure of the XML document into account when calculating the language models. Estimation of relevance for a given query term w first goes bottom-up, then top-down in the following steps:

1. To obtain the model of an element e, the first step is to obtain a model based on the text in the element, excluding text from sub-elements (that is, only text *children* of the element are considered). Given <p>The weather is <i>beautiful</i>, my friend.</p>, the language model for the p element is based on the text fragments "The weather is" and ", my friend.".

$$P(w|\theta_e) = (1 - \lambda_e^u) P_{\text{MLE}}(w|\theta_e) + \lambda_e^u \underbrace{P(w|\theta_{\text{type}(e)})}_{\text{collection-level background model}} \tag{3.13}$$

2. In addition to the maximum likelihood estimate, a collection-level background model is used. This background model can vary, depending on the type of element it is applied to (type(e)).

   Then the intermediate model θ'_e is calculated bottom-up by using the children's models and the original model of e:

$$P(w|\theta'_e) = \lambda_e^{c'} P(w|\theta_e) + \sum_{f \in \text{children}(e)} \lambda_f^c P(w|\theta'_f) \tag{3.14}$$

   The smoothing parameters λ^c are calculated based on the length of the corresponding elements. Given len(e) as the length of the text in element e, not counting the text that occurs in child elements and desclength(e) as the sum of the lengths of all descendants of e, we get:

$$\lambda_{e_i}^{c'} = \frac{\text{len}(e_i)}{\text{len}(e_i) + \text{desclength}(e_i)} \tag{3.15}$$

$$\lambda_{e_j}^{c} = \frac{\text{len}(e_j) + \text{desclength}(e_j)}{\text{len}(e_i) + \text{desclength}(e_i)} \tag{3.16}$$

   Ogilvie and Callan also hint at the possibility of setting the parameters depending on the child element's types, giving the example of increasing a title element's weight.

3. In the last step, the parent element's language model is incorporated in the final probability estimate (this is called *shrinkage*):

$$P(w|\theta''_{e_i}) = (1 - \lambda^p)\, P(w|\theta'_{e_i}) + \lambda^p P(w|\theta''_{\text{parent}(e_i)}) \tag{3.17}$$

Based on the probability estimates for single terms w, a formula for a complete query Q can be derived:

$$P(Q|\theta''_{e_i}) = \prod_{w \in Q} P(w|\theta''_{e_i})^{\text{tf}(Q,w)} \tag{3.18}$$

For INEX 2004, Ogilvie and Callan also incorporated length-based prior probabilities of relevance, where the probability of relevance is greater for longer elements (either linear, square, or cubic).

3.4 Vector-space-based approaches

Section 2.2.1 has already described the vector-space model for traditional information retrieval. Although it is more ad-hoc in nature than the probabilistic models, it is simple to understand and implement and delivers good results, so it stands to reason that it was adapted to XML retrieval. Several participants of the INEX evaluation initiative (see section 3.2) use the vector-space model as the basis for their retrieval engine; this section will focus on JuruXML and Gardens Point XML, two retrieval systems that have consistently been among the best.

3.4.1 Extended vector-space model

The approach of Crouch et al. (2006) is based on Fox' extended vector space model (Fox, 1985), which allows the incorporation of metadata in the vector model. The document collection is indexed at paragraph level, and the term vectors of the ancestors are derived from the vectors of the paragraphs by adding up the term weights.

3.4.2 JuruXML

Mass et al. (2002) presented their work on JuruXML at the first INEX workshop in 2002; this description is based on the version used for INEX 2004 and 2005 (Mass and Mandelbrod, 2005, 2006). The core idea is to create separate indexes for hand-selected core components; for INEX 2005, the core components were the structural elements from articles down to paragraphs. The search process goes as follows:

1. The search engine separately searches each index using a slight variant of standard tf-idf and the cosine similarity measure. The indexes are treated as separate vector spaces, so that term and document frequency are calculated

as usual *for each index*. The similarity in this step is calculated almost as in equation 2.3:

$$\mathrm{sim}_1(q,d) = \frac{\sum_{t_i \in q} d_{t_i} q_{t_i}\ \mathrm{idf}(t_i)}{||q|| \cdot ||d||} \tag{3.19}$$

The norms $||q||$ and $||d||$ are the number of unique terms in the query respectively in the document. The term weights defined as follows, where N is the total number of components (analogously for the query term weight q_t):[3]

$$d_t = \frac{\log(1 + \mathrm{tf}(d,t))}{\log(1 + \mathrm{avg}_{t_i \in d}\{\mathrm{tf}(d,t_i)\})} \cdot \log\left(\frac{N}{\mathrm{df}(t)}\right) \tag{3.20}$$

2. Next, the search engine applies automatic query refinement based on lexical affinities (Carmel et al., 2002) to improve precision without hurting recall. The elements retrieved in the first step are re-scored based on terms that occur in the elements close to the query terms.

3. The scores from the different indexes are not directly comparable, because they are based on separate vector spaces with different numbers of documents and average document lengths. Thus, the scores are normalized by dividing by the score of the query compared to itself (as a document): $\mathrm{sim}_1(q,q)$.

4. Finally, each element's score is then adjusted by using a document pivot; if s_e is the element's original score, s_a is the score of the corresponding document, and p is a document pivot value between zero and one, the new score is calculated as follows:

$$s = p \cdot s_a + (1-p) \cdot s_e \tag{3.21}$$

The result of this retrieval process is the *thorough* result, which is then filtered to obtain the *focused*, *relevant in context*, and *best in context* results.

For CAS results, the NEXI queries were automatically translated into queries in the XML fragments query language (Carmel et al., 2003). Depending on the sub-task – whether support and/or target elements should be treated as hints or requirements –, the translation was slightly different.

3.4.3 GPX

At the same workshop, Geva (2002) presented Gardens Point XML (GPX). The core idea is to only index nodes with text content (misleadingly called "leaf elements", although elements with mixed content are also meant) and derive the scores of their ancestors from the retrieval scores of the "leaf elements". Ancestors of the

[3]Note that the published version of this formula uses $\log(\mathrm{tf}\ldots)$ instead of $\log(1 + \mathrm{tf}\ldots)$, which would mean that query terms that only occur once do not contribute to the similarity. Private conversation with Yosi Mass confirmed that this is an error in the paper.

leaf elements are not indexed, but may still be retrieved. A manually assembled list of element types that are considered too small to be presented to the user are omitted from the index (in particular, inline markup like <i> for italics). For the new document collection used for INEX 2006, GPX was updated slightly to accommodate mixed-content elements (Geva, 2007). Retrieval is executed in the following steps:

- Standard retrieval on the indexed elements is performed. The score of the leaf elements is calculated using a variant of tf-idf that drastically boosts the score if several query terms occur in the same element:

$$L_0(e) = K^{\mathrm{coord}(q,e)-1} \sum_{t \in q} \frac{\mathrm{tf}(e,t)}{\mathrm{cf}(t)} \tag{3.22}$$

  n is the number of query terms that are matched in this element, and K is the boosting constant that, according to Geva (2006), can be anywhere from 5 upwards without much effect on retrieval quality. $\mathrm{cf}(t)$ is the *collection frequency* of term t, that is, the number of times it occurs in the document collection – note that this is *not* identical to document frequency, which counts multiple occurrences of a term in a document as one. For example, if a term occurs twice in document d_1 and once in document d_2, the document frequency is 2, but the collection frequency is 3.

- The score of the leaf elements is propagated up in the result tree to obtain scores for the elements that are not indexed, but can be returned. If a parent element has several relevant children, it is assumed to be more exhaustive, so its score is calculated by adding the scores of its children, multiplied by a decay function $D(n)$:

$$L(e) = L_0(e) + D(n) \sum_{c \in \mathrm{children}(e)} L(c) \tag{3.23}$$

  $L_0(e)$ is the original score of element e, $L(c)$ is the score of child c with a non-zero score (which has been calculated before), and $D(n)$ is a decay factor that depends on the number of children with matches. Geva (2006) gives values from 0.49 for a single relevant child) to 0.99 (for more than one relevant child) for this function, but the runs actually submitted to INEX use different values, for example, $D(n) = 0.1$ for all values of n. This indicates that this parameter is very important for fine-tuning retrieval quality. The role of the dampening function is to prevent over-scoring elements higher up in the tree – simple summation without dampening would lead to a parent's score always being at least as high as the children's scores.

- Finally, the root element's score is added to all its descendants' scores to reward elements from highly relevant documents. This is comparable to JuruXML's document pivot with $p = 0.5$.

Retrieval system	Model	Indexes	Overlapping
XIRQL	BM25	1	no
TopX	BM25	n	yes
GPX	Vector-space	1	yes
JuruXML	Vector-space	n	yes

Table 3.1: Overview of XML retrieval systems. "Overlapping" indicates whether the same term in a nested element can be counted for several index elements.

$$L = L_{\text{orig}} + L_{\text{root}} \tag{3.24}$$

For CAS queries, GPX adds the scores of support elements to the scores of the corresponding target elements. In order to fulfill strict requirements on the target elements, GPX performs filtering, that is, it discards all elements from the results that do not match the given target element path.

Both JuruXML and GPX ignore the small inline elements on the basis that they are not useful retrieval results. While this is undoubtedly true, they can still provide valuable hints about their ancestors, as the following chapter will show.

3.4.4 Document frequency

The vector-space retrieval systems make use of some form of document frequency of the terms. In traditional information retrieval, document frequency is defined as the number of documents a given term occurs in. In the case of XML documents, this simple definition may not be ideal, because the documents are no longer atomic; elements are indexed and retrieved as if they were documents. Due to element nesting, the same string in the XML document may be indexed under several elements – for example, given <sec>a <p>b</p></sec>, the term b occurs both in element /sec and in element /sec/p. Mass and Mandelbrod (2003) discuss that, if these elements are stored in the same index, this leads to a document frequency of 2 for term b, which is illogical, because the term only occurs once in the complete document. Counting the frequency at the XML document level may not lead to satisfactory results, either: In <sec><p>a</p> <p>a</p> <p>b</p></sec>, terms a and b have a term frequency of 1 for their respective sections, and they have a document frequency of 1. JuruXML addresses this by having different indexes and counting document frequency separately for each index. GPX has only one index, but since it does not place higher-level elements in the index, document frequency is not based on either complete documents or all elements.

3.5 Summary

This chapter provided an overview of the core ideas of XML retrieval and how it differs from traditional information retrieval. A selection of existing state-of-the-art retrieval methods was described; the field is still young, so there are many different methods without any one being superior to the others.

The following chapters will cover my contributions to XML retrieval, mostly focusing on content-only retrieval, but also briefly touching content-and-structure retrieval.

4 Base retrieval engine

"In that case we should have to commence our investigation from a fresh basis altogether."

(The Adventure of the Cardboard Box)

Currently, there is no accepted baseline for XML retrieval; entirely different approaches yield results of comparable quality, there is no approach that is clearly better than the others. The existing retrieval engines of the INEX participants are generally too complex to be reimplemented in a reasonable time frame without mistakes (and in many cases, the published descriptions are clearly incomplete or even wrong). Furthermore, they typically incorporate various orthogonal techniques to improve retrieval quality, from phrase matching to the exploitation of structure. Although the INEX results show that these search engines can provide good results, the exact reason is hard to isolate.

For this reason, I created a search engine based on well-established information retrieval methods, with minor changes to apply them to XML element retrieval. At first sight, this might seem like a questionable decision, but as the evaluation in chapter 7 will show, this search engine is comparable or even superior to the search engines of the INEX participants, provided the parameters are chosen well.

First, section 4.1 discusses the design goals and assumptions about the documents and users of the retrieval system. Next, sections 4.2 and 4.3 describe the retrieval engine that will be the base for the extensions discussed in chapter 6 and the baseline to compare against.

4.1 Assumptions and design goals

4.1.1 Assumptions

Although I strive to minimize assumptions about the documents, it is impossible to completely avoid them. The following assumptions are the same assumptions that are implicitly made for almost all research on content-oriented XML element retrieval, in particular, for the INEX workshops:

Collection is document-centric: I assume that the documents in the collection are mostly document-centric; they may have small parts that are data-centric (typically metadata about the publisher and related information), but the main contents is contained in free-text sections.

XML structure reflects logical structure: I assume that the structure of the XML document, as represented in the DOM tree, reflects the structure that the document author intended for the text. For example, if a text is modeled as book—chapter—section, there should be section elements inside chapter elements inside a book element. This is the case for DocBook, the IEEE collection, and the Wikipedia collection, for example, but XHTML does not abide by this.

Fragments of documents are useful retrieval units: This is the central assumption behind element retrieval in general; if parts of the documents do not stand on their own, it makes little sense to present them to the user. This assumption is somewhat weakened for the "in context" tasks, where the fragments are displayed in the context of a document, but even in this case, it is highly desirable for the user to pick and choose the relevant parts of the document.

Only the text content is relevant for retrieval: The search engine ignores all attributes and their values, because the text they contain is typically not presented to the user when the document is rendered (this is the case for both the IEEE and the Wikipedia document collection used at INEX). If the search engine retrieved a result where the query terms only occur in an attribute value, and thus in text that is not displayed, the user would certainly be confused.

Spamming is not an issue: In uncontrolled environments like the World Wide Web, malicious document authors may tune their documents so that they get ranked higher, even if they are not as relevant (Schwartz, 1998). I assume that this is not the case, that is, that the authors strive to inform the readers, not to game search engines. In a collection of published books, this is assured by the editorial process.

A basic assumption about the user of an XML element retrieval engine is that he is a specialist in the subject area of his query, at least to the degree that he can understand an isolated part of a document.

Although there are many possible retrieval tasks, this thesis will only address the ad-hoc thorough retrieval task in order to keep the scope manageable. The result for the other tracks and tasks, like *focused* and *in context* are typically derived from the thorough results, so any improvement of the thorough results is likely to also benefit the other tasks. Preliminary work (Dopichaj, 2006b) has shown that a straightforward implementation of focused retrieval can lead to good results.

These points are assumed for all parts of the thesis. If further assumptions are made for specific parts of the research, I will mention them where they are needed. Note that the list of assumptions does not include a detailed knowledge of the document schema.

4.1.2 Design goals

For the search engine developed for this thesis, the following design goals were given:

- *All* elements should be indexed, in order to be able to exploit these elements to improve retrieval quality as well as being able to answer all types of content-and-structure queries.
- Nevertheless, the index size should be reasonable – it can be expected to be significantly larger than for document-based retrieval, but a naïve approach would yield too big an index.
- The index structures should (mostly) be disk-based, in order to enable indexing and searching large collections.
- The search engine shall retrieve the smallest elements that still satisfy the user's information need (the *FERMI principle*; Chiaramella et al. (1996)).

The first goal in particular needs some rectification: As seen in the previous chapter, most existing XML search engines do not index extremely small elements, like italicized phrases. This has the advantage of taking less space for the index and (as a consequence) less time for retrieval processing, but there is a significant drawback. If the small elements are not indexed, they are not available for query processing, so not all queries can be answered; for example, a content-and-structure query might ask for all sections with "information retrieval" in their title. If this information is not stored in the index, it becomes completely infeasible to process this query.

Even for content-only queries, it may be useful to index small elements: They can provide important information about their ancestor elements, as the following section will show.

4.2 Interpreting XML

In traditional information retrieval, documents are flat, so they can be seen as a sequence of words. In XML retrieval, matters are not that clear anymore: The markup helps to structure the document, but it may also provide challenges for tokenizing; furthermore, some parts of the documents are not useful for retrieval, so they should not be included in the index.

4.2.1 Tokenization

One important question is how to determine token boundaries. The pattern-based tokenizers for flat text can be reused in this context, but the introduction of elements complicates matters: In many cases, whitespace between elements may only be used for formatting the input document and has no further significance. In other cases, however, it may be significant, for example in ASCII art or poems. In general, this cannot be deduced from the document alone, although its schema may provide this information (but it need not do this, so that all processors of the document must handle this themselves). Should the tokenizer consider all tags token boundaries? Neither "yes" nor "no" is an answer that is applicable in all cases. On the one hand, data-centric parts of XML documents can omit whitespace between the fields represented by the elements because it has no semantic significance. <first-name>Ricardo</first-name><last-name>Baeza-Yates</last-name>, for example, should be parsed as two tokens, Ricardo and Baeza-Yates, and not as one, RicardoBaeza-Yates, although that would be the logical text content of the enclosing element according to the XML standards. On the other hand, sometimes it may be useful to use markup within words, which of course should be regarded as a single token in this case. If the tokenizer does this, however, it violates the assumption that the index terms of an element should also contain all index terms from the sub-elements. Due to the inherent problems with the second approach and the rare occurrence of intentional in-word markup, it may be preferable to always consider tags as token boundaries.

4.2.2 What to index

Not every part of an XML document is worth indexing; comments are not visible when the document is displayed and should not contain any information, so they should be omitted from the index. Likewise, processing instructions are only present for technical reasons, to give hints for specific processors of the document, so they too should not be indexed. Entities like &copyright; should be indexed as their replacement text, because they are merely a form of macro substitution; in the rendered form, they are not visible anymore.

It is more debatable whether element and attribute names should be represented in the index; on the one hand, they are not part of the data, but on the other hand, they may well be represented in the rendered output of the document in some way. For example, the author element might be prepended by the string "Author:" in the rendered document. If this term does not occur in the index, searchers may well be surprised that some strings cannot be found although they are displayed – and elements that match the query terms may be displayed with the matching parts omitted. Unfortunately, what exactly is displayed depends not only on the XML document, but also on the stylesheet, and different stylesheets might display slightly different data; at best, one could index the XML documents with regard to a certain stylesheet. In practice, however, this is not likely to cause major problems, because most of the relevant data is contained in text nodes, so the mismatch is small.

Another problem is the handling of attribute values: Should they be indexed like the regular text of the element, separately, or not at all? As seen before, XQuery Full-Text specifies that attributes are not indexed at all, but that may not be the best approach in all circumstances. Under the assumption that attributes contain metadata, it seems reasonable to index them in such a way that the corresponding element is not found if the query terms occur only in its attribute values, but only if the attribute is directly addressed in the query.

The text nodes contain the main content of XML documents, so they should definitely be indexed. Since the basic unit of retrieval is the element, the elements must be represented in the index. On the other hand, not every element is a useful retrieval result, for few searchers would be content with a list of two-word inline elements that exactly match their query. Thus, the search engine must filter out unsuitable elements, either at indexing time or when the retrieval results are prepared for presentation. The choice of which elements are possible results is crucial to good retrieval quality.

4.3 Retrieval model

This section will discuss the design decisions for my search engine based on the material presented in the preceding chapter. For this thesis, I will use the vector-space model and the closely related BM25 model. INEX results have shown that vector-space search engines are among the top-performing search engines, so I can assume that this choice is reasonable. The search engine described in this thesis is a minor adaptation of the standard information retrieval techniques to XML retrieval – as the evaluations in the next chapter show, the results are competitive.

In any case, in the context of document-based information retrieval, Zobel and Moffat (1998) have executed experiments that indicate that there is no single best method for retrieval: They compared 720 variations of similarity measures, including BM25, the cosine similarity measure, and many unnamed variants. For evaluation of retrieval quality, they used several TREC test collections. They come to the following conclusions:

- There is no single best method that works in all scenarios.
- It is not even possible to identify parameters of the application scenario in which a given measure works well.
- The results using different evaluation measures are not consistent – a similarity measure that works well up to rank 20 may have a bad top-rank performance.
- The results are sensitive to minor changes in the formulas, such as which logarithm base is used.

This implies that I have to make a somewhat arbitrary decision about which similarity measure to use for my search engine (the resources that are necessary to perform a similar large-scale experiment are too great). I chose to examine the BM25 measure, which is a benchmark for document-based information retrieval, and the Lucene measure, which proved to be successful for my INEX 2005 submissions (Dopichaj, 2006b).

Thus, the index provides a mapping from each term to the elements that contain this term. Furthermore, it contains metadata about the document's original structure so that it is possible to give the exact position of the results in the form of (simplified) XPath, like /book[1]/chapter[2]. (This path is needed for the INEX evaluations; in realistic scenarios, the search engine will use a numeric identifier to directly retrieve the corresponding XML fragment from an XML database.)

Much research has been invested in standard document-based information retrieval, so I – like many other search engine authors – will reuse standard information retrieval models and index structures with adaptations to provide XML-specific enhancements.

As a baseline to compare my results with, I use a standard information retrieval method. The recursive content of every element in the collection, from a complete book to a single highlighted word, is indexed as a pseudo-document. This index is then searched using a standard similarity measure such as BM25; the result is a ranked list of elements.

This baseline makes no use of the tree structure of the XML documents, so one should hope that improvements of the results are possible. To use the structure, the search engine must perform a few additional steps. During index construction, the structure of the documents is stored in the *metadata index*. Before the similarity measure is applied, the tree structure of the original documents is reconstructed from the results and the metadata index; thus, the similarity measure can now not only work on a single element, but also on other elements that are related to the element. After the similarities have been calculated, the document trees are again broken apart and the elements are ranked according to their individual similarities. See figure 4.1 for the complete retrieval process.

The search engine is currently a content-only system, so it ignores the content-and-structure parts of NEXI. Furthermore, each query is treated as a set of words; phrases and should-have terms are treated just like all other terms, and terms that should not occur are simply ignored (that is, results containing these terms are not removed from the query).

Thus, the query (from topic 355)

```
+"Best Actress" +"Academy Award" -Supporting -nominated winner film
```

is interpreted as

```
Best Actress Academy Award winner film.
```

Naturally, the query terms are subject to the same stop words and tokenization as the indexed documents.

4.3.1 Base similarity function

As a baseline, I will use a search engine that indexes all elements as if they were documents and then uses a similarity function from flat information retrieval with as few adaptations as possible. Chapter 7 will show that this simple approach can yield surprisingly good results, provided the length normalization function is adapted to be more suitable for XML retrieval.

In document-based information retrieval, length normalization already plays an important role and many different variants have been developed (Lee et al., 1997). For element retrieval, this normalization becomes even more important: With elements as the retrieval units, the range of typical result content lengths is much wider

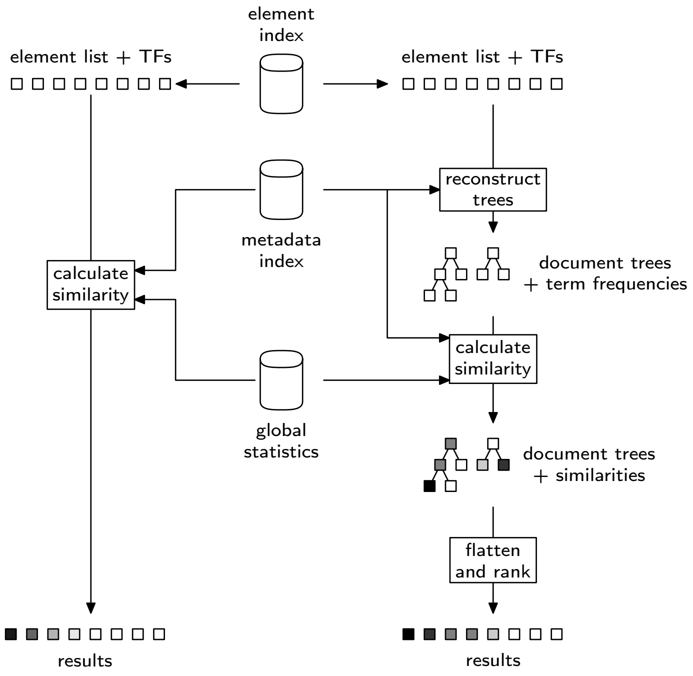


Figure 4.1: Retrieval process. On the left, the base retrieval process is shown, on the right, the conceptional retrieval process with structure-based similarity is shown. The shading of the nodes reflects the similarity value.

than for document-based retrieval – an element can contain little text, for example, an italicized phrase, or it can be a complete article. Thus, the length normalization functions typically used in document retrieval should not be used without change for element retrieval. Kamps et al. (2005) show that it is not sufficient to simply omit elements shorter than a given threshold from the index; instead, the search engine should use an extreme length prior. On the other hand, GPX shows that a search engine can yield good results only using a cutoff and no explicit length normalization, so it is unclear whether the findings can be generalized from the language-modeling approach they use to other methods.

Using a cutoff at indexing time – that is, omitting elements shorter than the cutoff – is not possible if extremely short elements are needed for improving retrieval quality, like the enhancements described later in this chapter. Thus, I will focus on the length normalization functions. Lucene's length normalization function, $1/\sqrt{\mathrm{len}(d)}$ favors very short documents extremely; this is not a problem for document retrieval, where such short documents simply do not exist. In element retrieval, however, the elements can be only a few words long, but these elements are not useful retrieval results. To avoid this, I will modify the length normalization function as follows, where k is a parameter that determines where the switchover from constant to decreasing occurs:

$$\mathrm{lnorm}(e) = \begin{cases} \frac{1}{\sqrt{k}} & \text{if } \mathrm{len}(e) \leq k \\ \frac{1}{\sqrt{\mathrm{len}(e)}} & \text{otherwise} \end{cases} \tag{4.1}$$

Okapi BM25 has two parameters k_1 and b that affect the influence of the length normalization. With these parameters, it is possible to tune the length normalization; the default parameters were determined on TREC data, which has different characteristics than XML documents, so different combinations are probably needed.

4.3.2 Term weighting

The issue of global term weights in XML retrieval is still largely unsolved. In flat information retrieval, the term weights are typically determined based on the document frequency of the terms on the grounds that this value is correlated with the discriminatory power of a term (a term that occurs in almost all documents does not help to retrieve relevant documents).

It is difficult, however, to adapt this to element retrieval, as Mass and Mandelbrod (2003) discuss: They argue that instead of document frequency, a *component frequency* should be used, that is, the number of elements that contain a given term. Due to the nesting of elements, this can lead to surprising anomalies. A term that occurs at a deeper level in a document will get a higher component frequency than a term that occurs only at a higher level. This in turn implies that the "deeper" term gets a lower weight, which is far from intuitive. Mass and Mandelbrod argue that document frequency does not have straightforward characteristics, either, because it ignores that elements (and not documents) are the unit of retrieval. They go on to propose the use of element-name-specific counting; they partition the index by element type and determine the component frequency *per sub-index* – for example, the index structures record that the term dog occurs 19,451 times in elements of type sec and 87,978 times in elements of type paragraph. This approach leads to more bookkeeping effort for the search engine, but it is doubtful whether this leads to improved retrieval quality.

Instead, I will contrast document frequency with *element frequency*, that is, the number of elements that contain a given term in their recursive text content. Comparing element frequencies with document frequencies, I can see that the relationship is roughly linear for the lower-frequency terms (up to a document frequency of about 7,000). In this range, the element frequency is about ten times the document frequency, except for a few outliers. Above this limit, the relationship appears to be exponential, but there are too few data points to confirm or reject this for sure. In any event, considering that there are about 17,000 documents, the terms for which the relationship is not linear are unlikely to be useful for retrieval because of their low discriminatory power (they occur in more than 40 percent of the documents). The outliers in the lower range are special-purpose terms like "fig" (for references to figures) and numbers; these, too, are unlikely to be useful query terms.

It might be, however, that although these terms are useless at the document level, they are useful at the element level; this will be investigated in chapter 7. In any event, one advantage of element frequency over document frequency is that element frequency works even if there is no one-to-one correspondence of XML documents to logical documents. For example, the distributed version of the IEEE collection contains all articles from one volume of a journal in a single XML document – document frequencies will clearly be misleading in this case, because the logical documents in this case are the articles. (For INEX, each article is assumed to be in a single document, so this does not play a role.)

4.3.3 Full similarity calculation versus reconstruction

My baseline does not make use of the structure of the XML documents, so more advanced methods are needed. For vector-space or BM25 retrieval, two approaches have proved to be successful at INEX: JuruXML creates separate indexes for different types of elements; this is only feasible if there are few element types in the collection, and even then only the upper levels (for example, article—section—subsection) are typically included because of the overhead this implies. GPX has a similar shortcoming: it indexes only the elements at or below a certain level that partitions the document. For example, one might choose to only index sections, because all body text is contained in the sections (there is no text between sections); the missing upper levels of the documents can be reconstructed at retrieval time by combining the retrieval status values (RSVs) of the missing elements.

Since the aim is to be able to retrieve *all* elements, even from the lowest levels (see figure 4.2), neither of these approaches can be used without modification. Adapting the JuruXML approach to create separate indexes for *all* element types is completely infeasible. In principle, the GPX approach can be extended to index all elements; indeed, with the modifications introduced in Geva (2007), this would be straightforward.

A disadvantage of GPX's approach, however, is that the full term frequency

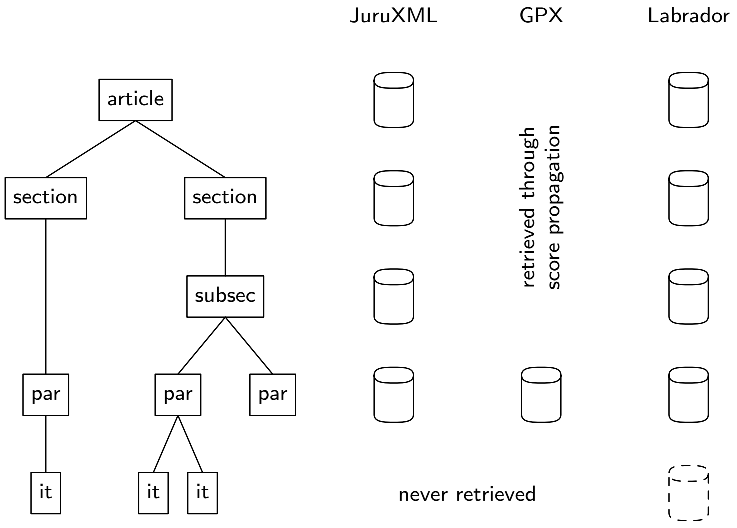


Figure 4.2: Indexing strategies for XML retrieval based on the vector-space model. Only the par and it nodes in the XML tree have text children; the it inline markup elements are not indexed and can never be retrieved. JuruXML has an index for each level and combined the results, whereas GPX only indexes nodes with text children and propagates the scores upwards. Labrador, the system described in this thesis, indexes *all* elements, but it does not retrieve the it elements – it only uses them to adjust the scores of ancestors.

information is not available for the inner nodes – in particular, the number of co-occurring terms, which leads to a high boost, is lost, which can lead to significant differences because the leaf-level similarity function rewards co-occurring terms dramatically. For example, searching for a and b in the document `<x>a b</x>` will lead to the following score for element /x:

$$K^1 \left(\frac{\mathrm{tf}(d,a)}{\mathrm{cf}(a)} + \frac{\mathrm{tf}(d,b)}{\mathrm{cf}(b)} \right)$$

If the structure is `<x><y>a</y> <y>b</y></x>` the score is lower although the effective contents is the same:

$$D(2) \left(\frac{\mathrm{tf}(d,a)}{\mathrm{cf}(a)} + \frac{\mathrm{tf}(d,b)}{\mathrm{cf}(b)} \right)$$

$(K \geq 5,\ D(2) < 1)$

This problem would be even more pronounced if even small elements were indexed; due to fragmentation, the boosting of co-occurring terms would hardly ever be applied.

It is possible, however, to achieve a comparable index size reduction if a difference-based indexing approach as presented in chapter 5 is used and the term frequencies are reconstructed before the similarity function is applied. This might be a bit more expensive at runtime – after all, now the term frequencies have to be stored in main memory even for the top-level nodes –, but the ability to retrieve all elements is worth this minor cost. Thus, my search engine is based on the availability of full term frequency information for all elements; it is the same as the baseline search engine with additional modifications to exploit the structure of the XML documents.

4.4 Summary

This chapter has described the design goals that form the basis for the further research that will be presented later in this thesis. To better isolate the influence the effects of the retrieval enhancements that will be described later, a simple adaptation of BM25 to XML retrieval was introduced to provide a base retrieval result. This base search engine will also deliver the baseline to compare the enhancements against – as the evaluation in chapter 7 will show, this is a reasonable choice.

5 Implementation of the base retrieval engine

Sherlock Holmes sat moodily at one side of the fireplace cross-indexing his records of crime.

(The Five Orange Pips)

For obvious reasons, it is infeasible to do a full-text scan of all documents whenever a query is posed to the retrieval system; to get acceptable performance, the search engine needs index structures that facilitate quick access to all relevant documents.

This chapter describes and analyzes various optimizations of XML full-text indexes based on previous work. In particular, it will describe difference-based storage of inverted lists for the XML documents and suitable storage of metadata about the elements; the main idea is to omit data that can easily be derived from other data that is stored in the index (Dopichaj, 2007c).

- The search engine supports retrieval of *all* elements in the collection, even very short ones that are frequently omitted from indexes. The aim is to reduce the size of the index significantly without negatively affecting retrieval time or retrieval quality.
- The focus of this chapter is not retrieval quality (this will be addressed in a later chapter), but index size and retrieval speed, so the evaluation is strongly biased towards the latter two. The index structures are flexible enough to support a variety of retrieval models and similarity measures; the current implementation supports BM25 and Lucene similarity measures as described in the preceding chapter, but it could be adapted to other approaches like language modeling easily.
- The chapter provides a detailed analysis of the implications of the space-saving index structures and the various trade-offs between space and time.

Although it is an interesting problem in its own right, this chapter does not address the combination of full-text with structural retrieval (so-called *content-and-structure* retrieval). In this form of retrieval, the user also specifies structural constraints like "the following keywords should appear in the bibliography". The indexes hold all information that is needed to evaluate such queries, but direct access to elements with specific structural properties is not possible, so a straightforward extension would probably lead to bad performance.

5.1 Existing XML index structures

Many of the optimizations that were proposed for atomic-document retrieval are not applicable for element retrieval without modification, because they assume that each document is independent of each other document.

Luk et al. (2002) classify and give an overview of many different indexing options for XML as of 2002; most of these are not applicable to XML element retrieval because they imply a data-centric view. *Segment-based indexing* divides the document into segments that typically do not overlap; extensions support nested document parts (as used in XML documents), but these extensions only support retrieval of the root element. For *tree-based indexing* methods, the index structures reflect the tree structure of the original XML documents.

Most of the researchers working on XML retrieval pay little attention to space or time savings; a commonly used approach is to store the indexes in an SQL database (Geva, 2006; Theobald, 2006).

The GPX retrieval engine (Geva, 2006) only stores nodes with text children in the index, but does not attempt to obtain the correct term frequencies for their ancestors. Instead, it calculates their retrieval score based on a completely different formula than that for the leaves, by simply summing the children's scores and applying a dampening factor. Although evaluation results indicate that this model works well in the tested scenarios, it does not support the retrieval of small elements at all – they are simply not indexed on the grounds that they are no useful retrieval results –, so if a query demands that some text occurs in a title element, GPX's index structures cannot answer this query. The core idea is close to segment-based indexing, but through the use of score propagation to reconstruct higher-level elements, GPX can return elements of various granularities.

Most of the research on index structures optimized for semistructured data goes back to the early days of XML or even SGML. The idea of calculating the term frequency values of inner nodes from the term frequency values of their children is not new. Shin et al. (1998) use this approach (but they do not discuss mixed-content elements). Their storage structure for the inverted lists is not particularly space-efficient: For each entry in the inverted list, they store the document ID, a unique element ID that also encodes the position in the document, the element's level in the document's DOM tree, and an ID of the element type. This storage format leads to a lot of redundancy, as it is repeated for each term in a given element.

Furthermore, although the idea of a simple element ID that can be used to calculate the parent's ID using a simple formula is nice, their implementation is simply not practical for arbitrary document collections. They assume a k-ary tree structure – that is, each element can have only up to k children. A fixed limit for the number of children is problematic, because it has to be rather large in order to accommodate all conceivable documents. If a single value is used for all documents, it would have to be large: For the INEX test collections used for evaluation, k would have to be 1023 (IEEE collection) respectively 8503 (Wikipedia collection). Even if k is determined separately for each document, new problems arise: Because k has to be known to determine the parent's ID, it has to be stored somewhere for each document, either redundantly in the inverted list or in a separate index structure.

This implies that the element IDs get large, and they do not lend themselves to delta encoding, so they take up much space in the index. The element type is obviously redundant, and, although the authors state that they use a favor a length-based normalization for scores, they do not mention where this is stored; either it too is part of the index, in which case it is another redundant number in the index, or it is stored separately in an unspecified place.

Lee et al. (1996) present a space-saving full-text index structure for structured documents. The basic idea is to reduce storage requirements by not storing term occurrences that can be derived from other index nodes. Unfortunately, their index structures only support boolean queries, so they do not provide for term frequencies. Furthermore, they do not take into account that non-leaf nodes may also contain text in mixed content, so their index structures are not applicable in all circumstances.

Myaeng et al. (1998), too, ignore mixed-content elements for their index structures. Furthermore, they store the complete element type information for each term occurrence; this is a significant waste of space. Jang et al. (1999) and Shin et al. (1998) also describe difference-based indexing, but they store element-level metadata redundantly in the inverted list.

5.2 Index structures

The index structures shall support simple keyword queries, that is, queries that comprise a set of words. The aim of the retrieval engine is to retrieve all elements in the collection that contain at least one of the query words and then rank the elements based on a similarity function.

For determining the candidate elements and determining the similarity to the query, the following information is needed for effective and efficient retrieval (this is mostly based on standard work in information retrieval (Witten et al., 1999)):

- Inverted lists, containing references to the elements along with the corresponding term frequencies.

- Metadata, for example information about the positions of the elements in their documents, typically represented by XPaths.

- The lexicon, containing information about the terms, including document frequencies and the pointers to the inverted list.

Furthermore, the similarity function may need additional information; the BM25 function also needs the length of an element to calculate its RSV; this, too, will have to be stored in the metadata.

Using these index structures, the search engine can efficiently answer a query. The query q is composed of a set of query terms $t_1, \ldots, t_n$. For each query term t_i, the search engine must perform the following steps:

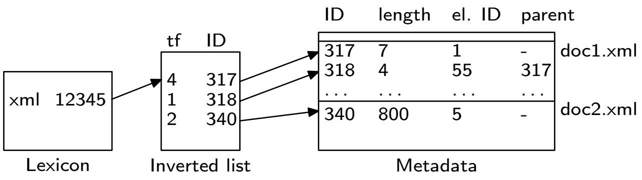


Figure 5.1: Index structures. For each query term, the lexicon is consulted to find the corresponding entries in the inverted list file, which is then read to determine the candidate elements and their term frequencies. Using the metadata, the elements are grouped by document, and the final similarities are calculated.

1. Retrieve the lexicon entry for t_i. This provides us with the term's weight and the entry point in the inverted list file.

2. Retrieve the inverted list, starting at the entry point from the lexicon.

3. Determine the tree structure of the document from the inverted list entries and the metadata.

4. Update the term frequencies based on the tree structure.

After this, the standard process for calculating the similarity of each element can be used, and XML-specific postprocessing operations can be applied to improve the ranking (see the next chapter). Figure 5.1 illustrates the index structures that are needed to support the retrieval process.

The remainder of this section will describe these aspects of XML indexing.

5.2.1 Tokenization

Before looking at the index structures themselves, I will discuss *tokenization*, because it has important implications for the index structures. In order to index a text, the indexer must first break it into *tokens*, typically based on words. These words are the entry points to the inverted lists.

For tokenization, the parser considers every sequence of Unicode letters and digits as a term. Additionally, all tags in the input are considered token boundaries; for example, the XML fragment <fn>John</fn><ln>Doe</ln> is parsed as the two tokens John and Doe, *not* as the single token JohnDoe.

As later sections will show, this is important for difference-based storage, because it ensures that all ancestors of an element e include at least as many instances of the terms contained in e as e itself. Formally, for all elements e with the respective parent parent(e) and all terms t, with $\mathrm{tf}(e,t)$ the number of occurrences of t in e, the following equation holds:

$$\mathrm{tf}(\mathrm{parent}(e),t) \geq \mathrm{tf}(e,t) \tag{5.1}$$

```
<au><fnm>Gary</fnm><snm><ref>Yngve</ref></snm></au>
```

(a) From the document's metadata (attributes omitted for clarity). This should be parsed as Gary Yngve (not GaryYngve).

```
<!ELEMENT au (cny|cty|ead|pc|san|sbd|str|url|aff|appel|deg|fn|fnm|ref|role|snm)*>
```

(b) Corresponding DTD entry, with references expanded. No #PCDATA occurs, so this is element content.

```
<au>
   <fnm>Gary</fnm>
   <snm><ref>Yngve</ref></snm>
</au>
```

(c) The element from 5.2a, reformatted to be more human-readable. This should be parsed in the same way.

```
<sec><st>P<scp>revious</scp> W<scp>ork</scp></st>
```

(d) From the body. This should be parsed as Previous Work (not P revious W ork).

```
<!ELEMENT st (#PCDATA|tmath|...|fig)*>
```

(e) Corresponding DTD entry, with references expanded, abridged. Character data is allowed inside this element, so this is mixed content.

Figure 5.2: Two fragments from tg/2002/v0346.xml. Tags should be seen as token boundaries in the first case, but not in the second case.

Because of mixed content (that is, sub-elements embedded within text), the term frequencies of the parent may be greater than the sum of the term frequencies of all its children:

$$\mathrm{tf}(e,t) \geq \sum_{f \in \mathrm{children}(e)} \mathrm{tf}(f,t) \tag{5.2}$$

Although this assumption does not always model the intention of the document author – for example, it fails if inline markup is used inside a word, like <i>high</i>light –, it works in the majority of cases and is frequently made in XML retrieval research. Usually, tags as token boundaries work well for data-centric parts of documents, for example, document metadata; see figure 5.2a for an example. Data-centric segments are characterized by the absence of mixed content; the data is split into fields, and the fields are (at most) separated by whitespace.

It turns out, however, that always using tags as token boundaries does not work well for the IEEE collection; in particular, titles often use small capitals (scp elements) in the middle of words, see figure 5.2. This is particularly problematic for the enhancements described in the following chapter, because they assume that the words in titles are particularly relevant.

A straightforward approach would be to create a list of element names that should not be seen as token boundaries; this has the disadvantage that it is schema-dependent and that the same element name may be a token boundary in some cases, but not in others. In the presence of a DTD, validating XML parsers must inform the application about which white space occurs in *element content*, that is, elements that can only have element children (Bray et al., 2006b, section 2.10); this white space is assumed to be only for making the XML source code more readable, see figure 5.2c for an example. If a DTD is available, this information could be used, but it is troublesome to parse a DTD, and if no DTD is available, this cannot work.

Instead, the same idea can be applied to the document: If an element has element content (or just white space between its child elements), its child elements are considered token boundaries, otherwise they are not. This alone would violate the containment assumption, because the children may contain text that is not contained in the parent; in figure 5.2d, the scp children contain the tokens revious and ork, whereas the parent only contains Previous and Work. To avoid this, the search engine do not index the child elements (and their descendants) whose tags are not token boundaries. This has the drawback that these elements cannot be retrieved, but, under the assumption that the markup occurs in the middle of words, they are not suitable retrieval units anyway. If the schema and document authors follow conventions, this heuristic will correctly detect all element content, but it might in unlikely situations detect some content that should really be mixed content as element content; this will rarely occur in practice and can thus be neglected without ill effects.

The heuristic works well for the IEEE collection, DocBook documents, and other documents in schemas that follow widely adopted practices (that is, rendering elements does not introduce extra white space in mixed content). INEX 2006 and 2007 used a new document collection that was artificially converted from the original markup, the Wikipedia collection (Denoyer and Gallinari, 2006). The conversion suffers from several technical problems, among them the arbitrary insertion and removal of white space, which makes it impossible to make the right choice about using tags as token boundaries (Beigbeder, 2007). For this reason, tags are always considered as token boundaries for this collection.

After the text has been separated into tokens, I remove stop words and apply the Porter stemming algorithm (Porter, 1980) as implemented by the Snowball project[1].

5.2.2 Lexicon

The lexicon provides the entry point from the query terms to the inverted list. For each term that occurs in the document collection, it stores the document frequency of this term and a pointer to the position in the inverted list file where this term's inverted list starts.

While indexing, there is little choice but to store the lexicon in main memory: Documents typically contain thousands of words, and for each term, the indexer must determine whether the term is already in the lexicon (in this case, it uses the old ID for the inverted list). Otherwise, it assigns the next ID to the term and add it to the lexicon. Even with an efficient index structure such as a B-tree, this will require two to three seeks, which would increase indexing time.

[1] see http://snowball.tartarus.org/

While searching, the time to access the lexicon is not critical, because most queries only refer to less than ten words. Thus, it is possible to perform the search on a machine with less main memory than the machine used for indexing.

5.2.3 Inverted lists

A large part of the index is stored in the inverted lists; compared to traditional information retrieval, the size of a document in the index is much larger: It also needs to record information about *where* (that is, in which element) inside a document a given term occurs. A straightforward approach is to simply index the textual content of each element separately, but the cost is prohibitive (because of nesting, the size is several times the size of the document-level index).

Many search engines for XML retrieval do not index small elements up to a given length in tokens, but this does not alleviate the problem much; the size is still about six times as large as the document-level index. One should note that the authors of these search engines typically do not give index size as an argument in favor of omitting these elements. Instead, they argue that these elements are never good retrieval results that the users want. Although this may be the case for typical element retrieval scenarios, this alone is not a sufficient reason, as it is always possible to filter these elements from the results.

One should consider that some retrieval approaches use the small elements to improve the quality of the retrieval results, even if they are not returned to the user (see the next chapter). In addition, if the scope is broadened just a little to include content-and-structure queries, the small elements may be vital to achieve acceptable retrieval results: Content-and-structure queries often reference metadata elements like authors' names or titles. These metadata elements are by their nature short, and if they are not included in the index, it becomes impossible to answer such queries.

Thus, there are good reasons for including *all* elements in the index, if it can be ensured that this does not increase its size too much. The *difference-based* method I will describe in this section is still significantly larger than a document-level index, but this is unavoidable; at least it tries to minimize redundant storage of information.

The basic idea is to omit data that can be derived by using the tree structure of the XML documents. Thus, for an element e, only the term frequencies that result from the text nodes directly below e are stored, *excluding* text contained in child elements. Thus, for elements with child elements, the stored term frequency $\mathrm{stf}(t, e)$ of term t in element e is:

$$\mathrm{stf}(e,t) = \mathrm{tf}(e,t) - \sum_{f \in \mathrm{children}(e)} \mathrm{tf}(f,t) \tag{5.3}$$

For elements without children, the stored term frequency is the real term frequency: $\mathrm{stf}(e,t) = \mathrm{tf}(e,t)$. Figure 5.3 illustrates this.

It may happen that a term does not occur in text node children of e, but only in element children; in this case, the stored term frequency is zero, and no inverted

```
<section>
  <title>Inverted lists </title>
  <p>Inverted lists are an <emph>index structure</emph>.</p>
</section>
```

(a) Indexed document.

Term	Path	ID	tf	stf
index	/section	1	1	0
index	/section/p	3	1	0
index	/section/p/emph	4	1	1
inverted	/section	1	2	0
inverted	/section/title	2	1	1
. . .				

(b) Inverted lists for this document. Observe that the section element's entries can all be omitted.

Figure 5.3: Example of indexing.

list entry for this element is stored in the index. In the extreme case, if the element does not have text nodes as children, it will not occur in the inverted lists at all. This happens for higher-level structural elements, like chapters.

During retrieval, a postprocessing step is needed to reconstruct the original term frequencies from the stored term frequencies, as figure 5.4 shows.

Using this storage technique leads to significant savings in the number of entries in the inverted lists and, consequently, in the total size of the inverted lists. Although a postprocessing step is required after retrieving the term frequencies from the inverted lists, the cost of this step is offset by the reduced time needed to read the entries from disk.

5.2.4 Building the inverted lists

The document collections to be searched can get large, too large to be indexed in main memory alone. Thus, it is necessary to keep large parts of the data on disk while indexing.

5.2.5 Metadata

The lexicon and the inverted list only contain information about term occurrences, but not about the structure of the XML document. This is a deliberate design decision; many search engines use index structures that store the structural information in the inverted lists, but this unnormalized form of storage increases the size of the index.

Instead, the schema is normalized and only the IDs of the elements are stored in the inverted list. This alone is obviously not sufficient: At least, the search engine

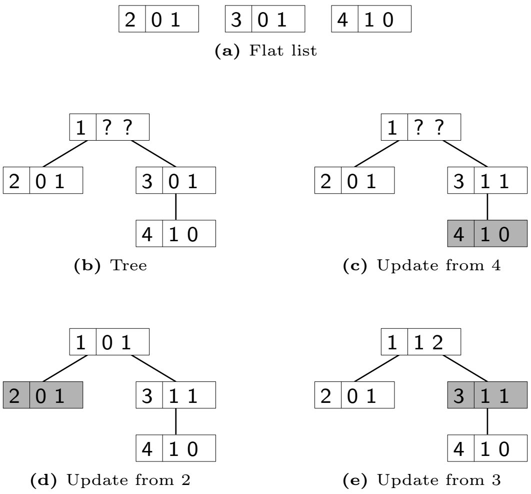


Figure 5.4: Reconstruction of term frequencies. Each box contains the element ID and the frequencies for "index" and "inverted". First, the stored term frequencies are retrieved as a flat list (a), next, the document structure is reconstructed from the metadata, including missing nodes (b). Finally, the tree is traversed bottom-up; for each node, the term frequencies are added to the parent's term frequencies (c,d,e).

needs to get the name of the document and the position of the fragment inside the document – typically an XPath expression – to display the results.

Even before that point, it may be necessary to obtain more detailed information. For example, if the search engine must avoid presenting nested elements, like a chapter and a section from that chapter, to the user in its result list, it needs data about results' parent–child relations, and exploiting small elements, as mentioned earlier, also requires this information. Many similarity measures, for example Okapi BM25, use the length of an element's text to adjust its similarity. And, of course, if only the differences of the term frequencies are stored for inner elements, the search engine must know the structure of the documents.

Thus, the search engine needs to store element-specific information in the index in a space- and time-efficient format. The simplest approach is to simply store the information per fragment ID:

- The structural information can be reduced to the ID of an element's parent (encoded as the difference to the parent's ID to save space).
- The element's length can be stored as-is.
- If the XPath is needed for identification purposes, the index must also contain the ID of the tag name.

The IDs of the elements themselves need not be stored, the metadata entries are simply stored in order, so that the ID can be derived from the position in the list. This is possible because the search engine must read the complete metadata for a document anyway to obtain the structural information.

A minor optimization is to apply the same technique used for the inverted lists and only store the difference to the child elements' lengths for a parent element. In this case, the search engine still has to keep elements whose text is completely contained in children in the metadata index, because the link to the parent's parent and the tag ID must still be available.

5.2.6 Index compression

At the level of actually storing integral values in files, space can be saved if some characteristics of the numbers to be stored are known: If most of the numbers are small, it is advisable to use a coding method that stores small values in fewer bits at the expense of using more bits than necessary for larger values that occur infrequently.

The implementation uses *unary coding*, which stores $n-1$ of one-bits followed by one zero-bit to store a number $n \geq 1$, γ *coding*, which breaks the number into an exponent part and a remainder: $n = 2^{\lfloor \log n \rfloor} + (n - 2^{\lfloor \log n \rfloor})$. The exponent is the stored as the unary code for $\lfloor \log n \rfloor + 1$, where $\lfloor \cdot \rfloor$ is the floor function, followed by the value of $n - 2^{\lfloor \log n \rfloor}$ in binary, in as many bits as needed. The δ *coding* uses the same breakdown of the number, but stores the exponent in γ coding instead of unary. Table 5.1 gives examples for the encodings; see Bentley and Yao (1976) and Elias (1975) for details on the δ and γ encodings.

n	Unary	$\lfloor \log n \rfloor$	$n - 2^{\lfloor \log n \rfloor}$	γ	δ
1	0	0	0	0	0
2	10	1	0	10 0	100 0
3	110	1	1	10 1	100 1
4	1110	2	0	110 00	101 00
100	11...110	6	36	1111110 100100	11011 100100

Table 5.1: Examples for the various encodings. For small numbers, unary encoding uses the least space, for large numbers, the situation is reversed. For 100, the unary encoding uses 100 bits, γ encoding uses 13 bits, and the δ encoding uses 11 bits.

It is customary to store the *difference* of the document IDs in the inverted lists instead of the IDs themselves; for example, if a term occurs in documents 2, 6, 7, and 24, the following values would be stored in the index: 2, 4, 1, 17. Assuming that terms occur in clusters, it is to be expected that the numbers are small in magnitude. This assumption has proved to be accurate for traditional information retrieval, and it is even more relevant for XML retrieval, for the following reasons:

- Term occurrences are more localized in XML retrieval if a consecutive numbering of elements is used, because all elements from the same document – whose IDs are close to each other – are typically about the same topic, so they contain similar terms.

- Furthermore, the term frequencies of elements that are deeper in the document tree tend to be extremely small in magnitude because the elements are quite short.

Depending on the expected distributions of values, different encoding functions can be used (Witten et al., 1999). With difference-based indexing, one can expect many term frequencies to be close to 1, so a unary encoding is most suitable. The differences of the document IDs are somewhat bigger, so the search engine uses a δ coding.

Much more data is read from the metadata index than from the inverted lists, so the decoding overhead of the bit-based unary, δ, and γ codings is more noticeable. These encodings are expensive because they require expensive shift and mask operations. Thus, I will also evaluate a byte-based encoding as an alternative: A sequence of bytes with the high bit set followed by a byte with the high bit clear forms the value; the lower seven bits of each byte are concatenated to obtain the value.

The encodings described above strongly favor small numbers, but the element name IDs are assigned in the order they occur in the document collection. Depending on the collection, the numbers can get quite large; the INEX Wikipedia collection, for example, has 1257 different element names. This will be even more pronounced if different collections with diverse schemas are stored in the same index.

	IEEE		Wikipedia	
	Baseline	Diff	Baseline	Diff
unary	204	47	659	151
γ	198	50	666	159
Huffman	189	47	631	151

Table 5.2: Storage sizes of the term frequencies in the inverted lists using different encodings, in megabits. Note that the numbers to not include padding between the runs; the Huffman figure excludes the size for the code table.

The first thing that comes to mind is to assign the element name IDs in decreasing order of frequency in the collection, instead of in sequence. This requires another pass over the metadata index at indexing time, but does not affect retrieval time. Although the global distribution may not be optimal in the general case – different documents may use different tags, depending on the author or the schema –, it should work for the test collections, both of which use a single schema with a limited vocabulary.

Arithmetic coding (Witten et al., 1987) cannot be used: When decompressing, it assumes that *all* data is read sequentially; here, however, the search engine must be able to read isolated runs from the inverted files without also reading all the preceding runs. Thus, it would only be possible to use arithmetic coding within a single run, but – because the runs are relatively short – it is unlikely to achieve a good compression ratio then.

Huffman coding (Huffman, 1952) is more realistic for this scenario, because it is based on a static model of the frequencies, so the same decoding table is used for all runs in the inverted file. The frequency information can be collected while writing the final merged run file, and then another pass is needed to copy this run file to the final file with the Huffman coding.

Experiments show that for the high-frequency values up to five, the Huffman encoding takes the same number of bits for storage. On the other hand, there are large gaps between extant term frequency values in the higher ranges; thus, using a fixed coding table or function, the encodings will use more bits than is necessary.

Table 5.2 shows that Huffman coding is indeed the most effective encoding for the term frequencies, as expected. The differences can be significant for the baseline – 20 percent for IEEE and 5 percent for Wikipedia –, but for the difference-based encoding, it is virtually indistinguishable from the unary encoding. This behavior can be explained by the extreme bias towards low term frequency values in the difference-based index.

Thus, there is little benefit in using Huffman for encoding the term frequencies, but the additional overhead while indexing suggests the use of the simpler unary encoding.

For compressing the metadata index, the situation is different: Every entry references an element name ID, and the element names appear with drastically different frequencies in the collection; for example, by far the predominant element name in the IEEE collection is it (for "italics"), followed by p (for "paragraph"). Together, the ten most frequent element names (out of 178 distinct element names in total) account for more than 50 percent of all occurrences. The distribution is even more skewed for the Wikipedia collection, where the ten most frequent element names (out of 1257) account for more than 80 percent of all occurrences.

Obviously, a suitable encoding can save a lot of space. A first approach could be to order the element name IDs by decreasing frequency and then use one of the above-mentioned fixed encoding functions like unary. This, however, would be far from optimal at least for the Wikipedia collection, where 788 element names occur only once. Because of the assumption of a fixed distribution, unary coding compresses the first few ranks very well, but fails miserably for the lower ranks, whereas γ encoding works reasonably well for the very top and the very bottom, but uses too many bits in the middle of the range. Huffman encoding can save about 8 percent in this case.

5.3 Evaluation

To show that the index structures are indeed useful, I will evaluate its properties on standard test collections. As mentioned before, the main focus is on index size and retrieval time; retrieval quality will be addressed in a later chapter. I made sure that all versions of the search engine yield exactly the same retrieval results.

5.3.1 Implementation and test environment

The retrieval system is implemented in Java. The tests were executed on a 3.2 gigahertz Pentium 4 system with one gigabyte main memory on an ext3 file system on two SATA hard drives in a RAID 0 under Ubuntu Linux 6.10.

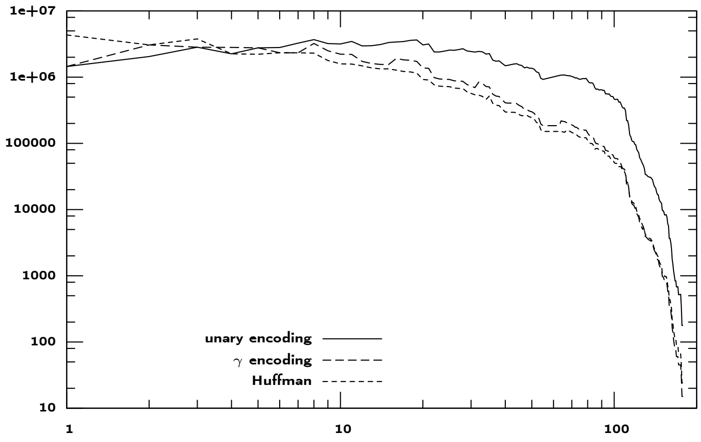
As the baseline, I use an index with all elements stored in the inverted lists, with the inverted lists compressed using bit-based compression and the digest compressed using byte-based compression. I compare that baseline to the difference-based indexing scheme with two variants of the compression of the metadata, bit-based and byte-based.

To make the comparison fair, most of the code is shared in the implementations. One notable difference is the propagation of term frequencies: In the baseline version, it is not needed at all, so I made sure that the corresponding code is not executed at all, in order to avoid the (slight) penalty it incurs.

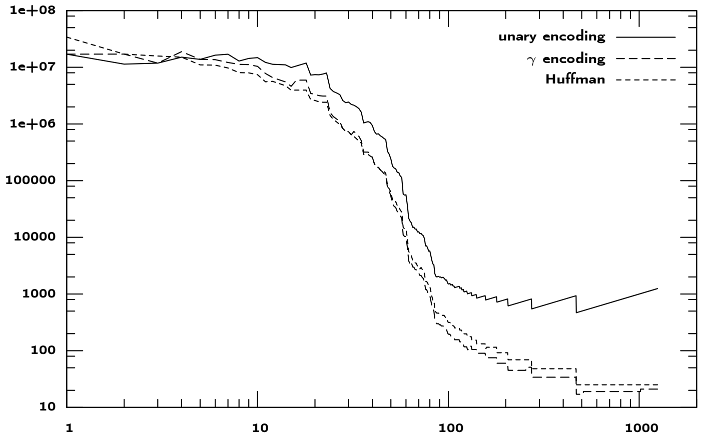
To smooth out random effects on retrieval time, all topics were executed in sequence five times and the mean time was used for the evaluation. (Note that the executions of the same topics were not adjacent, but all the other topics were in between; this avoids possible caching effects.)

The Java interpreter was called with the `-server` option, and the maximum and initial heap size were set to the same value to avoid expensive reallocation of memory.

Because there could be a slight change of the index sizes if the documents are added in a different order (file systems typically do not guarantee any specific order



(a) IEEE. Unary encoding is best at the start, but rapidly loses ground to the other encodings. Huffman encoding takes roughly 15 percent less space than γ encoding.



(b) Wikipedia. The sawtooth (unary) and step (γ, Huffman) behavior of the graph is caused by the atypical number of element names that occur only once.

Figure 5.5: Plot of the space usage (in bits) of the element name ID using various encoding functions over the frequency-based rank of that element name. The most frequent element is at the left, the least frequent element is at the right. The total space used for the IDs is roughly equal to the area under the graphs.

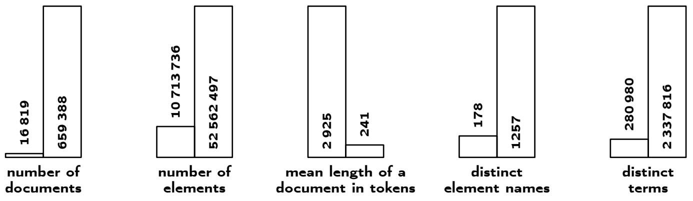


Figure 5.6: Test collections statistics. The bars in each group are, from left to right, the IEEE collection (2005) and the Wikipedia collection (2006). The token count excludes stop words.

when traversing directories), they were indexed from the distributed archive files, in the order they appear there.

5.3.2 Test collections

The evaluation will be based only on the test collections from INEX 2005 and INEX 2006. Figure 5.6 gives an overview of the properties of these collections. The two collections differ in several important qualities:

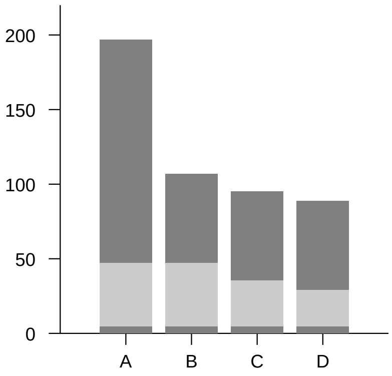
- The Wikipedia collection has about forty times as many documents as the IEEE collection, but the average length of a document is only a tenth.
- The Wikipedia collection has about eight times as many distinct index terms.
- Although the Wikipedia collection has many more distinct element names, the average number that is used in each document is smaller.

If the results are similar for these different collections, it is safe to assume that comparable behavior will be observed for other document-centric collections.

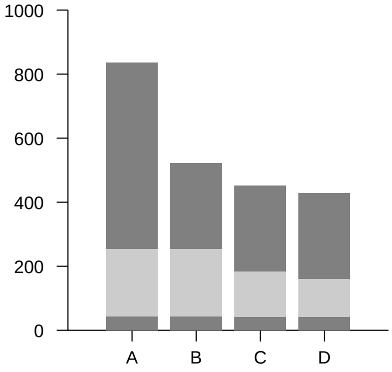
5.3.3 Index size

It is clear that less storage is required if less data is stored in the index. Figure 5.7 shows that the savings are significant: The index size is reduced to 56 percent of the baseline index size for the Wikipedia collection (from 871 megabytes to 485 megabytes) and to 50 percent for the IEEE collection (from 202 megabytes to 99 megabytes). For comparison, Geva (2007) reports the index size for the Wikipedia collection as 15 *giga*bytes – about 30 times the size of the index described here (no numbers are available for other search engines).

The lexicon and the metadata are not different between the baseline and the difference-based index. The size of the lexicon is negligible compared to the other parts of the index, but the metadata has a size comparable to that of the inverted list. Using a bit-based encoding for the metadata file reduces the size of it by about 25 to 30 percent, but increases retrieval time by about 10 to 15 percent.



(a) INEX 2005 test data (IEEE)



(b) INEX 2006 test data (Wikipedia)

A Baseline

B Difference-based inverted list, byte-compressed metadata

C Difference-based inverted list, bit-compressed metadata

D Difference-based inverted list, Huffman-compressed metadata

The boxes are, from bottom to top, the lexicon, the metadata index, and the inverted list. The mapping file from element name to ID is too small to be visible at this scale.

Figure 5.7: Index sizes, in megabytes.

Re-ordering the element names by decreasing frequency of occurrence in the collection only leads to minor savings of about 0.2 percent of the metadata index size for IEEE and less than 0.001 percent for Wikipedia with byte-based encoding. This is no big surprise for the IEEE collection, which only has 178 distinct element names, but for Wikipedia, a better compression ratio could have been expected. However, of the Wikipedia collection's 1257 element names, 1052 (more than 80 percent) occur at most three times each, because of peculiarities of the conversion program (it regarded all text occurring in angle brackets as element names, leading to element names like stdio.h). Because the vocabulary of the original markup is quite limited, the most-used element names have low numbers anyway, so the savings that can be expected are rather low. For the bit-based encoding, the savings are more significant: about 12 percent (Wikipedia) and 16 percent (IEEE).

5.3.4 Retrieval time

For timing, I use the official query sets from INEX 2005 and 2006, using the *title* field of the topics (that is, the keyword-based query without structural constraints, see Trotman and Sigurbjörnsson (2005)).

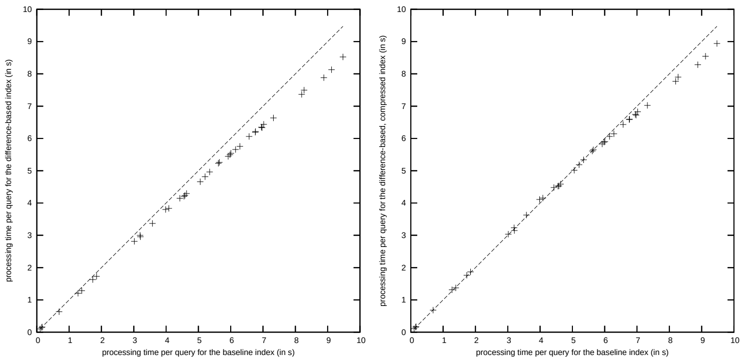
One cannot expect retrieval time to drop, because the reduced storage requirements and shorter read times are offset by the reconstruction of the missing elements. Retrieval time should not, however, increase compared to the baseline. This is confirmed by the measurements, as figure 5.8 shows: The retrieval time does not increase for any topic, in fact, retrieval time is reduced by about 5 percent. Figure 5.9 shows that indeed the time needed to read the inverted list entries from the index is reduced, but extra time is required to reconstruct the entries.

One important observation is that the search engine spends a significant portion, from 55 up to 85 percent, of the total time it needs to process a query in obtaining metadata.

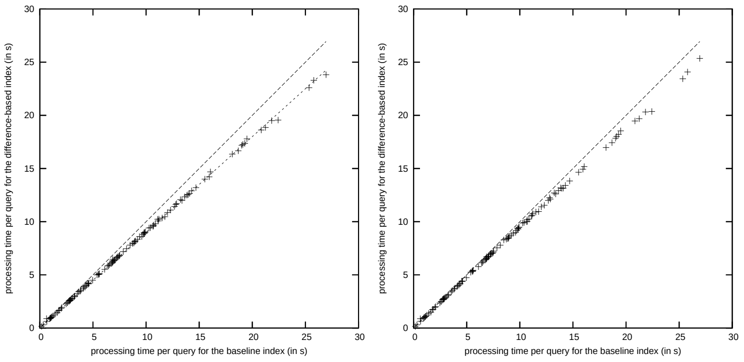
As a side note, the time needed to create the index is reduced by about 40 percent for difference-based indexes, because the inverted lists are significantly shorter, so less data has to be sorted on disk. Indexing the complete Wikipedia collection takes about 80 minutes, indexing the IEEE collection takes about 25 minutes.

5.3.5 Comparison to traditional information retrieval

Witten et al. (1999) give figures for the index size in relation to the size of the original documents for traditional information retrieval. The size of the index structures – inverted files and auxiliary files – is about 10 to 25 percent of the size of the documents for four collections with different characteristics. Compared to the uncompressed XML files, XML retrieval indexes are in the same range, at about 10 to 15 percent, although they store considerably more data. The main reason is, of course, that XML is a very verbose format; the markup takes up a large fraction of the file, and whitespace is often used to make the XML file more attractive at the source level.

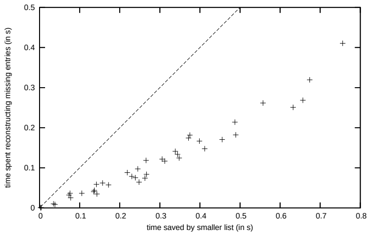


(a) INEX 2005 test data (IEEE)

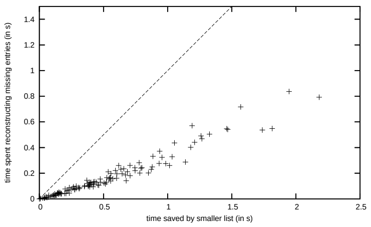


(b) INEX 2006 test data (Wikipedia)

Figure 5.8: Retrieval time: baseline versus difference-based (IEEE collection). Each point represents the time needed to process one particular query; for points that lie below the dashed line, retrieval on the difference-based index is faster. The diagrams on the left compare the baseline to the difference-based index with byte-compressed metadata, the diagrams on the right compare the baseline to the difference-based index with bit-compressed metadata.



(a) INEX 2005 test data (IEEE)



(b) INEX 2006 test data (Wikipedia)

Figure 5.9: Explanation why the retrieval time does not change much: The time needed to read the inverted list entries is reduced (x axis), but this is offset by the time needed to reconstruct the nodes that are not stored in the index (y axis). Points below the dashed line indicate a net saving for that query, points above the line indicate a net loss.

5.4 Indexing process

Because the indexer should only do a single pass over the documents, indexing is done in the following steps:

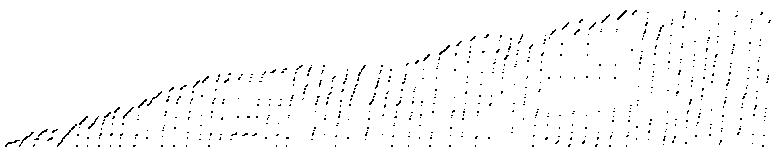
1. Each document is parsed, tokenized, and corresponding tuples of term ID, document ID, and further information (for example, term frequency) are stored in a heap of size n in main memory, ordered by term ID and document ID.

2. When the heap is filled, it is flushed to disk to form a single *run* of inverted list entries. Obviously, the tuples on disk are not completely sorted by term ID, document ID.

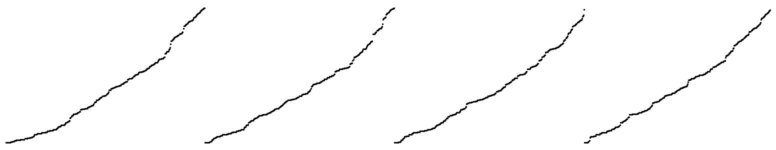
3. When all documents have been parsed, an n-way *merge sort* is executed on the stored on-disk tuples; n runs are merged to form a new sorted run, which is written to a new intermediate file. If the new intermediate file contains more than one run, this step is repeated.

4. Finally, the last intermediate file, which by now only contains a single run, is copied to the final inverted list file, omitting the (now unnecessary) term IDs and using delta encodings as far as possible. Also, the dictionary is updated to contain a pointer to the entry point in the inverted list file for each term.

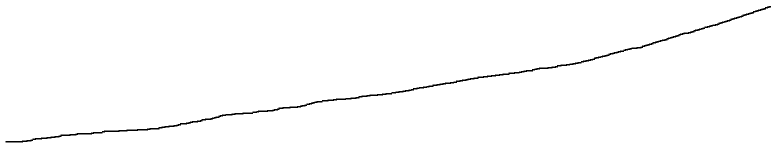
Thus, the intermediate inverted lists are effectively in a different format from the final result (and are typically slightly larger). Figure 5.10 illustrates the disk-based merge sort on real data.



(a) The 61 initial runs of length 16 plus overhead. Observe how the maximum term ID per run grows as more of the document has been read.



(b) After the first round of merges, 4 runs remain.



(c) The final inverted list.

Figure 5.10: Disk-based merge sort, run length 16, for a single XML document (the "Linux Print2Win mini-HOWTO"). The horizontal axis denotes the position in the inverted list file, the vertical axis the term IDs.

5.5 Performance versus accuracy

In XML retrieval, the number of indexing units is much larger than in flat information retrieval, as there are many elements per document. Naturally, this leads to a larger index and also to an increased retrieval time, because more data has to be read from disk, more main memory is needed for the results, and more calls to the similarity method have to be performed. Even worse, for anything but "thorough" results, the document's tree structure has to be re-built from the results, and it has to be traversed to decide which elements to keep.

As later sections will show, a straightforward approach yields response times that are barely acceptable, measured in tens of seconds. Although not the main focus of this thesis, I will examine how one can reduce retrieval time without sacrificing retrieval quality. There are many approaches to this problem for flat information retrieval, but these approaches assume that the retrieval results are independent of one another so that any subset of them can be omitted from the results. This no longer holds for XML retrieval, where several element results are part of the same document; only if the similarity measure makes no use of the document structure will it be possible to exclude some of a document's results from the similarity calculation without the risk of worsening results.

Geva (2007) describes how the version of GPX used at INEX 2006 keeps only a subset of the leaf elements before proceeding to the reconstruction of the inner nodes. For the official results, only the highest-scoring 3000 leaf elements were kept, and later experiments showed that keeping a higher number of elements leads to better retrieval quality at the expense of taking significantly more time. Furthermore, GPX excludes all terms that occur more than 100,000 times in the collection from the similarity calculation – they are basically seen as stop words. Theobald (2006) discusses top-k element retrieval in detail for a BM25-based search engine.

As the following chapter will show, my search engine uses a two-stage approach to retrieval which relies on an "all or nothing" approach to intermediate results: Either all relevant elements from a document are part of the retrieval results or none are. This prevents the use of the element-based top-k approaches, so I developed a document-based approach to this problem for INEX 2006 (Dopichaj, 2007d).

5.6 Summary

This chapter has shown that it is possible to significantly reduce the space overhead of XML full-text indexes without affecting the results or increasing retrieval time – in fact, retrieval time is slightly reduced compared to the baseline. The index structures support retrieval of *all* element types, including very short elements. This enables the search engine to support queries that target these small elements, and it is necessary for the enhancements described in the following chapter.

6 Exploiting small elements

"It was most suggestive," said Holmes. "It has long been an axiom of mine that the little things are infinitely the most important."

(A Case of Identity)

The previous chapters have shown that there is a variety of approaches to XML element retrieval. Many of these approaches are combinations of separate ideas, combining, for example, term proximity with relevance feedback mechanisms and novel similarity measures exploiting the structure of the XML documents. Some of these approaches have been shown to be effective in the INEX evaluation workshops, but it is impossible to single out the components that are responsible for the good retrieval quality.

So far, little work has been done concerning the influence of the components of XML retrieval systems. One reason is that the different components may interact in unexpected ways, so that their behavior in isolation is different from the behavior that can be observed if they are applied in combination. This is no reason to abstain from performing such an analysis completely; although the insights gained by a limited evaluation might not hold in different contexts, they can still be seen as a guideline for further research.

This chapter describes several approaches to exploiting small elements like section titles or italicized phrases. Finally, various ways of using small elements to improve retrieval quality and their respective advantages and disadvantages are discussed.

6.1 Small elements in standard information retrieval

One thing most of the existing retrieval systems have in common is that they do not make use of very small elements, like a single word in italics markup. Obviously, these elements are not reasonable retrieval results, because they would not help the user satisfy his information need. On the other hand, they can still be useful for estimating the relevance of their ancestors.

One particularly striking example is the use of titles. Titles are usually short, and they are mostly useless as a retrieval result, but they contain the essence of what the enclosing document is about. Cutler et al. (1997, 1999) make use of the markup in HTML documents by using separate indexes for different types of small elements, not only titles:

Document title, tag title (this document-level title can occur at most once),

high-level titles, tags h1 and h2,

low-level titles, tags h3 to h6,

strong, tags strong, b, em, i, u, dl, ol, and ul,

anchor, tag a (the text of incoming links), and

plain text for text outside the above elements.

Each term in the input documents is stored in exactly one of these indexes; if a term is contained in nested tags from different categories, the one higher in the list is used. For example, given <title><strong>term</strong></title>, term is indexed as a title and not as a "strong" word. The anchor class is a special case: It captures not terms from the same document, but terms from the anchor text of other documents that link to the indexed document. For example, if file cue.html is indexed and another file contains the fragment <a href="cue.html">cue</a>, then the term cue is indexed in the anchor field of cue.html.

In effect, instead of having a term vector for each document, their system has a term matrix: For each term, six term-frequency values are stored per document. When executing a search, importance values must be defined for each of the categories. To obtain the term weight, the scalar product of the importance vector and a term's weight vector for each document is calculated, and this term weight is then used as input to the cosine similarity measure.

The best importance vector they found was a weight of 8 for the "strong" class, 6 for the "high-level titles" class, 1 for the "low-level titles", 8 for the "anchor" class, 4 for the "document title" class, and 1 for plain text. This combination yielded a 26-percent improvement of the 11-point recall–precision average over their baseline system.

Liu et al. (2004) presented similar work at TREC, using an HTML document's title element, the first bold and head tags, and the first n words of straight text. Their evaluation shows minor improvement over their baseline system.

These approaches have in common that the different types of fields are stored in separate indexes; at retrieval time, a search process is performed on each of these indexes and the scores are combined. Robertson et al. (2004) contrast linear score combinations of separately indexed fields with the adaptation of the term frequency values based on what HTML markup a term is embedded in (for example, if a term is embedded in a title element, its weight is doubled). Their results indicate that the latter approach yields better results for the test cases.

6.2 Small elements in XML retrieval

Overall, the research on exploiting small elements in standard information retrieval shows that they can provide valuable hints about their ancestor elements. The experiments were based on document-level retrieval, and they worked on a known document schema (HTML); it remains open how they can be adapted to work for XML element retrieval.

For INEX 2005, I developed a retrieval method that exploits small elements to improve retrieval quality (Dopichaj, 2006b, 2007a). In independent work, Ramírez et al. (2006a) investigated the influence of small elements for the INEX collection; they used hard-coded rules like "use the maximum score of an element and its subordinate st element". On the INEX 2005 test collection and their language modeling retrieval engine, they showed that they can improve the baseline significantly.

6.2.1 Preliminaries

In this thesis, I will concentrate on a single type of small element, the title of a section. The reason for this is that the more influence factors there are, the less it is possible to draw clear conclusions; furthermore, my previous work shows that titles are the most promising variant of exploiting small elements. Therefore, the aim is to investigate the following orthogonal hypotheses:

- Titles can be used to identify highly relevant sections.
- It is possible to identify title elements without having to specify a schema-specific list of title element names.

For making use of titles, two things are necessary: a predicate $\mathrm{istitle}(e)$ that determines whether an element e is a title element and a means to change the similarity if a title was detected, the similarity adaptation function $\mathrm{tsim}(q, e, t)$.

The predicate $\mathrm{istitle}(e)$ is true if and only if element e is assumed to be the title of its parent element. Realistically, there is at most one title per element (this assumption simplifies the adaptation functions):

$$\mathrm{istitle}(e) := \forall c \in \mathrm{children}(\mathrm{parent}(e)) : \neg\, \mathrm{istitle}(c) \tag{6.1}$$

The tsim function determines how the degree of the title's parent element e should be changed; it is unlikely that every match in title elements should result in the same change of similarity. For example, if the query is "computer graphics" and the documents are computer science journals, the presence of "computer" alone is not a good indicator of relevance.

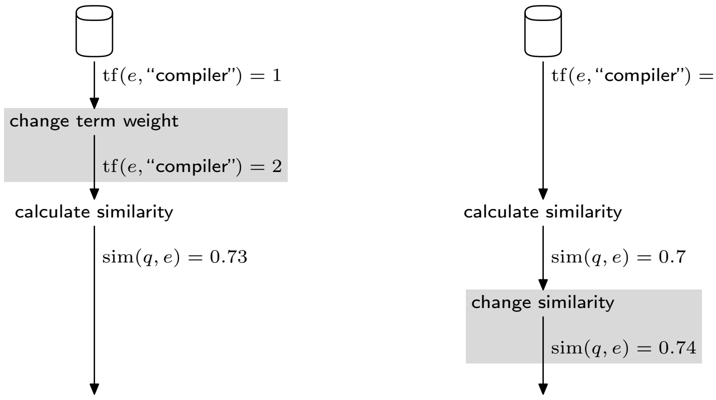
6.2.2 Example

To show that titles can indeed be useful indicators of relevant sections, I will now give an example from the INEX collection, topic 168. The information need for that topic is "Find documents or elements relating to the New Zealand Digital Library project", and the query is "New Zealand Digital Library Project". Table 6.1 gives the base result list for this topic along with relevant titles and assessments. Obviously, the baseline results are already good – there is not a single irrelevant element in the top ten. In this example, the titles are clearly useful hints, although the most relevant element – the complete article r2074 – is not a section with a

File	XPath expression	RSV	Section title (if applicable)	r
r2045	/article[1]/bm[1]/app[2]/sec[2]/lc[1]	36.8		1
r2045	/article[1]/bm[1]/app[2]/sec[2]	35.1	*Digital library* research in Asia	1
r2074	/article[1]/bm[1]	34.4		1
r2045	/article[1]/bm[1]/app[2]	31.0		1
r2074	/article[1]/bm[1]/vt[1]/p[1]	30.9		2
r2074	/article[1]/bm[1]/vt[1]	30.9		2
r2045	/article[1]	28.9		1
r2074	/article[1]/bm[1]/vt[2]	28.7		2
r2074	/article[1]/bm[1]/vt[2]/p[1]	28.7		2
r2074	/article[1]/bdy[1]/sec[2]	28.7	The *New Zealand Digital Library*	6
r2045	/article[1]/bdy[1]	28.2		1
r2045	/article[1]/bm[1]	28.2		1
r2045	/article[1]/bdy[1]/sec[4]	28.1	Semantic interoperability	1
r5022	/article[1]/bdy[1]/sec[1]	27.2	(empty title)	0
r2074	/article[1]	27.2		9
r2022	/article[1]/bdy[1]/sec[6]	27.0	Managing complexity in a distributed *digital library*, pp. 74-79	3
r2022	/article[1]/bdy[1]	27.0		1
r2022	/article[1]	26.7		1
r6067	/article[1]/bdy[1]/sec[3]/ss1[1]/p[3]	26.5		0
r2066	/article[1]/bdy[1]/sec[1]	26.0	(empty title)	0

Table 6.1: Topic 168 results baseline results ($b = 0.2$, $k_1 = 1$). The query is “New Zealand Digital Library Project”. All articles are from co/1999, except r5022, which is from co/1996. Keyword matches in the titles are *in italics*. The relevance value r indicates how useful the searcher found the corresponding retrieval result, the higher, the better.

```
<sec><st>Compiler</st> <p>...</p></sec>
```



(a) Term weight adaptation **(b)** Similarity adaptation

Figure 6.1: Two basic options to exploit title elements. Here, e is the sec element, and "compiler" is a query term. The first step is the retrieval of the term frequency from the index, and the similarity measure uses the frequency to calculate the RSV.

title. However, a section from that article has a title that matches the query almost perfectly (/article[1]/bdy[1]/sec[2]), and it is still assessed highly relevant.

Indeed, if the search engine only searched titles and returned the corresponding section, it could easily find three quite relevant hits among the top twenty. In combination with a regular retrieval method, the results should be very good.

6.3 Adaptation methods

From the related work cited above, the following options arise (see figure 6.1):

Similarity adaptation: The similarities are calculated from the term weights as normal, and the final RSVs are then adapted (Cutler et al., 1997; Ramírez et al., 2006a; Dopichaj, 2006b, 2007a).

Term weight adaptation: The term weights for terms occurring in titles are changed before the normal similarity measure is applied. For practical reasons, the term frequencies are changed (Robertson et al., 2004; Dopichaj, 2007b).

In theory, these two could be combined, but this does not appear to be very sensible. Much variation is possible, so an exhaustive evaluation is impractical. In the following sections, I will give an overview of the concrete implementations that are used for the evaluation.

6.3.1 Similarity adaptation

In XML retrieval, *any* kind of element can be a valid retrieval result, and it would be unreasonable to assume that only elements with titles – in this case, sections – can be useful results. Thus, the search engine must still calculate a similarity for every element. A new similarity measure could be developed that takes into account title information, but it would be hard to incorporate this into the base similarity function, and the search engine would be restricted to only this specific function. Since the baseline retrieval from section 4.3 proves to yield high-quality results, I will instead use a sort of postprocessing step:

$$\mathrm{sim}(q,e) = \begin{cases} \mathrm{tsim}(q,e,t) & \text{if } \exists t \in \mathrm{children}(e) : \mathrm{istitle}(t) \wedge \mathrm{sim}(q,t) > 0 \\ \mathrm{sim}(q,e) & \text{otherwise} \end{cases} \tag{6.2}$$

The tsim function determines the final similarity of elements that have a hit in the title. The aim is to push particularly good elements to the top of the result list, if possible to the top ten. Thus, the search engine increases the scores of elements that appear to be particularly relevant to the query and leaves the other scores as they are; in particular, no score should ever be decreased in this postprocessing step:

$$\mathrm{tsim}(q,e,t) \geq \mathrm{sim}(q,e) \tag{6.3}$$

The tsim function may also make use of the number of query terms that occur in the element, that is, the coordination factor.

To determine the quality of the title match, element information has two items that can be used to determine the quality of a match: the retrieval status value (RSV) and the coordination factor. Obviously, the RSV is more exact, because it takes into account term weights and other important factors, but it also incorporates some form of length normalization.

There are various options for increasing the RSV of a title's parent element p:

- **Simple:** Multiply the RSV of p by a constant factor $o > 1$ (Dopichaj, 2007a).

$$\mathrm{tsim}(q,e,t) := o\,\mathrm{sim}(q,e) \tag{6.4}$$

- **Restrictive:** Only multiply by the boost factor if there are as many hits in the title as in the complete section:

$$\mathrm{tsim}(q,e,t) := \begin{cases} o\,\mathrm{sim}(q,e) & \text{if } \mathrm{coord}(q,t) = \mathrm{coord}(q,e) \\ \mathrm{sim}(q,e) & \text{otherwise} \end{cases} \tag{6.5}$$

- **Dynamic:** Use a degree of the title's match quality to vary the boost factor. The intention is to make sure that titles that more closely match the sections they are titles of should move its parent further up.

$$\mathrm{tsim}(q,e,t) := \mathrm{sim}(q,e)\left(1 + o\frac{\mathrm{coord}(q,t)}{\mathrm{coord}(q,e)}\right) \tag{6.6}$$

- **Adding:** Add the title element's RSV to its parent's RSV (this is a simple form of linear score combination):

$$\mathrm{tsim}(q, e, t) := \mathrm{sim}(q, e) + \mathrm{sim}(q, t) \tag{6.7}$$

Many more similarity adaptation functions are conceivable, but the ones given here are useful to demonstrate important influence factors. Figure 6.2 illustrates the adaptation methods.

6.3.2 Term weight adaptation

The other adaptation method modifies the *input* of the similarity function instead of its output. This is achieved by modifying the local term weights, which are typically derived from the term frequencies. The core idea is to increase the weights of terms that (also) occur in title elements.

Robertson et al. (2004) describe the linear combination of term frequencies for documents that are split into (non-overlapping) fields like title, abstract, and full text. Their approach is only meant for document-based retrieval, but it can be adapted to semi-structured retrieval (Dopichaj, 2007b): If a term t appears in the title element e_t of section element e_s, the term frequency $\mathrm{tf}(e_s, t)$ of t can be increased to obtain a higher weight:

$$\mathrm{tf}'(e_s, t) = \mathrm{tf}(e_s, t) + o\,\mathrm{tf}(e_t, t) \tag{6.8}$$

The parameter $o > 0$ defines how much the term frequency should be increased. The similarity of e_s is then calculated using $\mathrm{tf}'(e_s, t)$, but the changed term frequencies are not propagated to the ancestors of e_s. This is similar to what is done in the implementation of the base retrieval engine when the term frequencies are restored from the stored term frequencies. Given the set of children $C = \mathrm{children}(e_s)$ and the title element e_t, the term frequency reconstruction formula can be changed to:

$$\mathrm{tf}'(e_s, t) = \mathrm{stf}(e_s, t) + \sum_{c: c \in C \wedge c \neq e_t} \mathrm{tf}(c, t) + (1 + o)\,\mathrm{tf}(e_t, t) \tag{6.9}$$

This shows that term weight adaptation can be done at retrieval time without much overhead.

If it is not necessary to be able to retrieve the title elements themselves (because they are not considered useful retrieval results) and a restriction of o to integers is acceptable, the term frequencies can instead be changed in the index (this is closer to the approach by Robertson et al.). In this variant, the term frequency calculated in equation 6.8 is stored in the index, and the term frequencies for the title element are not stored at all.

The index-based approach to term weight adaptation has the advantage that a reduction of both index size and retrieval time is possible: Most obviously, the postprocessing can be avoided entirely (but this is unlikely to be significant). More important is that fewer elements are indexed, which implies that the index gets

RSV	Document	XPath expression	Query matches		
1.3	doc1	/article/section/title	*	*	*
0.9	doc2	/article/section	*		*
0.5	doc1	/article/section	*	*	*
0.4	doc2	/article/section/title	*		

(a) Base result.

RSV	Document	XPath expression	Query matches		
1.8	doc2	/article/section	*		*
1.3	doc1	/article/section/title	*	*	*
1.0	doc1	/article/section	*	*	*
0.4	doc2	/article/section/title	*		

(b) Adaptation method “simple”, $o = 2$.

RSV	Document	XPath expression	Query matches		
1.3	doc1	/article/section/title	*	*	*
$0.5 \cdot 2 = 1.0$	doc1	/article/section	*	*	*
0.9	doc2	/article/section	*		*
0.4	doc2	/article/section/title	*		

(c) Adaptation method “restrictive”, $o = 2$. Now, doc2 has more query term matches in the section than in the title, so it is not boosted.

RSV	Document	XPath expression	Query matches		
$0.9 \cdot (1 + 1/2) = 1.35$	doc2	/article/section	*		*
1.30	doc1	/article/section/title	*	*	*
$0.5 \cdot (1 + 3/3) = 1.00$	doc1	/article/section	*	*	*
0.40	doc2	/article/section/title	*		

(d) Adaptation method “dynamic”, $o = 1$. Now the boost factor depends on the number of matches.

RSV	Document	XPath expression	Query matches		
$0.9 + 1.3 = 2.2$	doc2	/article/section	*		*
1.3	doc1	/article/section/title	*	*	*
$0.5 + 0.4 = 0.9$	doc1	/article/section	*	*	*
0.4	doc2	/article/section/title	*		

(e) Adaptation method “adding”.

Figure 6.2: Effects of the adaptation methods.

smaller, so that less data has to be read from disk at retrieval time and fewer results have to be processed.

The disadvantage lies in the loss of flexibility (parameters must be set at indexing time) and the inability to retrieve elements that are not stored in the index. Although title elements are not usually useful retrieval results, they may be the target of content-and-structure queries, for example, "Retrieve all documents which have a section titled 'information retrieval'". If the title elements are not indexed, the constraint cannot be verified, leading to less precise results.

In this thesis, I will use the same indexes for term weight adaptation as for similarity adaptation and use a similarity adaptation function to incorporate the term frequency adaptation ($e + o \cdot t$ denotes the addition of the term frequency vectors):

$$\mathrm{tsim}(q, e, t) := \mathrm{sim}(q, e + o \cdot t) \tag{6.10}$$

This is not completely equivalent to the index-based method described above, but retrieval quality should be similar, and the implementation is simplified considerably.

6.4 Title detection

The following sections will present two basic methods for determining whether an element is a title element: one based on the element name and a schema-independent heuristic based on simple statistics.

6.4.1 Name-based title detection

The core idea of semantic markup is that the markup conveys information about the element's contents that is not directly related to the final presentation; for example, instead of embedding a title in a boldface element, semantic markup would use a title element. For presentation to the user, the semantic element name is then mapped to a certain rendering, in this case a boldface font. Semantic markup has the advantage of enabling different renderings for the same markup – for example, if a style guide requires all titles to be rendered in italics instead, this is not a problem with semantic markup, whereas the original text has to be changed for presentational markup.

If semantic markup is used, the most straightforward way of implementing the istitle predicate is to use the element name of an element name(e) and compare it to a set T of known element names that denote

$$\mathrm{istitle}_{\mathrm{name}}(e) := \mathrm{name}(e) \in T \tag{6.11}$$

Unfortunately, the information about which elements are particularly relevant to searching is not encoded directly in the document schema (the schema languages simply do not support this), so this information must be obtained by other means. The most straightforward method is to manually inspect the schema and augment it with this information. This may be feasible (but tedious) if only a single schema

```
<!-- Title of Subsection -->
<xsl:template match="ss1/st">
    <h3><xsl:apply-templates mode="high" select="." /></h3>
</xsl:template>
```

Figure 6.3: A characteristic snippet from X-Rai's stylesheet for the INEX IEEE collection, obtained from http://svn.berlios.de/viewcvs/x-rai/trunk/xsl/ieee-article.xsl?rev=3&view=markup. This fragment suggests that ss1/st elements are lower-level headings.

is used for all documents in the collection; in more complex collections of multiple schemas, this may be too costly.

This effort can be avoided to a large degree in most cases: Remember that the documents will have to be displayed to the users, so there is typically at least one stylesheet available. This stylesheet translates from the semantic level of the markup to the visual level of the final display device, for example, HTML or XSL-FO. Although this process loses information, the rendered version of the document now maps the diverse markup elements of the document schema to a single well-defined schema; figure 6.3 provides an example of a typical stylesheet. Thus, instead of defining the sets of title element names for all input schemas, it is sufficient to define it for a single output schema. The stylesheets can then be parsed to obtain the element names from the input schema that are converted to title element names in the output schema. Determining title elements is out of the scope of this thesis, however, so I will not discuss this in detail.

6.4.2 Length-based title detection

It is not always possible or desirable to manually inspect the document schemas to determine semantically important element types, and stylesheets are not always available or easily parsable. In fact, using only the obvious candidates like st for section title may miss elements that act as titles in the document, but use seemingly incorrect markup – for example, a bold phrase at the start of a paragraph will stand out like a title to the users, but does not use the official tags.

It is thus useful to use a heuristic approach to automatically detecting titles at run time, using a rule like “a short element as the first child and at the beginning of a longer element is a title”. The heuristic will obviously not be entirely accurate with respect to the original markup, with two types of errors:

False positives: Some elements that are not marked up as titles will be found if they satisfy the condition.

False negatives: Not all elements marked up as titles will be found, for example, for unusually long titles.

Note, however, that this classification compares to the actual markup in the documents, that is, it assumes that all elements marked up as titles – and only those elements – really are titles (the markup matches the semantics). In reality, the situation may be different, because document authors do not always use the schema as intended (maybe because of ignorance, maybe because the schema does not provide the intended markup). A schema might, for example, not provide for titles of paragraphs. If an author wants to use a title for a paragraph, he must use a workaround like typesetting the title as a bold phrase in the beginning of the paragraph. An approach purely based on the schema will not recognize it as a title, but a structural approach will. Thus, it has a higher likelihood of success even if the markup is presentational instead of semantic.

Using the textual definition that "a short element that is the first child and occurs at the beginning of a longer element is a title", it is necessary to examine two levels in the document tree to determine whether element t is a title element. The title detection predicate must determine

- whether the potential title element t occurs at the start of its parent,
- whether it is short enough to be a title, and
- whether the parent element is long enough.

Thus, the predicate is defined as follows:

$$\text{istitle}_{\text{length}}(t) := (\text{pos}(t) = 0) \wedge (\text{len}(t) \leq k) \wedge (\text{len}(\text{parent}(t)) > 2\,\text{len}(t)) \quad (6.12)$$

The parameter k determines the maximum length in terms up to which an element is considered short enough to be a title. The function pos(t) gives the starting position of the element t inside its parent element; it yields 0 if t is not preceded by text. The length threshold k must be chosen carefully to both match long titles and exclude short paragraphs at the beginning of sections. The condition in the formula that the parent element must be at least twice as long as the potential title element is rather permissive; it will rather result in false positives than in false negatives.

In previous work (Dopichaj, 2006b, 2007a), I used a fuzzy predicate instead of a Boolean predicate with a fixed parameter; the predicate yielded a value between 0 and 1 instead of Boolean values. The motivation was that this more closely models the human interpretation: there is a degree of shortness instead of a fixed threshold. Additionally, fuzziness is useful if the results of several patterns are applied at the same time (but applying several patterns in combination proved not to be effective, anyway). Unfortunately, this approach has more parameters to adapt – the form and slope of the curve to determine the degree of shortness. This makes a thorough evaluation more difficult. Furthermore, later work (Dopichaj, 2007b) showed that a fixed threshold does not yield worse results than a fuzzy threshold. For these reasons, I will not further discuss a fuzzy approach to patterns.

6.5 Efficiency

The additional postprocessing step that is needed for exploiting title elements may appear to be expensive in terms of retrieval time, because the document's DOM tree has to be reconstructed from the retrieval results and traversed to adapt the similarity. However, in combination with the difference-based indexing method introduced in the preceding chapter, the reconstruction of the tree has to occur anyway, so the only overhead is the execution of a few additional simple calculations. Overall, retrieval time is smaller than for conventional index structures without postprocessing.

If even this overhead appears too costly and the index size should be reduced further, term weight adaptation can be done at indexing time instead of at retrieval time. Previous work (Dopichaj, 2007b) has shown that this can reduce the size of the index by about 5 percent for name-based patterns and reduce retrieval time by about 50 percent. This work was done before the implementation of difference-based indexing, so the results are obsolete (the index sizes reported in that paper are several times as large as the current index sizes). Cursory tests with the new indexing method indicate that the differences are now not relevant anymore.

Considering that index-based term-weight adaptation is less flexible – it is not easily possible to adapt the parameters of the adaptation function – and not all elements are retrievable anymore, I will only focus on term-weight adaptation implemented as a similarity function.

6.6 Summary

This section discussed the retrieval engine that is the basis of the evaluations in this thesis. First, an overview of approaches exploiting titles in traditional information retrieval was given, and these approaches were then adapted to XML element retrieval systematically. Several approaches to adapting similarity values based on title matches were introduced, and two approaches to detecting title elements were presented. The following chapter will present the evaluation setup and evaluate the utility of the enhancements on INEX test data.

7 Evaluation of retrieval quality

"You know my methods. Apply them!"

(The Hound of the Baskervilles)

To determine whether exploiting title elements, as described in the preceding chapter, really improves retrieval quality, experiments have to be performed. In XML retrieval, there is no way around INEX (see section 3.2): there simply is no comparable alternative.

To understand what exactly the evaluation results mean, one must first understand the metrics used to evaluate retrieval quality (section 7.1). In the following section, I briefly discuss the evaluation of my official INEX submissions (section 7.2). Next, the best parameters and the general behavior of the base retrieval system has to be tested (section 7.3); unfortunately, no generally accepted baseline has been established in the XML retrieval community. After the optimal parameters of the baseline have been found, it is finally possible to evaluate whether title elements can indeed improve retrieval quality (section 7.4). The evaluations based on the INEX data have various shortcomings, so I closes this chapter with a discussion of these shortcomings and a proposal for how a more suitable evaluation could be set up (section 7.5.1).

7.1 Evaluation metrics

In order to determine how good a retrieval engine is, a qualitative measure of the quality of a search engine's retrieval result must be determined from the results and relevance assessments.

In traditional information retrieval, *recall* and *precision* are used. Recall measures the engine's ability to retrieve as many relevant documents as possible, and precision measures the ability to retrieve as few irrelevant documents as possible. Given d_r the number of relevant documents that were retrieved, d_a the total number of relevant documents, and d_R the total number of retrieved documents, recall r and precision p are defined as follows:

$$r = \frac{d_r}{d_a} \tag{7.1}$$

$$p = \frac{d_r}{d_R} \tag{7.2}$$

Frequently, precision is given not for *all* retrieved results, but only for the first n retrieved results, because the first few results are the most interesting for the user. In this case, $d_R = n$ and d_r is the number of relevant documents among the first n retrieved. Reporting recall at different rank values does not make sense because it would turn out to be exactly the same as precision (as long as there are enough relevant documents). Recall and precision work on a binary model of relevance, so a document either is relevant or it is not, there is no notion of documents being relevant to a certain degree.

Binary relevance may not always be sufficiently detailed, so Järvelin and Kekäläinen (2002) introduced the *cumulated gain* measures. Assessments are done in degrees, that is, instead of "is (ir)relevant", a numeric value is assigned to each document that denotes a degree to which this document is relevant. For example, a completely relevant document might get a value of 1, whereas a document that only satisfies a part of the information need might get 0.5. Each retrieval result gets a score from the assessment values, so a result list is mapped to a list of scores, the *gain vector*. The cumulated gain at position i is then calculated as the sum of all gain values up to position i. For a gain vector $G = \langle 1, 0.5, 0, 0.75 \rangle$, the associated cumulated gain values are 1, 1.5, 1.5, and 2.25. These values can be used for comparing systems: the higher a value is, the better the corresponding system is.

7.1.1 Relevance in XML retrieval

For XML retrieval, a one-dimensional notion of relevance was deemed to be too crude because of the nesting of elements: If a paragraph is relevant, then logically its ancestors (section, chapter, complete document) must also be relevant. This is obviously incompatible with the FERMI principle, which states that the search engine should retrieve the most specific element that satisfies the information need. Thus, if the paragraph is such an optimal element, its ancestors (although still relevant) should be regarded as worse retrieval results.

To model this, two relevance dimensions were introduced, *specificity* and *exhaustiveness* (Fuhr et al., 2003). The specificity value of an element denotes to what degree a result is focused on the information need; the higher the specificity, the lower the amount of text in the element that does not help to satisfy the information need. The exhaustiveness represents to what degree the whole information need is addressed. If only one of two aspects of the query is addressed by an element, that element gets a lower score than one that addresses both.

From 2005 on, the assessor used a virtual yellow highlighter to select highly specific passages in the online assessment interface. This was used for calculating the specificity value of the elements, which was obtained by dividing the number of characters that were highlighted in the element by the total character count in the element. After that, the assessor had to set the exhaustiveness for all elements with a specificity greater than zero. The choices ranked from "highly exhaustive" (2) to "too small" for elements which should not be retrieved (0). The latter was applied recursively to all sub-elements – an element that is too small to be retrieved cannot contain any elements that are not too small.

In 2006, specificity was determined the same way as in 2005, but exhaustiveness was abandoned completely. This was mainly done to make the assessment procedure less burdensome, and the hope is that specificity alone is a good measure of retrieval quality.

7.1.2 Extended cumulated gain

Based on the two-dimensional assessment scale and the cumulated gain measure, XML-specific evaluation metrics were designed. Kazai and Lalmas (2006) adapted cumulated gain to the two-dimensional relevance scale of INEX by introducing *quantization functions* that amalgamate exhaustiveness e and specificity s into a single value.

$$\begin{aligned} \text{quant}_{\text{strict}} &= \begin{cases} 1 & \text{if } e = 2 \text{ and } s = 1 \\ 0 & \text{otherwise} \end{cases} && (7.3) \\ \text{quant}_{\text{gen}} &= e \cdot s && (7.4) \\ \text{quant}_{\text{genLifted}} &= (e+1) \cdot s && (7.5) \end{aligned}$$

The quantization function $\text{quant}_{\text{strict}}$ rewards only the very best elements, that is, elements that are fully exhaustive and highly specific. The difference between $\text{quant}_{\text{gen}}$ and $\text{quant}_{\text{genLifted}}$ is that the former treats all elements assessed as "too small" ($e = 0$) as completely irrelevant, whereas the latter rewards their retrieval a little; it was introduced because several assessors used the "mark all subelements as too small" liberally to ease the assessment burden (this is far less work than manually inspecting every small element).

The basis of the evaluation for *thorough* results is the *full recall base*, which consists of all the elements that were assessed as relevant. The relevance value for an element is obtained by applying the quantization function to the assessment, and the cumulated gain measure can then be applied without modification, now called *extended cumulated gain* (nxCG). For better comparability across topics, a normalized version (nxCG) was introduced. In this version, the cumulated gain reached by a retrieval engine is divided by the corresponding cumulated gain in the ideal recall vector (that is, the vector obtained by ordering the assessed elements by decreasing relevance). Given xCG[i] as the cumulated gain at rank i and xIG[i] as the cumulated gain of the ideal run, nxCG[i] is defined as follows:

$$\text{nxCG}[i] = \frac{\text{xCG}[i]}{\text{xIG}[i]} \tag{7.6}$$

For each point, the nxCG value can be interpreted as the degree to which the obtained result is ideal. This definition of nxCG does not take the ordering of the results up to the measured rank into account; a result list and its reversed version would achieve exactly the same value. In reality, searchers are mostly interested in the first few results – according to Jansen et al. (1998), 58 percent of all web searchers only look at the first result page with ten results, more than 75 percent look at only one or two pages –, so the gain values for higher ranks should have

ID	Path	r
1	/art[1]/sec[1]	0.1
2	/art[1]/sec[1]/p[4]	2
3	/art[1]/sec[2]	1.4
4	/art[1]/sec[2]/p[1]	1
5	/art[1]/sec[1]/p[2]	0.5

The table contains all relevant elements along with their relevance value; a higher value indicates a better result.

(a) Full and ideal recall bases.

Ideal run				Real run			
Path	r	xCG	nxCG	Path	r	xCG	nxCG
2	2	2	1	3	1.4	1.4	0.7
3	1.4	3.4	1	–	0	1.4	0.41
4	1	4.4	1	1	0.1	1.5	0.34
5	0.5	4.9	1	2	2	3.5	0.71
1	0.1	5	1	5	0.5	4	0.8

(b) Calculation of nxCG.

Figure 7.1: Example for nxCG calculation for the *thorough* retrieval task.

a higher weight. *Mean average nxCG* at rank r addresses this by calculating the average nxCG values for ranks 1 to r:

$$\text{MAnxCG}[r] = \frac{1}{r}\sum_{i=1}^{r} \text{nxCG}[i] \tag{7.7}$$

Figure 7.1 gives an example of nxCG calculation. The official INEX results page reports nxCG at the ranks 5, 10, 25, and 50 and for all quantization functions for the user-oriented tasks, in particular the *focused* task.

In 2006, although the metrics remained formally the same, the assessment procedure was reduced to only the highlighting part for specifying specificity. This reduced the time needed to do the assessments, but no exhaustiveness value was available any more, so it was set to 2 for all elements. Using this set of assessments turned out to have the disadvantage that small elements only a few words long were evaluated as highly exhaustive, which is clearly undesirable at least for user-oriented tasks. Thus, a second recall base was prepared that set the exhaustiveness of link elements to “too small” (this applied to all elements of the following type: collectionlink, wikipedialink, redirectlink, unknownlink, outsidelink). This choice is rather arbitrary, as there are many other elements that are useless as retrieval results. In reality, whether a given element is a useful retrieval result can only be determined by the assessor, using the exhaustiveness dimension. The removal of arbitrary results from the recall base is just a workaround for the loss of this assessment dimension and cannot completely compensate for it. For this reason, the results reported in this thesis will be based on the original full recall base.

Many more evaluation measures have been developed in the context of INEX: The more user-oriented measures *effort-precision* and *gain-recall* (Kazai and Lalmas, 2006), which measures the effort a user has to make in order to reach a given level of cumulated gain. As the name implies, it is modeled after standard recall and precision, with changes for the multigraded relevance. *Mean average effort-precision* (MAep) is the mean of the effort-precision values at natural gain-recall points (whenever a relevant element is found in the result list). *Q and R* was introduced because the number of relevant elements varies between topics, so averaging the xCG scores does not yield smooth results. Piwowarski (2006) introduced the Expected Precision Recall with User Model (EPRUM) as a measure that is based on a clear user model. EPRUM approximates user behavior using a probabilistic model with various parameters that can be tuned to imitate different types of users. Pehcevski and Thom (2006) introduced Highlighting XML Retrieval Evaluation (HiXEval), an evaluation measure that is based solely on specificity (that is, it ignores the exhaustiveness dimension of the assessments). Thus, the retrieval task shifts from the principle of retrieving the smallest element that satisfies the information need to retrieving elements that contain as little irrelevant information as possible. They justify this decision by stating that several people assessing the same topic agree more on specificity than on exhaustiveness. (In fact, INEX 2006 dropped the specification of exhaustiveness from the assessment interface.)

Apart from the current official INEX metrics, there are also the metrics used in previous years – inex_eval, inex_eval_ng, precision recall with user modeling, tolerance to irrelevance –, as well as various other metrics that have never been considered official. Kazai and Lalmas (2005) give a brief overview of various metrics as of 2005.

Apart from the *thorough* retrieval task, there are measures for the other tasks like *focused*, *relevant in context*, and *best in context*.

This incomplete list of evaluation measures illustrates that there is no clear consensus on how to measure retrieval quality; new measures are presented at the INEX workshops every year. The evaluations in the following sections are based on the nxCG measure, because it is the official measure of INEX and the other measures are not clearly superior for the purposes of this thesis.

In order to ensure consistent evaluation of results, INEX provides a Java implementation of the nxCG, ep/gr, PRUM, and EPRUM metrics as open source software, EvalJ.[1] EvalJ takes as input the relevance assessments that are available for download and a submission file in the INEX format. This makes it possible to perform evaluation outside the INEX schedule.

EvalJ is hard to integrate in other software, so I reimplemented the nxCG measure for my retrieval system. I made sure that my version of the evaluation gives the same results (although at a slightly higher numerical accuracy).

7.1.3 Test collections

The INEX workshops used a collection of IEEE computer society[2] journal and transactions articles through 2005, where later versions of the collection are supersets of

[1] http://evalj.sourceforge.net

[2] http://www.computer.org

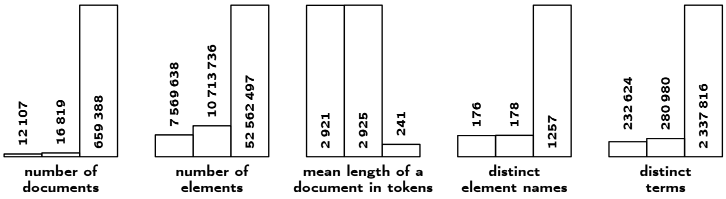


Figure 7.2: Test collections statistics. The bars in each group are, from left to right, the IEEE 1.4 collection (2004), the IEEE 1.9 collection (2005), and the Wikipedia collection (2006). The token count excludes stop words.

earlier versions (new volumes were added). From 2006 on, a conversion of the English version of Wikipedia was used (Denoyer and Gallinari, 2006). The evaluations in this thesis will be based on the collections from 2004, 2005, and 2006. Figure 7.2 gives an overview of various characteristics of the document collections.

For each year of the workshop, a new set of topics was created by the participants, consisting of a longer description of the information need and a query in NEXI format. The number of topics varied: in 2004, there were 40 CO topics (34 have been assessed), in 2005, there were 40 topics (29 assessed), and in 2006, there were 130 topics (114 assessed). For my evaluations, I will only use content-only topics.

The assessment procedure has changed against the years: In 2004, the assessors had to manually select both specificity and exhaustiveness on a scale from 0 to 2 for each element in the recall base. In 2005, a highlighting approach was introduced; the assessor used a virtual highlighter to mark relevant passages in the documents to denote specificity. In the next step, the exhaustiveness had to be set for each element as in 2004. From 2006 on, exhaustiveness was dropped from the assessments, only the highlighting approach to selectivity was retained.

Note that I will use nxCG for the evaluations on the INEX 2004 test collection, even though nxCG was not the official evaluation measure at the time. This is possible because the data that was collected for the assessments is compatible, and it makes the results presented in this thesis more consistent and comparable. The results may not be as meaningful as the results for the other collections, but it is still interesting to see differences of behavior compared to the 2005 results, which are based on almost the same document collection.

7.2 Official INEX results

First, I will examine the official result of the two INEX workshops where submissions based on the research presented in this thesis were submitted (2005 and 2006; INEX 2007 results were not available in time). Keep in mind that the implementations differ somewhat from the descriptions in this thesis – they are earlier versions described by Dopichaj (2006b) and Eger (2005). Although the evaluation only partially applies to the current version, it is still important not to ignore the results; in contrast to the evaluations in the following sections, the implementation and chosen parameters cannot possibly have been the result of tuning to the test collection.

From the INEX 2005 results, I can observe the following points:

- My submissions are consistently at the top for the first few result pages, although the overall quality over the 1500 highest-ranked results is only mediocre compared to the other contestants.
- Relative scores with “strict” interpretation of the relevance assessments are lower, but still rather good.
- Pattern-based runs outperform the baseline runs (but not dramatically).

Overall, this matches my design goals and assumptions; the first few result pages are the most important ones to the searcher, and this is precisely the area in which my runs perform best.

My INEX 2006 results suffered from mediocre performance compared to the other participants' results. This was caused mainly by a bad choice of parameters for the similarity function, the following sections will show that much better results can be achieved using the same base retrieval engine.

7.3 Parameter tuning for the base retrieval engine

There are many similarity measures, both for traditional information retrieval and for XML retrieval. In this section, I will examine how well a naïve approach to XML retrieval, as introduced in chapter 4, works. For this, all elements are indexed as if they were separate documents and the following similarity functions are used: BM25 and Lucene similarity. For both of these similarity measures, I will tune the parameters to suit XML retrieval; the default parameters are good for standard information retrieval, but will probably have to be adapted for this new scenario. The best parameter combination will then be used as the basis for further evaluation.

7.3.1 Lucene similarity measure

The Lucene similarity measure gave good results at least in 2005. In this section, I will evaluate two global weighting methods – element and document frequency – and a parameterizable version of Lucene's length normalization function:

- Standard length normalization:

$$\mathrm{lnorm}_{\mathrm{luc}}(d) = \frac{1}{\sqrt{\mathrm{len}(d)}} \tag{7.8}$$

- Standard length normalization with a constant value up to length l:

$$\mathrm{lnorm}_{\mathrm{const}}(d) = \frac{1}{\sqrt{\max(\mathrm{len}(d), l)}} \tag{7.9}$$

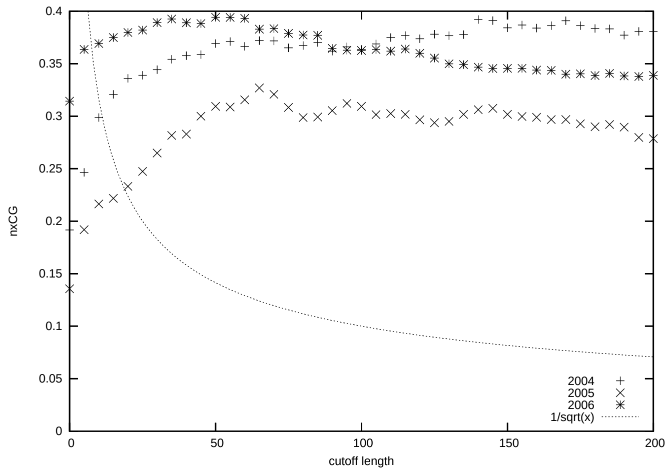


Figure 7.3: Lucene retrieval quality (nxCG@10), using document frequency. For reference, a plot of the Lucene length normalization function is included in the plot.

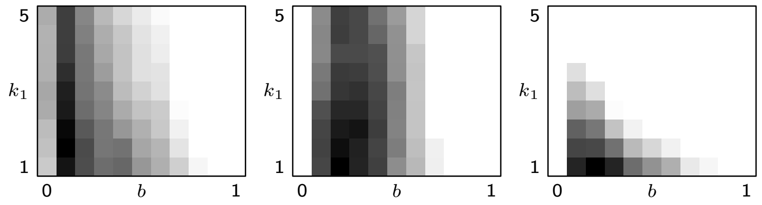
The following parameter combinations have to be tested, using $\text{lnorm}_{\text{const}}$ (for 0, $\text{lnorm}_{\text{const}}$ is effectively $\text{lnorm}_{\text{luc}}$):

$$\underbrace{\{\text{df}, \text{ef}\}}_{\text{gf}} \times \underbrace{\{0, 5, 10, \ldots, 195, 200\}}_{\text{lnorm}}$$

In my INEX submissions, I used a non-linear adaptation of Lucene's function (Dopichaj, 2006b) – elements shorter than about 50 tokens basically get an RSV of 0. This length normalization function leads to inferior results in all my experiments (in particular at higher ranks), so I do not include this version in my evaluation.

Tuning the length normalization is crucial to good performance, and what version is the best depends on the document collection. As figure 7.3 shows, for the IEEE collection, a soft threshold of 65 tokens yields the best results, whereas for the Wikipedia collection, a lower value of about 50 is better. This can be explained by the different typical lengths of the documents in the collections: IEEE articles are much longer than Wikipedia articles, so the relevant parts are also longer (but this might also be a side effect of the assessment procedure).

For INEX 2004, the results are significantly worse than the best official results; it is unclear what the reason is. For INEX 2005, the Lucene similarity measure can exceed the best official submission at rank 10 (my own submission also using Lucene with a different length normalization function). For INEX 2006, the best Lucene results are about 10 percent worse than the best submitted results.



(a) INEX 2004, element frequency

(b) INEX 2005, element frequency

(c) INEX 2006, document frequency

Figure 7.4: Parameter tuning for BM25; the darkness of each field corresponds to nxCG at cutoff rank 10. In each map, black corresponds to the maximum and white corresponds to 10 percent more than the minimum. The horizontal axis corresponds to b, from 0 to 1, and the vertical axis corresponds to k_1, from 1 to 5.

The results for the different global weighting functions are close to one another. This indicates that it does not matter whether document or element frequency is used with the Lucene similarity measure.

7.3.2 BM25 similarity measure

For BM25, length normalization is controlled by the parameters b and k_1. Permissible values for b are in the range $0 \dots 1$, where 0 means "no length normalization" and 1 means "maximum influence of length normalization". The larger k_1 gets, the closer the local term weight gets to the raw term frequency.

According to Spärck Jones et al. (1998), $b = 0.75$ and k between 1.2 and 2 work well on the TREC data, but it is unlikely that these parameter combinations can be transferred unchanged to XML retrieval. Theobald (2006) uses $k_1 = 10.5$ and $b = 0.75$, but the TopX approach is sufficiently different from mine to warrant further exploration.

The following parameter combinations should be tested (the full range for b and a reasonable range for k_1):

$$\underbrace{\{0.0, 0.1, \dots, 1.0\}}_{b} \times \underbrace{\{1, 1.5, 2, \dots, 4.5, 5\}}_{k_1}$$

Figure 7.4 shows the results for the three test collections. It is obvious that a good choice of parameter b is much more critical than a good choice of k_1. In general, lower values of b work better than higher values, with the exception of $b = 0$ (that is, no length normalization). Compared to the best parameter values for traditional information retrieval ($b = 0.75$ and $k_1 = 1.2$), the best value of b for element retrieval is much lower (somewhere between 0.1 and 0.2), so the influence of length normalization is reduced.

Each parameter space has a global maximum; the parameters for this maximum are close for the different test collections, but not identical. In particular, it is surprising to see that the best parameters for 2004 and 2005 differ noticeably.

The reason is that in my usage scenario, length normalization also fulfills the purpose of selecting the right result granularity (should a chapter or a paragraph be ranked higher?). What happens is that for maximum length normalization ($b = 1$), very short elements are pushed to the front of the result lists, typically leading to a list of section titles or titles of cited works. This is obviously a bad result. With length normalization completely disabled ($b = 0$), there is a strong bias towards the longest elements, that is, complete articles or their bodies. For values of b between the extremes, the results are much more balanced; they are a mixture of sections, complete articles, and other elements. Although an occasional title does occur in the top ranks, this is the exception rather than the rule and does not do much harm. In fact, if all elements of fewer than ten terms are removed from the results, retrieval quality drops dramatically.

The best choice for the global frequency function depends on the document collection: Element frequency is best for the IEEE collection, whereas document frequency is better for the Wikipedia collection.

Using element frequency as the global frequency consistently leads to higher results than using document frequency for the IEEE collection (2004 and 2005). Although this is consistent with the original formula, this result is somewhat surprising: Element frequency is not simple to interpret – terms that occur in deeply nested elements have a higher element frequency than terms that do not.

The explanation lies in a peculiarity of the BM25 formula: For terms that occur in more than half of all documents, the term weight w_i is negative so that the presence of these terms actually decreases the RSV:

$$w_i = \log \frac{N - \mathrm{df}(t_i) + 0.5}{\mathrm{df}(t_i) + 0.5} \tag{7.10}$$

To circumvent this problem, the term weight is generally set to 0 if it is negative, which means that these terms are treated as stop words.

In the IEEE collection, there are many terms that occur in more than half of the documents, so they cannot contribute to the RSV. There are, however, no terms for which the element frequency is high enough to obtain a negative weight, so this particular problem does not occur.

One might argue that terms that occur so frequently are useless for retrieval, but this is not necessarily the case for element retrieval: The terms “IEEE”, “volume”, and “computer” basically occur in all documents, so they have no discriminatory power at the document level. On the other hand, they may well be useful for element retrieval. For example, if a user searches for “IEEE conferences”, elements that mention both terms are likely to be relevant, but elements that only mention “conferences” will have a high rate of false positives.

For the 2006 data, the behavior of element and document frequency is roughly identical, with document frequency being slightly better. This discrepancy is somewhat puzzling: what characteristic affects this? In the Wikipedia collection, the

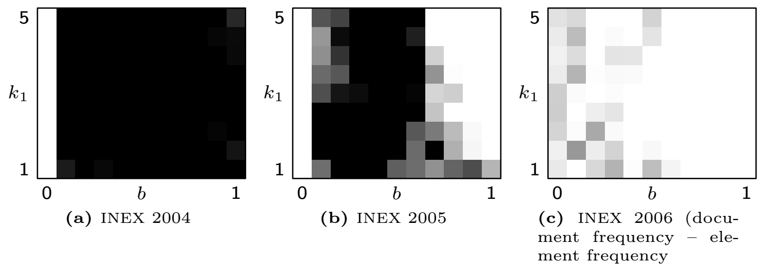


Figure 7.5: Choice of global frequency for BM25. The heat maps show the difference between the results for element frequency and the results for document frequency; each square corresponds to one combination of b and k_1. White squares denote no change or better results for document frequency, all other shades of gray denote the degree of improvement when using element frequency.

topics of the documents are more diverse, so there are no terms (apart from stop words) that occur in more than half of the documents, so the problem of negative term weights does not occur. The only outlier in this respect is the term “0”, which occurs in almost all documents’ header.

Figure 7.5 illustrates the effect of the global frequency for all tested combinations of b and k_1.

7.3.3 Comparison with the official submissions

I will compare the quality of the base retrieval engine with the maximum of all official submissions to that year’s workshop. That is, for each rank, the nxCG value averaged over all topics for each submission is calculated, and we use the maximum as the comparison run; the resulting curve does not correspond to a real run, but it gives us an indication of where the baseline stands with respect to the others. Lucene results are excluded because they are exceeded in all cases by BM25 results.

From the INEX 2005 results, one can see that unmodified BM25 already yields high-quality results, even compared to the official submissions. This is somewhat alarming, as it shows that the methods tailored to XML retrieval fail to be better than the general-purpose algorithms.

Further tuning resulted in the values presented in table 7.1. For INEX 2005, there is a noticeable increase in retrieval quality, whereas for INEX 2006, the increase is less pronounced. For INEX 2004, the optimum result of the base retrieval engine is significantly worse than the best submitted run. This is surprising, considering that the 2004 and 2005 collections basically use the same document collection. It should be noted, however, that the assessment procedure has changed between these rounds of INEX. Figure 7.6 shows the results for the 2005 and 2006 collections compared to the maximum of the submissions for all ranks and shows that the good

	Parameters			nxCG@10	
Test collection	b	k_1	gf	base	max
INEX 2004	0.08	1.5	ef	0.4669	0.5099
INEX 2005	0.20	1.0	ef	0.3368	0.3037
INEX 2006	0.18	0.8	df	0.4332	0.4294

Table 7.1: Best parameters and evaluation results for the different test collections. In all cases, the Lucene similarity measure yielded worse results. The "base" column displays the value for the base engine, the "max" column displays the maximum of all official submissions in that year. The maximum from 2005 is my own submission.

quality at rank 10 is not completely isolated.

In a real-world scenario, there are usually no relevance assessments available, so it is impossible to find the optimal parameter values. However, the values for the 2005 and 2006 test collections are close in magnitude although the collections are very different; thus, one can assume that these values are good starting points for other collections.

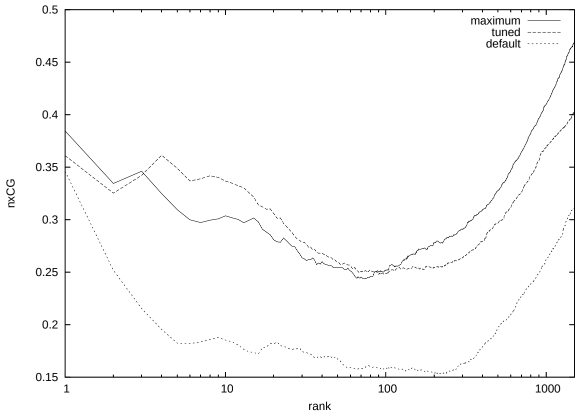
7.4 Evaluation of title-element exploitation

Now that the best baseline using plain BM25 has been established, I can examine whether exploiting title elements (see chapter 6) can indeed improve retrieval quality. Because of the time needed to run the evaluation, it is infeasible to check the complete parameter space for every variation of title detection and similarity adaptation. Thus, I will use the optimum parameter combinations from the previous section for the evaluation (preliminary test indicated that the maximum retrieval quality is achieved with the same parameters).

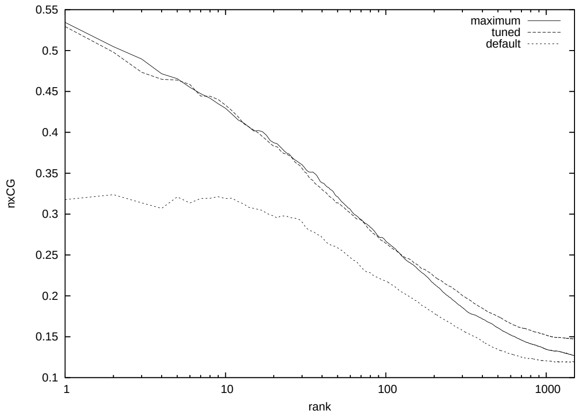
7.4.1 Analysis of potential

In order to find out whether small elements can indeed be helpful to improve retrieval quality, I will examine old INEX assessments. In contrast to the evaluation I will present in the following sections, this analysis does not depend on any specific retrieval algorithm; thus it gives a first indication of the potential of exploiting titles without (much) interference of the baseline search engine.

Although I try to be independent of the search algorithm, I still need to make an important assumption: The set of candidate results is only determined by keyword matching; any element that does not include any of the query terms is not considered as a result (but stemming is performed). Most search engines are based on this assumptions, notable exceptions are search engines that use some form of query expansion. In principle, it would be possible to perform the same experiments with a preceding query expansion step, but it seems unlikely that the results would differ significantly.



(a) INEX 2005, tuned is $b = 0.2, k_1 = 1$. The base retrieval engine is better than the best submissions up to about rank 100 (with the exception of the top ranks). Below that rank, performance gets significantly worse, possibly due to the pooling problems.



(b) INEX 2006, tuned is $b = 0.18, k_1 = 0.8$. Although the baseline does not quite reach the top status, it is close.

Figure 7.6: Two of the base BM25 runs compared with the maximum run ("maximum"). The BM25 run with $b = 0.75$ and $k1 = 1.2$ ("default") shows what can be achieved without parameter tuning and the "tuned" BM25 run shows the best parameter combination for the test collection.

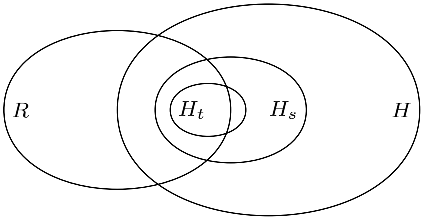


Figure 7.7: The sets of elements used in the analysis of potential: The relevant elements R, the retrieved elements H, the retrieved section elements H_s, and the retrieved section elements with matches in their titles H_t.

Thus, the following sets should give an indication of what can be expected of exploiting title elements (see also figure 7.7):

- The assessments provide information about relevant elements for the queries in the test collection, yielding the set of relevant elements R.
- Results H from a completely indexed collection.
- Results constrained to section elements H_s.
- Section elements with hits in title elements H_t; titles are determined by name-based title detection.

Naturally, the candidate set for title elements is a subset of the complete candidate set, so it is not possible to get improved recall (indeed, recall can be expected to get worse). This does not mean that recall will worsen in the actual retrieval scenarios; the elements in H_t is merely intended to be placed higher in the result list. On the other hand, if precision is improved – that is, if the fraction of relevant elements is higher in H_t than in H –, this yields a good indication that the overall principle is sound and can be exploited in search engines. To make sure that the increase in precision is not simply because section elements have a higher chance of being relevant than random elements, I will also compare to the set of hits in section elements H_s.

The candidate sets were constructed as follows: For the complete candidate set, all elements in the collection were indexed, a keyword-based search was performed, and all elements with a non-zero score were included in the set. The candidate sets exploiting small elements were created in two steps: First, a search on an index including only the small elements was performed. Second, the title elements were replaced by their parent section elements.

The precision was determined by dividing the number of relevant elements (that is, elements with a non-zero exhaustiveness in the assessments) by the total number of elements retrieved:

$$\frac{|H_x \cap R|}{|H_x|} \tag{7.11}$$

	INEX 2004		INEX 2005		INEX 2006	
Set H_x	Relevant	Precision	Relevant	Precision	Relevant	Precision
H	17,755	0.00264	12,631	0.01408	54,323	0.00673
H_s	4,362	0.00320	3,480	0.02228	8,050	0.00571
H_t	1,153	0.01752	721	0.04917	2,899	0.01896

Table 7.2: Precision average over all assessed topics. The count of relevant elements is $|H_x \cap R|$, precision is $\frac{|H_x \cap R|}{|H_x|}$.

Table 7.2 shows that the average precision increases significantly. Sections with hits in titles are more than five times as likely to be relevant for the 2004 data set, more than two times for the 2005 and 2006 data sets. A per-topic comparison of precision confirmed that the difference is indeed significant; with the exception of a single topic for the 2005 data, *all* topics get a higher precision when sections with hits in titles are retrieved.

Of course, these numbers do not actually say much about the retrieval quality, but they show that matches in titles can indeed be useful hints to find relevant sections. With the simple binary relevance scale, nothing is said about the degree of relevance, and of course the sets are unordered. The exact effects on retrieval quality can only be determined if the title detection is combined with adaptation functions to obtain rankings to be evaluated.

7.4.2 Adaptation methods

First, I will examine the various adaptation methods, as introduced in section 6.3, using name-based title detection. The adaptation methods introduced in chapter 6 (simple, restrictive, dynamic, adding, and term-frequency-based) will be evaluated with various settings for the corresponding parameters, where applicable. For *simple* and *restrictive*, the parameter $o \in \{1.1, 1.2, 1.3\}$ is used, for *dynamic*, $o \in \{0.1, 0.2, 0.3\}$ is used (due to the different interpretation of o, these values result in similar changes). For term frequency adaptation, I used $k = 1$.

There are two parts of the results that are important regarding the result quality:

- The mean nxCG value at rank 10, averaged over all topics.
- The number of topics where the nxCG value at rank 10 is better or worse compared to the baseline.

Both parts are important to evaluate the changes introduced. If a single topic is worse by a large margin and all other topics are slightly better, the overall average may be the same although there is a drastic difference.

Table 7.3 shows the results for the retrieval quality as measured by the nxCG measure at rank 10. For the 2004 test collection, every change appears to decrease retrieval quality; at best, it stays the same as for the baseline. Likewise, no noticeable improvement of retrieval quality can be seen for the 2006 collection;

Adaptation method	o	nxCG@10
best submission		0.5099
base		0.4669
tf		0.4655
restrictive	1.1	0.4603
dynamic	0.1	0.4600
dynamic	0.2	0.4582
restrictive	1.2	0.4565
restrictive	1.3	0.4470
simple	1.1	0.4445
dynamic	0.3	0.4351
simple	1.2	0.4273
simple	1.3	0.3938
adding		0.3935

(a) INEX 2004

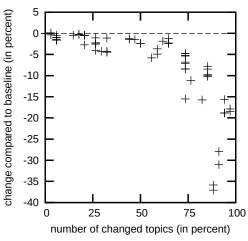
Adaptation method	o	nxCG@10
restrictive	1.2	0.3460
tf		0.3436
dynamic	0.1	0.3433
restrictive	1.1	0.3429
restrictive	1.3	0.3416
dynamic	0.2	0.3397
base		0.3368
simple	1.1	0.3355
dynamic	0.3	0.3332
simple	1.2	0.3153
simple	1.3	0.3079
best submission		0.3037
adding		0.2701

(b) INEX 2005

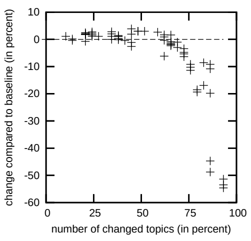
Adaptation method	o	nxCG@10
tf		0.4350
base		0.4332
best submission		0.4294
dynamic	0.1	0.4278
restrictive	1.1	0.4273
restrictive	1.2	0.4194
simple	1.1	0.4159
dynamic	0.2	0.4138
restrictive	1.3	0.3985
dynamic	0.3	0.3968
simple	1.2	0.3911
simple	1.3	0.3618
adding		0.3242

(c) INEX 2006

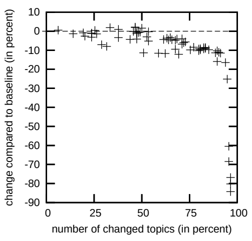
Table 7.3: Retrieval quality for the different adaptation methods, using name-based title detection. Also listed are the base result and the best official submission result. Results are sorted according to the nxCG measure, with best results at the top of the lists. Obviously, there are few combinations of adaptation method and parameter that are better than the base result. Apparently, exploiting title elements does not work for the given retrieval methods.



(a) INEX 2004



(b) INEX 2005



(c) INEX 2006

Figure 7.8: Change of retrieval quality against percentage of changed topics. Each point stands for a combination of title detection strategy and adaptation method. Points above the dashed line indicate an improvement over the baseline, points below it indicate a reduction of retrieval quality.

term-frequency-based adaptation yields a minuscule improvement that can probably be attributed to random effects. Only for the 2005 test collection, there is an improvement that might be significant – *restrictive* adaptation with $o = 1.2$ yields a 2.7-percent improvement over the base result. This change, however, is probably too small to actually increase user satisfaction (if it is even noticed).

Among the adaptation methods, *adding* and *simple* adaptation generally lead to reduced retrieval quality, and *term-frequency-based* adaptation is the adaptation method with the best results if all three collections are considered.

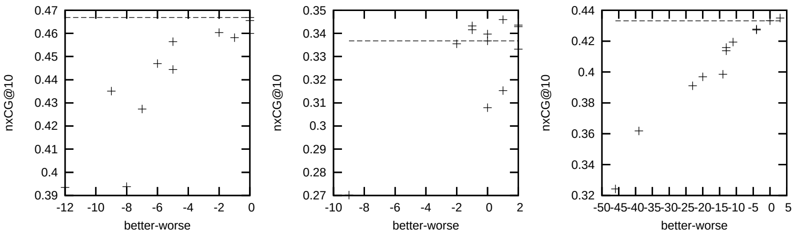
Figure 7.8 shows that parameter combinations with a large number of changed topics (close to 100 percent) lead to a significant degradation of retrieval quality in the given scenario. This appears to imply that for some of the topics, nothing can be gained if titles are exploited. Of course, this does not mean that the base results are already perfect for these topics, but it is a strong indication that other means than title exploitation are needed to improve them.

Average nxCG values over all topics are certainly an important measure, but it is also important to see how many topics were improved or degraded. Figure 7.9 shows that a clear correlation only exists for the INEX 2006 data; for the INEX 2005 data, there is clearly no correlation.

7.4.3 Section-based evaluation

One possible cause of the lack of improvement is that section results are only a small subset of the result set. Only sections are affected by exploiting title elements, so if there is no section element close to the top ten, nothing will change. To see if this is the case, I will now examine results where all elements but section elements have been removed; this evaluation will show whether at least sections alone are reordered usefully by exploiting titles.

Figure 7.10 shows that this is indeed mostly the case. Points above the dashed line



(a) INEX 2004 **(b)** INEX 2005 **(c)** INEX 2006

Figure 7.9: Correlation between average retrieval quality at rank 10 and the difference between improved and degraded topics. A horizontal coordinate of 2 means, for example, that two more topics were improved than were degraded. The dashed line indicates the baseline retrieval quality. There appears to be a strong correlation for 2006. For 2004, the correlation is less clear, and for 2005, there does not appear to be a correlation.

indicate that a given combination of adaptation method and parameter improved the results for section-based retrieval. Most interesting are the top-left and bottom-right sections of the diagrams. Points in the top-left section are points that lead to worse retrieval quality if all results are considered, but to better retrieval quality when only sections are considered, and the converse is true for bottom-right points. For all collections and adaptation methods, there is only one combination that falls in the bottom-right section: *restrictive* adaptation with $o = 1.1$. Many more combinations are now improvements.

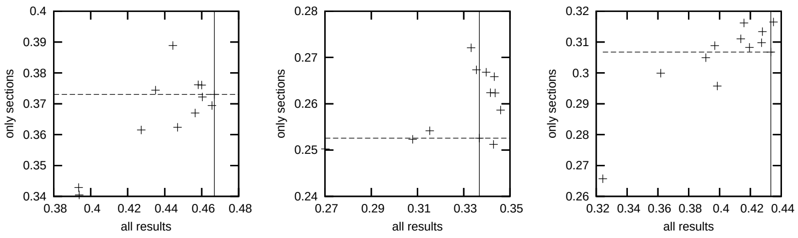
Overall, an improvement of retrieval quality is now possible for all test collections: For 2006, a 3-percent improvement is achieved using *term frequency* adaptation, for 2005, a 7.7-percent improvement is achieved using *dynamic* adaptation with $o = 0.3$, and for 2004, a 4-percent improvement is achieved with *simple* adaptation and $o = 1.1$. Overall, better results are possible for all collections using *simple* adaptation and $o = 1.1$ (even if not the maximum).

This shows that the sections are reordered to yield a better result, but there are non-section elements high in the result lists that are more relevant than the sections.

7.4.4 Stability

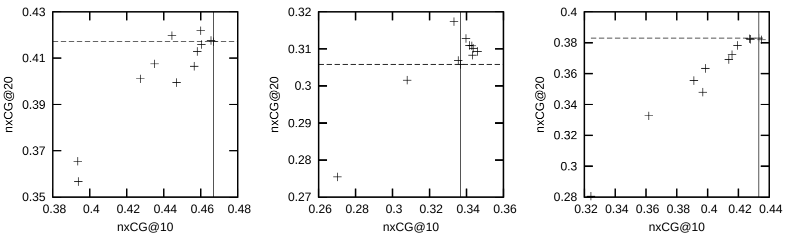
Another interesting aspect is the stability of the results. So far, all evaluations were based on nxCG at rank 10; if the methods and the evaluation were stable, then the relative order of the adaptation methods at rank 20 should be the same (even if the absolute nxCG values change). However, as figure 7.11 shows, this is not the case. Only for the 2006 test collection, there is a reasonable correlation between the scores at these ranks, for the other test collections, the behavior is far from clear.

Again, the overall improvement is best for 2005 data: with one exception, the



(a) INEX 2004 **(b)** INEX 2005 **(c)** INEX 2006

Figure 7.10: Comparison of all results versus only section results, based on nxCG@10. Each point represents a combination of adaptation method and parameter; the base result is at the intersection of the lines.



(a) INEX 2004 **(b)** INEX 2005 **(c)** INEX 2006

Figure 7.11: Comparison of retrieval quality at ranks 10 and 20. The base result is at the intersection of the lines; points to the top right of this point are better both at rank 10 and at rank 20. If the results were stable, the points should form a straight line; clearly, this is not the case.

		Percentiles		
Collection	Median	75th	95th	Maximum
INEX 2004 (st)	3	4	6	44
INEX 2005 (st)	3	4	7	44
INEX 2006 (title, name)	2	2	4	6,862

Table 7.4: Statistics of the lengths of title elements in terms (excluding stop words). In general, titles are shorter than 8 terms, with few exception. The improbable maximum value for the Wikipedia collection (2006) is caused by a conversion error.

adaptation methods that are better at rank 10 are also better at rank 20, but the order of them changes. Furthermore, the largest improvement at rank 20 (4.7 percent) is achieved by the simple adaptation method with $o = 1.2$, which was previously 6.4 percent *worse* than the base retrieval engine.

All in all, the results are too inconsistent to draw any definite conclusions, but no major improvement with any of the adaptation methods can be observed for the INEX test collections.

7.4.5 Title detection strategies

From the results, one can see that exploiting titles yields a small increase in retrieval quality using the best parameters for the base retrieval engine. This improvement is probably too small to be noticeable to the end user.

Even so, I will now examine the behavior of the length-based title detection strategy. In other settings, the exploitation of title elements has shown its potential, so it is useful to find out whether a schema-independent approach to title detection works.

First, the length threshold has to be determined. Titles are typically short; table 7.4 shows what lengths occur in the collections. For all test collections, 95 percent of all title elements are at most seven terms long (excluding stop words), with a median of two or three. Thus, I will test the length thresholds 2, 4, 6, 8, and 10.

Surprisingly, length-based title detection outperforms name-based title detection in many cases. The difference is marginal and certainly not statistically significant; however, it shows that length-based title detection can indeed be a viable alternative to name-based title detection.

The differences in retrieval quality compared to name-based title detection can be explained by the inevitable errors made by the heuristic. Table 7.5 shows that length-based title detection does not indeed detect only titles; in fact, for the IEEE collection, less than a quarter of the elements detected as titles are in fact st elements. However, with the exception of extremely short paragraphs (about 12 to 13 percent), all top-rank elements are some form of highlighting (bold, italics, the title of the journal ti). For the Wikipedia collection, title detection works considerably worse: The actual title element is only on rank 2, with about 13 percent

	INEX 2004		INEX 2005		INEX 2006	
Rank	Element	Rel. freq.	Element	Rel. freq.	Element	Rel. freq.
1	st	0.22	st	0.23	collectionlink	0.33
2	p	0.13	p	0.12	title	0.13
3	b	0.11	b	0.10	unknownlink	0.12
4	it	0.09	it	0.09	item	0.08
5	ti	0.07	ti	0.07	outsidelink	0.06
count	82		82		92	

Table 7.5: Element names detected as titles with length-based title detection, threshold 4. Row "count" is the total number of distinct element types detected as titles.

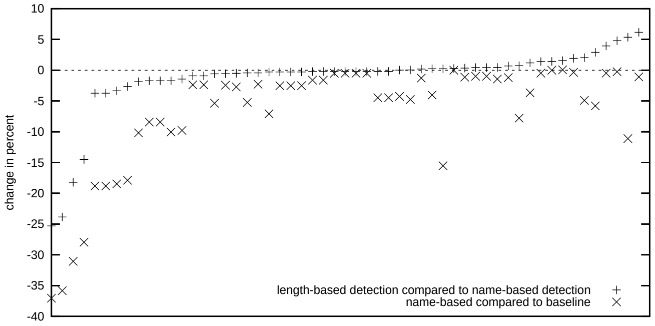


Figure 7.12: Relative change of retrieval quality of length-based title detection compared to name-based title-detection (only INEX 2004). Each point stands for a combination of adaptation method and length threshold. The situation is similar for 2005 and 2006.

relative frequency; three out of the top five are some form of link elements, which together account for more than half of all detected titles.

These numbers are, of course, not a direct indication of retrieval quality: if the elements wrongly detected as titles belong to an element with a low RSV, their detection as a title will not do much harm.

Figure 7.12 shows that using length-based title detection does not affect retrieval quality much for most adaptation methods, no matter what length threshold is used. The exception is the additive adaptation method; this method, however, already leads to bad results. One might think that the closeness of name-based and length-based title detection results from the fact that name-based title detection does not change the results very much. However, as the figure shows, the difference between length-based and name-based title detection is small even if there is a large difference between name-based title detection and the base result.

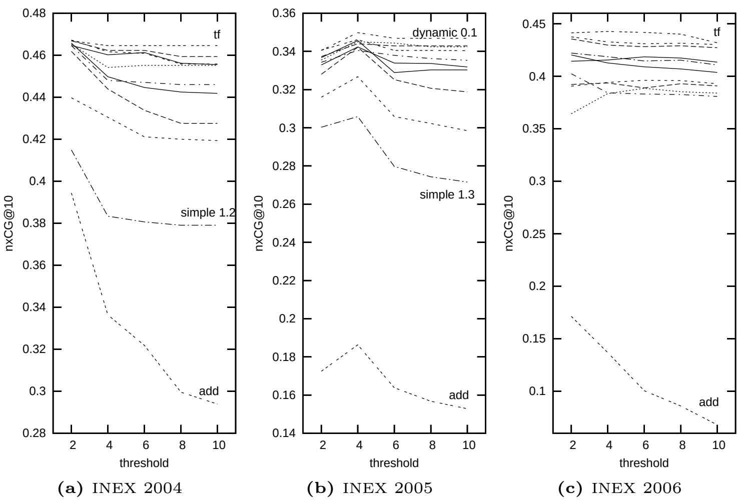


Figure 7.13: The influence of the length threshold on retrieval quality. Every line represents an adaptation method and parameter.

The exception is addition-based similarity adaptation; if this method is used, retrieval quality drops significantly (more than 10 percent). However, this adaptation method does not work at all anyway, so it does not matter much that bad results are made worse.

The influence of the length threshold on the result quality is interesting, as figure 7.13 shows. In general, lower thresholds work better for adaptation methods that degrade results (in particular the addition-based adaptation method). This is unsurprising, because the lower thresholds match fewer elements, so the result is changed less.

The INEX 2005 test collection differs from the other collections in that it has a pronounced peak at length threshold 4. Considering that this is the only collection on which title exploitation with name-based title collection really worked, this value appears to indicate that there is indeed a best value for detecting titles. Naturally, this value is likely to be collection-specific.

7.4.6 Per-topic evaluation

The results show that title exploitation does not always work on the given test collections. To see why that is, I will now look at two selected topics where this method degraded retrieval quality.

One topic where exploiting title elements does not work at all is topic 205 from 2005. The information need is stated as follows:

> I am writing an essay on the influence of new media icon Marshall McLuhan on digital technologies. I'm seeking information describing how McLuhan's views have influenced current digital technologies. To be relevant, a retrieved item should discuss some aspect of Marshall McLuhan's visionary ideas or famous one-liners in the context of current digital technologies. Retrieved elements that merely cite some of McLuhan's work are non-relevant, as are elements that discuss ideas not originating from McLuhan.

The corresponding query is "marshall mcluhan", and it is clear that these query terms are unlikely to occur in titles. If someone discusses McLuhan's works, it will most likely occur in a section like "Related work". Thus, it is no surprise that title exploitation does not help in this case. In fact, it even hurts because of bogus matches in sections about "marshaling" (they match because of stemming).

Another topic that does not work at all is topic 291 (INEX 2006). This topic's content-only query is "Olympian god or goddess", but the information need is really "I want some figures showing a representation of an Olympian god or goddess". It is no surprise that the results for this query are bad, because only figures are considered relevant. Exploiting title elements is likely to degrade retrieval quality because it pushes sections to the front, and sections are definitely irrelevant. This information need is really not suitable for a content-only query.

Thus, there are many reasons why title exploitation may fail. The query may demand elements that are not sections (topic 291), or it may be that the query terms are unlikely to occur in titles (topic 205). In many cases, it is simply the preference of the searcher that determines whether title exploitation may work. Some searchers regard sections as a good retrieval units, whereas others favor complete articles or paragraphs. Unfortunately, it appears to be impossible to predict from the query alone whether title exploitation may work, so the only feasible solution is to let the searcher decide.

7.5 Discussion

It is surprising to see how well a simple adaptation of standard information retrieval techniques can work for XML retrieval. Simply indexing all elements as if they were documents and applying BM25 with the right parameters can lead to better results than the best official submissions. One should keep in mind that the optimal parameters were determined after the fact by evaluating a large range of combinations on the assessed test data; the real submissions do not have the advantage of this fine-tuning.

On the other hand, the best parameters are very similar for the INEX 2005 IEEE collection and the Wikipedia collection, and minor deviations from the optimal results do not decrease retrieval quality much. Considering that these collections are very different from one another, it seems plausible to assume that using $b = 0.2$

and $k_1 = 1$ will work reasonably well in other situations. It is surprising that the best parameters are different for the INEX 2004 collection, which is almost identical to the 2005 collection. It is not clear what the reason is, but it should be kept in mind that I used an evaluation measure that was not official back then.

Name-based patterns have shown their potential as an addition to a language modeling retrieval system, as demonstrated by Ramírez et al. (2006a,b) on the INEX 2005 data. In previous publications, I have demonstrated that heuristic approaches, too, can improve retrieval quality (Dopichaj, 2006b, 2007a,b) on the same test data.

The results in the context of this thesis, however, indicate that these findings do not apply in all circumstances: Although small elements can improve the retrieval quality a lot if they are applied to medium-quality baseline methods (Ramírez et al. (2006a) report a maximum nxCG value of 0.3 at rank 10, whereas the base method in this thesis exceeds 0.33), they offer little benefit if the baseline is well-tuned.

On the INEX 2005 collection, a maximum improvement of the nxCG value of about 3.7 percent can be observed, using length-based title detection with a threshold of 8 and a boost factor $o = 1.1$. This improvement is presumably not noticeable to a user. In any event, the results are not statistically significant at any acceptable p level. For 2004 and 2005, the number of topics is too small to make any definite statements. For 2006, there are barely enough topics for a useful evaluation, but here, the results are far from convincing; one should keep in mind that not all of the assessed topics are actually useful for exploiting title elements.

The situation is slightly better if section-based retrieval is performed, but this only shows that sections are not the best retrieval results for the test collections at hand.

The results clearly do not show that the retrieval enhancements presented in this thesis can improve retrieval quality significantly. However, the evaluations cannot be considered a final indication of the potential of these approaches. In the remainder of this section, I will discuss in how far the choice of the test collections could affect the results and briefly describe the characteristics of an evaluation setup that better matches the intended use cases.

7.5.1 Suitability of the test collection

The evaluation shows that the proposed enhancements do not lead to a significant increase of retrieval quality. Keep in mind, however, that the test collections and evaluation metrics that are used at the INEX workshops do not entirely reflect the intended application area, and other potential problems may affect the results:

- Both the IEEE articles and the Wikipedia articles are rather short and self-contained so that it is unlikely that a fragment of such an article is more relevant than the article itself.

- The two collections differ in so many aspects that it is impossible to attribute the difference in retrieval quality to a single difference.

- The assessment process is not the same in different years, which makes it hard to do a comparison.

- Relevance assessments are generally subjective; in the cases where several people assessed the same topic, the assessments were quite different (Trotman, 2005; Pehcevski and Thom, 2006).

- Runs that are evaluated, but were not included in the pooling process may suffer if they retrieve elements that are not in the pool. Whereas this effect has been shown to be minimal in the context of TREC (Zobel, 1998), no study has been made in the context of INEX, but problems have been reported (Trotman et al., 2007a).

- The assessment interface differs from what a user of the retrieval system would see; it does not use ranking and is document-based, so the relation to real-world scenarios is unclear.

The last point needs further explanation: The unranked presentation of the results is inherent to the pooling approach that has successfully been used for traditional information retrieval evaluation for years. In the context of element retrieval, however, there is the problem that the pool does not reflect the retrieval results. Even if the pooled results only contain a single paragraph from a document, the assessor must assess the complete document. This in itself is a minor technical problem, but it seems likely that the assessment can be different from the assessment that would be obtained if the isolated paragraph were presented; if the paragraph is shown in the context of the document, the assessor may – consciously or not – use this context to rate the element's relevance.

It is clear that even the INEX organizers and participants have not yet reached consensus on how to evaluate the effectiveness of XML retrieval systems: Through the years, various metrics were adopted and abandoned, and even the basic retrieval tasks for the ad-hoc track are far from being fixed (INEX 2007 dropped the *thorough* task, which previously was the only task that had been done in every year). This is not avoidable, considering that XML retrieval is still a relatively young research area, but the lack of clear definitions makes it hard to do meaningful comparisons between systems.

In general, it is questionable whether the results from batch evaluations – as done in the INEX ad-hoc track – contribute to user satisfaction. Hersh et al. (2000) compare several systems' performance on TREC data in batch and interactive experiments and come to the conclusion that there are significant differences in the results. In XML retrieval, the differences are likely to be even more pronounced, because the assessment user interface displays the results in a different fashion than an XML retrieval system would – the element results are shown in the context of the complete document. This is likely to affect the assessment: the users can take the surrounding material into account when judging the relevance of an element.

Buckley and Voorhees (2000) discuss what it takes to draw conclusions with a sufficiently low error rate. The retrieval scenarios in this thesis are closest to their notion of web retrieval – it is very difficult to know how many relevant documents exist in total, so precision at a cutoff level of 10 to 20 should be used. In this scenario, precision is replaced by nxCG, but the reasoning is the same. To achieve a reasonable error rate, they suggest using 100 queries, which implies that only INEX 2006 data can be used to obtain reasonable conclusions (2004 and 2005 together have only 63 queries); unfortunately, the IEEE collection more closely matches the assumptions made in this thesis.

Overall, even document-based retrieval evaluation has problems, despite having a rather long tradition. For INEX, the problems are amplified by a number of new problems, partly specific to XML, partly due to the resources being much more limited than for TREC. Evaluations in INEX data are certainly far from worthless, but they should be interpreted with care.

7.5.2 Design of a more suitable experiment

The current INEX test collections are not ideal for evaluating the effects of title elements, and no suitable test collection exists at the moment. Thus, a new test collection will have to be designed for more meaningful evaluation, and the following aspects have to be considered:

- The *test collection* should consist of larger works whose sub-parts are understandable in isolation, like collections of technical books. Such collections actually exist (Dopichaj, 2006a), but they are of commercial value, so it will be hard to get access to them for research.

- The *authors* of the documents in the test collection should take care to choose meaningful titles and generally use the markup sensibly. For commercial books, this will usually be ensured by the editors.

- The *users* should have enough background knowledge to understand isolated chapters or sections from the document, and their queries should be focused enough that a single section can satisfy the information need.

- The *evaluation process* should reflect the satisfaction of the user based on the results. Thus, it should take into account the way results will be presented to the user.

The experimental setting described here is probably infeasible at the moment, but it might eventually be possible to incorporate this in INEX.

7.6 Summary

This chapter described the basics of XML retrieval evaluation and applied the evaluation methods to my base retrieval engine and the extended version. The results are somewhat surprising: contrary to previous published results, exploiting title elements does not lead to a significant improvement of retrieval quality (although minor improvements can be seen). This is mostly caused by the good base retrieval engine – it is astonishing how good the results are if standard information retrieval techniques are applied. Furthermore, I have discussed various reasons why evaluations based on the INEX test collection may not be entirely meaningful in the context of this thesis and described how a more suitable experiment could be designed.

8 Using background knowledge for content-and-structure search

"You see I have a lot of special knowledge which I apply to the problem."

(A Study in Scarlet)

Although the main topic of this thesis is content-oriented XML retrieval, this chapter briefly discusses preliminary approaches to content-and-structure XML retrieval. In particular, I focus on the application of *background knowledge* about the document collections: If further knowledge about the schemas is available and it is feasible to perform manual modeling of similarity measures, improved retrieval quality can be expected. First, this chapter discusses potential uses for specialized similarity measures in section 8.1 – in particular, similarity measures from case-based reasoning – and then describes a concrete example on how a specific similarity measure can be applied to XML retrieval in sections 8.2 and 8.3. Section 8.4 then describes why the available test collections are not suitable for a quantitative evaluation.

8.1 Specialized similarity measures

So far, this thesis has focused on the document-centric parts of XML documents – after all, they are the most important part of the digital libraries that are the main use case for this thesis. However, even texts typically have data-centric parts, for example the information about the author and the publisher, or even information items in ordinary text, contained in special markup.

Most XML retrieval engines only use the XML documents themselves and possibly their schemas for indexing and retrieval, because this information is readily available. If, however, the users of the system can provide further background knowledge, this could be used to better match the users' concept of similarity (Dopichaj, 2004):

- The retrieval engine can use explicit knowledge about the application domain.
- It can collect data on the system usage to adapt itself to the user base as a whole or to specific users.

8.1.1 Current state of XML retrieval

The various standards for describing the structure of XML documents – for example, DTD, XML Schema, and Relax NG – provide means to restrict the structure of valid

XML documents. For example, they allow the specification of permitted parent-child relationships of element types or what type of character data a given element may contain (for example, free text, integer, or date).

These descriptions are merely syntactic, however: They only specify the structure and permitted data types, but no meaning. This is enough information for parsing an XML document, but for assessing the similarity of documents and queries, it can be important to know whether an integer is a person's age or his weight.

Most of the current XML retrieval engines, including the one presented in this thesis, have virtually no provisions for using background information: They either regard all "textual" XML contents – even if they are specified to be a number by the schema – as a sequence of tokens and compare them using information retrieval techniques, or they have different methods of similarity calculation for the schema-specified data types.

8.1.2 Applying CBR similarity measures to XML retrieval

The XIRQL query language (Fuhr and Großjohann, 2001), an extension of the XML Query Language (XQL), goes further than that: It introduces data types and corresponding *vague predicates* for calculating similarity at the element level. These low-level similarities are then propagated upwards in the document tree by *augmentation.*

These concepts resemble the concept of similarity measures from (structural) CBR (Lenz et al., 1998; Richter, 2003). Structural CBR is based on structured data in attribute–value form and applies the local-global principle to calculate object similarity: First, the similarities of the objects' details are calculated using *local* similarity measures – corresponding to vague predicates –, and the results are combined into a *global* similarity for the objects using *amalgamation functions* – comparable to augmentation.

The CBR community has considerable experience in creating and refining similarity measures, hence it is worthwhile to transfer these techniques to the field of semi-structured retrieval. The following list highlights a few research results that are worth considering:

- Similarity measures for many data types and application domains have already been developed.
- Basic building blocks and methods for developing new similarity measures are available.
- Methods for learning local similarity measures from user feedback have been developed (Stahl and Gabel, 2003).

Despite the striking parallels, there are differences that have to be considered before these techniques can be applied to XML retrieval. For example, the structure of typical XML documents is less rigid than in structural CBR, which poses the problem of selecting the proper local similarity measure for an element based on context. This might be achieved by specifying XPath expressions for selecting similarity measures (for example, //person/age versus //planet/age), but how broad or narrow these paths should be needs further examination.

In summary, CBR has a lot to offer for improving XML retrieval, but further research is needed in order to integrate the techniques seamlessly into the existing XML retrieval engines. In the remainder of this chapter, I will discuss a specific example of a CBR similarity measure that can be applied to XML retrieval (Dopichaj, 2005).

Many existing approaches use the XML markup to some extent. Markup can be used at several levels in an XML document schema: Block-level markup can be used to embed metadata (like authors' names) and to represent the structure of the document; examples include body in (X)HTML and section in DocBook (Walsh and Muellner, 1999). Inline markup is used on single words or (short) phrases to convey the meaning or intended representation of the marked-up contents.

Inline markup can be useful for indexing and similarity calculation, because it may hint at the correct way to interpret an element's contents; making good use of inline markup is the main focus of this chapter.

8.2 Generalizing content-and-structure queries

This section will motivate why there is a need to make better use of inline markup in content-and-structure XML retrieval. I do this by providing several example scenarios that are inadequately supported by the existing query languages and retrieval engines. The context of all scenarios is the collection of Linux Howto documents collected by the Linux Documentation Project (TLDP)[1]. The documents are marked up in DocBook, an XML- (and SGML-) based markup language for the creation of computer-related texts.

Example 1 *Adam does not know the details of DocBook markup, but he can distinguish various basic types of search terms. When he searches for information about the command shell* `bash`*, he should be able to specify that the word* `bash` *is only relevant if it is used as a technical term; in particular, it is of no interest if "to bash" occurs in normal text.*

Example 2 *Betty knows DocBook well, but she is interested in a higher level of abstraction, because she knows that any of several element types might contain the relevant text. For example, when she searches for information about* `save`*, she is interested in* commands *and* menuitems *(among others), but not on hints about saving paper.*

The users in these examples would benefit from a level of detail between simple keyword-based search and complex XML path queries. Typical query languages do not support this intermediate level: Either they are purely keyword-based or they require detailed knowledge of the relevant tag names, like XIRQL, XQuery, and NEXI.

[1] http://www.tldp.org

Example 3 *Charlie has performed a search and found a document fragment that almost, but not quite, satisfies his information need. He proceeds to search for similar documents.*

No major XML retrieval engine directly supports documents as queries. It is possible to transform the document to a query, but this will lead to one of two problems:

- The converted query is too specific and matches only the original document (if the markup is converted to XPath constraints); this can easily happen if the input document is short and contains detailed markup.
- The converted query is too general so that the semantic information contained in the markup is lost.

What is needed for good results in this example is a retrieval engine that supports some form of fuzzy element matching.

Example 4 *Book author Dorothy wants to mention the shell* `bash` *in her text, but she is not sure whether* productname *or* application *is the appropriate markup.*

Given the wealth of elements provided by DocBook, it is not surprising that the semantics of some elements are very similar, so it is frequently hard to choose. Another problem lies in the authors' laziness or less than perfect knowledge of DocBook: An examination of the Linux "Howto" documents revealed that technically incorrect markup is fairly common. Even the DocBook reference concedes that this problem exists: "`Emphasis` is often used wherever its typographic presentation is desired, even when other markup might theoretically be more appropriate." Because of this, retrieval engines should support approximate matching of elements.

Example 5 *Eric considers the amount of markup necessary for something as simple as command-line input to be excessive and omits all but the top-level tags.*

This is a real problem at least in the Linux Howtos; the reason for this 'lazy' markup is probably the high number of semantic markup options available to the author that cause work without much apparent benefit (the rendered presentation might not change anyway). It is unrealistic to expect the retrieval engine to reconstruct the missing elements, but it can make sure that equivalent fragments with complete and incomplete markup compare almost equal.

8.3 Element relationship

In order to address the problems mentioned in the previous sections, I introduce the concept of *element relationship* which allows us to partially substitute elements with other, similar elements in the retrieval process.

The first two examples from the previous section illustrate that searchers need a level of abstraction above that of element names: In both cases, the searchers were

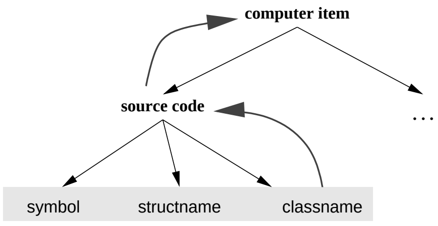


Figure 8.1: Gradual generalization when searching specific element types.

willing to supply details on the markup structure, but not at that level of detail. It seems reasonable to form groups of related tags and offer an input field for each of them. The number of groups should be small, because otherwise it is still too complicated. For DocBook, the following list might be a reasonable starting point:

- Computer-related text (for example, user input, program output and listings)
- Emphasized text (for example, text marked as emphasized, keywords and index terms)
- Metadata (for example, author and revision information)
- "Normal" text (everything else)

These categories are not necessarily free of overlaps, but as later sections will show, this poses no problem and can indeed be used to our advantage. The result is an interface that is still usable without having to learn a complex query language, but offers more power than simple keyword-based languages.

If an existing document is used as a query ("more like this"), the query obviously contains elements instead of categories. Due to the problems with ambiguous or misused markup, searching for element contents *only* in elements of the same type may lead to omitting many good matches. On the other hand, simply searching for the contents in *all* text contents in the documents, no matter what markup is used, sacrifices precision.

In this case, it is useful to gradually generalize the element, that is, matches contained in the same type of element receive the highest score and going up in a hierarchy of categories reduces the score. Figure 8.1 illustrates this principle: A search for text marked up with classname would first search all classname elements, then (at reduced score) all **source code** (symbol, structname, . . .), then all **computer items**, and so on.

8.3.1 Facets of element similarity

The question that arises at this point is: What can the element grouping be based on? There is no single aspect that can be used in isolation to calculate the similarity of two element types. Instead, there are several, somewhat related options:

Tag names. Ideally, element names should convey their meaning without any further information; XXL (Theobald and Weikum, 2002) makes this assumption and uses a separate ontology to relate these names. In my experience, meaningful names (that is, names that correspond to unabbreviated words) for XML tags are the exception rather than the rule. Very often, cryptic abbreviations like qandadiv from DocBook or st from the IEEE collection are used, and considering that even humans have problems interpreting these names without further information, it seems unrealistic to expect computers to manage that task.

Syntactic restrictions. XML schema languages like DTD, XML Schema, or Relax NG provide provisions for defining the syntactic structure of the documents in that schema, in particular the permitted nesting of elements. In DocBook, for example, the element copyright may only contain the elements holder and year. This information is easily parsable, but its use for the ERG is limited: In most cases, either all inline elements are allowed as sub-elements or none.

Semantics. Considering the previous remarks, it appears to be necessary to use further information to establish semantic relations between element types, for example, grouping related element types (see figure 8.1). This information is typically available in the form of documentation aimed at authors of documents, but actually making use of that information can be very time-consuming.

Contents. If a significant number of documents is available, one can use statistical methods based on the contents of the XML elements. One simple approach would be to use statistics of character classes like upper/lower case letters, digits, etc. to differentiate the element categories; for example, UNIX paths typically contain a disproportionate number of slashes ("/"). More sophisticated approaches could use the words both in the element and in its context in order to obtain classifiers.

Visual appearance. Normally, document-centric XML is meant to be rendered for presentation to the user. The number of semantic inline tags typically exceeds the number of available formatting options of the output format, so many tags are represented in the same way. While much of the semantics contained in the markup is lost, the mapping is not arbitrary: Even though several unrelated tags might be represented in the same way, *related* tags usually have the *same* formatting. In DocBook, for example, the computer-related entities like filenames, computer input/output, and environment variables are all likely to be rendered in a fixed-width font. The transformation from XML to the rendered representation can be specified in XSLT style sheets.

Each of these aspects can be used as the basis for a similarity measure comparing two elements. Instead of creating a similarity matrix containing the similarities of all pairs of elements, a more compact representation that it is comprehensible to a human reader would be useful. Reconsidering the examples, one can see that some form of categorization (with overlapping categories) would be most useful. The number of elements in DocBook (and most other XML-based languages) is too high for a single level of categories to be sufficient – the result would be either too many or too broad categories.

The solution is to use an almost hierarchical representation, where categories can contain sub-categories ("almost" because of overlaps). This keeps the number of members in each category low but enables us to take the query categories from higher-level categories.

The *element relationship graph* (ERG) is a directed, acyclic graph. The nodes are labeled with either an element name (*element nodes*) or a category label (*category nodes*). Element nodes may have several incoming edges (because categories may overlap). The category nodes are partitioned into *aspect sets* corresponding to the aspects mentioned above; no two nodes from different aspect sets have a direct connection. In essence, this means that there are sub-graphs that are disjoint except for the element nodes.

As mentioned in the context of aspects, the construction of an ERG can only be automated in some cases. In the case of a graph based on element semantics, there is no option but to create the graph manually, based on the documentation. Considering that there are typically hundreds of elements in a given schema – about 300 in DocBook –, this may seem like a daunting task.

It is rarely necessary to start from scratch, given nothing more than a list of element names and descriptions: For didactic reasons, tutorials and reference material for a schema normally describe the elements in related groups. For DocBook, for example, there is a section about "Logical Divisions: The Categories of Elements in DocBook" in the reference manual (Walsh and Muellner, 1999, section 2.5), and a quick reference card where the easily parsable XML source code is available. Thus, while the task is still far from trivial, it turns out to be manageable, as section 8.3.3 will show.

8.3.2 Element similarity in the element relationship graph

The graph described so far provides information about (almost hierarchical) relationships of elements and newly-introduced categories, but it does not quantify element similarity. A first approach could be to define the distance of two elements – which can be seen as the inverse of similarity – to be the shortest path between them, ignoring the direction of the edges. This approach is not entirely satisfactory, however, because not all possible paths are equal: The information that the elements structname and classname are both rendered in the same font is not as meaningful as the information that they are both in the semantic group **source code** (see figure 8.2).

Bergmann (1998) examined a similar problem in the context of similarity measures for taxonomies in structural case-based reasoning, which only needs minor modifications to be used in this context. We start by labeling each inner node n with a number $c_n \in [0, 1]$ denoting the *coherence* of the group formed by the direct descendants. Furthermore, the values must satisfy the following condition: If the inner node a is an ancestor of d, $c_a < c_d$ must hold. This condition ensures that similarity can never increase if the level of generality is increased.

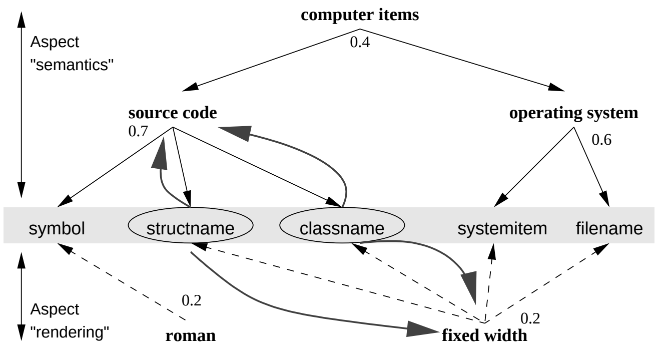


Figure 8.2: Calculating similarity in the ERG. The names in the gray bar are tags, the labels in the upper part are (semantic) concepts, the labels in the lower part are presentation styles.

Given two different nodes n_1 and n_2, the search engine can then easily calculate their similarity: It needs to find the set of their closest common ancestors A; the similarity is $\max_{a \in A}(c_a)$. In figure 8.2, the nearest set of common ancestors for structname and classname is $\{\textbf{source code}, \textbf{fixed width}\}$, and the resulting similarity is $\max(c_{\textbf{source code}}, c_{\textbf{fixed width}}) = \max(0.7, 0.2) = 0.7$.

The difference between this approach and an approach based purely on distance is obvious for systemitem and classname: The closest common ancestor based on path length is **fixed width**, but **computer items** has a higher coherence value, so the resulting similarity is 0.4 instead of 0.2.

8.3.3 Constructing an example element relationship graph

To show that it is feasible to construct an ERG, I created one for DocBook and the aspects of visual appearance and contents.

Visual appearance of the output is determined by XSL style sheets, so I took the official ones[2] and wrote a script to derive a graph from them, using the style sheets for transforming to HTML as a basis. I am concerned about the markup of inline elements, so I only used the file inline.xsl to avoid unnecessary clutter of the resulting graph.

The style sheets also contain templates not directly tied to HTML tags for modularization. For example, the template inline.italicmonoseq indicates that the text is both italic and monospace; having this intermediate node in the graph has the advantage of expressing that *both* features are present. If the ERG contained two separate links instead, only one of them would be used for similarity calculation, so some information would be lost. Of course, there are also links from the corre-

[2] http://docbook.sourceforge.net/projects/xsl/ (version 1.66.1)

sponding HTML elements to this intermediate node, so that elements having only one of these are still similar to elements having both.

The effort needed was low: It took one person less than two hours total, and a significant fraction of that time was spent removing templates that are not relevant in this context. Removing these nodes is only necessary for making the graph easier to comprehend; leaving the additional nodes in the graph would not result in worse similarity calculation.

Next, I looked at a possible semantic grouping of the elements based on the quick reference mentioned previously[3]. It contains 46 overlapping groups of elements, but not all groups are relevant in this context; only the 32 groups containing inline elements are of interest. The authors categorized the elements with a focus on quick look-up, not on semantic similarity, so I needed to modify them slightly.

Then I successively merged the low-level categories until I reached the high level of abstraction mentioned above (computer-related text, emphasized text, metadata, normal text). DocBook's main application area is computer texts, so it is not surprising that the **computer items** category is the most complex one, with 11 sub-categories in three levels.

The last step is to assign the coherence values. I found it easiest to create an internally consistent labeling (on a scale from 0 to 1) for the sub-graph of each aspect. When merging the sub-graphs, I then assigned a weight to each of them denoting the relative importance. For example, the aspect of semantics is much more important than the aspect of presentation, so the weights were 1.0 and 0.4. The final coherence value of a category is then determined by multiplying the preliminary coherence value and the importance of the corresponding sub-graph. This approach makes it easy to adjust the relative importance later without needing to revisit all nodes individually.

Overall, the construction of an initial version of the ERG took less than one day. Of course this original version may well need to be refined based on feedback from users in everyday use.

8.3.4 Search process

One important issue that has not been addressed yet is the actual retrieval process from query to results.

The query is formulated on the basis of high-level categories, and although the documents contain detailed markup, the element relationship graph is used for grouping the element contents into categories at the same level as the query. For each category, the similarity is calculated as follows: $\mathrm{sim}_{\mathrm{global}} = \sum \mathrm{sim}_i \, w_i$.

The weights w_i with $\sum w_i = 1$ represent the relative importance of category c_i. For example, in the example scenario, **computer items** are very important compared to free text, as they have stricter, that is, less ambiguous, semantics.

The index described in chapter 5 contains all necessary data for answering queries using an ERG. Retrieval is executed as normal on the full text as normal, but the term frequency data in combination with the XPath expression of the element is

[3] http://www.dpawson.co.uk/docbook/qrefplain.xml

used to determine whether the query term is marked up with markup from the query. If additional index structures can be used, it is possible to further reduce retrieval time (Dopichaj, 2005).

8.4 Evaluation

To put these concepts to test, a suitable test collection would be needed. The test collections from INEX are not suitable for this purpose: The document schemas have a small vocabulary that mostly allows for visual markup. The only data-centric parts are the metadata (author information) and the bibliography, and these parts do not really lend themselves to specialized similarity measures.

For INEX 2005, we implemented a prototype version of element-relationship graphs (Eger, 2005; Dopichaj, 2006b), with a rudimentary ERG derived from the DTD. However, for the reasons outlined above, the performance of the runs with the ERG was not better than the performance of the runs without it.

8.5 Summary

This chapter has outlined how content-and-structure retrieval can be improved by introducing similarity measures from case-based reasoning. As an example, a similarity measure for ontologies was adapted to be used for vague matching of structural constraints in queries. This is achieved by introducing element relationships, which can be used to determine how similar two elements from a given schema are.

I have shown that the construction of the ERG for a reasonably well-documented XML-based language can be accomplished in very short time, and that the increase in index size is tolerable. One important component that is still missing is an experimental verification of the retrieval quality of this approach.

9 Conclusions and future work

Thus encouraged, our scientific friend drew his papers from his pocket, and presented the whole case as he had done the morning before.

(The Hound of the Baskervilles)

9.1 Conclusions

In this thesis, I have examined various aspects of content-only element retrieval, ranging from indexing and other implementation aspects to similarity calculation and XML-specific improvements.

The most surprising result of this thesis is that standard information retrieval techniques work very well for XML retrieval with only minor tweaking; the experiments show that approaches developed specifically for XML retrieval are no better than the standard methods. This suggests two possible explanations: Either content-only XML element retrieval is really not much different from document-based information retrieval, or – more likely – good ways to make use of the XML structure have not yet been discovered.

Obviously, a wealth of findings have been reported in the context of INEX and other venues; many of the publications in this area have shown more or less pronounced improvements over a baseline of the respective authors' choice. This does not imply that these findings are invalid, but they highlight an important problem in XML retrieval evaluation in general: There is no generally accepted baseline to compare with; if every author chooses his or her own baseline, it is not surprising that this baseline can be beaten, but this tells us little about the absolute quality of the improvements.

Furthermore, it is important to realize that a retrieval system is a complex entity with innumerable parameters. As I have experienced myself, much to my dismay, is that an apparent improvement based on a certain configuration of parameters can turn out to lead to worse retrieval quality if the configuration is slightly changed.

The complexity of a retrieval system is in my opinion a major hindrance to the ability to reproduce other researchers' results; I tried to implement the GPX search engine to have a valid baseline. It turned out that various important points were missing in the published description of the system. Even with clarifications from Shlomo Geva, the author of the corresponding papers, I did not manage to reproduce the results closely. So many things that play a role are left unspecified, starting from seemingly irrelevant matters such as tokenization. Obviously, a paper is rather short, so one cannot expect a complete description, but even a long description such as this thesis is unlikely to be complete enough to allow an exact reimplementation.

Also, as the INEX workshops have shown, bugs in the implementation are frequent, often changing the results dramatically. If these bugs have a negative impact on retrieval quality, they are likely to be found, but if they have a positive effect, nobody will even get the idea of searching for them.

In line with this observation, my own attempts at exploiting the structure of the XML documents have shown mixed results: In some parameter configurations, they lead to an improvement of retrieval quality, whereas in other configurations, they show no improvement or even a decrease of retrieval quality. I have speculated about possible causes for this behavior, but there is little that can be said with confidence, other than that the properties of the documents play an important role.

Overall, it appears not to be advisable to use any variant of these extensions. Even if they can achieve minor improvements in some special cases, it is unclear what exactly characterizes these cases; thus, it is not possible to predict how the extensions will behave in a new context. Considering that the enhancements introduce several new parameters – adaptation method, title detection, parameters of both –, the added complexity and instability offsets the minor improvements of retrieval quality that may be possible.

Considering that the standard methods yield good quality, it is important to see that new index structures can help to make retrieval more efficient. The new index structures achieve a significant reduction of space usage and retrieval time compared to a straightforward implementation. Overall, although there still is a large overhead compared to document-based retrieval, it is now much more manageable and can be used as a starting point for a retrieval engine that is useful for real-world use.

Although it was not possible to do a quantitative evaluation of the use of background knowledge, it still appears likely that much can be gained in the context of content-and-structure retrieval. A combination of type-specific similarity measures from case-based reasoning and standard information retrieval methods could be the ideal match for XML documents that contain text with some semantic markup.

9.2 Future work

Although the aim of the thesis has been achieved, this is only the beginning; more research needs to be done to understand XML retrieval better. Two areas are particularly worthy of attention: optimization of the search engine's performance and the support of content-and-structure search.

9.2.1 Implementation issues

Although the performance is acceptable for a research prototype, there is a lot of room for improvement. The system has been designed to be very flexible, so that different implementations for various core parts of the search engine can be replaced easily to compare them. This flexibility comes at a price – it demands a clear separation of the different subsystems, which prohibits certain optimizations.

Storing each result as an object in main memory is expensive both in terms of time and of memory usage; in long-running experiments with many queries executed in a row, Java's garbage collection takes more than 10 percent of the total time. Furthermore, to accommodate different similarity measures, these internal results contain data that is not needed – any specific similarity measure only uses a subset of the data. This makes the implementation simpler and less error-prone than using simple accumulators (Witten et al., 1999, chapter 4), but for a production system, this should be changed.

Measurements clearly show that for the index structures described in chapter 5, obtaining the metadata is the most costly operation, so applying top-k methods to avoid getting the metadata for all results seems promising (Dopichaj, 2007d). The rationale is that we are hardly ever interested in *all* retrieval results, but only in the best k; for typical interactive retrieval scenarios, k will typically be just large enough to cover a few result pages.

So far, execution of the queries is single-threaded, but it would be simple to parallelize the reading of the inverted lists and the similarity calculation. Preliminary experiments with multi-threading on a dual-core system suggest that the performance gain is small, but memory usage increases dramatically.

For extremely large collections like the world wide web, scalability on a single system is not enough, the execution should be distributed in a cluster. For this, the *MapReduce* programming model (Dean and Ghemawat, 2004) suggests itself. In this model, the various steps of execution can be executed on different nodes in the cluster, and the parallelization can be between different computers instead of different threads on the same computer.

When the results are presented to the user, it is necessary to access the contents of the documents rapidly. This is completely infeasible if the documents are stored as files in a file system; instead an XML database like the XTC research prototype (Härder, 2005; Haustein, 2005) is needed. The index structures of this database could also be used for faster content-and-structure search.

The current index structures are not update-friendly; if the usage scenario differs from the expected scenario of unchanging documents, the index structures will have to be adapted to deal with this. This is of major importance if the retrieval system is to be integrated in a database system.

9.2.2 Evaluation and user interfaces

Clearly, the INEX test collections and evaluation methods are not ideal for evaluating the effectiveness of title exploitation. As outlined in section 7.5.2, a new experiment should be set up, with a test collection that provides long texts (ideally books) that have largely self-contained parts.

Such an evaluation is obviously not easy to set up and should be done on a large scale to allow the comparison of different retrieval systems. Unfortunately, a useful document collection is probably of commercial interest, so it is doubtful whether it is possible to get access for research purposes.

Another problem for the evaluation is that few XML retrieval systems are in real-world use, so user interfaces are still a matter of debate. For content-only queries, a keyword-based query language appears to be most useful, because most people are familiar with them from web search engines. The output format for the results is another matter; in contrast to standard information retrieval, a flat result list might not be ideal. Probably, the result display should in some way reflect the structure of the original documents, but so far it is unclear exactly how this should be done. This is still largely an unresolved research problem; one of the basic questions is whether the results should be presented as a flat list or grouped by document. The best way to present the results will depend on the specific use case.

If it is not even known how the results should be displayed to the searchers, it is obviously not possible to determine whether the results would be useful.

9.2.3 Further forms of XML retrieval

As chapter 3 outlined, there are many facets of XML retrieval, ranging from pure data retrieval to content-only retrieval. The work in this thesis mainly focuses on content-only retrieval; the next logical step would be to make the step towards content-and-structure retrieval by adding content constraints to the full-text queries.

Chapter 8 presented preliminary work in this area, and there have been several Master's theses touching this area (Eger, 2005; Hartel, 2007). Without access to suitable test collections for further analysis (the INEX collections are not suitable), however, this work is necessarily rather speculative and difficult to quantify.

It will be difficult to create a useful test collection for content-and-structure queries as long as there is no consensus about the details: The current query languages like NEXI and XIRQL are clearly not fit to be used by end-users, but it is hard to imagine what a good query language supporting complex structural queries could look like.

Bibliography

"Did she give any references when she came?"

(The Adventure of the Veiled Lodger)

Amer-Yahia, Sihem; Botev, Chavdar; Buxton, Stephen; Case, Pat; Doerre, Jochen; Holstege, Mary; McBeath, Darin; Rys, Michael; Shanmugasundaram, Jayavel. XQuery 1.0 and XPath 2.0 full-text. 2006. W3C Working Draft 1 May 2006.

Baeza-Yates, Ricardo; Ribeiro-Neto, Berthier. *Modern Information Retrieval.* Addison Wesley, Harlow, Essex, England, 1999.

Beigbeder, Michel. Structured content-only information retrieval using term proximity and propagation of title terms. In *INEX 2006 proceedings.* Springer, 2007.

Bentley, Jon Louis; Yao, Andrew Chi-Chih. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.

Berglund, Anders. Extensible stylesheet language (XSL) version 1.1. 2006. W3C Recommendation 05 December 2006.

Berglund, Anders; Boag, Scott; Chamberlin, Don; Fernández, Mary F.; Kay, Michael; Robie, Jonathan; Siméon, Jérôme. XML path language (XPath) 2.0. 2007. W3C Recommendation 23 January 2007.

Bergmann, Ralph. On the use of taxonomies for representing case features and local similarity measures. In *GWCBR 1998 proceedings.* Universität Rostock, 1998.

Boag, Scott; Chamberlin, Don; Fernández, Mary F.; Florescu, Daniela; Robie, Jonathan; Siméon, Jérôme. XQuery 1.0: An XML query language. 2007. W3C Recommendation 23 January 2007.

Bos, Bert; Çelik, Tantek; Hickson, Ian; Lie, Håkon Wium. Cascading style sheets, level 2 revision 1. 2006. W3C Working Draft 06 November 2006.

Bray, Tim; Hollander, Dave; Layman, Andrew; Tobin, Richard. Namespaces in XML 1.0 (second edition). 2006a. W3C Recommendation 16 August 2006.

Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; Maler, Eve; Yergeau, François; Cowan, John. Extensible markup language (XML) 1.1 (second edition). 2006b. W3C Recommendation 16 August 2006, edited in place 29 September 2006.

Brownell, David. *SAX2 – Processing XML Efficiently with Java*. O'Reilly, 2002.

Buckley, Chris; Voorhees, Ellen M. Evaluating evaluation measure stability. In *SIGIR 2000 proceedings*, pages 33–40. ACM, 2000.

Carmel, David; Farchi, Eitan; Petruschka, Yael; Soffer, Aya. Automatic query refinement using lexical affinities with maximal information gain. In *SIGIR 2002 proceedings*, pages 283–290. 2002.

Carmel, David; Maarek, Yoelle S.; Mandelbrod, Matan; Mass, Yosi; Soffer, Aya. Searching XML documents via XML fragments. In *SIGIR 2003 proceedings*, pages 151–158. ACM, 2003.

Carmel, David; Maarek, Yoëlle; Soffer, Aya. XML and information retrieval: a SIGIR 2000 workshop. *SIGIR Forum*, 34(1), 2000.

Chiaramella, Yves; Mulhem, Philippe; Fourel, Franck. A model for multimedia information retrieval. Technical Report FERMI ESPRIT BRA 8134, University of Glasgow, 1996.

Connolly, Dan; Khare, Rohit; Rifkin, Adam. The evolution of Web documents: The ascent of XML. *World Wide Web Journal*, 2(4):119–128, 1997.

Croft, W. Bruce. Language models for information retrieval. In Dayal, Umeshwar; Ramamritham, Krithi; Vijayaraman, T. M., editors, *ICDE 2003 Proceedings*, pages 3–7. IEEE Computer Society, 2003.

Crouch, Carolyn J.; Khanna, Sudip; Potnis, Poorva; Doddapaneni, Nagendra. The dynamic retrieval of XML elements. In *INEX 2005 proceedings*, pages 268–281. 2006.

Cutler, M.; Deng, H.; Maniccam, S. S.; Meng, W. A new study on using HTML structures to improve retrieval. In *ICTAI 1999 proceedings*, pages 406–409. IEEE, 1999.

Cutler, Michael; Shih, Yungming; Meng, Weiyi. Using the structure of HTML documents to improve retrieval. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*. 1997.

Dean, Jeffrey; Ghemawat, Sanjay. MapReduce: Simplified data processing on large clusters. In *OSDI 2004 proceedings*. 2004.

Denoyer, Ludovic; Gallinari, Patrick. The Wikipedia XML corpus. *SIGIR Forum*, 40(1):64–69, 2006.

Dopichaj, Philipp. Exploiting background knowledge for better similarity calculation in XML retrieval. In *BNCOD 2004 proceedings, Volume 2*, pages 126–127. Heriot-Watt University, 2004.

Dopichaj, Philipp. Element relationship: Exploiting inline markup for better XML retrieval. In *BTW 2005 proceedings*, pages 285–294. GI, 2005.

Dopichaj, Philipp. Element retrieval in digital libraries: Reality check. In *SIGIR XML Element Retrieval Methodology Workshop 2006 proceedings*, pages 1–4. 2006a.

Dopichaj, Philipp. The University of Kaiserslautern at INEX 2005. In *INEX 2005 proceedings*, pages 196–210. Springer, 2006b.

Dopichaj, Philipp. Improving content-oriented XML retrieval by applying structural patterns. In *ICEIS 2007 proceedings*, pages 5–13. INSTICC, 2007a.

Dopichaj, Philipp. Improving content-oriented XML retrieval by exploiting small elements. In *BNCOD 2007 workshop proceedings*, pages 68–74. IEEE, 2007b.

Dopichaj, Philipp. Space-efficient indexing of XML documents for content-only retrieval. *Datenbank-Spektrum*, 7(23), 2007c.

Dopichaj, Philipp. The University of Kaiserslautern at INEX 2006. In *INEX 2006 proceedings*, pages 223–232. Springer, 2007d.

Doyle, Arthur Conan. *The Complete Illustrated Strand Sherlock Holmes. The Complete Facsimile Edition.* Midpoint Press, 2007.

Eger, Benedikt. *Entwurf und Implementierung einer XML-Volltext-Suchmaschine.* Master's thesis, University of Kaiserslautern, 2005.

Elias, Peter. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

Fox, Edward A. Composite document extended retrieval: an overview. In *SIGIR 1985 proceedings*, pages 42–53. ACM, 1985.

Fuhr, Norbert; Großjohann, Kai. XIRQL: A query language for information retrieval in XML documents. In *SIGIR 2001 proceedings*, pages 172–180. ACM, 2001.

Fuhr, Norbert; Malik, Saadia; Lalmas, Mounia. Overview of the INitiative for the Evaluation of XML Retrieval (INEX) 2003. In *INEX 2003 pre-proceedings*. Available at http://inex.is.informatik.uni-duisburg.de:2003/PrePreceeding.pdf, 2003.

Geva, Shlomo. Extreme file inversion. In *INEX 2002 proceedings*, pages 155–161. 2002.

Geva, Shlomo. GPX – gardens point XML IR at INEX 2005. In *INEX 2005 proceedings*, pages 240–253. Springer, 2006.

Geva, Shlomo. GPX – gardens point XML IR at INEX 2006. In *INEX 2006 proceedings*, pages 137–150. Springer, 2007.

Geva, Shlomo; Sahama, Tony. The NLP task at INEX 2004. *SIGIR Forum*, 39(1):50–53, 2005.

Gövert, Norbert; Kazai, Gabriella. Overview of the initiative for the evaluation of XML retrieval (INEX) 2002. In *INEX 2002 proceedings*. 2002.

Härder, Theo. XML databases and beyond – plenty of architectural challenges ahead. In *ADBIS 2005 proceedings*. Springer, 2005.

Hartel, Christoph R. *Improving XML Retrieval by Exploiting Content and Structure*. Diploma thesis, University of Kaiserslautern, 2007. To appear.

Haustein, Michael P. *Feingranulare Transaktionsisolation in nativen XML-Datenbanksystemen*. Ph.D. thesis, University of Kaiserslautern, 2005.

Hersh, William; Turpin, Andrew; Price, Susan; Chan, Benjamin; Kramer, Dale; Sacherek, Lynetta; Olson, Daniel. Do batch and user evaluations give the same results? In *SIGIR 2000 proceedings*, pages 17–24. ACM, 2000.

Hors, Arnaud Le; Hégaret, Philippe Le; Wood, Lauren; Nicol, Gavin; Robie, Jonathan; Champion, Mike; Byrne, Steve. Document object model (DOM) level 3 core specification, version 1.0. 2004. W3C Recommendation 07 April 2004.

Huffman, David A. A method for the construction of minimum-redundancy codes. *Proc. of the I. R. E.*, 40(9):1098–1101, 1952.

ISO. Information processing – text and office systems – standard generalized markup language (SGML). ISO 8879, 1986.

Jang, Hyunchi; Kim, Youngil; Shin, Dongwook. An effective mechanism for index update in structured documents. In *CIKM 1999 proceedings*. 1999.

Jansen, Bernard J.; Spink, Amanda; Bateman, Judy; Saracevic, Tefko. Real life information retrieval: a study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.

Järvelin, Kalervo; Kekäläinen, Jaana. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

Kamps, Jaap; de Rijke, Maarten; Sigurbjörnsson, Börkur. The importance of length normalization for XML retrieval. *Information Retrieval*, 8:631–654, 2005.

Kay, Michael. XSL transformations (XSLT) version 2.0. 1999. W3C Recommendation 23 January 2007.

Kazai, Gabriella; Lalmas, Mounia. Notes on what to measure in INEX. In *Proceedings of the INEX 2005 Workshop on Element Retrieval Methodology*. 2005.

Kazai, Gabriella; Lalmas, Mounia. INEX 2005 evaluation metrics. In Fuhr, Norbert; Lalmas, Mounia; Malik, Saadia; Kazai, Gabriella, editors, *INEX 2005 proceedings*, pages 16–29. Springer, 2006.

Kazai, Gabriella; Lalmas, Mounia; de Vries, Arjen P. The overlap problem in content-oriented XML retrieval evaluation. In *SIGIR 2004 proceedings*, pages 72–79. ACM, 2004.

Larsen, Birger; Tombros, Anastasios; Malik, Saadia. Is XML retrieval meaningful to users? Searcher preferences for full documents vs. elements. In *SIGIR 2006 proceedings*, pages 663–664. ACM, 2006.

Lee, Dik L.; Chuang, Huei; Seamons, Kent. Document ranking and the vector-space model. *IEEE Software*, 14(2):67–75, 1997.

Lee, Yong Kyu; Yoo, Seong-Joon; Yoon, Kyoungro; Berra, P. Bruce. Index structures for structured documents. In *DL 1996 proceedings*, pages 91–99. 1996.

Lehtonen, Miro. Designing user studies for XML retrieval. In *SIGIR XML Element Retrieval Methodology Workshop 2006 proceedings*, pages 28–34. 2006.

Lehtonen, Miro; Pharo, Nils; Trotman, Andrew. A taxonomy for XML retrieval use cases. In *INEX 2006 proceedings*, pages 413–422. Springer, 2007.

Lenz, Mario; Bartsch-Spörl, Brigitte; Burkhard, Hans-Dieter; Wess, Stefan, editors. *Case-Based Reasoning Technology: From Foundations to Applications*. Springer, 1998.

Liu, Yiqun; Wang, Canhui; Zhang, Min; Ma, Shaoping. Finding “abstract fields” of web pages and query specific retrieval – THUIR at TREC 2004 web track. In *TREC 2004 proceedings*. 2004.

Lu, Wei; Robertson, Stephen; Macfarlane, Andrew. Field-weighted XML retrieval based on BM25. In *INEX 2005 proceedings*, pages 161–171. Springer, 2006.

Luk, Robert W.P.; Leong, H. V.; Dillon, Tharam S.; Chan, Alvin T.S.; Croft, W. Bruce; Allan, James. A survey in indexing and searching XML documents. *Journal of the American Society for Information Science and Technology*, 53(6):415–437, 2002.

Malik, Saadia; Kazai, Gabriella; Lalmas, Mounia; Fuhr, Norbert. Overview of INEX 2005. In *INEX 2005 proceedings*, pages 1–15. Springer, 2006.

Malik, Saadia; Trotman, Andrew; Lalmas, Mounia; Fuhr, Norbert. Overview of INEX 2006. In *INEX 2006 proceedings*, pages 1–11. Springer, 2007.

Mass, Yosi; Mandelbrod, Matan. Retrieving the most relevant XML components. In *INEX 2003 proceedings*, pages 58–64. 2003.

Mass, Yosi; Mandelbrod, Matan. Component ranking and automatic query refinement for XML retrieval. In *INEX 2004 proceedings*, pages 73–84. Springer, 2005.

Mass, Yosi; Mandelbrod, Matan. Using the INEX environment as a test bed for various user models for XML retrieval. In *INEX 2005 proceedings*, pages 187–195. Springer, 2006.

Mass, Yosi; Mandelbrod, Matan; Amitay, Einat; Maarek, Yoelle; Soffer, Aya. JuruXML – an XML retrieval system at INEX '02. In *INEX 2002 proceedings*, pages 73–80. 2002.

Melton, Jim; Muralidhar, Subramanian. XML syntax for XQuery 1.0 (XQueryX). 2007. W3C Recommendation 23 January 2007.

Moffat, Alistair; Sacks-Davis, Ron; Wilkinson, Ross; Zobel, Justin. Retrieval of partial documents. In *TREC*, pages 181–190. 1993.

Myaeng, Sung Hyon; Jang, Don-Hyun; Kim, Mun-Seok; Zhoo, Zong-Cheol. A flexible model for retrieval of SGML documents. In *SIGIR 1998 proceedings*, pages 138–145. ACM, 1998.

Ogilvie, Paul; Callan, Jamie. Using language models for flat text queries in XML retrieval. 2003.

Ogilvie, Paul; Callan, Jamie. Hierarchical language models for XML component retrieval. In *INEX 2004 proceedings*, pages 224–237. Springer, 2005.

Pawson, Dave. *XSL-FO*. O'Reilly, 2002.

Pehcevski, Jovan; Thom, James A. HiXEval: Highlighting XML retrieval evaluation. In *INEX 2005 proceedings*, pages 43–57. Springer, 2006.

Piwowarski, Benjamin. EPRUM metrics and INEX 2005. In *INEX 2005 proceedings*, pages 30–42. Springer, 2006.

Ponte, Jay M.; Croft, W. Bruce. A language modeling approach to information retrieval. In *SIGIR 1998 proceedings*, pages 275–281. ACM, 1998.

Porter, Martin F. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

Ramírez, Georgina; Westerveld, Thijs; de Vries, Arjen P. Using small XML elements to support relevance. In *SIGIR 2006 proceedings*, pages 693–694. 2006a.

Ramírez, Georgina; Westerveld, Thijs; de Vries, Arjen P. Using structural relationships for focused XML retrieval. In *FQAS 2006 proceedings*. 2006b.

Richter, Michael M. Fallbasiertes Schließen: Vergangenheit, Gegenwart, Zukunft. *Informatik-Spektrum*, 26(3):180–190, 2003.

Robertson, Stephen. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520, 2004.

Robertson, Stephen; Zaragoza, Hugo; Taylor, Michael. Simple BM25 extension to multiple weighted fields. In *CIKM 2004 proceedings*, pages 42–49. ACM, 2004.

Robertson, Stephen E.; Walker, Steve. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR 1994 proceedings*, pages 232–241. ACM, 1994.

Salton, Gerard; Allan, James; Buckley, Chris. Approaches to passage retrieval in full text information systems. In *SIGIR 1993 proceedings*, pages 49–58. ACM, 1993.

Schwartz, Candy. Web search engines. *Journal of the American Society for Information Science*, 49(11):973–982, 1998.

Shannon, Claude E. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:51–64, 1951.

Shin, Dongwook; Jang, Hyuncheol; Jin, Honglan. BUS: an effective indexing and retrieval scheme in structured documents. In *DL 1998 proceedings*, pages 235–243. 1998.

Sigurbjörnsson, Börkur; Kamps, Jaap. The effect of structured queries and selective indexing on XML retrieval. In *INEX 2005 proceedings*, pages 104–118. Springer, 2006.

Spärck Jones, Karen. IDF term weighting and IR research lessons. *Journal of Documentation*, 60(5):521–523, 2004.

Spärck Jones, Karen; Walker, Steve; Robertson, Stephen E. A probabilistic model of information and retrieval:development and status. Technical report, Computer Laboratory, University of Cambridge, 1998.

Stahl, Armin; Gabel, Thomas. Using evolution programs to learn local similarity measures. In *ICCBR 2003 proceedings*. Springer, 2003.

Theobald, Anja; Weikum, Gerhard. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT 2002 proceedings*, pages 477–495. Springer-Verlag, 2002.

Theobald, Martin. *TopX – Efficient and Versatile Top-k Query Processing for Text, Structured, and Semistructured Data.* Ph.D. thesis, Universität des Saarlandes, 2006.

Theobald, Martin; Schenkel, Ralf; Weikum, Gerhard. An efficient and versatile query engine for TopX search. In *VLDB 2005 proceedings*, pages 625–636. 2005.

Theobald, Martin; Schenkel, Ralf; Weikum, Gerhard. TopX and XXL at INEX 2005. In *INEX 2005 proceedings*, pages 282–295. 2006.

Trotman, Andrew. Wanted: Element retrieval users. In Trotman, Andrew; Lalmas, Mounia; Fuhr, Norbert, editors, *Proceedings of the INEX 2005 Workshop on Element Retrieval Methodology*, pages 63–69. 2005. See http://www.cs.otago.ac.nz/inexmw/.

Trotman, Andrew; Pharo, Nils; Jenkinson, Dylan. Can we at least agree on something? In Trotman, Andrew; Geva, Shlomo; Kamps, Jaap, editors, *Proceedings of the SIGIR 2007 Workshop on Focused Retrieval*, pages 49–56. 2007a.

Trotman, Andrew; Pharo, Nils; Lehtonen, Miro. XML-IR users and use cases. In *INEX 2006 proceedings*, pages 400–412. Springer, 2007b.

Trotman, Andrew; Sigurbjörnsson, Börkur. Narrowed extended XPath I (NEXI). In *INEX 2004 proceedings*, pages 16–40. Springer, 2005.

Voorhees, Ellen M. TREC: Improving information access through evaluation. *Bulletin of the American Society for Information Science and Technology*, 32(1):16–21, 2005.

Walsh, Norman; Muellner, Leonard. *DocBook: The Definitive Guide.* O'Reilly, Sebastopol, 1999.

Wilkinson, Ross. Effective retrieval of structured documents. In *SIGIR 1994 proceedings*, pages 311–317. ACM, 1994.

Witten, Ian H.; Moffat, Alistair; Bell, Timothy C. *Managing Gigabytes.* Morgan Kaufmann, 1999.

Witten, Ian H.; Neal, Radford M.; Cleary, John G. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, 1987.

Zobel, Justin. How reliable are the results of large-scale information retrieval experiments? In *SIGIR 1998 proceedings*, pages 307–314. ACM, 1998.

Zobel, Justin; Moffat, Alistair. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, 1998.

Zobel, Justin; Moffat, Alistair; Sacks-Davis, Ron; Wilkinson, Ross. Efficient retrieval of partial documents. *Inf. Process. Manage.*, 31(3):361–377, 1995.