Dipl.-Ing. Mag. Marcus Hassler

Linguistically Enhanced Information Retrieval of Structured Documents

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

Studium der Angewandten Informatik

Alpen-Adria Universität Klagenfurt

Fakultät für Technische Wissenschaften

1. Begutachter
   O. Univ.-Prof. Dipl.-Ing. Mag. Dr. Roland Mittermeir
   Alpen-Adria Universität Klagenfurt / Institut für Informatik-Systeme

2. Begutachter
   Ao. Univ.-Prof. Mag. Dr. Günther Fliedl
   Alpen-Adria Universität Klagenfurt / Institut für Angewandte Informatik

March / 2009

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende wissenschaftliche Arbeit selbstständig angefertigt und die mit ihr unmittelbar verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle aus gedruckten, ungedruckten oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für wissenschaftliche Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.

Die während des Arbeitsvorganges gewährte Unterstützung einschließlich signifikanter Betreuungshinweise ist vollständig angegeben.

Die wissenschaftliche Arbeit ist noch keiner anderen Prüfungsbehörde vorgelegt worden. Diese Arbeit wurde in gedruckter und elektronischer Form abgegeben. Ich bestätige, dass der Inhalt der digitalen Version vollständig mit dem der gedruckten Version übereinstimmt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Dipl.-Ing. Mag. Marcus Hassler   Klagenfurt, 3. March 2009

Acknowledgements

Over the years, this work consumed much energy, patience, and brainpower not only of myself but also of many others (= **POWs**). Without that helping hands and good ghosts, this project would never have come to existence, not to mention its completion. I would like to express my deepest gratitude to all of them (= **compensation for loss**).

First of all I have to thank Prof. Roland T. Mittermeir (= **boss no. 1**), who provided me not only with a job (= **food**) but also with his knowledge, wisdom, and farsightedness (= **ideas**). Although he had always a great deal to do, his door was never locked for discussions and counseling. I always looked up to him because of his righteousness, tolerance, and kindness.

Much thanks deserves Prof. Günther Fliedl (**= boss no. 2**), who encouraged me more than once that I am doing the right things the right way. In numerous discussions I gained deep insights into natural language processing and computational linguistics (= **more ideas**). His continuous guidance and open-mindedness left its clear mark on this thesis. Besides that, he also gave me a job (= **more food**).

I am as well deeply indebted to Prof. Abdelhamid Bouchachia for his professional advice on several chapters. With his help most of the English words could be tamed to form proper and readable sentences (= **eloquence**).

Thanks to my colleagues at the Department of Informatics Systems who have always been a great support to me. Their assistance, company, humor, and addiction to coffee (= **fun**) created a friendly working environment. Special thanks go to my friend Dr. Karl Wiggisser, who – before he cold-blooded left me behind alone in the office – had been a valuable conversationalist to exchange views with (= **more fun**).

I bow deeply in respect to the honorary gentlemen of the hekkas guild, which constantly refreshed my hope about hekkn is the best thing one can do to save the world. Their way of life, ideology, and taunting were one reason for me to finish this work. Interestingly, best ideas always came up in the monthly sauna subcommittees... (= **hekkas.com**)

```
\_0_/
  |
 / \
```

Since not all people I want to thank are fluent in English, I will now switch to Deutsch, damit mich auch wirklich alle Angesprochenen verstehen:

An dieser Stelle möchte ich mich bei meinen Eltern Ursula und Herbert für Ihre jahrelange Unterstützung und Motivation bedanken (= **faith**). Jedoch sei angemerkt, dass meine Mutter sich gegen Ende der Arbeit nur noch auf monatlicher Basis nach dem jeweiligen Ist-Stand erkundigte.

Auch möchte ich Linde und Hans recht herzlich dafür danken, dass Sie mich in den letzten Monaten meiner Fertigstellung mit offenen Armen bei sich aufnahmen und beherbergten (= **shelter**).

Gesondert jedenfalls muss der Dank an meine Lebensgefährtin Jutta (= **big boss**) entrichtet werden. Hier sind Dankesworte alleine voraussichtlich nicht ausreichend, hier werden wohl Taten folgen müssen. Im Gegensatz zu meiner Mutter hat sich Ihre Abtastfrequenz meines Ist-Standes gegen Ende hin jedoch eher indirekt proportional verhalten (= **patience**).

Equipped with a single big boss remaining, lots of experiences with former bosses, a pool of new ideas, a well-fed body, eloquence, fun stuff, a wonderful website, faith, shelter, and patience, the course is now set for a prosperous future.

Abstract

This research sheds light on structured document retrieval and its challenges. In this paradigm, documents are no longer perceived as 'flat' content containers. Rather, their content is structured into a hierarchy of various levels of granularity. By considering parts of documents instead of its entirety, the users' query can be answered more precisely and with better focus. This implies that traditional retrieval methods have to be enhanced by exploiting the structure as additional source of information. However, the structural dimension of documents demands for tailored models that incorporate structure in the representation and retrieval process.

The aim of this thesis is to develop a retrieval system that is capable of satisfying complex user queries containing both, constraints on the content and structure. Documents are first transformed into a generic XML document format that optimally supports retrieval tasks. It consists of three structural elements. These are the document, its (sub-)sections, and its fragments (smallest retrievable units). Each element includes a metadata and content part. A cascade of natural language processing steps analyzes the textual contents and extracts relevant index terms. The analysis involves extended tokenization, supporting token types and multi-tokens, and filtering of functional, content-related, and domain-specific stopwords. The single-term index is supplemented by multi-term indices of composite nouns, named entities, formulaic speech, and full forms of acronyms. These patterns plus additional processing rules relevant to information retrieval (as applied during tokenization) are extracted from the documents automatically. As a consequence of splitting a single document into numerous parts according to its structure, mechanisms, e.g. classification and clustering, to organize these parts are needed. Being a user-centered approach, classification automatically assigns them to pre-defined classes that may be created, populated, and navigated on demand. Clustering partitions document components that are not pre-classified into groups using similarity measures (e.g., edit distance). It is used to speed up retrieval by organizing clusters into a hierarchy. During search, clusters (and descendant clusters) consisting of irrelevant documents to the query are simply ignored.

In order to validate the concepts developed within (the theoretical part of) this thesis, a prototype called X-DOSE – XML-Document Oriented Search Engine – has been implemented. Based on a client-server architecture, the system handles indexing, retrieval, classification, and clustering tasks. X-DOSE has been evaluated using the XML documents of the INEX collection. The empirical results indicate the appropriateness and importance of the various stages of processing proposed in the context of this dissertation. Moreover, other aspects have been investigated such as the pros and cons of natural language processing steps involved, enhancement of query languages for structured documents, and hybrid similarity measures.

Keywords: Information Retrieval, Structured Document Retrieval, Search Engines, XML, Natural Language Processing, Text Mining, XML Classification, XML Clustering, Document Mining, Retrieval Evaluation, X-DOSE

Zusammenfassung

Die vorliegende Dissertation beschäftigt sich mit dem Auffinden von Informationen in strukturierten Dokumenten (Structured Document Retrieval) und den damit verbundenen Herausforderungen. Dokumente werden nicht als Hüllen für 'flache' Inhalte verstanden. Vielmehr bestehen sie aus einer hierarchischen Anordnung von Inhalten auf verschiedenen Abstraktionsebenen. Da bestimmte Teile von Dokumenten meist genauere Antworten auf Benutzeranfragen beinhalten, müssen traditionelle Methoden des Information Retrievals angepasst werden, um zusätzliche Strukturinformation einzubeziehen. Diese strukturelle Dimension verlangt nach speziell dafür zugeschnittenen Modellen, die die Struktur eines Dokuments sowohl bei der Repräsentation als auch beim Retrievalprozess berücksichtigen.

Das Ziel der vorliegenden Dissertation ist die Entwicklung eines Softwaresystems zur Beantwortung komplexer Benutzerabfragen, die sowohl inhaltliche als auch strukturelle Bedingungen enthalten. In einem ersten Schritt werden zu durchsuchende Dokumente in ein generisches XML Dokumentformat transformiert, das für Retrievalzwecke optimiert ist. Es besteht aus drei Grundelementen für die Dokument-, (Unter-)Kaptitel-, und Fragmentebene (kleinste beziehbare Informationseinheit). Jedes dieser Elemente beinhaltet einen in sich geschlossenen Metadaten- und Inhaltsdatenblock. Der Inhalt, vorliegend in Form von natürlichsprachlichen Texten, wird von einer Sequenz hintereinander geschalteter 'Natural Language Processing' Komponenten verarbeitet. Inhaltstragende Wörter werden extrahiert und in den Index aufgenommen. Zu den zentralen Analyseschritten zählen einerseits die erweiterte Tokenisierung, die sowohl eine Typisierung von Tokens als auch Zusammenschlüsse von Einzeltokens zu Multitokens durchführt, und andererseits die Filterung von Stoppwörtern auf funktionaler, inhaltsbezogener und domänenspezifischer Ebene. Der so entstehende Einzelwortindex wird in weiterer Folge durch Mehrwortindizes ergänzt, die Eigennamen, Nominalkomposita, Redewendungen und ausgeschriebene Akronyme inkludieren. Diese Textmuster werden, zusammen mit sprachspezifischen Regeln (wie etwa bei der Tokenisierung verwendet) automatisch aus vorhandenen Dokumenten gewonnen.

Die Aufgliederung eines Dokuments in eine Vielzahl kleinerer Einheiten führt notgedrungen zu einer Explosion an zu verarbeitenden Dokumententeilen. Um dieser Situation entgegen zu wirken, sind Mechanismen wie Klassifikation und Clustering notwendig. Sie arbeiten große Datenmengen auf und (re-)organisieren sie für spätere Zugriffe. Aufgabe der

Klassifikation ist die Zuteilung von Dokumentteilen in vordefinierte Kategorien, die vom Benutzer selbst angelegt, ergänzt und für den Zweck der Navigation genutzt werden. Clustering hingegen stellt eine weitere Möglichkeit der automatisierten Organisation dar. Dabei werden Kategorien, in diesem Zusammenhang Cluster genannt, durch Zusammenfassen ähnlicher Dokumentteile vom System selbst definiert. Dies geschieht unter Zuhilfenahme von Ähnlichkeitsmetriken wie etwa der 'edit distance'. Durch die hierarchische Verknüpfung dieser Cluster miteinander entsteht eine Baumstruktur, die effektiv zur Beschleunigung des Retrievalprozesses genutzt wird. Cluster (und untergeordnete Cluster), die lediglich aus nicht relevanten Dokumentteilen bestehen, werden von der Suche ausgeschlossen.

Um die theoretischen Aussagen dieser Arbeit validieren zu können, wurde der X-DOSE (XML-Document Oriented Search Engine) Prototyp entwickelt. Das System basiert auf einer Client-Server Architektur und verarbeitet eingehende Anfragen zur Indizierung, Abfrage, Klassifikation und Clustering. Die Effizienz und Effektivität von X-DOSE wurde anhand von Echtdaten (XML Dokumente des INEX Textkorpus) sowie von genormten Aufgabenstellungen beurteilt und anschließend mit jenen von ähnlichen Systemen verglichen. Die empirischen Ergebnisse bestätigen die Relevanz und Korrektheit der in dieser Arbeit vorgeschlagenen Vorgangsweise. Darüber hinaus wurden auch andere Erkenntnisse, wie beispielsweise die Vor- und Nachteile einzelner Textanalyseschritte, Erweiterungen von Abfragesprachen für strukturierte Dokumente und hybride Ähnlichkeitsmaße, gewonnen.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

List of Acronyms and Abbreviations

AST	Abstract Syntax Tree
BEP	Best Entry Points
CAS	Content-And-Structure
CO	Content-Only
COS	Content Only + Structure
CSIRO	Commonwealth Scientific and Industrial Research Organisation
CSV	Comma Separated Value
DTD	Document Type Definition
HIS	Hyperwave Information Server
HTML	HyperText Markup Language
HyREX	Hyper-media Retrieval Engine for XML
HySpirit	HYpermedia System with Probabilistic Inference for the Retrieval of InformaTion
IEEE	Institute of Electrical and Electronics Engineers
INEX	INitiative for the Evaluation of XML Retrieval
IR	Information Retrieval
MathML	Mathematical Markup Language
MS	MicroSoft
MUC	Message Understanding Conference
NEXI	Narrowed Extended Xpath I
NLP	Natural Language Processing
PADRE	PArallel Document Retrieval Engine
PDF	Portable Document Format

POS	Part-Of-Speech
RMI	Remote Method Invocation
RSV	Retrieval Status Value
SDR	Structured Document Retrieval
SQL	Structured Query Language
STTS	Stuttgart-Tübingen TagSet
TEI	Text Encoding Initiative
TnT	Trigrams'n'Tags
URL	Uniform Resource Locator
VQL	Verity Query Language
VSM	Vector Space Model
X-DOSE	XML-Document Oriented Search Engine
X-RAI	XML Retrieval Assessment Interface
xCG	eXtended Cumulated Gain
XIRQL	XML IR Query Language
XML	eXtensible Markup Language
XOR	XML Oriented Retrieval language
XSLT	eXtensible Stylesheet Language Transformation
XXL	fleXible XML search Language

Chapter 1

Science must begin with myths,
and with the criticism of myths.

Karl Popper

Introduction

Over the past several years the amount of available online documents grew rapidly. Repositories like Wikipedia[1] or the IEEE digital library[2] provide a huge source of information which is constantly increasing and continually changing. To manage these data loads, automatic mechanisms for searching and organizing are indispensable, demanding sophisticated methods to represent, store, organize, and retrieve information from these sources. Therefore, methods from the fields of Information Retrieval (IR) [246, 21] and document mining [119, 160] have been adapted to meet these requirements.

Traditional information retrieval is concerned with unstructured documents. From a structural point of view these systems operate on a 'flat' document structure, neglecting any kind of internal document organization (e.g., chapters, sections) and content distinction (e.g., texts, figures, tables). The whole content of a document is treated as a single piece of information which is represented by a single object (i.e., weighed feature vector). As a result, the retrieval process triggered by a user query ends up with a ranked list of documents corresponding to the query terms they contain (see Figure 1.1). However, often a user is more interested in a specific piece of information that is only part of a document. For instance, information needs may range from a set of sections of a book to a single figure with a certain caption. Retrieving the whole book at once does not satisfy the users interest because the coverage of the full document is usually too broad, leading to further focused search and browse within the document. The danger here is to miss documents containing satisfying information, because the overall similarity of (rather long) documents might be too low given a specific request for information. The following scenario sketches this problem:

Assume there is a collection of 1.000.000 documents available. Within this set there is a short newspaper article ($\sim$100 words) dealing about the recently discovered black hole that is nearest to earth, entitled 'Nearest Black Hole found!'. Another document is written by Stephen Hawking about phenomena in the universe called 'Black Holes and Baby Universes

[1]http://www.wikipedia.org (16.02.2008)
[2]http://www.computer.org/portal/site/csdl/index.jsp (16.02.2008)

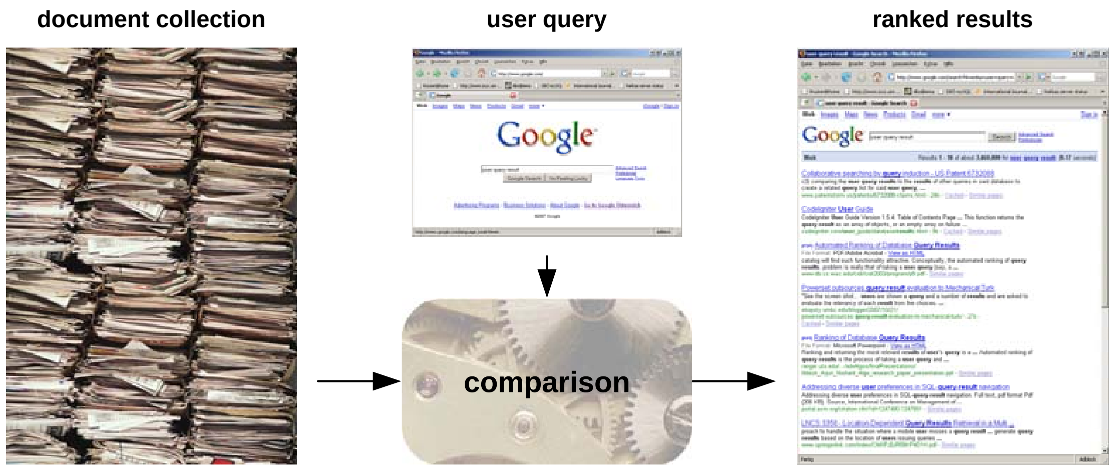


Figure 1.1: The process of information retrieval

and Other Essays'. The length of this document is about 50.000 words, where chapter eleven ($\sim$4.000 words) describes black holes in detail, containing figures and tables that explain the anatomy of black holes. In order to search the collection, a search engine (e.g., google) analyzes all documents and offers a keyword-based query interface. A user who is interested in the general nature and functioning of black holes might enter the search terms 'black hole physics functioning'. As a result the system returns a list of documents. Within this result the newspaper article is probably ranked as one of the top ten matches, whereas Stephen Hawking's book is located at the very end of the list. Why would that be the case? Because the estimated one hundred words of the newspaper contains twice the terms 'black hole', on average $\frac{2}{100}$. In contrast, the terms 'black hole' are mentioned two hundred times in the chapter of the book, on average $\frac{200}{50.000} = \frac{2}{500}$. As a consequence, the book is ranked[3] much lower than the newspaper article although the query terms occur more frequently in the book. Would chapter eleven be treated separately as a unit of about 4.000 terms, the average occurrence of 'black hole' in the chapter $\frac{200}{4.000} = \frac{2}{40}$ exceeds that of the newspaper article by far. Thus, the book chapter itself would be ranked higher than the newspaper article.

Structured Document Retrieval (SDR) suggests a solution to this problem by including structural information inherent in documents to improve the results of search engines. The goal is to retrieve not only complete documents, but also parts of documents that are relevant to the given user query. These parts are explicitly defined by the logical structure of the document (e.g., document, chapter, section, paragraph, figure). Thus, the content of relevant parts irrespectively of their granularity is returned. In this context, the idea is to adapt and apply mechanisms from traditional information retrieval along with extra source of

[3]For the sake of illustration relative term frequencies are used instead of applying any standard weighing formula.

knowledge mainly of structural nature. Such additional structural information assists the retrieval process in three ways:

- It enables structure-aware experts to express very specific queries, including explicit constraints on the content and structure.
- It allows to achieve highly focused search results without the need of further search and browse activities by the users.
- It reduces the amount of data that has to be searched by ignoring unwanted structural elements (e.g., no figures, no tables, only paragraphs) and documents not fulfilling structural constraints.

In many cases structural information is already included in documents and can be (semi-) automatically extracted from these sources: For instance, HTML [6] webpages contain tags like `<h1>`, `<h2>`, `<p>`, `<img>`. LaTeX [7] sources contain markup such as `section`, `subsection`, `figure`, `table`. MS-Word [10] and PDF [8] documents support `heading1`, `paragraph`, `embedded image` formatting. However, these formats mix up structure and layout information. To cope with that problem, the eXtensible Markup Language (XML) [4] came into existence. XML offers an appropriate format to represent hierarchically organized knowledge within semi-structured text documents. XML schemata [11] and the former Document Type Definition (DTD) [5] provide a stable framework for defining a valid structure of XML documents. This allows to establish a standard to uniformly access information emanating from different sources. Since most structured document retrieval systems operate on XML documents, structured document retrieval and XML retrieval are used synonymously.

Whereas classical information retrieval treats documents as atomic units, XML suggests a tree-like view. The root node represents the entry point to the document. The content of the document is (mainly) located at the leaf nodes of the tree. Inner nodes are understood as intermediate nodes in between the root and the leaf nodes. For example, consider a <document> node (root node) with several `<section>` nodes (inner nodes), where each section consists of multiple `<paragraph>` nodes (leaf nodes). From this point of view an XML document can be seen as a structured set of XML components or XML elements, where each component is accessible via a unique path starting at the root node.

Besides the information retrieval discipline, document mining becomes more and more an issue in processing large information repositories. Automatic grouping of similar documents or parts of documents is able to enhance the performance in both regards, computational complexity and retrieval quality. Therefore, document classification and document clustering mechanisms need to be adapted to fit the paradigm of hierarchically structured documents.

This chapter sketches the main challenges of structured document retrieval and discusses various aspects pertaining to XML retrieval. In line of these challenges, issues of representation, storage, retrieval, and organization of structured documents are covered. A brief overview of the structure of this work concludes the chapter.

1.1 Problem Identification and Motivations

In addition to traditional information retrieval requirements, SDR systems face several new challenges inherent in structured documents only. Due to the internet, a real flood of quite diverse document formats (e.g., HTML, XML, PDF) became widely accepted. Also, the ways of structuring documents themselves have been evolving independently. As a consequence, XML documents of different sources are highly heterogenous due to their multiform structure. Till now, no standards and not even guidelines are available that conveniently describe document structures. Often, documents come without any schema specification, so possible (sub)structures can therefore only be intuitively guessed. In order to process XML documents efficiently without being too restrictive, retrieval systems have to consider these structural specificities.

Having solved the problem of heterogeneity, the kind of storage has to be decided. In this context not only the content but also the structure and metadata of XML documents have to be considered in indexing and searching. The element hierarchy has to be retained, supporting fast access to document parts as well as to sets of parts (i.e., document subtrees). Several proposals for storing structured documents have been made, ranging from relational databases to native object-oriented XML databases. However, for the sake of structured document retrieval all of these proposals have to be adjusted to find a balance between functionality and performance [153].

Representation of the content and structure of documents strongly relies on the storage mechanisms. Since XML implements a hierarchical structure, the content is mostly (but not necessarily) restricted to the leaf nodes. The content of an inner node is recursively derived from the contents of its descendant nodes. For the indexing procedure, the level of granularity is defined implicitly by the document structure itself. During retrieval, the granularity of the result elements stands for a challenge. For instance, if a section and a paragraph within this section are both relevant to a query, which of them should be returned to the user – the section, the paragraph, or both? Yet, a user searching for information about an author may not be satisfied by returned components referring to the authors' name only. Thus, the target components may differ from the components returned. Clearly, ranking based on relevance of individual components has to take the inclusion relationship and possibly the explicit user's preference into account. One strategy is to return the so-called Best Entry Points (BEP) [151] serving as starting points for further document browsing activities. Related to the granularity aspect is the context of XML components. A paragraph might be relevant to a query because of its preceding and succeeding paragraphs only. The same paragraph in another context may be completely irrelevant. Thus, context, even being not included in the answer directly, has to be accounted for.

Retrieval systems often operate in highly dynamic environments where documents are constantly added, removed, and altered during runtime. Therefore, the indexing process has to be fast and the calculated indices have to be kept stable. Corpus-based weighing techniques such as the inverse document frequency of the vector space model (described in Section 2.3.3) need adaptations to avoid constant re-computations of millions of representations each time a document changes.

In order to retrieve XML components that are relevant to a query, proper representations of both, the query content and the components' contents, are essential. Much work on Natural Language Processing (NLP) has already been carried out in the field of traditional information retrieval to enhance the quality of textual representations. However, in the context of flat document retrieval NLP methods have shown only minor effects on the retrieval quality but have had a high impact on the (computational) complexity. With structured documents, the portions of textual content are generally much smaller (e.g., paragraphs). This fragmentation offers new possibilities to apply NLP techniques that improve, extend, and refine indexing and matching procedures of short text passages. Statistically motivated patterns (e.g., multi-terms such as composite nouns or named entities) can be effectively used to make content representations more meaningful and matching more precise. Text mining provides methods that extract valuable patterns from large amounts of text automatically.

A major benefit of exploiting structured documents is that users are able to express very specific information needs. For that purpose, SDR systems use query languages that support formulations of complex queries containing combinations of structure and content constraints. Initially, existing query languages were extended to handle structural information. Then, new query languages dedicated to SDR were developed. Very often, users must express their information need explicitly. In addition to the query language, users need to know the underlying structure of the target documents. This may be easy for domain experts, but not for general users. Another difficulty is that most of these languages are quite complex, making it hard to apply them directly [233]. Newer approaches rely on graphical interfaces that assist users to formulate their queries. The query is generated automatically by the interface avoiding incorrect syntax and semantic errors.

Once a query is formulated, the system computes the set of relevant XML components. The retrieval process consists of matching, ranking, and filtering of document components. While in traditional information retrieval a query is compared to a document only once, structured document retrieval has to match the query with each component of a document. These multiple similarity computations per document require efficient mechanisms that reduce the number of comparisons and ensure fast system response. One solution is that structural constraints are applied as a filter for XML components. Generally, sequencing filters does reduce the data load considerably.

Processing of retrieval tasks can be supported and improved by document mining techniques in two ways: First, document classification enables users to create groups of related XML components, which allows browsing and searching such groups. Basic requirement to do so is the definition of a non-trivial similarity function that compares two document trees. Second, document clustering identifies groups of similar documents called clusters without human intervention. Comparing a query to these clusters instead of single documents also reduces the number of similarity computations considerably.

Of particular importance in structured document retrieval is the ranking of the results. It has to reflect multiple matches of the structure, content, and metadata of single components as well as (i.e., structural match, content match) structural relationships among the results. Based on the tree-like representation, an adequate presentation of the results is needed to prevent users from getting lost. Because multiple components even of single documents may be retrieved, browsing functionality of documents containing relevant information is indispensable. Graphical interfaces assist users with highlighting and focusing relevant components while explaining the underlying structure and context.

All of the aspects mentioned above are strongly related to each other. For instance, the representation of documents and their storage are interdependent. Queries are formulated on the basis of the underlying document structure, which in turn defines the result presentation during browsing. Thus, structured document retrieval systems have to consider each of these aspects to perform well. Having identified the crucial points of structured document retrieval, the next section provides an overview of the approach taken in this work.

1.2 Overview of the Approach

The focus of this thesis is on processing XML structured documents for information retrieval, where the main issues include document representation, storage, content representation, indexing, retrieval, classification, and clustering of XML components based on their content and structure.

As mentioned earlier, the formulation of complex queries requires the competence of the users in the documents domain, structure, available metadata, and query language. Hence, the main target user group of SDR systems is considered to be domain experts (e.g., academic research, jurisprudence area). But even without deeper knowledge (e.g., queries without any structural constraints), the retrieval results are more useful than complete documents for two reasons. First, parts of documents usually provide more precise answers to specific questions than whole documents. Second, parts of documents are already focused by the search engine so there is no need for further search and browse activities within documents.

Bearing in mind the challenges presented in the previous section, a system for structured document retrieval has been developed. This system, X-DOSE, is based on a scalable client-server architecture which allows for distributed computing. Initially, new documents are mapped onto a generic document format and stored in a relational database. A graphical interface assists users in formulating their queries using XOR, an extended version of the XML-Oriented-Retrieval language [96]. Given a query, the system returns a result set of ranked XML components to the client. Users are able to browse documents, where relevant components are focused and highlighted applying relevance-based colors. A graphical representation helps users to focus on the results retrieved without losing track of the documents' structure. Adding additional constraints identified during inspection of results enables iterative query refinement that yields more focused results. In the sequel, brief descriptions of the basic system features are presented:

Document Transformation In order to process heterogenous collections efficiently, the system transforms documents temporarily into a generic XML format. This format retains the logical structure and supports metadata information on each structural level, including back-references to the corresponding parts in the original document. The benefits of the mapping procedure are simple and uniform data access, reduction of structural ambiguities, and general processing steps.

Storage Based on the generic XML format, transformed documents are stored in a relational MySQL[4] database. Structure, content, and metadata are maintained separately to improve access times. The original content of the transformed documents is also kept in the database, mainly for two reasons:

1. Transformed documents may refer to documents that do not allow direct access to certain parts (e.g., postscript documents). In response to a query, the content stored is displayed to the user as a hint where to find the relevant information within the original source.
2. It allows database-supported full text searches, post processing, and re-indexing without accessing the original source document again.

Document representation Documents are internally represented as ordered document trees consisting of XML components, where the content is restricted to the leaf nodes only. During retrieval, the content of inner nodes is computed on-the-fly using the contents of its descendant nodes. Additional metadata information associated with XML components supports the identification of relevant results.

[4]http://www.mysql.com (02.10.2008)

Content Analysis and Representation For the sake of efficient text representation, methods from the field of natural language processing are exploited. Tokenization, stemming, tagging, and stopword filtering are applied to extract index terms from textual contents. In addition to that, multi-terms such as named entities, composite nouns, formulaic speech, and full forms of acronyms are identified to improve content representation and matching.

Pure textual content is represented as weighed feature term vector, where the Vector Space Model (VSM) [213, 211, 214, 21] provides well-established formulae for weighing and comparison. However, the vector space model applies corpus-based statistics for term weighing (e.g., the number of documents including a term). This notion has to be reconsidered in the context of structured documents, since weights of descendant nodes cannot be simply combined to form weights of ancestor nodes. Instead, plain term frequency vectors are stored and propagated upwards. Weighing is done on-the-fly during retrieval after the propagation process.

Classification and Clustering Two techniques, classification and clustering of document components, are applied to improve retrieval tasks. At the user-side, classification according to predefined categories supports browsing, search restriction to a small subset of elements, and automatic identification of similar contents. It is based on a similarity function that compares two XML components regarding both the content and structure. Clustering, on the other hand, generates a hierarchy of clusters used to reduce the number of comparisons needed during retrieval. Therefore, cluster representatives reflect the features of the cluster's components. Given a query, only similar clusters (through their representatives) are subject to retrieval.

Retrieval Retrieval of XML components includes mechanisms for searching content, structure, and metadata information. Each type of information is treated differently, then a combined relevance value is computed based on the similarities of these types. Additionally, a set of user-specific query parameters allows to tune the results. Explicit constraints on the structure and metadata are applied for pre-filtering of XML components to speed up retrieval process.

The system is evaluated using real world data provided by the INitiative for the Evaluation of XML Retrieval (INEX) [82]. The next section describes the logical organization of the work.

1.3 Thesis Outline

The thesis comprises two main parts. The first part (Chapters 2 to 7) provides the theoretical background of structured document retrieval and introduces concepts and methodologies that are used in the development of a SDR system. The second part (Chapters 8 and 9) describes the system and its evaluation using real world data. Detailed information about indexing, retrieval, classification, clustering, and evaluation procedures are given. Each chapter comprises a related work section, reviewing previous research that has been conducted in this field. In particular, the chapters are presented as follows:

Chapter 2 presents the basic concepts and methodologies structured document retrieval systems rely on. Key issues of successful systems are described and brought into context. The chapter tries to answer questions like: What are the proper units to be indexed and which units are to be retrieved? How is the content within a document tree represented? Where are the bottlenecks and possible solutions?

Chapter 3 covers document format, storage, and representation issues. Focusing on document-centric XML documents, a generic document schema is proposed which supports highly sophisticated user queries addressing the content, structure, and metadata. In addition to that, the underlying relational MySQL database is explained. An adequate representation for XML documents is described, relying on various term space considerations.

The goal of any retrieval system is to return relevant information. In written documents this information is expressed in the form of natural language text. Chapter 4 introduces fundamental natural language processing methods that transform these texts into a computable form. Analysis steps are described and brought in relation to each other, where one steps' output serves as input for a subsequent step. Tokenization, tagging, stemming, and stopword filtering are explained and extended to achieve optimal outputs. Finally, the autonomous tasks are assembled in a sequence that transforms natural language texts into weighed feature term vectors.

Based on the previous chapter, Chapter 5 deals with more complex natural language processing tasks. It is concerned with high-level pattern mining to extract multi-terms (e.g., named entities) and to identify and resolve acronyms. The process chain described previously is extended to include these tasks. Due to this process, multi-term indices are created that improve retrieval performance.

Classification as a means of document organization is introduced in Chapter 6. The main issue is how two document trees are compared to each other. Based on this similarity, k-Nearest-Neighborhood classification is used to assign unseen documents to pre-existing categories. Initial experiments using an extra dataset for XML document classification yielded good results.

Chapter 7 is concerned with document clustering. In contrast to classification, clustering aims at finding commonalities among documents without user intervention. Thereby, similar documents are grouped into clusters. The difficulty is to define an appropriate representation of a cluster that accounts for all features of the documents assigned to it. In this work, an approach based on supertrees is proposed. Pruning of uncommon branches is applied to increase the similarity between a cluster representative and single documents. Several similarity measures are formulated and compared to each other. In order to validate the theoretical assumptions in this chapter, a set of experiments was run on a special dataset dedicated to the evaluation of XML document clustering. The preliminary results were promising. Further investigations on the improvement of retrieval tasks due to clustering are carried out in the evaluation chapter.

In Chapter 8, the architecture of X-DOSE, the XML-Document Oriented Search Engine, is described in detail. X-DOSE is based on a client-server architecture using a MySQL database. While the client provides a user-friendly graphical interface, the server implements the processing procedures for indexing, classification, clustering, and retrieval. The whole system is written in Java, where the modular design supports replacements and extensions of single system components in a simple and neat manner.

Chapter 9 provides an extensive evaluation of X-DOSE. An official set of about 17.000 IEEE computer science articles provided by INEX [90] serves as basis to measure the performance of the approach. Besides experiments that measure the accuracy due to natural language processing, comparisons to similar systems using the same dataset are carried out.

Finally, Chapter 10 summarizes the theoretical and experimental results achieved. Contributions of the work in the field of structured document retrieval are highlighted and open issues are identified. An outlook of future research directions concludes the thesis.

Chapter 2

Information is the currency of democracy.

Thomas Jefferson

Structured Document Retrieval: An Overview

This chapter introduces structured document retrieval and provides the theoretical background of this work. The focus is put on preconditions and difficulties of applying information retrieval on structured documents.

2.1 Introduction

Information retrieval has become an important discipline in computer science. A vast amount of textual information and data is freely available to everyone, demanding for automatic methods that help to find and navigate through it. Additionally, information is no longer plain and unstructured. It has become a conglomerate of content, structure, and metadata. The main challenge now is to apply information retrieval techniques that fit this kind of information. The discipline that aims at resolving these difficulties is called structured document retrieval.

The main idea of structured document retrieval is to exploit the structure together with the content of documents to improve retrieval performance. This allows users to additionally retrieve parts of documents, so-called document components, that better fit their information needs instead of whole documents (e.g., an entire book). Thus, the aim of structured document retrieval is to return

- document components of varying granularity (e.g., the entire document, a chapter, a subsection, a single paragraph/table/figure) that are
- relevant to the user regarding their content, structure, and metadata.

Thus, focused retrieval – as structured document retrieval is often called – is concerned with finding the best entry points in structured documents [151]. These entry points are serving as starting points for further search and browse activities by the user.

The advantages of applying structured document retrieval are obvious: Large documents often mask relevant information because most of the entire document is irrelevant to a user query. Thus, parts of documents often provide better and more focused answers to queries regarding a certain topic. Moreover, precise queries can be formulated addressing not only the content, but also the structure and available metadata. Numerous studies have been conducted which highlight the improvement of retrieval effectiveness applying structured document retrieval [257, 151, 156].

Text documents can be regarded as structured in several ways: The simplest (implicit) structure is according to the linear order of words, sentences, and paragraphs. Second, a structure is given by the links a document contains (e.g., hyperlinks, cross-references, citations). Also, temporal or spatial relationships within multimedia documents define a certain structure. Anyway, the most obvious way to define a documents' structure is to utilize the logical structure inherent in the documents' content (e.g., chapter, section, paragraph). Structural elements are defined as labeled nodes (optionally containing content) that are organized in a hierarchy. In structured document retrieval, this latter logical structure is exploited to define the units used for indexing, storing, and retrieving. According to the logical structure, textual information can be distinguished into three different categories [21, 195]:

Unstructured information refers to a raw text, optionally containing markup for syntactic purpose only (e.g., formatting tags in HTML files). Every piece of information can be placed everywhere within the document. Placement in certain positions and/or markup tags do not indicate any semantic information about the content. Search results are whole contents (documents) that are relevant to a query to a certain degree.

Structured information refers to text which is formatted in a strict manner (e.g., database-like records in CSV files). Additional metadata explicitly defines the type, length, and other attributes of the content. Here, a piece of information is allowed only in a certain place associated with a certain semantic expressed in the metadata. Search results are data objects (e.g., data tuples) that are exact answers to queries addressing the content according to the metadata. Without knowledge about the underlying metadata and thus about the document structure, relevant information cannot be extracted.

Semi-structured information strikes a balance between these categories defined above (e.g., XML files). As with structured information, extra metadata describes the format of the content. However, it allows partial matching and missing elements. User-defined markup tags identify the meaning of the encapsulated content. Relationships between content containers are specified via nesting and references. Thus, the structure of a text is expressed without being too restrictive with content attributes (type or length restrictions, arbitrary nesting) and placing of different elements (e.g., sections, paragraphs, figures). Query results are parts of documents that match constraints on both, the content and the structure.

Retrieval models combining information on textual content and the logical structure are called structured text retrieval models [21, pp. 62]. When speaking of structured text documents, in the remainder of this thesis we rather mean semi-structured documents. It is widely accepted by the scientific community that these terms are used synonymously. The reason for this is that in contrast to strictly formatted database systems textual documents are either considered structured or unstructured at all. In this work the term structured text document is used, although it more correctly refers to semi-structured text document.

While the logical structure provides documents with hierarchical levels of granularity, and hence more precision can be achieved by means of focused retrieval, it does, however, imply more requirements on the representation, storage, indexing, matching, retrieval, ranking, and graphical user interface. Because documents and queries are structured, standard retrieval models such as the vector space model (see Section 2.3.3) are not adequate anymore. Of course structured document retrieval involves the same tasks as traditional information retrieval. However, several aspects of information retrieval applied to flat documents have to be reconsidered and cannot be straight-forwardly applied on structured documents. Rather these methods have to be adapted to fit the structured document paradigm. In the past, structured document retrieval approaches focused on one of three issues [86]:

- The structural approach enriches traditional content-based approaches by including a structural constraint component. Documents are understood as an ordered set of independent nodes. This allows to restrict terms to certain document nodes by specifying additional constraints on the structure. However, these models are based on the boolean retrieval model and do not support weighing of index terms and ranking. A survey of these approaches is given in [185].
- The content-based approach represents a document as a sequence of plain text segments. A first attempt in this direction is passage retrieval [41, 257, 148], which is closely related to traditional content-based information retrieval. Only few researchers combined explicit structural information and content-based retrieval [86].
- The tree-matching approach represents both documents and queries as ordered labeled trees. Thus, document retrieval is understood as an approximate tree-matching problem. Work about this approach can be found in [219, 220].

Like in traditional information retrieval it is necessary to first identify and define the major tasks of structured document retrieval. These include the following aspects:

(a) Document format: Crucial for the performance of any retrieval system is its underlying document collection and the document format used. Besides processing and performance issues, it further has to support textual and binary (e.g., multimedia data) contents, fit the structural expressiveness requirements, and optionally include metadata at certain structural levels.

(b) Representation: Not all nodes in a document provide a meaningful portion of information because they might be too small or relevant only in a certain context. As a first step so-called indexing nodes are identified, defining the basic components that are able to be retrieved. Textual content of the hierarchically structured documents is generally restricted to the leave nodes. Hence, mechanisms to represent the content of inner indexing components have to be defined.

(c) Ranking: Related to the previous aspect, a scoring function to match document components and queries expressing their similarity and thus their relevance is needed. Based on the score the final results are ranked and returned to the user.

(d) Retrieval granularity: An important question is whether the retrieval units must be known ahead of time or are dynamically decided by the system itself. Based on the above score, the elements to be returned have to be decided.

(e) Query language: In order to formulate a users' information need a query language has to support complex constraints on the content, structure, and metadata.

(f) Result presentation: The way results are presented is a key issue and has to be considered early in the design phase. Once ranked, the results are displayed showing their context of appearance together with their relevance score. Browsing as a means to further explore documents containing multiple result components is inevitable.

(g) Evaluation metrics: Measuring the performance of a newly proposed retrieval system is essential to show its benefits. To be able to compare the results achieved against similar systems, a well-defined document collection, a set of user queries, valid results of the queries, and proper evaluation metrics are required.

In the sequel this chapter discusses these aspects.

2.2 XML Document Format

The XML document format [4] has become an emerging standard for the representation of text documents. Simply being a text document, the content is marked up using self-defined tags which can be further enriched by metadata via attributes. The tags themselves must not overlap, providing a hierarchical structure of the document. From this point of view an XML document can be seen as a semi-structured text document that can be described as a tree of nodes. The root node corresponds to the first XML element. Each child node recursively starts a new branch of the tree. Generally, the content of the document is contained in the leaf nodes. However, XML allows nodes to contain both content and child nodes. These nodes are referred to as mixed content nodes.

XML documents may contain any kind of contents in their nodes, even bytecode. It further allows to combine metadata and content information in a single document. Based on their application area, XML documents can be separated into two different categories [85]:

Data centric XML stores fully structured information in a database-like style. It is mainly used for data interchange. Two parties are enabled to exchange data based on their defined document scheme (DTD or XML schema) and an eXtensible Stylesheet Language Transformation (XSLT) [9] file. The XSLT file is used to translate documents instantiating one schema into documents instantiating a different one. Generally, the order among XML elements in such documents is not crucial.

Document centric XML, in contrast, deals with hierarchically structured full-text documents. As with data centric documents, a DTD or XML schema specifies valid document structures. But this kind of documents is not supposed to be transformed. Rather the schema is used to check whether a document is valid or not. In general, the order of XML elements plays a central role (e.g., the introduction always comes before the conclusion).

The difference between data and document centric documents is not a sharp one. Data centric documents may involve an order among their child nodes, whereas document centric documents may also contain unordered metadata like the author's name or section titles. But commonly it is clear whether XML documents are used in a data or document centric way. This work focuses on processing document centric XML documents, where the order of elements is of importance. However, special types of nodes dedicated to store metadata are interpreted in a data centric point of view.

Using XML for structuring documents yields several advantages. (1) XML allows to mirror domain and application specific document structures using a set of self-defined tags. (2) These tags can be further extended by metadata information in the form of self-defined attributes. (3) Moreover, XML supports internal and external linkage (XLink [2] and XPointer [3]). (4) XML elements in the structure are addressed via their unique path (XPath [53]) starting at the root element of the document.

The definition of the structure is located in a DTD [5] or in a newer XML schema file [11], to which XML document instances (the actual XML documents) are referring. Based on the structural specifications, XML instances can be checked automatically whether they are valid or not according to the schema given. In contrast to DTD, XML schemata also support the definition of various data types and formats to restrict the content within XML document structures.

Listing 2.1: XML example code

```
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="bookdef.xsd">
    <author>M. Hassler</author>
    <title>XML Retrieval</title>
    <abstract>This work ...</abstract>
    <chapter title="Introduction">
        <section title="Section 1">XML is one ...</section>
        <section title="Section 2">
            This section ...
            <subsection title="SubSec 2.1">These days ...</subsection>
            <subsection title="SubSec 2.2">Based on ...</subsection>
        </section>
    </chapter>
    ...
    <chapter title="Outlook">Crucial for ...</chapter>
</book>
```

Listing 2.1 shows a simple XML document and its representation as an ordered tree (Figure 2.1), where the order among child nodes is of importance. In the first two lines of code the specification file for valid structures, here an XML Schema file named `bookdef.xsd`, is referenced.

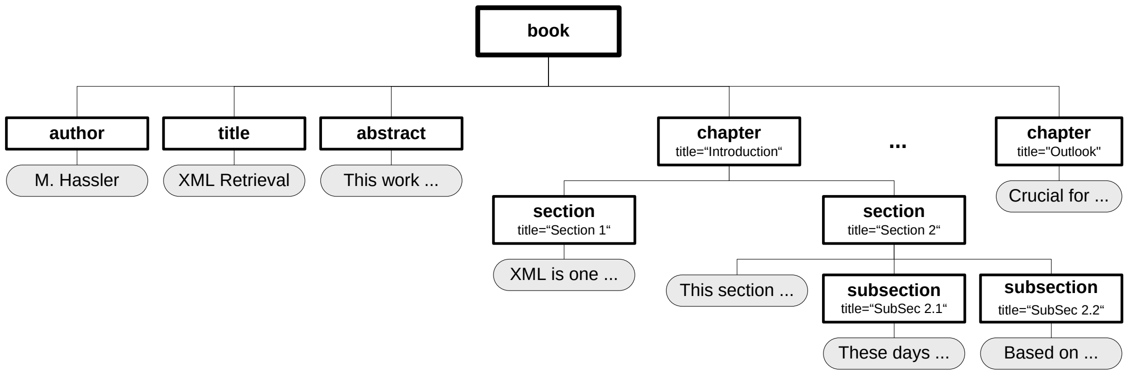


Figure 2.1: XML representation as an ordered tree

Gray shaded nodes in the XML tree refer to the plain content of the document. All other nodes contain only structural information. One might note that only a single node (`Section 2`) contains both, content and structure. Such a node is called a mixed XML node. Anyway, the content of XML structured documents is generally restricted to the leaf nodes. Additional information in form of metadata comes with the `title` attributes in the `chapter`, `section`, and `subsection` nodes. In the example the elements `author` and `title` are not implemented as attributes of the book element, which would also be a possible solution.

The XPath expression to address the content of the first subsection `SubSec 2.1` is given by `/book[1]/chapter[1]/section[2]/subsection[1]`.

Generally, there are no standards or even guidelines available that describe how documents are properly structured. Even the tagsets used for markup vary strongly from one domain to another. This simple example already indicates the difficulties one has to face when confronted with XML documents from heterogenous sources or different structure definitions (e.g., mixed nodes, attribute versus element, etc.).

Exploiting the XML format for information retrieval purposes has several advantages [168]: (1) more precise search by providing additional information in the elements; (2) unified access to documents from heterogenous sources; (3) powerful search paradigm using structural, content, and metadata specification; (4) data and information interchange to share resources and to support cooperative search.

Within the structured document retrieval paradigm, the structure provided by XML documents is consulted to define the units for indexing and retrieval. That is the reason for using XML retrieval as synonym for structured document retrieval or focused retrieval. In this context the goal of XML retrieval can be reformulated as retrieving document components (XML elements) of arbitrary granularity that are relevant to a certain user query instead of whole documents only.

2.3 Related Work

This section outlines several aspects and work related to structured document retrieval. It covers historic developments and depicts currently probed approaches in this field.

2.3.1 Database and Information Retrieval Perspective

There are two main communities who have concentrated their efforts on the computation of semi-structured XML documents: the database community and the information retrieval community [167, 195].

The database-oriented approach tries to tackle the problem of querying structured information in a database like manner. Most of the information on the internet exists in form of HTML documents and/or is stored in relational databases. Both information sources can be represented using XML (e.g., see [1]) as an intermediate document format. This opens the door to apply database-like search techniques which are further extended to support proximity search and ranking.

Simply speaking, XML documents are plain text files. Thus, information retrieval approaches can be applied straightforward to search and retrieve information from these documents. As soon as XML turned out to be closely related to HTML, the information retrieval community focused their research on this area [195]. A first attempt was to simply ignore the additional markup, which led to lower retrieval performance [167]. More elaborated approaches include extra indices of the structure, support regular expression matching of XML paths and contents, and affiliate basic semantics with certain structural elements. However, these extensions were not as straightforward as assumed, leaving much space for further research.

2.3.2 Fragment and Passage Retrieval

Early work on returning parts of documents instead of full documents in response to user queries was done by John O'Connor [188, 189]. The idea is to segment a text into so-called text passages and then to apply standard indexing and retrieval models. In this context a text passage is defined as a set of 'consecutive words', a sequence of words in 'reading order'. According to this approach, documents are considered as linear sequences of text fragments. Passages are mostly defined as non-overlapping and do not exactly match the underlying document structure. Later work by Mittendorf and Schäuble [181] focuses on query-independent text segmentation using Hidden Markov Models.

Another work, although based on passages but closer related to the notion of logical structure, was proposed by Burkowski in [41]. In this approach a document is represented using multiple lists of non-overlapping passages, where each list corresponds to a certain level in the hierarchical structure expressed through tags (e.g., headlines, sections, paragraphs, authors). Queries are formulated as retrieval command strings, implementing a text algebra based on a defined containment model. The final result set is ranked applying a statistical ranking strategy based on document length, term frequency, and inverse document frequency.

Salton et al. [212] probed several passage sizes which are evaluated using the SMART retrieval system. Similar experiments are conducted by Callan [42] who uses paragraph-sized and window-sized passages in the INQUERY retrieval system. His evaluation results show that a combination of document- and passage-level outperforms the others.

Further experiments done by Wilkinson [257] confirm that combined results of whole documents and parts of documents enhance retrieval. In [258] he and his colleague found that a page size of about 1000 bytes define useful fragments.

Kaszkiel and Zobel [148] came to the same conclusion as Wilkinson, showing that a fixed length passage approach is efficient and robust. However, in subsequent experiments they found that passages of arbitrary length improve retrieval performance significantly.

Based on these findings, classical passage retrieval methods assume a window of fixed length which is slid stepwise over the whole text of a document [50]. In each position the distribution of words is analyzed. Passage boundaries are found if the distribution of words within two subsequent window positions changes significantly. Having identified the passages, it is straightforward to apply standard information retrieval strategies on the corpus of text passages instead of the documents. However, passages do not exactly reflect the logical structure of documents intended by their authors. Thus, retrieval performance strongly depends on the applied segmentation of a text into passages. Also, splitting texts into passages predefines all possible units for indexing and retrieving in a static way. This is one of the reasons why the logical structure of documents is believed to improve the efficiency of retrieval systems [50].

Grossman and Frieder [110] briefly review initial work in passage retrieval and discuss approaches as marker-based passages, dynamic passage partitioning, and merging passage-based similarity measures.

2.3.3 Retrieval Models

Language Models

Ogilvie and Callan [190] propose a tree-based generative language model approach for ranking document components. In their work they use probabilistic context free grammars to estimate the probability of parse trees for sentences within certain components. The probability of a specific parse tree, and thus its ranking, is computed as the product of the probabilities of all rules applied in creating this (sub)tree. For each leaf node in the document tree the language model is estimated directly from the text attached to the node. Therefore, three different ranking models, the Kullback-Leibler Divergence, the Generative Language Model, and the Maximum-Likelihood Estimate are evaluated. In order to represent inner nodes, linear interpolation is applied, where the parameters for combining language models of children to ancestor nodes are learned from a large data repository. These parameters conform to the augmentation factor described by Fuhr et al. [83]. Smoothing, the re-estimation of the probabilities in a language model, improves the estimates by including knowledge based on sample data.

Kamps et al. [145] suggest to support language-model based ranking strategies by priors, smoothing functions, and cut-off values. While priors allow non-content features such as the length of a document component to be included in the scoring mechanism, smoothing is used to increase the relevance score of small components containing only a small subset of the query terms. Without smoothing, larger components containing more query terms are favored. In order to avoid too small components to be retrieved they propose cut-off values for the minimal length of retrieved elements. Their experiments show that extreme length normalization is an important issue in XML retrieval as a means to improve retrieval performance.

Bayesian Models

Piwowarski et al. [202] use Bayesian networks for the sake of XML retrieval. Complex queries are decomposed and translated into elementary subqueries, where each subquery refers to a single component. Subqueries are implemented as Bayesian networks (directed acyclic graphs). In the graph, nodes represent XML components (tag names) that are interconnected via arcs representing relations between the components. The joint probability of a set of i components $\{x_i\}$ is given by

$$P(\{x_i\}) = \prod_i P(x_i|x_i\ parents)$$

where x_i *parents* denotes the parents of node x_i in the network. The content of document components is represented using external models. The relevance score $P(d|q)$ of a document d with respect to query q is given by

$$P(d|q) = \sum_{i=1..n} P(x_1, x_2, ..., x_n|q)$$

In order to answer a user query, the evidence is added to the network as shown in Figure 2.2. For every target element to be retrieved, the set of elementary networks is connected to

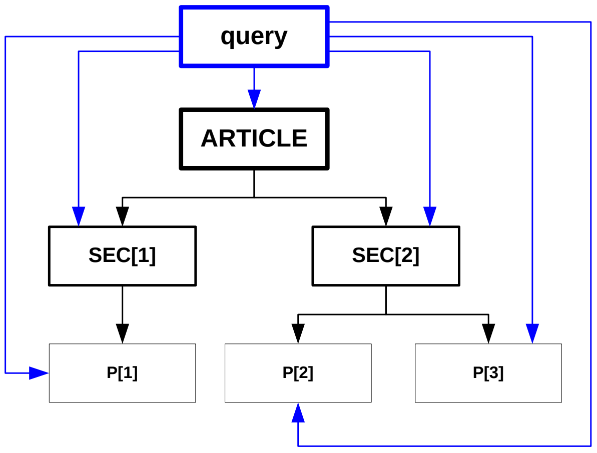


Figure 2.2: Structured document retrieval using Bayesian networks [202]

compute its global score. As an example, the relevance of $sec[1]$ in the figure is given by:

$$P(sec[1]|q) = \sum_d P(d|q)P(sec[1]|d,q)$$

Additional connection parameters are estimated from the corpus using the Estimation Maximation algorithm [63].

The Vector Space Model

Introduced by Salton in 1968, the vector space model [213, 211, 214] provides a solid framework that allows to compute a degree of similarity between a document and a query. The model represents documents and queries as term vectors with associated term weights. Term weights w are calculated by combining term statistics (term frequency tf) and corpus-based statistics (inverse document frequency idf) using the Equations 2.1–2.3 [21, pp. 27–30].

$$tf_{i,j} = \frac{freq_{i,j}}{\max freq_{l,j}} \tag{2.1}$$

$$idf_i = \log \frac{N}{n_i} \tag{2.2}$$

$$w_{i,j} = tf_{i,j} \cdot idf_i \tag{2.3}$$

The actual weight $w_{i,j}$ of a term i in a document j is computed as the normalized term frequency $tf_{i,j}$ times the inverse document frequency idf_i. $freq_{i,j}$ refers to the raw term frequency of term i in document j, and $\max freq_{l,j}$ is the maximum raw frequency of any term l in document j. idf_i can be seen as a terms' discriminative power among the documents. In the formulae N is the total number of documents in the system, whereas n_i stands for the number of documents containing term i. The effect of the inverse document frequency is to downweigh terms that occur frequently throughout the whole collection, and the other way round, to increase the importance of terms that are rare.

Queries are represented and weighed in the same way. The similarity of a document d_j and a query q is defined as the cosine measure between the two weighed feature vectors (Equation 2.4). t denoted the total number of terms.

$$sim(d_j, q) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{|\overrightarrow{d_j}| \cdot |\overrightarrow{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{j=1}^{t} w_{i,q}^2}} \tag{2.4}$$

Vector Space Model based Approaches

Grabs and Schek [106, 105] propose a model based on vector spaces generated on-the-fly for flexible XML retrieval. The main idea is that content on different levels in a document tree is considered to be of different importance with respect to a query. Therefore, index nodes are identified that are indexed and retrieved. More distant nodes are treated as less important than nodes closer to the index nodes. Term weighing is done using the traditional $tf \cdot idf$ formula, where the notion of idf is extended to inverse element frequency. For their experiments at INEX 2002 Grabs and Schek distinguish single-category retrieval, multi-category retrieval, and nested retrieval (which they call flexible retrieval), reflecting the complexity of the queries (single-category $\leq$ multi-category $\leq$ nested). Single-category retrieval refers to retrieval of index nodes addressed through a single path expression (e.g., paragraphs in main sections `/ARTICLE/SEC/P`). Multi-category retrieval extends retrieval to disjunct sets of index nodes (e.g., paragraphs and figures in main sections `/ARTICLE/SEC/P` or `/ARTICLE/SEC/FIG`). The aim of nested retrieval is to provide efficient and consistent retrieval over arbitrary combinations and nestings of XML components. It enables queries that are restricted to whole subtrees (e.g., everything within main sections `/ARTICLE/SEC/*`). Subtrees may comprise besides multiple categories additional non-index nodes at higher levels and complex containment relations. Augmentation is applied to downgrade term weights according to Fuhr et al. Indices and statistics are preprocessed and stored for leaf elements only. Representations of inner nodes are computed on-the-fly when needed.

Another approach based on the vector space model is proposed by Mass and Mandelbrod in 2004 [175]. In order to overcome the difficulties occurring in nested components, they create separate indices for components on the same level (e.g., all documents, all sections, all paragraphs). For each of these indices, formulae from the vector space model are used for weighing and for similarity computation. During query processing, all indices are searched, resulting in multiple result sets of varying granularity. As a final step, a document pivot factor is applied to combine and re-rank the retrieved result sets. In their experiments Mass and Mandelbrod show a 30%–50% improvement of mean average precision compared to previous retrieval results achieved in [174].

Liu et al. [164] also propose a $tf \cdot idf$ model for structured document retrieval. XML documents are represented using Ctrees, which is a compact tree representation for the whole document collection. Queries are expressed in the NEXI language (see Section 2.3.5) used at the INEX workshop in 2003 and represented as query trees. The content of XML elements is represented as weighed feature vectors applying a combination of term frequency, inverse element frequency, label weights, and weights for query term modifiers. During retrieval, the query is decomposed into a set of retrieval components which are grouped according to their structural specificities. XML components matching a subquery are evaluated independently and relevance scores are assigned. The overall relevance of an XML component is obtained by merging all subquery relevance results to a single score. Finally, a ranked list of XML elements is returned to the user.

2.3.4 Retrieval Units

Hatano et al. [125] addresses the problem of determining proper retrieval units of XML documents. By defining XML elements as retrievable (meaningful) or not retrievable (meaningless), the number of targeted portions of XML elements can be reduced. Kazai et al. refers to meaningless contexts as stop contexts [152] (e.g., XML elements carrying only layout information). This distinction leads to improvements during indexing, matching, and retrieval. The remaining (meaningful) portions of XML documents are called coherent partial documents. They are identified automatically based only on the topology of a document tree. Thus, no explicit definition of the structure in form of a DTD or XML schema file is needed. A context node is defined to be an ancestor node which does not have sibling nodes of the same type (element name). To avoid too small contexts of text nodes, a minimum distance of two nodes (grandparent) is demanded.

2.3.5 Query Languages

XIRQL [84, 85, 87, 86] is an XML query language based on XQL [206] which is extended by information retrieval concepts for weighing, data types and vague predicates, relevance-oriented search, and semantic relativism. These functionalities are implemented via special XIRQL operators. Boolean operators enable the user to combine subqueries to more complex ones. The final XIRQL query is transformed into an underlying path algebra defined in [85, 86], which is then optimized and executed.

Theobald and Weikum [237] propose XXL (fleXible XML search Language) for querying structured documents. In XXL a query is internally represented as a directed labeled data graph, including a set of wildcard placeholders for path constraints and single element names. Besides the standard set of operators on strings and other simple data types, a special tilde operator $\sim$ is used to enable semantic similarity matching based on ontologies. A graphical interface named Visual XXL GUI helps formulating complex user queries.

Proposed by O'Keefe and Trotman [191], NEXI (Narrowed Extended Xpath I) [240] is developed as a query language used at INEX since 2004. Since XPath seems to be too complex for end-users to formulate their information needs [195], NEXI implements only a subset of XPath to address certain XML elements (hence its name narrowed). Because XPath does not support information retrieval like matching, NEXI is extended by an `about()` predicate that enables similarity-based content matching.
For example, the NEXI query `/doc[about(., computer)]//sec[about(./par, apple)]` searches all documents `<doc>` that contain the term `computer` and returns sections `<sec>` with paragraphs `<par>` that contain the term `apple`. The semantics of NEXI queries is not defined; instead, the retrieval engine has to deduce it, which is why NEXI differs from database-oriented query languages. Introducing NEXI at INEX significantly improved retrieval results: The error rate of experts formulating complex queries dropped from 63% (in 2003) to 12% (in 2004) [241].

Geva et al. [96] introduced XOR, the XML Oriented Retrieval language, which is intended to replace NEXI as query language in INEX. XOR is based on the previous NEXI syntax and therefore backwards compatible. Extra functionalities are added to support advanced information retrieval tasks. These extensions include a negation operator for content matching (`-about(...)`), logical operators for query combination (`AND`, `OR`, `NOT`, `ANDNOT`, `SUPPORT`), path matching extensions (`strict`, `vague`), term extensions (e.g., part-of-speech tags, case of letters), logical operator qualifiers (`strict`, `vague`), and additional predicates (e.g., `LinkTo()`, `LinkFrom()`, `Contains()`, `lt()`, `eq()`, `gt()`). Based on these definitions, an open-source parser for the XOR language was developed. The parser is able to translate XOR queries into reverse polish notation supporting the implementation of back-end processors. However, XOR is not supposed to be used as an end-user query language. It is meant to be an underlying language model supported by natural language query interfaces and query generators. Reported results of four different baseline systems using XOR instead of NEXI show significant retrieval improvements.

2.3.6 Performance

Fuhr and Gövert [79, 80] address an issue regarding the performance of structured document retrieval systems. They suggest index compression to scale down storage requirements for indexing XML documents. In their work, inverted files with compressed paths are used to speed up database access. Further, a new data structure called XS tree is proposed, which stores structural information of a document in a very compact form.

2.4 Index Objects

The first step of searching structured documents is to define the way documents are indexed. This index provides the basis for applying weighing and ranking formulae, which allow similarity-based matching of XML components to queries. Generally, there are two possible options: the development of a completely new weighing scheme; or the adaptation or generalization of existing ones (e.g., language models, Bayesian models, vector space model).

According to Fuhr and Großjohann [85], the long experience with existing weighing and ranking strategies in traditional information retrieval favors the second option. Thus, the atomic units or index objects of a document must be identified. According to Fuhr and Großjohann [85] this results in two benefits: first, traditional information retrieval models can be applied for processing these elements; and second, only index objects are returned to the user.

Starting with the assumption that textual contents are restricted mainly to the leaf nodes of the document tree, these leafs are obvious candidates for atomic units. However, their granularity might be to fine grained as retrieval result [85]. Just assume that a query searching for `M. Hassler` returns only the component `<author>M. Hassler</author>`, not including any additional information about the document he wrote. Without any context, this turns the result completely useless. Hence, Fuhr and Großjohann propose to rely on a hand-crafted definition of disjunct index nodes (also called index objects or contexts). These index nodes are identified either by (1) analyzing the underlying DTD or XML schema, or alternatively (2) according to the type of content (e.g., `chapter` elements).

The root of a document itself is defined as the uppermost index node. The contents of all remaining non-index nodes are indexed at their nearest ancestor index node.

Figure 2.3 shows an example document and its corresponding index nodes (dashed boxes). In order to get the complete content of an arbitrary index node, the contents of all descendant index nodes are propagated upwards (arrows) and are combined with the content of the current index node itself.

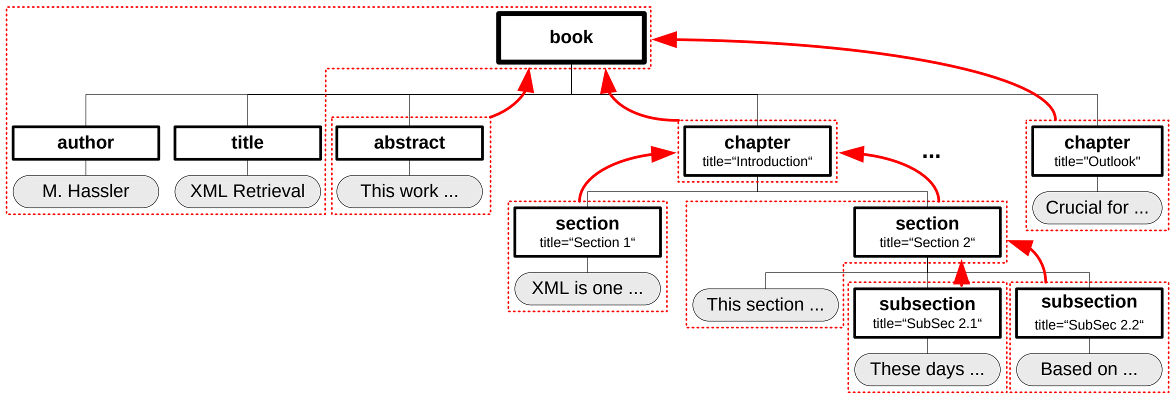


Figure 2.3: Example XML document tree with disjoint index objects

An automatic identification of appropriate index (and retrieval) nodes is proposed by Hatano and his colleagues [125]. The idea is to select only nodes that have sibling nodes at the same level in the hierarchy. These nodes are characterized by the same path starting at the root node. Figure 2.4 depicts the identified indexing paths `/book/chapter`, `/book/chapter/section`, and `/book/chapter/section/subsection` of the previous example. Again, the root node is defined as prime index node. In order to avoid unidentified index nodes in document instances where only a single element is implemented (e.g., only a single chapter within a book), the identification can be shifted to an analysis of the DTD or XML schema instead. However, in cases where a definition of the structure is missing this option might not be available. Thus, multiple chapters in documents are indexed whereas a single chapter is not. In the above example one can see that applying this approach misses at least the `<abstract>` component as an index object. If other nodes similar to `<abstract>` occur frequently within the document domain, an automatic approach seems not to be the best solution.

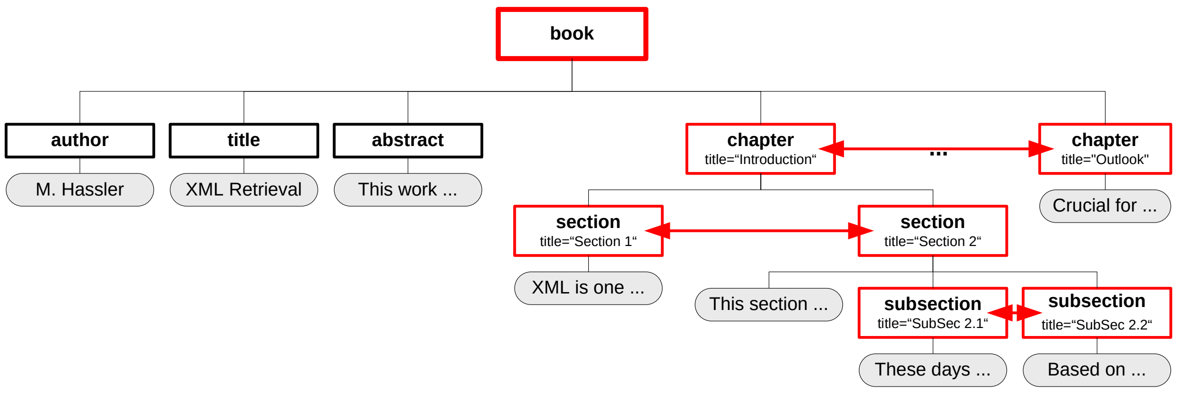


Figure 2.4: Automatic identification of disjoint indexing units

The reason for indexing only a selection of nodes results in two main advantages:

- The number of nodes which have to be indexed is reduced. Especially in the context of structured document retrieval this increases performance during the indexing, matching, and retrieval processes.
- Meaningful portions of information are defined independently of user queries. This ensures that the system does not return XML components that are too small to be interpreted without contexts.

2.5 Content Representation and Weighing

In order to represent structured documents for retrieval purposes, many approaches have been proposed [85, 151, 44, 156]. Good surveys are given by Luk et al. [168], Pal [195], and Pinel-Sauvagnat and Boughanem [201].

The representation is always guided by the efficiency of processing queries and retrieving relevant parts of documents. While many structured retrieval systems rely on persistent XML databases, the representation varies from one system to another. Furthermore, mechanisms for indexing and ranking document components during retrieval are different. Approaches proposed by the information retrieval community are similar to those investigated by the database community to some degree. However, while the latter class of approaches aims at dealing with the boolean model, the former is concerned with more advanced models that allow to handle both structure and content of documents. From this point of view it is reasonable to combine both paradigms, that is fast database processing and sophisticated similarity computation to efficiently and accurately compute retrieval results.

Having identified the index objects described in the previous section, methods borrowed from flat document retrieval can be applied to represent and weigh the content of these nodes.

Traditionally, content-based retrieval systems rely either on the boolean model or on the vector space model [213, 211] to represent the content as a bag of words. Extensions of these models have been proposed, e.g., the fuzzy boolean model, knowledge-aware models, or the extended vector space model [21].

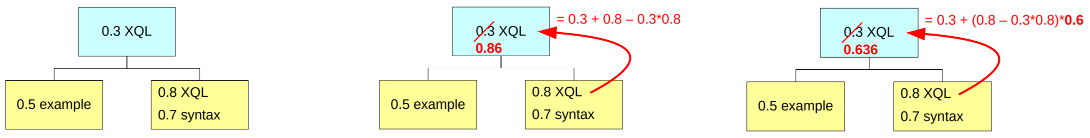
However, all of these indexing models ignore the logical organization of texts expressed by the documents' structure. In order to apply existing models to structured document retrieval the content of components is represented and stored using flat indexing. Beyond that mechanisms dealing with the hierarchical relationships among document components have to be applied.

For illustration purpose this section relies on the vector space model (described in Subsection 2.3.3) to represent the content as a weighed feature term vector. The actual weight of a term is calculated by combining term statistics and corpus-based statistics (Equations 2.3 – 2.2). The challenge that arises at this point is to aggregate multiple disjoint feature vectors of descendant index objects to a single feature vector of their parent index object. For this purpose two different solutions are feasible [17]:

Propagation of term statistics To compute the weight of terms for a given node, raw term statistics (e.g., term frequency, document length) of its descendants are accumulated (i.e., summed up). Weighing formulae are applied afterwards to yield the final term weights (on-the-fly weighing [106]). In other words, term weights are calculated independently of the weights associated with the node's descendants. Thus, no recombination of term weights (using empirical parameters) is needed and existing formulae can be applied without being altered.

Propagation of term weights In contrast to the propagation of term statistics, term weights of a node are computed by aggregating the weights of its direct descendants without explicitly referring to their term statistics. Therefore, mechanisms for combining multiple term weights have to be defined.

A concept proposed by Fuhr and his colleagues to combine the representations of lower components with higher ones, first described in [83], is called augmentation. The idea is to augment the content of a component by the contents of its children. Figure 2.5 explains the concept of augmentation in a brief example. The left tree represents an XML document consisting of three components (Figure 2.5a). The root element contains the term `XQL` and consists of two children elements (mixed content node). The first child states the term `example` and the the second child mentions the terms `XQL` and `syntax`. The numbers in front of the terms are the independently computed terms weights of each component. The root node of the tree in the center contains already the propagated term weight for `XQL` (Figure 2.5b). Here, the weight of child components adds to the ancestors weight by applying the inclusion-exclusion formula presented in [27, pp. 20]:



(a) Initial term weigths **(b)** Aggregated term weights **(c)** Aggregated term weights with augmentation factor of 0.6

Figure 2.5: Propagation of term weights (cf. [86])

$$P(e) = P(C_1 \vee ... \vee C_n) = \sum_{i=1}^{n}(-1)^{i-1}\left(\sum_{i\leq j_1 \leq ... \leq j_i \leq n} P(C_{j_1} \wedge ... \wedge C_{j_i})\right)$$

However, this example shows already the dilemma: the term weight of the ancestor always exceeds the weight of the child. The solution proposed by Fuhr et al. is to introduce so-called augmentation weights ($\in [0;1]$) for downgrading propagated weights. In the right tree (Figure 2.5c) an augmentation factor of $0,6$ is applied. While applying this factor still increases the weight of the ancestors term, it still keeps the descendants weight higher. By allowing different augmentation weights in different index nodes, domain and collection dependent tuning of term weight propagation is possible.

Since term weights already include corpus-based statistics such as the inverse document frequency, representations of root nodes applying augmentation diverge from flat document representations on the same content using the same weighing formulae.

In this work the first approach, namely the propagation of term statistics, is followed. This decision is mainly taken for one reason: The content representation of the root node exactly matches the traditional flat information retrieval approach. Due to this correspondence it is possible to test and compare the performance of structured document retrieval to traditional information retrieval engines on the document level. Whichever propagation is used, one has to be aware of its most important implication: Due to the aggregation of information from descendant nodes to ancestor nodes, a relevant child component implies relevance of all its ancestor components – at least to a certain degree. The calculation of XML component representations carried out in this work can be summarized as a four-step process:

1. Disjoint index objects are identified
2. Contents of index objects are represented and stored in form of term statistics (local representation)
3. Starting at the leaf nodes, term statistics are propagated upwards the document hierarchy (global representation)

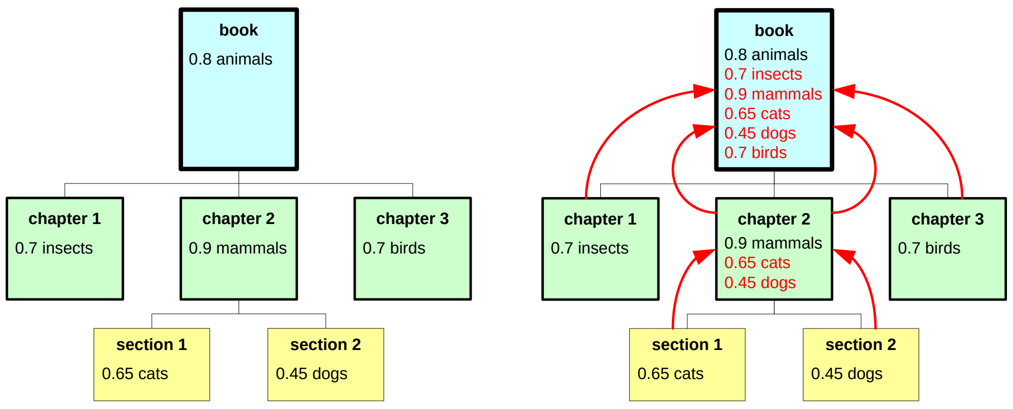
4. The final representations are computed by weighing the propagated term statistics using corpus-based or path-based statistics (weighed feature vector)

2.6 Querying and Ranking

Representing index objects as described in the previous section is a basic requirement of querying and subsequent ranking of document components. The concept of searching within structured documents is based on the comparison of each components' representation with a query that is represented in the same way. In contrast to flat documents, each query is compared multiple times (once for each component) to a single structured document. Hence, scalability depends on efficient algorithms to manage these vast amount of comparisons

Sticking with the vector space model (see Subsection 2.3.3), the similarity of a document component and a query representation is defined as the cosine measure between the two weighed feature vectors (Equation 2.4). However, computing the similarity of single XML component contents stored in the database independently is not sufficient. It is important to include the contents of descendant nodes as well. To illustrate this issue consider the example in Figure 2.6a. The same document with its aggregated representations indicated is given in Figure 2.6b. For illustration purpose the term weights are passed upwards without being changed. Note that this refers to maximal weights at the ancestors, hence propagation is meant only to reduce propagated weights. If someone wants to know something about pets, this document (Figure 2.6b) is queried for instance with the weighed terms `0,90 cats` and `0,80 dogs`. Figure 2.6c gives the similarities (cf. Equation 2.4) of all document components and this query, showing an interesting result: Both sections are relevant to the query reaching a similarity of 74,7% and 66,4%. Chapter 2 is less relevant than both sections (65,5% similarity). From a focused retrieval point of view this comes quite unexpected, because the second chapter is the smallest component that fully covers the query. Thus, it is expected to be the optimal (top-ranked) result to this query. The whole book itself is relevant to a degree of 44,9%, which again meets a users' expectation.

From this example it seems to be necessary to include further mechanisms that reflect structural relationships more accuratly. In structured document retrieval every index object is a potential query result item. Hence relevant descendant components imply that ancestor components are also relevant to the same query (even to a less degree), overlap of query results is unavoidable. This ability puts additional requirements on the retrieval mechanism, which has not only to return relevant components but also components of the correct granularity [146]. Neither should the components be too small nor should they be too large. From this point of view, the highest rank should be assigned to the second chapter in Figure 2.6. One possible solution is to redesign the propagation strategy. However, this



(a) Initial tree **(b)** Aggregated tree

document components

weighted term vectos	book	chapter 1	chapter 2	section 1	section 2	chapter 3	query
animals	0.80	-	-	-	-	-	-
insects	0.70	0.70	-	-	-	-	-
mammals	0.90	-	0.90	-	-	-	-
cats	0.65	-	0.65	0.65	-	-	0,90
dogs	0,45	-	0.45	-	0.45	-	0.80
birds	0.70	-	-	-	-	0.70	-
$\vec{d_j} \bullet \vec{q}$	0,95	-	0,95	0,59	0,36	-	
$\lvert\vec{d_j}\rvert, \lvert\vec{q}\rvert$	1,75	0,70	1,20	0,65	0,45	0,70	1,20
sim(d_j,q)	**0.449**	**-**	**0,655**	**0,747**	**0.664**	**-**	

(c) Cosine similarities

Figure 2.6: Query matching in structured documents

may lead to weighing inconsistencies at the representation level. Another approach combines the computed similarity values of ancestors and descendants directly. The final score, often called Retrieval Status Value (RSV), has to take account of these issues.

As stated above the retrieved units in structured document retrieval are not whole documents but document components. This leads to difficulties in query matching, because a query is no longer matched against one (large) document feature vector. Now it is compared to a lot of (respectively small) leaf representations, as well as to many inner nodes with accumulated – though not necessarily small – feature vectors. Using a term space that consists of all document terms, smaller portions of text consequently lead to sparser representations. This sparseness of feature vectors has to be considered, because single term matching results in high relevance values. Generally, natural language processing techniques have not significantly improved the performance of flat document retrieval [55, pp. 227]. However, in the domain of structured documents dealing with small portions of text retrieval quality is believed to benefit from deeper linguistic analysis.

2.7 Query Language

In the past several languages for querying XML documents have been proposed (XPath [53], XQL [206], XQuery [28], XML-QL [66]). An overview of the features of five of these languages (LOREL, XML-QL, XML-GL, XSL, and XQL) is provided by Bonifati and Ceri in [29]. Two other surveys are given by Deutsch et al. [67] and Luk et al. [168].

However, these languages are still lacking some important features. Missing concepts range from ignoring data types and absence of term weighing functionality to unsupported similarity matching [84]. Also, the complexity of these languages is too high and the syntax is too complicated for users to express their information needs [259]. Anyway, these languages provide basic building blocks and good starting points for the development of more appropriate information retrieval query languages. The requirements put on a query language can be summarized as [87, 86]:

Weighing Document and query term weights enhance retrieval performance.

Relevance-oriented search If no specific result elements are specified, the system should return the most specific document components.

Data types and vague predicates Data types refer to semantic categories that explain the content of an XML component more clearly (e.g., `PersonName`, `Title`, `PlainText`). Data types allow for better search results in case these are addressed in a correct manner. Special search predicates defined on certain data types provide sophisticated matching functionality.

Structural vagueness If a user searches for a certain value without caring about the underlying document structure, the system should be able to generalize structures to fit the users information need. This includes attribute versus element distinction, generalization of ancestor-descendant relations, as well as similarity of element and attribute names. Furthermore, data types provide another kind of generalization regarding elements and attributes.

Based on these requirements more appropriate higher-level query languages like XIRQL, XXL, NEXI, or XOR (see Section 2.3.5), to name just some of them, have been developed. Intended to bridge the gap between pure structure-oriented and content-based retrieval, all of these languages have their own syntax and semantic, but in general, their basic functionalities remain the same. However, only NEXI and XOR are supposed to be interpreted by a retrieval engine only. Listing 2.2 presents an example query in XOR.

Listing 2.2: XOR query example

```
1 //book[about(.,XML) ANDNOT about(.,XSLT)]//*//subsection[about(.,These days)]
2     AND
3 //book[eq(.//author,M. Hassler) OR about(.//*,Retrieval)]
```

In the listing, the XOR query is composed of two main subqueries (line 1 and 3). The first subquery retrieves `subsection` elements containing `These days` which occur in books that treat `XML` but not `XSLT`. The second subquery retrieves `book` elements where the author is exactly `M. Hassler` or where any element is about `Retrieval`. Finally, the two subqueries are combined via an `AND` condition. The order of `AND`-combined subqueries is implicitly defined: The subquery addressing components of larger granularity (line 2 returns `book` elements) functions as filter for the subquery addressing smaller components (line 1 returns subsections). As a result, a set of `subsection` elements within `book` elements is returned to the user.

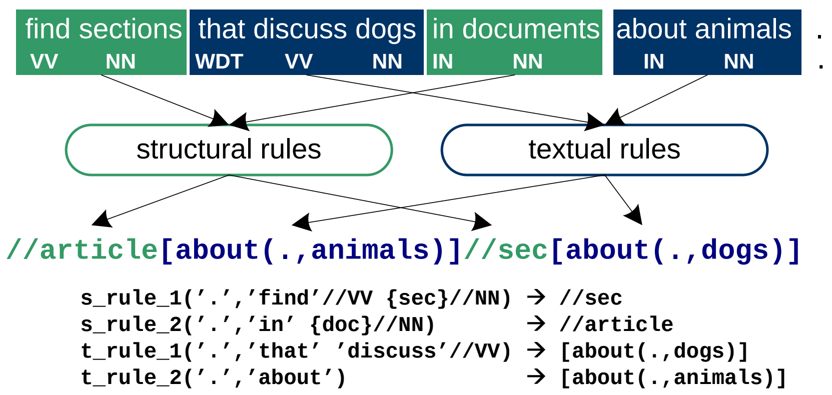
This example shows that formulating such queries is definitely not an easy task and even error-prone for expert users. Hence, this language is not intended to be applied directly by users. It is supposed to be supported by query generators offering an (interactive) user interface to formulate queries that are both, syntactically and semantically correct.

Novel approaches for querying XML documents are presented by Woodley et al. [259]. In their work the goal is to transform natural language queries into the NEXI query language automatically. Although different, the approaches described comprise four common query processing steps: (1) detection of structure and content constraints; (2) mapping of structural constraints onto corresponding XML tags; (3) deriving of NEXI-conform content requirements; and (4) NEXI query formulation. Three different approaches were tested in the natural language query track at the INEX workshop in 2005 [233]. An approach proposed by Hassler is based on template matching of words and part-of-speech tags. Therefore, patterns are used to extract two kinds of rules handling the structure (`s_rule`) and the content (`t_rule`) (see Figure 2.7). Woodley and Geva suggest a shallow syntactic analysis before applying similar template matching. Tannier includes deep syntactic analysis complemented by semantic rules concerning the structure and content. Their experiments showed promising results, even outperforming the performance of a baseline system.

2.8 Result Presentation and Browsing

In contrast to traditional retrieval results (i.e., a ranked list of documents), the returned items in structured document retrieval (i.e., a ranked list of document components) are no longer independent of each other. The result set may include:

- elements of varying granularity (e.g., whole documents, sections, single paragraphs or figures)



```
s_rule_1('.','find'//VV {sec}//NN) → //sec
s_rule_2('.','in' {doc}//NN)       → //article
t_rule_1('.','that' 'discuss'//VV) → [about(.,dogs)]
t_rule_2('.','about')              → [about(.,animals)]
```

Figure 2.7: Query analysis with template matching [233]

- multiple disjunct elements of the same document
- overlapping elements that are in structural relationships (ancestor - descendant of the same document)

Of course, this puts special requirements on the visualization of these results. A structured document retrieval system should foremost retrieve the most specific component of a document (its best entry point) answering a given query [51]. Starting from this element, other relevant elements in the document are explored by browsing.

In [109, 108], Großjohann et al. propose a user interface for both, query formulation and result presentation. The query formulation interface called HyGate [81, 86] aims at constructing queries in the XIRQL query language in a query-by-example way. The result presentation interface displays the retrieved document components to the user, reflecting relationships among elements stemming from the same document. The resulting components are depicted in a 2D graphic called TreeMap, where each element is displayed as a rectangle. Child components are drawn as nested within their ancestors rectangle (see Figure 2.8). Brightness (white means not relevant, dark means highly relevant) is used to reflect a components achieved score. The concept of TreeMaps is further advanced to Partial TreeMaps (see Figure 2.9), where elements that are not included in the result set are removed for better readability.

This approach is extended by Fuhr et al. [87] to cover two additional aspects: (a) structural relationships among result components (from the same document), and (b) size of the result components. TextBars (see Figure 2.10), as these graphical representations are called, visualize a document as a bar that is segmented according to its underlying structure. Above the bar the title of the document is given. The length of the bar reflects the size of the document (number of words). According to the size of an XML component, red separators indicate the borders of different components (index nodes). Segments are colored using different shade

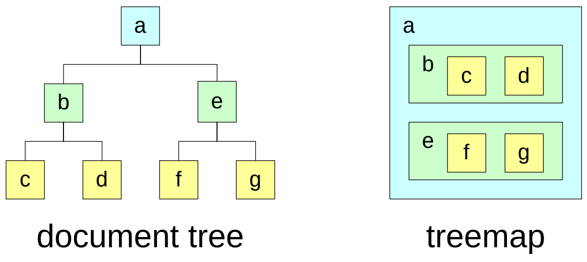
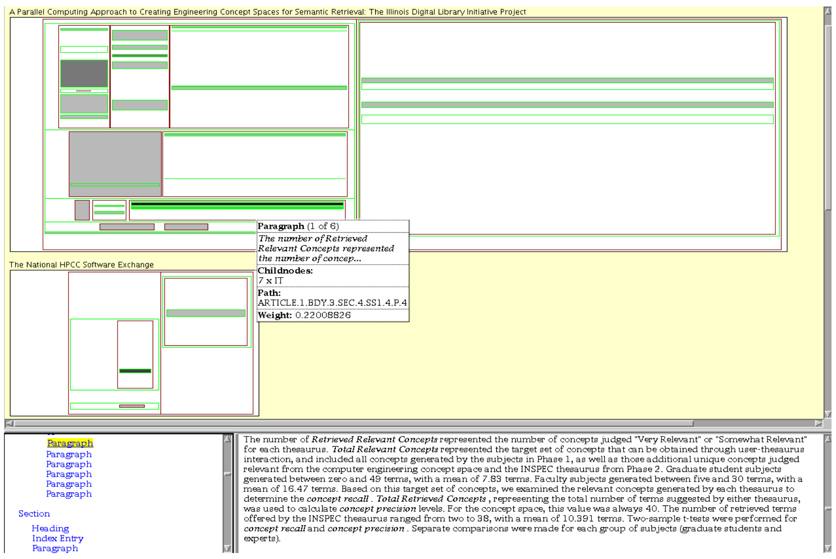


Figure 2.8: XML tree and its TreeMap [109]


A Parallel Computing Approach to Creating Engineering Concept Spaces for Semantic Retrieval: The Illinois Digital Library Initiative Project
The National HPCC Software Exchange
Paragraph
Paragraph
Paragraph
Paragraph
Paragraph
Paragraph
Section
Heading
Index Entry
Paragraph

Paragraph (1 of 6)
The number of Retrieved Relevant Concepts represented the number of concep...
Childnodes:
7 x IT
Path:
ARTICLE.1.BDY.3.SEC.4.SS1.4.P.4
Weight: 0.22008826

The number of *Retrieved Relevant Concepts* represented the number of concepts judged "Very Relevant" or "Somewhat Relevant" for each thesaurus. *Total Relevant Concepts* represented the target set of concepts that can be obtained through user-thesaurus interaction, and included all concepts generated by the subjects in Phase 1, as well as those additional unique concepts judged relevant from the computer engineering concept space and the INSPEC thesaurus from Phase 2. Graduate student subjects generated between zero and 49 terms, with a mean of 7.83 terms. Faculty subjects generated between five and 30 terms, with a mean of 16.47 terms. Based on this target set of concepts, we examined the relevant concepts generated by each thesaurus to determine the *concept recall* . *Total Retrieved Concepts* , representing the total number of terms suggested by either thesaurus, was used to calculate *concept precision* levels. For the concept space, this value was always 40. The number of retrieved terms offered by the INSPEC thesaurus ranged from two to 38, with a mean of 10.391 terms. Two-sample t-tests were performed for *concept recall* and *concept precision* . Separate comparisons were made for each group of subjects (graduate students and experts).

Figure 2.9: Result presentation with Partial TreeMaps [87]

factors reflecting their relevance to the query, where white means not relevant at all and black means totally relevant.

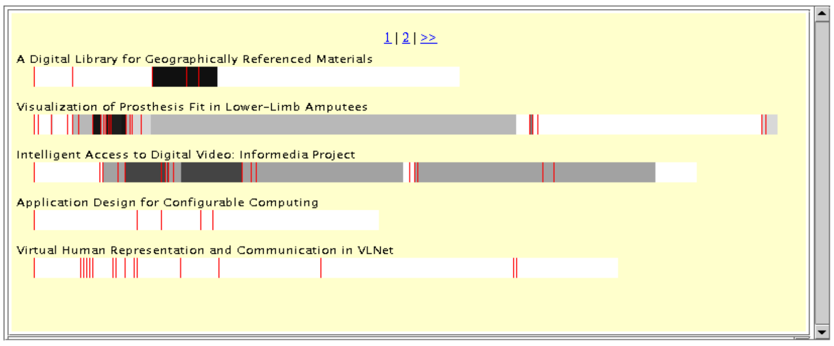

1 | 2 | >>
A Digital Library for Geographically Referenced Materials
Visualization of Prosthesis Fit in Lower-Limb Amputees
Intelligent Access to Digital Video: Informedia Project
Application Design for Configurable Computing
Virtual Human Representation and Communication in VLNet

Figure 2.10: Result presentation with TextBars [87]

2.9 Retrieval Evaluation

A key issue in information retrieval, and thus in structured document retrieval, is the evaluation of the retrieval performance. A solid evaluation of any IR-related system requires (1) a predefined document test collection (corpus), (2) a set of tasks a system has to perform (queries, topics), and (3) precisely defined evaluation metrics reflecting the retrieval quality. These preconditions enable system developers to determine the performance of a system and create the basis for an impartial comparison of heterogenous systems dedicated to the same tasks.

Currently much research is done in the field of structured document retrieval evaluation. An initiative that is devoted solely to this topic is the INitiative for the Evaluation of XML retrieval (INEX) [82, 89, 91, 90, 88, 92], organized by the European DELOS[1] Network of Excellence for Digital Libraries.

One of the main tasks of INEX is to provide a framework for querying and retrieving XML document components not only via content but also via structural constraints. It includes

- a large collection of real-world XML documents,
- a set of user queries called topics,
- relevance-assessed results of experts for each topic, and
- evaluation measures.

Every year the INEX workshop held in Dagstuhl Castle in Germany brings together experts from around the world, which are discussing aspects of retrieval methodology and models, evaluation measures and strategies, query languages, and future research directions. Each participating research group evaluates their system and reports the achieved results at the workshop. The workshop is structured into several tracks. The last workshop held in December 2007 embraced the following tracks [92]:

Ad Hoc Retrieval Standard INEX track evaluating the retrieval performance of structured document retrieval systems.

Book Search Investigates book-specific relevance ranking, user interfaces and behavior, special issues such as book indexes, and linkage to external sources such as metadata catalogue information.

Document Mining Tests machine learning methods for structured documents and evaluates their performance, focusing on XML classification and XML clustering.

Entity Ranking Compares and evaluates techniques for returning ranked XML components.

Heterogenous Collections Treats interoperability issues of documents from different sources (different structure, different tags, different coding).

[1]http://delos-noe.iei.pi.cnr.it/ (08.02.2008)

Table 2.1: Development of INEX

	Groups	Papers	Collection	Tracks
INEX 2002	49	23	12107 docs 494 MB 60 topics	Ad Hoc Retrieval
INEX 2003	46	28	12107 docs 494 MB 60 topics	Ad Hoc Retrieval, Interactive, Heterogeneous Collection, Relevance Feedback, Natural Language
INEX 2004	59	34	12107 docs 494 MB 60 topics	Ad Hoc Retrieval, Interactive, Heterogenous Collection, Relevance Feedback, Natural Language
INEX 2005	64	59	16819 docs 764 MB 87 topics	Ad Hoc Retrieval, Interactive, Heteterogenous Collection, Relevance Feedback, Natural Language, Document mining, Multimedia
INEX 2006	85	57	659388 docs 4,6 GB 130 topics	Ad Hoc Retrieval, Interactive, Heterogeneous Collection, Relevance Feedback, Natural Language, Document Mining, Multimedia, Use case studies, XML Entity Ranking
INEX 2007	105	60	659388 docs 4,6 GB 130 topics	Ad Hoc Retrieval, Heterogeneous Collection, Document Mining, Multimedia, Entity Ranking, Link the Wiki, Book Search

Link-The-Wiki Analyzes the approaches to automatic link discovery.

Multimedia track Focuses on the retrieval of multimedia components.

INEX was first organized in 2002 and involved around 50 participating groups from around the world. In the beginning INEX classified the approaches into three categories [103]: IR model-oriented, DB-oriented, and XML-specific. One year later the categorization was changed into model-oriented and system-oriented approaches [93]. Table 2.1 briefly summarized the historical development of INEX reflecting its increasing approval and importance throughout the scientific community.

2.9.1 Corpus

In the year 2002 a test collection containing 12.107 documents was created by INEX. The documents stemmed from 12 magazines and 6 transactions of the IEEE Computer Society covering a period of 1995–2002, with a total size of 494 megabytes of raw XML files containing no pictures or multimedia content. 169 different XML tags were used for markup. For two years the collection stayed nearly the same, only with some work spent on the correction of inconsistent syntax and structure of the XML files. On average each document contained 1.532 XML elements with an average nested depth of 6,9. In 2005, the collection was extended by 4712 new articles (16.819 documents) from the period 2002–2004, reaching a total size of 764 megabytes. Also, the number of unique tags increased to 192.

Preliminary results of this work were evaluated and presented at the INEX workshop in 2005. In order to be comparable and to show improvements achieved, this work sticks to the INEX 2005 document collection throughout all experiments.

In 2006 INEX adopted the Wikipedia document collection which is freely available on the internet comprising 659.388 XML documents. The documents, in contrast to the previous collection, make heavy use of internal and external linkage. In total the collection size is about 4,6 gigabytes of plain XML files, where each document on average consists of 161,35 XML elements in a nested depth of 6,72. The documents are structured according to the Wikipedia template, making use of about 5.000 different tags.

2.9.2 Topics

Each year the INEX topics (queries) used for the currently organized workshop are created by the participating groups themselves. Each group hands in a set of potential topic candidates from which the most appropriate are selected for evaluation. An example INEX 2005 topic is given in Listing 2.3. It consists of an initial topic statement, a title, a short description, a long description explaining the idea and the expected result, and its corresponding NEXI query.

INEX distinguishes two types of topics:

Content-Only (CO) queries that contain only constraints on the content. No structural information where to find the information nor any hint which elements should be returned is provided. The system has to search the collection and decides on its own which elements to return and how the ranking is done.

Content-And-Structure (CAS) queries allow to specify constraints on the content, the structure, or in most cases both. These queries require profound knowledge about the underlying document structure and are expressed in the NEXI language (`<castitle>` element in Listing 2.3).

Whereas the ranking for the CO queries is only a matter of content, CAS queries put much more effort on the ranking mechanism in order to combine multiple relevance values regarding the content and structure. For evaluation purpose CAS queries are interpreted in two ways: strict (SCAS), where only the specified target components are allowed to be returned; and vague (VCAS), where specified target components are treated more like preferences than strict conditions.

The optimal human answers are assessed manually by the participating groups. Each group evaluates two to three topics which takes about two weeks per topic. With the use of an online XML browser and markup tool, the whole collection is skimmed through for a topic and relevant answer elements are marked up and classified with respect to their exhaustivity and specificity. Results of participants assessing the same topic are cross checked to minimize

Listing 2.3: INEX topic example (2005)

```
<inex_topic topic_id="231" query_type="CO+S" ct_no="98">
    <InitialTopicStatement>
        I’m interested in the applications of markov chains in graph theory.
    </InitialTopicStatement>
    <title>
        markov chains in graph related algorithms
    </title>
    <description>
        Retrieve information about the use of markov chains in graph theory and in
        graphs-related algorithms.
    </description>
    <narrative>
        I have just finished my Msc. in mathematics, in the field of stochastic
        processes. My research was in a subject related to Markov chains. My aim is
        to find possible implementations of my knowledge in current research. I’m mainly
        interested in applications in graph theory, that is, algorithms related to
        graphs that use the theory of markov chains. I’m interested in at least a short
        specification of the nature of implementation (e.g. what is the exact theory
        used, and to which purpose), hence the relevant elements should be sections,
        paragraphs or even abstracts of documents, but in any case, should be part of
        the content of the document (as opposed to, say, vt, or bib).
    </narrative>
    <castitle>
        //(sec|p|abs)[about(.,+"markov chains" application "graph theory")]
    </castitle>
</inex_topic>
```

inaccuracies. After finalizing the topics assigned, INEX grants access to all available topic assessments.

2.9.3 Metrics

In order to measure the performance of information retrieval systems one needs to define formal metrics. In traditional information retrieval, the standard metrics recall and precision are used [21]. In contrast, result sets in structured document retrieval consist of document components of varying granularity which may contain overlapping elements. Thus, these metrics are not appropriate to reflect the retrieval quality in all its facets [240].

INEX proposes a set of metrics for the evaluation of such systems to deal with that difficulty. Hence the retrieval task in INEX is defined as returning XML components that are most specific and exhaustive [103], these two aspects have to be defined more clearly [93]. Exhaustivity expresses the extent to which an element covers the topic, and specificity means how focused an element is on the topic. Both dimensions are rated using a four-degree scale: (0) not, (1) marginally, (2) fairly, and (3) highly. Out of the 16 possible combinations, 10 meaningful *ES* tuple (Exhaustivity-Specificity) are taken [195].

$$ES = \{(0,0),(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\}$$

In order to apply different evaluation metrics, the two relevance dimensions are mapped onto a single relevance scale by a quantization function $f_{quant}(e,s) : ES \rightarrow [0,1]$. INEX 2002 used two variants of quantization functions [103], a strict one (Equation 2.5) and a generalized one (Equation 2.6). Later on, additional variants of f_{gen} are introduced with more focus put on either of the two dimensions.

$$f_{strict}(e,s) = \begin{cases} 1, if(e,s) = (3,3) \\ 0, otherwise \end{cases} \tag{2.5}$$

$$f_{gen}(e,s) = \begin{cases} 1, if(e,s) = (3,3) \\ 0{,}75, if(e,s) \in \{(2,3),(3,2)\} \\ 0{,}50, if(e,s) \in \{(1,3),(2,2),(2,1)\} \\ 0{,}25, if(e,s) \in \{(1,1),(1,2)\} \\ 0, if(e,s) = (0,0) \end{cases} \tag{2.6}$$

Additional metrics proposed by and used at INEX (precall, XCG, nXCG, T2I, and PRUM) are described in detail in [150, 195].

2.10 Summary

Online document repositories and digital libraries increasingly rely on structured documents. Hence, structured document retrieval methods become more and more important. This chapter introduced the challenges of searching within structured documents and provided answers how upcoming difficulties can be handled. A brief overview of ongoing research in the area is given. The main tasks of information retrieval, namely indexing and retrieval, were discussed in detail. As the presentation of retrieval results poses several difficulties one does not face in traditional information retrieval, two graphical approaches, treemaps and TileBars, were presented. Most important to prove the success of a new structured document retrieval approach is an extensive evaluation and comparison to similar systems. To do this, this work reverts to INEX, which is dedicated to the evaluation of XML retrieval systems. To assure comparability INEX provides a document collection, topics (queries and their assessments), and evaluation metrics.

Chapter 3

Simplicity does not precede complexity, but follows it.

Alan J. Perlis

Document Format, Storage, and Representation

This chapter is concerned with the most simple XML document format that could possibly work as a means to express structure, content, and metadata of text documents. By transforming documents into this generic schema the foundation for efficient storage and retrieval is laid. Typing, associated with tailored search predicates, supports sophisticated matching of information contained in different XML components. In order to compare the content of document components, the content of descendants is aggregated upwards contributing to the content of the ancestors, allowing to apply traditional information retrieval models. Therefore, two different modifications of the vector space model, one using a static term space and another one using dynamic term spaces, are presented. As a result documents of any source and of any format are represented in the same way, enabling fast and elaborate retrieval of documents and document components.[1]

3.1 Introduction

As soon as speaking of structured documents the question of 'what is structured' and 'how is the structure expressed' comes up. Usually, the logical structure of a document is covered by terms like 'chapter', 'section', 'paragraph', or 'figure', where the order of structural elements is of importance (i.e., the introduction always comes before the conclusion). Nesting of elements is used to summarize related contents under a common topic.

Unfortunately, there exists no standardized set of terms that describes the structure of a document. Also, no guidelines or limitations for proper structuring – as the number of allowed child elements or the maximum depth of nesting – are available. On the one hand this leaves much room to individually structure documents, on the other hand it complicates structured document retrieval tasks.

[1]Parts of this chapter have already been published in [124, 120]

Furthermore, the structure is tightly coupled to the intensions of an author in organizing the text. Important contents are introduced at the beginning and recur several times as the text evolves. Deeper nested contents provide more details, whereas contents at higher levels overview a topic. Different authors tend to structure their texts differently, so there is no consistency inherent in a set of documents written by different authors. Even single authors rarely stick to their way of structuring different documents over a longer period of time. This structural heterogeneity often leads to inconsistencies and ambiguities, especially in large-scale document management systems [172, pp. 201].

Besides heterogenous document structures, a wide variety of document formats are used, calling for diverse access methods. Both, heterogeneity of document structures and document formats make it hard for retrieval systems to perform well. Besides constraints on the content, queries addressing structured documents may contain additional restrictions on the structure or context (span of components). In the context of XML documents another issue arises: Documents on the net mostly come without any schema specification making it hard to guess which structural element contains which content.

In order to take care of these difficulties this work proposes to map incoming documents of arbitrary structure and format onto a single generic XML document format. The advantages of this procedure is that most of the inconsistencies and ambiguities can be resolved at a very early stage of retrieval. The consequences include:

- Homogenous documents regarding their structure and format (XML) exist.
- Common access methods to address the content of XML components are possible.
- Adoption of data types and sophisticated search predicates for specific contents.
- Definition of index objects based on a components' name or data type.
- Support of structural vagueness and semantic relativism of components [85] due to the mapping onto single distinct elements.
- Application- and domain-specific definition of retrieval units becomes possible.

In the sequel a brief overview of related work is given. All subsequent sections and chapters present the achievements obtained in this work. In Section 3.3, a minimal XML document schema that fits these requirements is defined. Efficient storage of the mapped documents using a relational database is described in Section 3.4. Based on the stored information, representation issues of arbitrary document components are outlined in Section 3.5. A short summary concludes the chapter.

3.2 Related Work

3.2.1 Typing of Structural Entities

Gövert [101] describes several extensions of search predicates that are added to the HyREX search engine. These predicates are based on different data types organized in a type hierarchy (see Figure 3.1). Each data type supports a set of search predicates which are inherited by

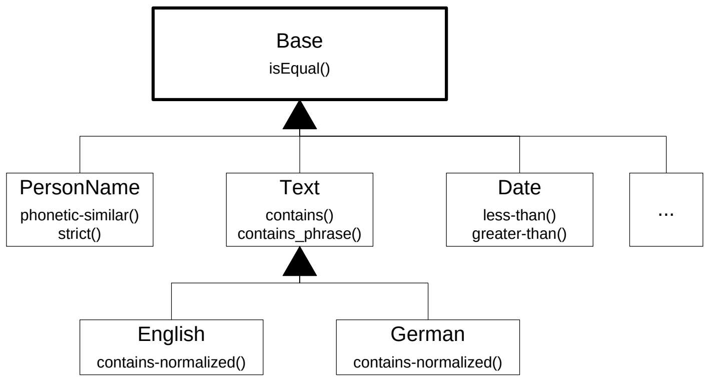


Figure 3.1: Inheritance hierarchy on data types (cf. [101])

its subtypes. The predicates implement type-based matching methods for names that sound similar or are abbreviated forms, geographical closeness of locations, interval restrictions of time and date specifications, similarities of chemical structures, and several grades of text processing.

3.2.2 Storage

Florescu and Kossmann

Florescu and Kossmann [73] describe eight different approaches how XML documents can be mapped onto relational database tables. They evaluated their approaches using approximately 80 megabytes of XML documents. Their results show that best performance is achieved by using separate attribute tables in combination with inlining the corresponding element values. Another finding is that more sophisticated approaches may hurt retrieval performance more than they help.

Schmidt, Kersten, Windhouwer, and Waas

Schmidt et al. [223] propose a data model based on complete binary fragmentation of the document tree. In order to store the XML documents they rely on the relational Monet database engine.

The Monet transform M_t decomposes an XML document tree into binary parent-child, node-attribute, and node-rank relations of the same type. M_t of a document d is given by the tuple

$$M_t(d) = (r, R, A, T)$$

where r is the root of the document, R is the set of binary relations between parent and child nodes, A is the set of binary relations between nodes and attributes, and T is the set of binary relations between nodes and their rank. Each of the R, A, and T relations is stored in separate database tables that are further decomposed according to the parent and child elements involved. The decomposition process of a document is linear in time and space.

For evaluation purpose three different document collections are used: ACM Anthology (46,6 MB), Shakespeare's Plays (7,9 MB), and Webster's Dictionary (56,1 MB). In their experiments they show that the size of the Monet XML format (44,2 MB, 8,2 MB, 95,6 MB) strongly depends on the original XML documents. Higher heterogeneity of the document structure increases the amount of database tables needed, peaking in 2587 tables for the Webster's Dictionary. Quantitative assessments show that the approach scales linear in the size of the XML document collection regarding both, storage and retrieval.

Grust

Grust [112] describes a relational index structure for XML documents. It allows a complete reconstruction of stored documents. Table 3.1 overviews the thirteen axes supported by XPath. The main focus of Grust settles on supporting four major axes exploited for retrieval, namely descendant, ancestor, following, and preceding relations.

Table 3.1: Semantics of XPath axes [112]

Axis	Result forest of a node v
`child`	direct element child nodes of v
`descendant`	recursive closure of `child`
`descendant-or-self`	like `descendant`, plus v
`parent`	direct parent node of v
`ancestor`	recursive closure of `parent`
`ancestor-or-self`	like `ancestor`, plus v
`following`	nodes following v in document order
`preceding`	nodes preceding v in document order
`following-sibling`	like `following`, same parent as v
`preceding-sibling`	like `preceding`, same parent as v
`attribute`	attribute nodes of v
`self`	v
`namespace`	namespace nodes of node v

Therefore, he suggests a five-dimensional descriptor

$$desc(v) = \langle pre(v), post(v), par(v), att(v), tag(v) \rangle$$

to encode structural elements, where *pre* (resp. *post*) is an ID assigned through a preorder (resp. postorder) traversal of the document tree, *par* is the parent node, *att* is a boolean value distinguishing XML attributes ($att = true$) and XML elements ($att = false$), and *tag* is the actual name of the XML attribute or XML element. One has to note that both pre- and postorder are unique within a document and can be used as primary keys. Both numbers are initialized with 1, so the root node of the document starts with $pre = 1$. The parents preorder ID *par* is stored for the sake of ancestor and descendant detection (root nodes are defined as $par = nil$). *par*, in combination with *pre* and *post*, allows efficient calculation of the following and preceding sibling. Both, *att* and *tag* are stored for name testing.

Grust points out that insertion of documents into the database is linear in time and size of the input source. By using an event-based parsing framework for XML documents (e.g., SAX) only very limited scratch space during storing is needed.

In order to store the content of the elements two, different solutions are feasible: (1) Inlining the content as a child node of its parent node (like an attribute); or (2) storing the content in a separate database table using (*pre*,*cdata*) tuples where *pre* is a foreign key pointing at the corresponding elements preorder ID. Experiments by Florescu and Kossmann [73] identified the second solution as superior, which is adopted in this work.

Hiemstra

Hiemstra [137] presents the fundamental ideas and starting points for building an XML retrieval system based on relational database systems. In his work he defines a basic language model for term weighing and introduces database tables that hold document term statistics, global term statistics, and XML document structures. For indexing purpose he applies methods described by Grust [112]. However, he substitutes the node identification based on preorder and postorder (assigned during two traversals) by starting and ending counters (assigned during a single traversal), keeping up fast ancestor-descendant detection.

3.3 The Most Simple XML Document Format

From now on, all methods and results described are achieved by the author as part of this work. One of the primary tasks of information retrieval is the handling of incoming information. Since this work relies on structured input documents, the usage of a generic

XML document format that satisfies the requirements of structured document retrieval is proposed.

An XML schema allows to define valid document structures independently of the XML document instances implementing it. The goal of this section is to identify the minimal requirements necessary to represent a structured document appropriately for information retrieval purposes. This first thing to be aware of is the different kinds of information coming along with structured documents:

Structure refers to the logical structure of a document usually covered by the terms 'chapter', 'section', 'subsection', etc. It defines a hierarchy where the order among the elements is of importance (i.e., the introduction always comes before the conclusion).

Content is associated with structural elements and defines what an element is about. Content can be regarded either as (1) flat content, typically referring to paragraphs, tables, and figures, or alternatively as (2) composite content that consists of recursively combined descendant contents of smaller granularity.

Metadata provides additional information which is – like content – assigned to structural elements. It describes the content more clearly without being part of it. Examples of metadata are the document's author(s), section titles, text languages, and figure dimensions.

In order to achieve accurate retrieval results, structure, content, and metadata have to be treated differently during both phases, indexing and retrieval.

3.3.1 Structure

The hierarchical structure of documents is usually covered by terms like chapters, sections, and subsections. Sticking to these definitions in a general domain leads to severe inconsistencies. For instance, scientific research articles do not provide chapters but consist of sections, subsections, and subsubsections only. Another problem occurs if these identifiers are abbreviated (sec, ssec) or cross-language documents are included (capitulo[2], Unterkapitel or Abschnitt[3], etc.). As a consequence more abstract definitions of content containers seem to be necessary.

Therefore, this work introduces a generic document structure (defined through XML schema) that consists of only three elements: `DOC` (document), `SEC` (section), and `FRA` (fragment). These three structural concepts manage to reflect any tree-like logical structure inherent in structured documents.

[2]Spanish for chapter
[3]German for section

Figure 3.2 illustrates the transformed document of the introductory example given in the previous chapter (see Figure 2.1). Titles are included for better comprehension. The DOC

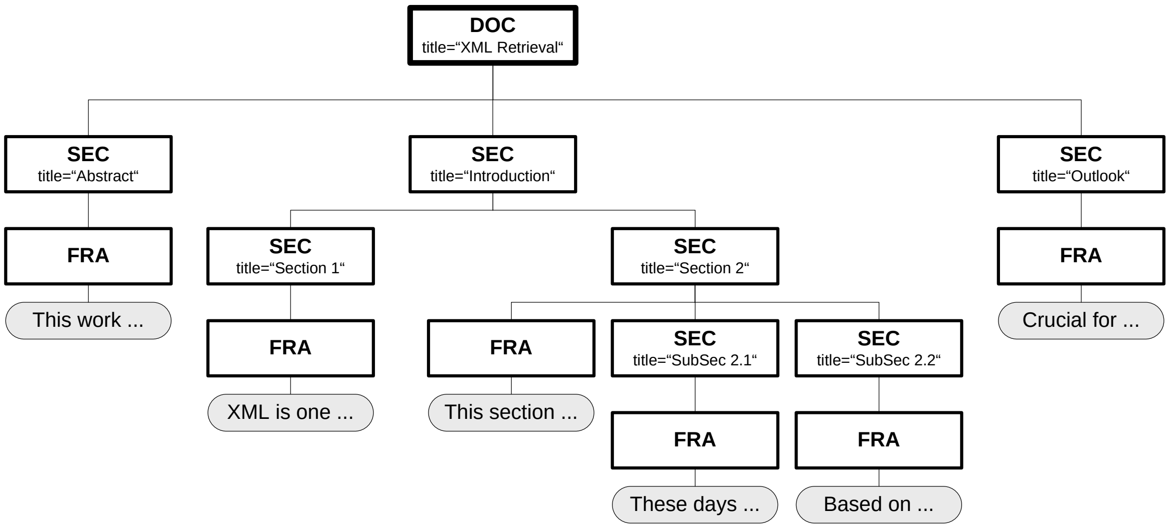


Figure 3.2: Example of a transformed XML document

element defines the root of the document. This enables all kinds of documents like books, articles, or news messages to be mapped onto this element. As content DOC elements may only comprise SEC and FRA elements. SEC is the main structural element of a document. Sections are defined recursively consisting of other SEC elements (e.g., subsections) and/or FRA elements (first child element of 'Section 2' in the Figure). This allows unlimited nesting of structural elements (e.g., chapter, section, subsection, subsubsection, segment). FRA elements are the only elements that carry flat content, defining the leaf nodes of the document schema. Fragments can be understood as basic building blocks acting as elementary containers for either plain text (which is the standard) or bytecode (e.g., inlined binary information like figures and multimedia objects).

From the information retrieval point of view fragments denote the atomic units of a structured document. Thus, they define the smallest units for indexing and retrieval. Their granularity strongly depends on the application area and domain requirements. In this work fragments are defined to fit the size of meaningful portions of texts that are retrievable. Some domains may require a deeper structure with much smaller fragments comprising sentences or even single terms, while others need fragments that fit even long text passages. In any case the granularity of fragments has a major impact on the number of elements in the transformed XML document and thus on the performance during indexing and retrieval.

Within documents links are supported as a means to express relations among elements. There exist two types of links, internal and external links. Internal links are links within the same document including citations (pointing to entries in the bibliography section), figure or table references within the text (pointing to the actual figure/table), or the table of contents (pointing to the beginning of sections/subsections). External links refer from one document to another like reference entries in the bibliography section (pointing to other documents), hyperlinks (pointing to URLs or email addresses), and file locations (pointing to specific files/directories).

Each of the three elements (`DOC`, `SEC`, `FRA`) is viewed as a tuple (*metadata*, *content*), where *metadata* refers to descriptive information of the node itself, while *content* refers to the content of the element (dotted rectangles in Figure 3.3). Generally, the first type of nodes requires database-supported matching during retrieval, while the second type is subject to partial matching (e.g., VSM). The strict distinction between metadata and content enables different

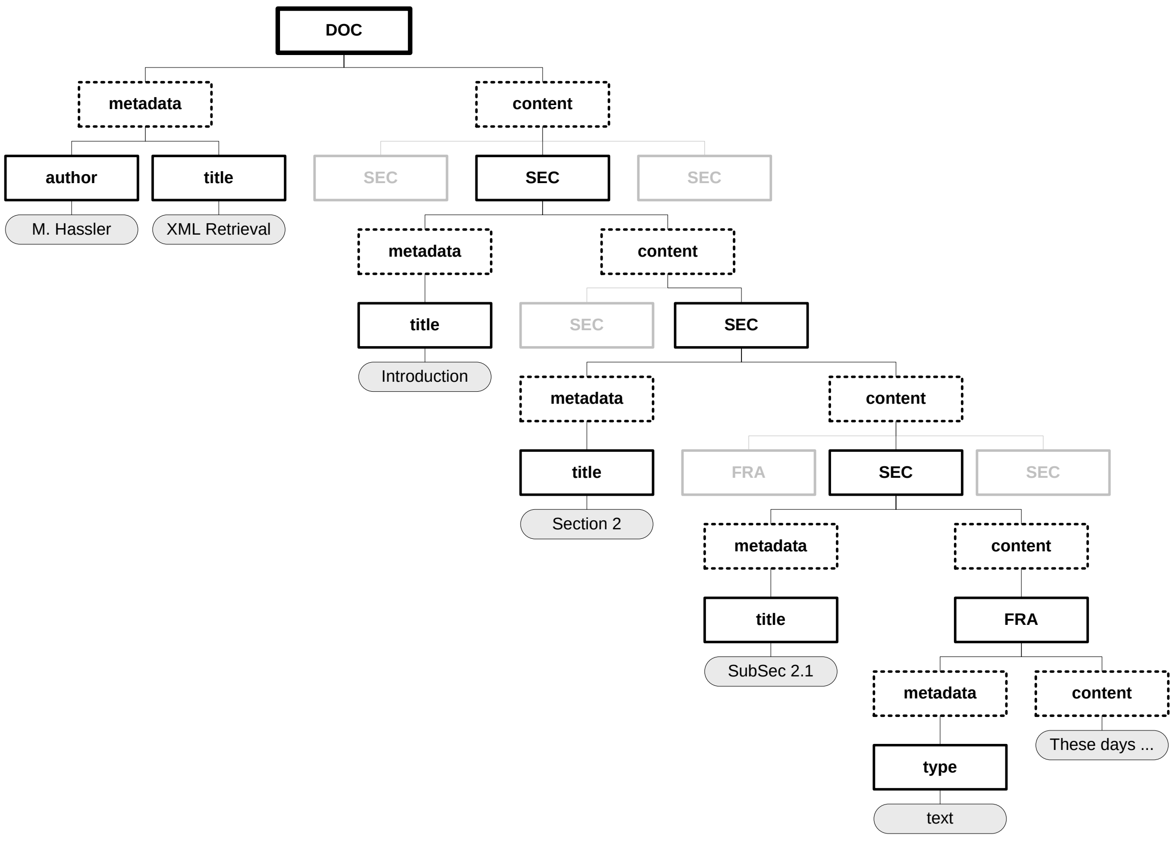


Figure 3.3: Expanded example of a transformed XML document

methods to match queries and document components. Thus, special retrieval focus can be put on each of these information sources.

3.3.2 Metadata

The metadata block of a node provides element-based complementary information. Examples of metadata information are a documents author or a sections title. A full list of the metadata fields for each structural element is given in Table 3.2.

Table 3.2: Supported metadata information

Element	Metadata
Document (DOC)	sourcepath, doc_type, title, author, subtitle, publisher, proceeding, year, keywords, isbn, price
Section (SEC)	sourcepath, sec_type, title
Fragment (FRA)	sourcepath, fra_type, title, language

Each of the elements (DOC, SEC, FRA) provides a sourcepath field that keeps a reference to the source of the information (e.g., URL). By this means the connection between a retrieved element of the transformed XML document and its original source document is established. In cases where a reference is not available (i.e., a section within an Adobe Acrobat PDF document cannot be addressed directly) the field may remain empty. Another metadata field available to all structural elements is the content type (doc_type, sec_type, fra_type) which is explained in the next subsection. The remaining metadata fields do not need further explanations. The reason for the title in the FRA metadata (usually paragraphs without titles) is that fragments may also refer to figures, tables, or multimedia objects.

Of course the proposed metadata fields have to be chosen according to the application area and domain. A natural extension of this metadata model is to allow multiple entries with the same name and sub-structured metadata. Although not exploited in this work, complex metadata information can be easily integrated (see Figure 3.4).

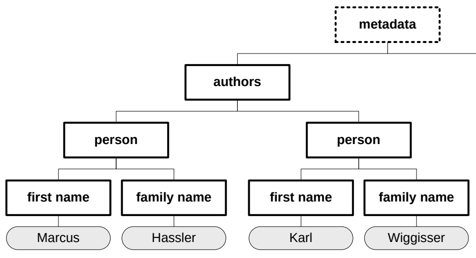


Figure 3.4: Structured metadata example

3.3.3 Content

The content block of DOC and SEC elements only supports SECs and FRAs as valid child elements. In contrast, the content of FRA elements is not further nested. Such a definition completely avoids mixed content nodes containing both, content and substructures. This in turn eases the implementation of retrieval systems regarding the storage, indexing, and retrieval functionalities.

As already mentioned, the proposed structure defines a fragment as the smallest structural unit which is in most cases a plain text block. However, restricting the content to paragraphs, enumerations, lists, tables, definitions, theorems, proofs, references, algorithms, equations, formulae, and code listings might seem too restrictive. It is important that a fragment could also refer to figures, pictures, videos, sounds, and other multimedia material. At this point the XML document format offers a possibility to include inlined binary data in form of bytecode in the document. Although such information has to be interpreted before being used it enables a single document to store any kind of contents.

Within a fragments textual content, semantic markup is useful to support additional

- layout information
- mathematical environments
- linkage
- footnotes
- etc.

Layout information is not exploited for retrieval purpose. It is reduced to a single `<emph>`-tag to express emphasis on a term or text passage. Mathematical expressions and formulae within a text are marked by `<math>`-tags. Both kinds of links, internal and external, are supported by a `<link>`-tag that encloses the text to be linked. This tag also specifies the type (`internal` or `external`) and the `destination` (anchor within the document or address to the external resource). Footnotes are resolved as internal links to special footnote sections inserted at the end of a document. A footnote is realized as a section because it may contain more than a single paragraph. This concept can easily be extended to fit sub-structures of more complex contents (e.g., table-like HTML markup), enabling more appropriate analysis and matching mechanisms. Although these sub-structure elements are queryable the smallest retrievable unit remains the whole fragment.

However, the content block of elements is not mandatory. This allows the inclusion of contents by exploiting their metadata information (i.e., if content is not read- or analyzable) only. Further, it can be used to outsource external contents by referring to it (e.g., a picture or dynamically generated content somewhere on the net), which in turn reduces the size of the XML documents.

3.3.4 Typing of Structural Elements

Proper retrieval results always involve a certain level of content interpretation and structural knowledge. It plays a central role in satisfying the users needs during retrieval. Therefore, semantic typing of content and metadata is applied to enhance retrieval performance. The proposed types in this section are developed to fit the context of this work. They were never regarded as a complete set and are intended to be extended according to the application domain.

Typing of Metadata

Being XML elements, metadata fields contain ordinary strings. Although XML schemata allow for rudimentary type specification of nodes (e.g., `xls:type='integer'`), generally the very same content is valid in many different fields. For instance, both the title and the ID of a document may be defined as simple strings. Clearly these fields have to be treated differently when being compared to a query. Thus, semantic information about the content of metadata fields is needed to match them appropriately. In order to achieve this, metadata types are exploited that are assigned to single metadata fields.

To allow a semantic interpretation of the content of an XML element, a type hierarchy is proposed by Gövert [101].

A similar model is applied for typing of metadata fields. The type hierarchy used is depicted in Figure 3.5. There, types are derived from a common base element. The first level in the hierarchy (gray-shaded rectangles) corresponds to database supported data types. Thus, they can be used to assign types to the columns of database tables. More specialized types in subsequent levels in the hierarchy have one of the basic database types as supertype (e.g., `PersonName` is a String). Accordingly, the assigned types also imply certain restrictions on the content and formatting (e.g., `DateTime` format, `URL` format) of metadata fields.

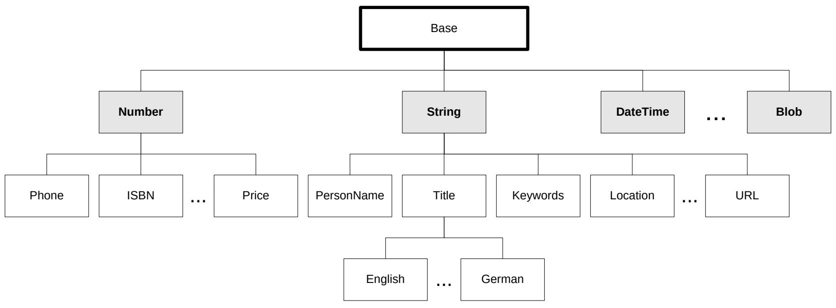


Figure 3.5: Hierarchical metadata types

Associated with the metadata types are custom-built implementations of predicates for comparison. This allows special searching of single metadata fields on different structural levels (e.g., section titles, phone numbers, author names).

For instance, the type `PersonName` identifies the content of a metadata field such as the name of a person. Generally, names include a family name and one or more forenames. In user queries, multiple forenames or unknown forenames are often abbreviated by their initials. Although family names and forenames cannot be distinguished automatically, initial letters of names can be matched during retrieval. Consider documents written by an author named `Albert Einstein` and a user query containing `A. Einstein`. In this case typing allows a similarity calculation ($0 \leq sim \leq 1$) between these two strings, resulting in a higher similarity value (e.g., $0,9$) than a comparison with `H. Einstein` (e.g., $0,6$). This is also true for abbreviated family names like `E. Albert`. However, more detailed metadata information (i.e., a `PersonName` that consists of a `PersonFamilyName` and `PersonForeNames`) is able to handle even complex comparisons in a correct manner (see Figure 3.4). `Keywords` separated by commas can be split and compared in a boolean manner. `Titles` might be analyzed and compared in more detail using stopword removal or (shallow) syntactic parsing. Also, high-level similarities of `Phone` numbers based on their area code or geographic closeness of `Location` types can be realized by these predicates.

Another important factor to find commonalities of queries and metadata information is the normalization of content. By transforming special types (e.g., times and dates, phone numbers, prices) to a common format, similarity computation becomes more exact. It is also simplified and speeded up.

Typing of Content

For indexing and retrieval purposes the content of `DOC`, `SEC`, and `FRA` is classified according to several content types. These types can be addressed within queries and enable to focus on particular subsets of elements. For instance, only `books`, sections of type `introduction`, or `references` might be searched. Figure 3.6 presents the content types supported for each content container (gray-shaded rectangles). Again, the proposed content types are not regarded as a final list and have to be chosen according to both, the application and the domain.

3.4 Storage

A central issue of retrieval systems is their performance. This includes the way documents are stored and accessed. Especially in the context of structured documents, efficiency during retrieval of components of any granularity must be provided. Generally, three different alternatives for storing XML documents have been proposed in the literature [153, 104]:

- Special purpose database systems (e.g., Rufus [228], Lore [15, 177], Strudel [70]) work best as they are scalable and meet the storage requirements to handle huge data loads. Unfortunately, these systems are tailored for special domains and applications. Thus, they are not very flexible.

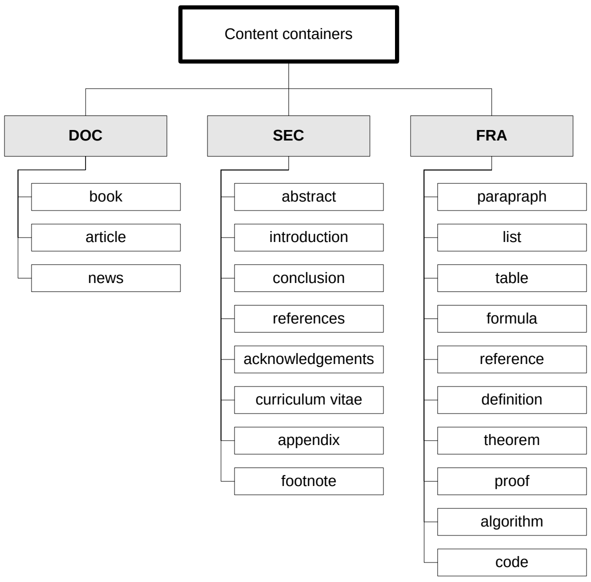


Figure 3.6: Content types

- Object oriented database systems (e.g., O_2 [23]) and native XML stores (e.g., Timber [142], eXist [178]) are optimally suited to store complex nested data structures. However, when querying large amounts of data, these systems are not able to compete with retrieval performance of special purpose database systems or relational database management systems.

- Relational database management systems have been well proven in the information retrieval domain. They provide maturity, stability, portability, and scalability [226]. Mapping XML documents appropriately to fit the relational paradigm seems to be a promising solution that meets the storage requirements for structured documents [111].

In this thesis a relational database is used to store the XML documents. The goal is to accelerate the access to various structural neighbors of a node in the document tree (descendants, ancestors, and siblings). In order to process the transformed XML documents efficiently not all nodes are stored in the database. Pure artificial elements used for content distinction, the `metadata` and the `content` block, are neglected. Further, the set of metadata elements is treated differently as described in the sequel.

This work departs from the idea of opening (*pre*) and closing (*post*) node identifiers introduced in [112, 137]. The access efficiency comes from the fact that *pre* and *post* descriptors are unique for a given document and, therefore, can be used conjointly with the ID of the document as primary key in the mapped relational schema. Both, *pre* and *post* are assigned straight-forward to the nodes of a document in a single preorder traversal (root first, then children from left to right) by keeping track of opening and closing tags. Figure 3.7 depicts an example with *pre* (number to the left) and *post* (number to the right) IDs assigned to each node.

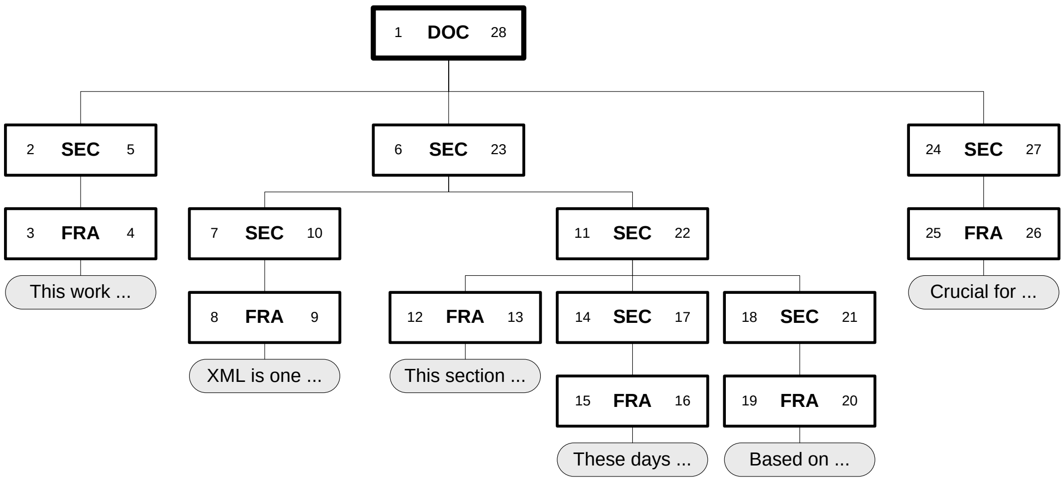


Figure 3.7: Transformed XML document with *pre* and *post* identifiers

During retrieval, *pre* and *post* identifiers imply a number of useful features:

- The root DOC always starts with $pre = 1$.
- Child nodes of the same parent node are continuously numbered, where *pre* of a subsequent element equals $post + 1$ of the preceding element (e.g., the three uppermost SEC nodes: 2-5,6-23,24-27).
- The number of all descendant nodes is given by $\frac{post-pre-1}{2}$.
- FRA elements define the leaf nodes of the tree and are identified through the equation $post - pre = 1$.
- Containment of two nodes (ancestor-descendant relation) is defined by pre_ancestor $<$ pre_descendant and $post$_ancestor $>$ $post$_descendant. Thus, no recursive computation to detect ancestor-descendant relations is needed.

Table 3.3 shows the structural entries for the document given in Figure 3.7. A structural entry is described by the tuple (*doc*, *pre*, *post*, *parent*, *tag*, *path*). *doc* refers to the document, *pre* and *post* are the numeric node identifiers, *parent* is the *pre* identifier of the parent node in the same document, *tag* is the XML tag naming the component, and *path* denotes the common path (see Definition 3.1) to the root of the document. The root element ($pre = 1$) has per definition no parent. *tag* is included for fast name lookup and access. For the sake of retrieval performance each entry includes its common *path*. For queries specifying the path of a document component, a great deal of recursive path generations using the *parent* and *tag* attributes is avoided.

Definition 3.1 [Common path]: The common path of an XML component is defined as the XPath to the root of the document (e.g., `/DOC[1]/SEC[3]/FRA[1]`) without positional information (e.g., `/DOC/SEC/FRA`).

Table 3.3: Relational entries for the structure

doc	pre	post	parent	tag	path
d_1	1	28	-	DOC	/DOC
d_1	2	5	1	SEC	/DOC/SEC
d_1	3	4	2	FRA	/DOC/SEC/FRA
d_1	6	23	1	SEC	/DOC/SEC
d_1	7	10	6	SEC	/DOC/SEC/SEC
d_1	8	9	7	FRA	/DOC/SEC/SEC/FRA
d_1	11	22	6	SEC	/DOC/SEC/SEC
d_1	12	13	11	FRA	/DOC/SEC/SEC/FRA
d_1	14	17	11	SEC	/DOC/SEC/SEC/SEC
d_1	15	16	14	FRA	/DOC/SEC/SEC/SEC/FRA
d_1	18	21	11	SEC	/DOC/SEC/SEC/SEC
d_1	19	20	18	FRA	/DOC/SEC/SEC/SEC/FRA
d_1	24	27	1	SEC	/DOC/SEC
d_1	25	26	24	FRA	/DOC/SEC/FRA

There are two possible ways to store the content of document elements [112]: (1) inline the content as a child node of its parent (like an attribute); or (2) store the content in a separate table with a foreign key pointing to the correct element. The second method has been proven to be superior [73] and is adopted in this work.

The content of nodes (in particular of fragments) is kept in a separate database table (see Table 3.4). Contents of inner nodes (`SEC` and `DOC` elements) do not have to be stored explicitly because they can be recovered dynamically from the descendants' contents. Finding a balance between retrieval performance and storage space, dynamically recovered inner node contents can be additionally (or temporarily) preserved in the content table. This allows to retrieve the content of an inner node by a single database query instead of computing it recursively from multiple descendant nodes.

Table 3.4: Relational entries for the content

doc	pre	data
d_1	3	This work ...
d_1	8	XML is one ...
d_1	12	This section ...
d_1	15	These days ...
d_1	19	Based on ...
d_1	25	Crucial for ...

The concept of separating content and structure opens up the possibility to maintain multiple contents for each structural element. Hence content can be represented in various forms, this is a major advantage for information retrieval. At least two (independent) content tables are needed: one for saving the original plain content and another one for saving its representation.

During indexing of a new XML document, plain contents restricted to leaf nodes are stored in the database first. Afterwards, content representations of the stored leafs are computed (and recalculated on demand) without accessing the XML file again. Additionally, contents and content representations of inner nodes can be calculated and stored in the database. Therefore, not every content in the database must have a corresponding content representation (i.e., inner nodes).

Since this work focuses on the representation of natural language texts, contents such as figures or formulae are unsuited for textual representation. However, additional representations of these contents (e.g., bitmaps) can be integrated easily by adding new representation tables.

In order to improve retrieval performance metadata handling is completely shifted to the database. This is achieved by grouping all metadata fields according to their element. Instead of having multiple entries in the structure and content tables, a single tuple (*doc*, *pre*, $meta_1$, ..., $meta_n$) lumps together the set of metadata fields in a single database entry. Metadata of elements (`DOC`, `SEC`, `FRA`) is stored in separate but similar tables as shown in Table 3.5 for sections.

Table 3.5: Relational entries for metadata (section level)

doc	pre	title	sec_type	...
d_1	2	Abstract	abstract	
d_1	6	Introduction	introduction	
d_1	7	Section 1		
d_1	11	Section 2		
d_1	14	SubSec 2.1		
d_1	18	SubSec 2.2		
d_1	24	Outlook	conclusion	

Although the same elements on different hierarchical levels may implement different sets of metadata (i.e., sections in proceedings may state an author but subsections do not), the approach taken in this work assumes that all elements of the same type (`DOC`, `SEC`, `FRA`) have the same metadata fields. This may lead to numerous *NULL* values (unavailable metadata for some elements) in the database. However, the whole set of metadata can be accessed by a single select operation. This both simplifies and speeds up querying of metadata considerably.

3.5 Document Representation

The representation of structured documents relies on XML documents that implement the generic schema introduced in Section 3.3. As a starting point, the three structural components document (`DOC`), section (`SEC`), and fragment (`FRA`) serve as index objects.

Since fragments are the leafs in the document tree, their content has to be indexed first. Therefore, methods from the field of natural language processing transform the textual content (e.g., paragraphs, lists) into term frequency vectors as described in the next two chapters. In order to index inner nodes (sections and documents) the representations of descendants are recursively merged using a propagation of term statistics (see Section 2.5). This operation is equivalent to process the concatenated contents of the descendant nodes.

The actual weighing of the term frequency vectors is carried out during retrieval using the traditional vector space model (Equations 2.3–2.4). Based on the weighed feature vectors the similarity between document components and user queries is computed. However, weighing includes the inverse document frequency (Equation 2.2) as a means to exploit corpus-based statistics. In the domain of structured document retrieval, the notion of 'document' has to be reconsidered: What elements are regarded as a valid 'document'? Is it the `DOC` component which comes closest to the original definition? Perhaps the fragments containing the content should be counted? Or, are all components (`DOC`, `SEC`, `FRA`) treated as separate 'documents'? Obviously, this choice has a major impact on weighing and ranking, and thus on the retrieval result. In this work two approaches are explored.

- The first approach treats all components (`DOC`, `SEC`, `FRA`) as equal units or 'documents'. Their contents are represented within the same term space, which is defined as the union of all terms stated in all components. Thus, it implements a single static term space.

- The second approach is more elaborated, utilizing different term spaces for components on different hierarchical levels. In order to group components according to their term space, the common path (see Definition 3.1) of an element is consulted. This approach is referred to as applying dynamic term spaces.

Both approaches are described in the next two subsections. Since the term 'document' is no longer valid for structured documents, the term inverse document frequency idf is substituted by inverse element frequency ief [106]. A final comparison of the two approaches outlines their behaviors.

3.5.1 Static Term Space

In a static term space all component representations are treated as equal. Neither a distinction between document, section, and fragment components is made, nor is the level in the document hierarchy considered. All components are processed as a collection of 'documents' assumed to be equally important to a query.

Taking this point of view, the vector space model needs only slight changes (Equations 3.1–3.4), where N (resp. n_i) is the total number of document components (resp. components stating term i).

$$tf_{i,j} = \frac{freq_{i,j}}{\max freq_{l,j}} \quad (3.1)$$

$$ief_i = \log \frac{N}{n_i} \quad (3.2)$$

$$w_{i,j} = tf_{i,j} \cdot ief_i \quad (3.3)$$

$$sim(d_j, q) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{|\overrightarrow{d_j}| \cdot |\overrightarrow{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{j=1}^{t} w_{i,q}^2}} \quad (3.4)$$

Because the cosine similarity implies a certain normalization in the space of document components [21, pp. 28], the arbitrary length of the contents is accounted for. Term weights computed on the document level are unequal to term weights achieved by applying the traditional formulae on the flat content of the same document. The reason for that is that the inverse element frequency is computed using a much higher number of document components than the inverse document frequency using the number of documents.

In order to support the calculation of ief_i the database stores (*doc*,*term*,*n*) tuples of the documents. As the number of entries in this table becomes quite large, a condensed table holds (*term*,*n*) tuples to improve performance during retrieval. If documents or document components are added, altered, or removed both data structures are updated accordingly.

3.5.2 Dynamic Term Spaces

Another approach is to exploit structural information to define multiple term spaces that better represent components on the same level in the hierarchy. Two assumption are that these components (1) are of similar granularity and length, and (2) use only a portion of the complete term space. To achieve this, the common path (see Definition 3.1) is used to cluster the components accordingly.

Based on the common path, the *context of a node* is defined as the set of all components having the same path (all chapters, all sections, all fragments within subsections, etc.). To fit this requirement, the vector space model adaptation is given by the Equations 3.5–3.8. The inverse element frequency $ief_{i,c}$ is calculated dynamically based on a node's context c, where N_c (resp. $n_{i,c}$) is the number of document components in context c (resp. components in the context c stating term i).

$$tf_{i,j} = \frac{freq_{i,j}}{\max freq_{l,j}} \tag{3.5}$$

$$ief_{i,c} = \log \frac{N_c}{n_{i,c}} \tag{3.6}$$

$$w_{i,j} = tf_{i,j} \cdot ief_{i,c} \tag{3.7}$$

$$sim(d_j, q) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{q}}{|\overrightarrow{d_j}| \cdot |\overrightarrow{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{j=1}^{t} w_{i,q}^2}} \tag{3.8}$$

For the sake of term weighing, different $ief_{i,c}$ values of the same term i in different contexts c are used. As a consequence, this strategy weighs the same term with the same term frequency differently depending on context c (e.g., chapter versus subsection). Clearly this approach puts more attention on the actual context during retrieval. If the unit of retrieval is defined explicitly, elements in this context are focused. Representations of elements in other contexts do not influence the result. This allows to perform focused retrieval on any level in the document tree.

Generally, term spaces of different contexts imply a containment constraint: The term space of components of larger granularity subsumes the term space of components of smaller granularity. This becomes clear since the `/DOC` term space obviously is the superset of the `/DOC/SEC` term space, which in turn is the superset of the `/DOC/SEC/SEC` term space. However, fragments on different levels in the hierarchy influence the term space of their ancestors considerably. A nice feature of this approach is that in contrast to a single static term space, the weighed feature vectors of `DOC` components exactly match the representations using the vector space model on flat documents.

Dynamic $ief_{i,c}$ calculation is accomplished by the database keeping track of (*doc*,*path*,*term*,n_c) tuples in a `localIEF` table. Compared to a static term space, the amount of database entries in this table is extraordinary high. To circumvent performance losses, a compressed data structure of (*path*,*term*,n_c) tuples is pre-computed and stored in a `globalIEF` table. Both tables `localIEF` and `globalIEF` are used to compute dynamic as well as static inverse element frequencies. More details are given in Chapter 8 describing the system implemented.

3.5.3 An Example: Static versus Dynamic Term Spaces

The best way to illustrate the differences of a static term space and multiple dynamic term spaces is by means of an example. Consider the query and document given in Figure 3.8. The document tree contains three fragments and their associated term weights. For demonstration purpose term weights in fragments are assumed to be equal $(0,5)$. Due to the propagation mechanism, term weights at higher levels are decreased (by subtracting $0,1$ for each level). Furthermore, the query contains a term x that never occurs in any of the document components.

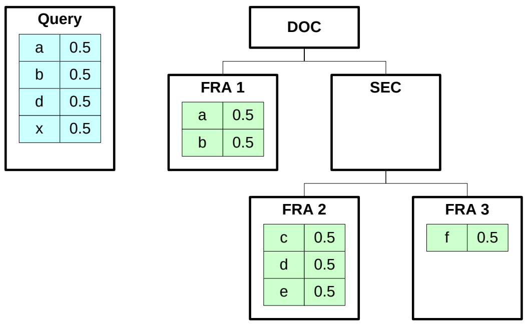


Figure 3.8: Initial document and query

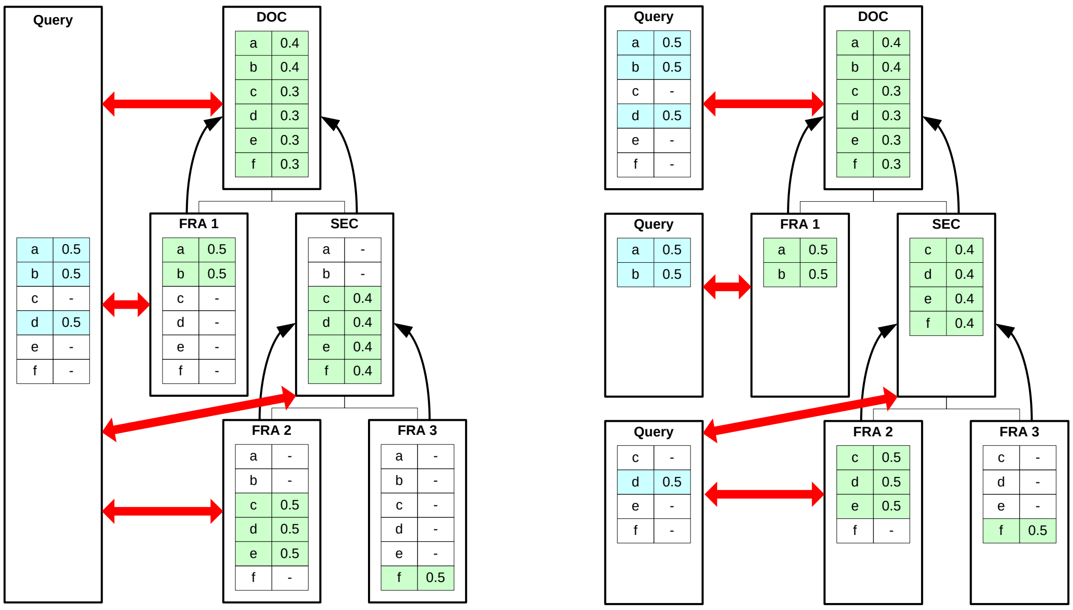
In order to compute the similarity $sim(d_j, q)$ of a component d_j and the query q, both representations have to be mapped onto a common term space (static or dynamic). Terms i not stated in a component j are weighed as zero ($w_{i,j} = 0$). Terms in the query that are not included in any of the components are neglected because they do not lie in the term space. The resulting feature vectors of d_j and q are of equal length and can be matched according to the $sim(d_j, q)$ formula (Equations 3.4 and 3.8).

Besides its impact on the term weights due to different inverse element frequencies (which is masked in the example) Figure 3.9 shows the effect of using either a static or dynamic term spaces.

In a static term space all components and the query are represented using a vector size of six terms. On the one hand this supports faster mapping (fixed length) and weighing (only a single ief_i). On the other hand feature vectors become sparser because the largest term space containing all terms of all components is used, resulting in a drop of performance (longer feature vectors) and similarity of smaller elements (see Figure 3.9a).

In contrast, using dynamic term spaces means that components and the query are represented according to the context currently being matched. In the figure, four term spaces are maintained: `/DOC`, `/DOC/FRA`, `/DOC/SEC`, and `/DOC/SEC/FRA` (the latter two are the same).

The resulting similarities of both approaches are given in Table 3.6. The first thing to note is that the ranking of the components according to their similarity stays the same. But



(a) Static Term Space **(b)** Dynamic Term Spaces

Figure 3.9: Static versus dynamic term spaces

Table 3.6: Similarities using static and dynamic term spaces

Context	static term space	dynamic term spaces
DOC	0,770	0,770
SEC	0,289	0,500
FRA 1	0,816	1,000
FRA 2	0,333	0,577
FRA 3	0,000	0,000

– as expected – the similarities of smaller components (especially fragments) gain much higher values using the dynamic term spaces. This improves focused retrieval by ranking more specific components of smaller granularity higher without effecting (decreasing) the similarity of larger components. A performance evaluation of both approaches using real world documents is carried out in Chapter 9.

3.6 Summary

This chapter introduced a generic document format for information retrieval and its XML implementation applied throughout this work. The generic format consists of only three components (document, section, and fragment), which serve as both index and retrieval objects. Documents from heterogenous sources are first mapped onto this schema for further processing. In order to store the documents, a relational database approach has been chosen.

Finally, representation issues are discussed. The focus is put on weighing and ranking, providing two strategies based on the vector space model: A single static term space approach and a more elaborated approach reverting on multiple dynamic term spaces are presented. An evaluation comparing these two approaches is conducted in Chapter 9.

Chapter 4

Any noun can be verbed.

Alan J. Perlis

Natural Language Text Representation

Whereas the previous chapter is concerned with the representation of the structure, this chapter introduces the basic tasks necessary to process the content of the leaf nodes of a document containing plain natural language texts. The goal is to represent textual content in a form that can be computed and compared efficiently. The result of this process is a vector of terms – actually of stems – associated with their frequencies of occurrence. The involved natural language processing tasks are tokenization, tagging, stemming, and stopword filtering. These tasks are described in detail.[1].

4.1 Introduction

Proper retrieval results depend on appropriate content representations for the documents searched. Therefore, methods from the field of natural language processing are applied to compute representations for plain texts. Several works [263, 250] are solely dedicated to investigate the effects of natural language processing tasks on the information retrieval performance. It is commonly agreed that this task is a very complex one, including various processing steps like tokenization, tagging, shallow parsing, stemming, compound splitting, etc. [32] The result of this processing chain is an appropriate representation of the text. In order to support retrieval tasks appropriately, the representation has to facilitate

- fast computation and comparison of representations,
- minimal storage requirements, and
- support of high recall and precision in searches.

A major design goal of natural language processing frameworks is to define (1) components that manage these tasks, their (2) correct processing sequence, where the output of one component serves as input for a subsequent one, and (3) appropriate data interchange objects. Components are tightly coupled to these data objects and cannot be used in isolation. Hence, an easy and approved possibility to transfer data is to utilize structured text files like

[1]Parts of this chapter have already been published in [122]

XML documents. The usage of such a generic data interchange format results in platform, application, and system independent components. A component can simply be plugged into an existing system at the moment it is required. Furthermore, XML output is easily adaptable and can be processed efficiently.

In this work the basic components accomplish the tasks of tokenization, tagging, term extraction, stemming, stopword filtering, and term frequency computation. The processing sequence is depicted in Figure 4.1.

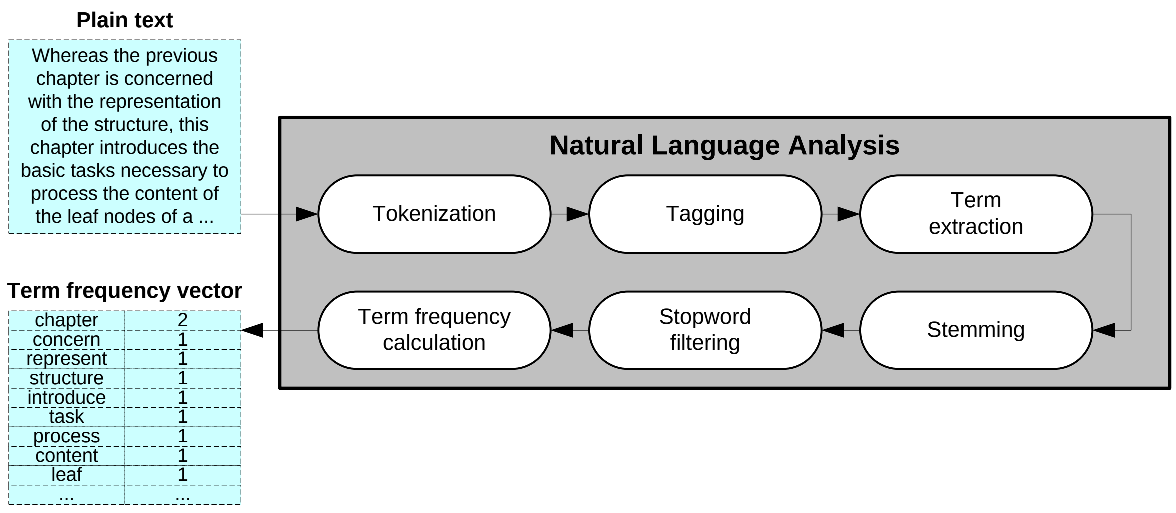

Plain text
Whereas the previous chapter is concerned with the representation of the structure, this chapter introduces the basic tasks necessary to process the content of the leaf nodes of a ...
Natural Language Analysis
Tokenization
Tagging
Term extraction
Stemming
Stopword filtering
Term frequency calculation
Term frequency vector
chapter 2
concern 1
represent 1
structure 1
introduce 1
task 1
process 1
content 1
leaf 1
... ...

Figure 4.1: Natural language processing sequence

Tokenization is the first task of natural language text processing [255, 75]. It segments the text into meaningful processing units called tokens. The output of the tokenizer serves as input for a tagger. During tagging syntactic word categories (e.g., verb, noun, adjective) are assigned to tokens as tags. Based on these tags terms that carry the meaning of the text are extracted and converted into lower case. Therefore, nouns and verbs are considered more important than determiners and prepositions. Afterwards a stemmer reduces each extracted term to its stem. By stemming, the number of distinct terms (size of the vocabulary) is reduced drastically. Supplementary it enables a matching of different forms of the same word. For instance, the terms `book`, `books`, `booking`, and `booked` are all reduced to the single stem `book`. In order to get rid of additional unwanted material (i.e., the term `computer` within computer science articles may not be very helpful) a list of stemmed stopwords is used for filtering. Finally, the frequencies of each stem are summed up to reflect its importance.

This chapter starts with presenting related work and natural language related delimination problems, highlighting the difficulties of boundary detection of both words and sentences. In the sequel the foundations of the natural language processing steps (tokenization, tagging, stemming, stopword filtering in Figure 4.1) are provided. These steps are necessary to compute content representations for information retrieval and document mining tasks.

4.2 Related Work

4.2.1 Character Sets

A well known language-related problem is the occurrence of special characters (e.g., Slavic diacritics, umlauts and sharp s in German, etc.). In order to work properly a well-designed and language-independent core system for natural language text processing has to support these character sets. Therefore, the Unicode Consortium was founded to develop, extend, and promote the usage of world-wide Unicode related standards to encode characters [243]. Further, the Unicode Consortium actively develops standards in the area of internationalization, including the definition of the behavior and relationships between Unicode characters. The aim of Unicode is to offer a language-independent solution for text data interchange. Currently it is regarded as the default standard that specifies the representation of texts in modern software systems.

4.2.2 Tokenization

One of the first authors dealing systematically with the problems of tokenization was Fox [75] in 1992. Besides standard tokenization, his work addresses special cases like words containing numbers, hyphens, special usage of punctuation marks including slashes and underlines, and the case of letters. For this sake, hand-written finite state automata are applied to resolve tokenization ambiguities. Further, the work exemplifies how a stopword list is constructed automatically to support the tokenization machinery.

Webster and Kit [255] provide an initial definition of the token concept. They concentrate on Chinese texts, where delimiters for words and sentences are not present. Two types of ambiguities are distinguished, conjunctive ambiguity (two or more neighboring words are adjacent, compound) and disjunctive ambiguity (a word in between is adjacent with a previous word and with a following word, double usage of the word in the middle). Two ways for disambiguation are proposed: (1) application of knowledge-based rules and (2) application of statistical-based methods borrowed from corpus-linguistics.

Grefenstette and Tapanainen [107] describe a regular expression based treatment of common tokenization difficulties, including special number formats (e.g., dates, fractions) and abbreviations. In order to identify sentence borders they run experiments using rules together with the corpus itself as a filter. First, all possible candidates for sentence ends (i.e., all strings ended by periods) are tested against a short list of exceptions and a set of identification rules.

Second, if no match is found during the first step, the string without the ending period is compared to all sentence internal words. In other words it is checked whether the term occurs as non-abbreviation somewhere within a sentence. In the case that both steps do not come up with a result, the token is marked as the last word of the sentence.

Giguet [98] addresses the problem of multilingual text tokenization for natural language diagnosis. The aim is to assign language names to sentences. Because there is no knowledge about the language in advance, a general and language-independent tokenization method is applied. Therefore, he proposes a rule-based tokenization approach, addressing common tokenization problems like elision and the difficulty of merging ordered sets of tokenization rules.

Strongly related to tokenization is the problem of sentence boundary disambiguation. Palmer and Hearst [197] point out several difficulties that arise if sentence end markers are not identified correctly. They propose a machine-learning approach based on neural networks to tackle this problem. Their implementation system Satz utilizes part-of-speech tags of preceding and following tokens around potential sentence end markers. These tag sequences are used to estimate the probability of a true end of the sentence. As the system relies on machine-learning, it operates in two phases: In a training phase, classification patterns provided by an expert are learned. The trained knowledge is then applied in a testing phase to recognize sentence boundaries in unknown texts. Satz executes fast in both, the training and testing phase, adapts to other corpora and languages, and operates robust regarding the input.

Guo presents in [116, 117] a one-tokenization-per-source approach that focuses on non-segmenting languages like Chinese. In contrast to segmenting languages as English or German, the text comes as a sequence of characters without any delimiters (e.g., blanks, punctuation marks) between words and sentences. Thus, the difficulty is the identification of token borders, which is not an easy task. His proposed approach is based on the assumption of a complete dictionary. Nevertheless, the sentence `todayissunday` still can be interpreted either as `to day is sun day` or `today is sunday`. To recover from this problem, Guo suggests to use the document itself as a filter to disambiguate these cases. The system is evaluated using the Chinese PH newspaper corpus, achieving high accuracy values of up to 99%.

Yamashita and Matsumoto [260] propose a language independent morphological analysis consisting of three steps: tokenization, dictionary lookup, and disambiguation. Their approach is applicable for both segmenting and non-segmenting languages. Segmenting languages are transformed to non-segmenting ones by simply removing all delimiters. They introduce the notion of a morpho-fragment, which is an intermediate representation between characters (graphic words separated by blanks in segmenting languages) and lexemes (logical units). Trie data structures are used to implement efficient dictionary lookup including morphological information. The final disambiguation step is realized via a hidden markov model. The approach is evaluated using the tagged Penn Treebank corpus (97% and 95% precision and recall), a Japanese tagged corpus (97% precision and recall), and the tagged corpus released by the Chinese Knowledge Information Processing Group (95% and 91% precision and recall).

Using a minimum of pre-built resources, Mikheev [180] tackles three different tokenization problems: sentence boundary disambiguation, disambiguation of capitalized words, and identification of abbreviations. With the help of automatically retrieved word frequency lists and a small set of heuristics, the corpus itself is used to dynamically infer disambiguation clues. In order to detect sentence borders unambiguous abbreviations (e.g., abbreviations followed by a lower case word) are looked up within the document. Based on these occurrences and local contexts ambiguous abbreviations are resolved. The disambiguation of capitalized words, which is an important task of text normalization, is closely related to the first problem. This step partly covers the task of named entity recognition, where three subtasks are distinguished: identification of unique identifiers (organizations, persons, locations, proper names, product names, acronyms), identification of temporal expressions (complete or partial times and dates), and identification of numerical expressions and quantities (monetary values, percentages). As before, the clues found in unambiguous occurrences are used to resolve ambiguous contexts. Sentence ends marked by either exclamation marks, question marks, or periods are identified as central processing units relying on the previous results, abbreviations, and capitalized words. The system achieves high performance exceeding 99% accuracy on the three tasks using the Brown Corpus.

4.2.3 Tagging Systems

The Brill Tagger

One of the earliest rule-based taggers that achieves similar results compared to stochastic taggers was presented by Brill [34, 35]. The approach, called transformation-based error-driven learning, operates in two phases: During a learning phase, tagging rules are automatically inferred from a corpus by trying to increase the overall accuracy. In a testing phase, each word is initially assigned its most likely tag without considering any context. Therefore, the tagger is provided with two clues: capitalized words tend to be nouns and most common word endings are known (e.g., words with the suffix `-ous` tend to be adjectives). Afterwards, tag assignments are iteratively changed according to the learned rules. An evaluation using the Brown corpus showed that the error rate drops to 3,5%.

Some extensions of the Brill tagger are described in [36]. They comprise tagging using lexical relationships, a new method to identify unknown words, and multiple tags for each word. Initial tagging knowledge is reduced to a minimal amount. The unsupervised learning algorithm is explained in detail in [38]. A good summary of Brill's work is found in [37].

Cuttings Hidden Markov Model Tagger

Cutting et al. [59] present a part-of-speech tagger that is robust, efficient, accurate, tunable, and reusable. The tagger is based on a hidden markov model to resolve a small set of ambiguity classes. A lexicon module provides information about common word tags, word suffix identification, and default ambiguity classes. Their experiments show that the tagger achieves an accuracy of over 96%.

TreeTagger

Schmid [221] presents a probabilistic part-of-speech tagger called TreeTagger. The tagger operates on second order markov models using binary decision trees to estimate *n*-gram probabilities instead of a maximum likelihood estimation (see Figure 4.2). Tree pruning is used to recursively remove leaf nodes below a weighed information gain. Initially, tags are

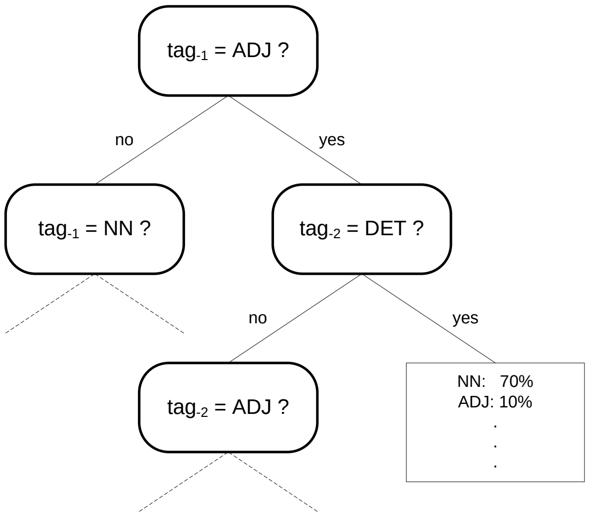


Figure 4.2: A sample decision tree [221]

assigned through a lexicon that includes full forms, suffixes, and default category entries. Afterwards, unigrams, bigrams, trigrams, and quatrograms are applied to assign correct tags. Evaluation results using the Penn Treebank corpus show good results of up to 96,34% accuracy.

Further developments of the TreeTagger are described in [222]. In order to fit the domain of German texts some additional processing steps are included: Smoothing to circumvent the sparse data problem is done with equivalence classes, a prefix lexicon is used in combination with suffixes, and capitalized sentence initial words are considered. In an evaluation using the large newspaper corpus Stuttgarter Zeitung the tagger achieves up to 97,53% accuracy.

QTag

Tufis and Mason [242] propose QTAG, a probabilistic part-of-speech tagger. Tiered tagging, as they call it, allows the usage of large tagsets that go beyond the computational power of average computer systems. This is accomplished by using a hidden tagset of smaller size that is post-processed (i.e., hidden tags are resolved by the original ones) during a later phase. Dictionary lookup is applied for initial tagging. Unidentified tags are analyzed by two morphological guessers, one applying suffix matching and another one which is linguistically motivated. An evaluation of the Romanian CTAG corpus results in over 97% accuracy.

The TnT Tagger

In 2000 Brants [31] introduces the statistical Trigrams'n'Tags (TnT) tagger. The TnT tagger is based on markov models, where the first step is processing maximum likelihood probabilities for unigrams, bigrams, trigrams, and lexical units. Smoothing, the linear interpolation of unigram, bigram, and trigram probabilities resolves the sparse data problem by avoiding zero probabilities. Unknown words are identified through suffix identification and capitalization clues. Experiments using the NEGRA and the Penn Treebank corpora showed that the TnT tagger outperforms other approaches reaching an accuracy of over 99%.

The Stanford Tagger

At Stanford University Toutanova and Manning [239] developed a maximum entropy tagger. According to its context, each word is assigned all possible tags through probabilities. The probability of a tag sequence is computed as the product of the conditional probability of each of the tags, resulting in a probability distribution. Out of this set of all feasible assignment sequences, the probability distribution that has the highest information gain (entropy) is selected. With special attention put on the correct identification of unknown words, the tagger achieves an overall accuracy of 96,86%.

4.2.4 Stopword Filtering

Fox [76] describes a method to systematically extract a list of stopwords from a document collection. In his work he relies on the Brown corpus that consists of over a million words from common English literature. Based on their frequencies terms occurring at least 300 times are extracted, resulting in an initial list of 278 terms. In a first step 32 non-stopwords are manually removed. 26 additional terms of lower frequency are manually added in a second step. Finally, the list is further extended by words that are considered as stopwords and which can be recognized with the same finite state machine (almost) for free. Therefore, he defines nine different adding criteria like words with different prefixes ended by `-body` (e.g., `anybody`, `nobody`), `-ly` (e.g., `clearly`, `fairly`), or `-self` (e.g., `itself`, `herself`). He ends up with a list of 421 stopwords for common English texts that "can serve as the basis for stop lists for specialized data bases" [76].

4.2.5 Stemming Systems

The Lovin Stemmer

Initial research in the area of word stemming is conducted by Lovins [165] in 1968. The proposed stemming algorithm is context sensitive, that is suffixes are only stripped if certain qualifications are fulfilled (e.g., minimum length of a remaining stem is two letters). Longest-match suffixes are removed first, implying an order among the set of reduction rules. The stemmer includes a list of 294 possible suffixes, a large list of exceptions, and a set of context sensitive rules (constraints) to avoid a wrong reduction of terms. The algorithm finalizes by applying rules for recoding incorrectly stemmed words (e.g., `-ll` $\rightarrow$ `-l`, `-iev` $\rightarrow$ `-ief`).

The Dawson stemmer

Compared to Lovins, the Dawson stemmer [62] uses a much larger list of about 1200 suffixes. These are implemented as character trees to ensure fast computation. In contrast to Lovins' stemmer no final recoding takes place. A partial matching routine ensures that nearly identical stems are reduced to a single one. This is done by sets of endings that, if removed, result in the same stem. For instance, the set $\{mit, miss\}$ summarizes the terms `admit` and `admiss`, where the shortest remaining stem `admit` is selected to serve as the single stem for all of the terms.

The Porter Stemmer

One of the best known stemmer is proposed by Porter [203]. The rules for stripping suffixes are organized in five distinct phases. Each of the phases succeeds the previous one, starting with phase 1 and ending with phase 5. Equations 4.1–4.6 depict some examples according to these phases. Phase 1 (Equation 4.1) reduces plurals and past participles. Subsequent phases further reduce the intermediate stems: phase 2 (Equation 4.2), phase 3 (Equation 4.3), and phase 4 (Equation 4.4). Finally, phase 5 tidies up some remaining suffixes (Equations 4.5 and 4.6).

$$\begin{aligned}
\textit{generalizations} &\rightarrow \textit{generalization} \quad (4.1)\\
\textit{generalization} &\rightarrow \textit{generalize} \quad (4.2)\\
\textit{generalize} &\rightarrow \textit{general} \quad (4.3)\\
\textit{general} &\rightarrow \textit{gener} \quad (4.4)\\
\textit{controll} &\rightarrow \textit{control} \quad (4.5)\\
\textit{probate} &\rightarrow \textit{probat} \quad (4.6)
\end{aligned}$$

In his experiments Porter shows that stemming applied for information retrieval reduces the size of the vocabulary by about one third.

The Paice/Husk Stemmer

The Paice/Husk stemmer [193] described by Paice is an iteratively operating rule-based stemmer. Each rule either deletes or substitutes word endings, making recoding rules obsolete. The rules are grouped according to the ending letters of the suffix, where their order within the groups is of importance. Some of the rules are dedicated only to words where no changes have yet been made. The rule format consists of five parts: an ending in reverse order, an optional intact flag, a zero elimination flag, an optional append string, and a continuation symbol defining the termination of the stemming procedure. Stemming is done by first selecting the corresponding group and applying the ordered rules until (1) a rule terminates the stemming, (2) no more rules are applicable, or (3) the group letter changes.

In his subsequent work Paice [194] presents a way to evaluate different stemmers. In an experiment he compares the performance of three different stemmers proposed by Lovin, Porter, and himself, using a self-generated test set of about 10000 words from the CISI document collection. The findings are that both, the Lovin' and the Porter stemmer tend to reduce words referring to the same concept to different stems. In contrast, the Paice/Husk stemmer tends to reduce words referring to different concepts to a single stem. However, without an explicit definition of an application domain neither of the stemmers clearly outperforms the others.

Krovetz' Extension of the Porter Stemmer

Krovetz [158] experiments with an altered version of the Porter stemmer. One of the major problems of Porter's work is that words with different meanings often are reduced to the same stem. To avoid this, Krovetz includes dictionary lookup that precedes each of the five phases in Porters stemming algorithm. If a match occurs, the stem included in the dictionary is taken instead of applying any rules, and the stemming procedure is instantly terminated. However, results only show slight benefits compared to the original version.

Thus, Krovetz proposes two other stemming approaches. First, an inflectional stemmer that only converts plurals to singular form, changes past tense to present tense, and eliminates `-ing` endings was developed. Each of the three tasks is preceded by a dictionary lookup. Second, a derivational stemmer was implemented that extends the inflectional stemmer by the fifteenth most frequent endings. In their experiments using four different document collections he shows that the derivational stemmer slightly outperforms the others (including the Porter stemmer).

The Croft Stemmer

Croft and Xu [58] present a corpus-specific stemming strategy. They apply an aggressive stemmer (Porter stemmer) that extensively reduces terms even with different meanings to the same stem. In their experiments the West collection and the Wall Street Journal collection are used. Based on the stems and a window size of 100 words, equivalence classes are formed for each stem using the unionfind algorithm (i.e., $< word, stem >$ pairs). Each word pair (i.e., $< word, word >$) within each equivalence class is evaluated according to Equation 4.7.

$$em(a,b) = \frac{n_{ab}}{n_a + n_b} \tag{4.7}$$

where n_a (resp. n_b) is the number of occurrences of term a (resp. term b) and n_{ab} denotes the number of occurrences of the terms a and b together in the same window. According to a $em(a,b)$ threshold new equivalence classes are formed, where the final 'stem' is defined by the shortest word in each equivalence class. The new corpus-specific $< term, stem >$ pairs are stored in a lookup table. Their results show a clear improvement of information retrieval applying the proposed stemming algorithm, even outperforming the results achieved by applying the Porter stemmer directly.

4.3 Natural Language Oddities

A properly working language processing system has to be aware of the different kinds of delimiters, printable characters, and special characters. Clearly, the depth of analysis has a strong impact on the system's performance regarding both, quality and time. An important point is an unambiguous terminology and support during all levels of language analysis. Last but not least domain specific knowledge plays a central role and has to be considered properly in advance. This implies a proper definition of allowed characters within processing strings. For instance, are units containing characters like `-`, `@`, `:`, `%`, `$`, `.` defined as correct words or not? The decision strongly relies on language, domain, and external knowledge which has to be provided to the language analyzer in some way.

Different character sets and the treatment of special characters always posed difficulties. Moreover, the discussion about upper- and lowercase letters has still not come to a conclusion. Most information retrieval systems based on keyword search are converting characters into lowercase. From the retrieval point of view this does not effect retrieval performance a lot [21]. Anyway, capital letters also have their own semantics which may be used to begin sentences, identify proper nouns, name institutions, or mark other kinds of tokens such as operation system specific commands. Obviously, a distinction between `Richard Brown` and `brown ink` may be of interest. In general there is no easy solution for the problem of accurate proper name detection [173, pp. 124].

One of the major difficulties in natural language processing pose the diverse possibilities to separate words and sentences. It includes common delimiters like blanks, tabulators, line feeds, and carriage returns. In addition to that, special characters may also be used to end up or recombine tokens, including apostrophes, hyphens, slashes, and punctuation marks. This section outlines example usages of these delimiters and points out possible treatments.

4.3.1 Difficulties Concerning Sentence Delimiters

Jackson and Moulinier pointed out that "Detecting sentence boundaries accurately is not an easy task" [141, pp. 9]. Usually we have no problem to detect the end of sentences with default markings like the period (`.`), question mark (`?`), and exclamation mark (`!`). Other candidates for sentence termination markings are the colon (`:`) and the semicolon (`;`). Especially in the context of irregular usage of those markers difficulties may arise (see Table 4.1). Without considering the actual context, an interpretation of punctuation marks at sentence borders may often be ambiguous. This diversity of marker usage gives a first insight

Table 4.1: Examples of (irregular) usage of sentence delimiters

Category	Example
sentence end	`This is the end...`
sentence ending number	`3 plus 4 is 7.  It ended 1977.`
sentence ending abbreviation	`Including mice, rats, etc.`
punctuation marks in text	`In Java !yellow means not(yellow).`
cumulative punctuation marks	`Is that true?!  Not this way!!!`
citation gap	`states that "one theory ...  saves the world"`

colon and semi colon	`Some animals:  cats and dogs.`
	 `Some cars:  the BMW; the Ford; and the Audi;`

into the complexity of natural language text processing. Each of the markers pose specific (language-dependent) peculiarities in the domain of text segmentation handling. In order to master the problem of sentence border disambiguation, a rule-based and linguistically motivated (language-dependent) multi-level approach is proposed. As a consequence fixed and static identification strategies often turn out to be not appropriate, suffering in (language (in-)dependent) tokenization.

Being the first step of natural language analysis, common tokenization strategies assume that the function of the period is easily identifiable. However, it takes over many different functions. One of its default functions is to decode sentence endings as the last character of a sentence ending word. But even this task of sentence detection based on border identification still seems not completely solved.

Usually the period is defined as the classical sentence delimiter in European languages. But on a closer look it delimits sentences (per definition it "draws the boundaries of something") only in combination with a blank. In the domain of pattern conceptualization it does not function in the same manner. Very often the period may also be used to format numbers, fix dates and times, compose complex formats (e.g., credit card numbers, postal codes), enumerate items (`the 4.rank`), identify variable names (`object.id`), abbreviate full words and phrases, and mark citation gaps (see Table 4.1).

The period itself does not uniquely function as a delimiter of sentences. Without a following blank (i.e., inside abbreviations or digits) it has a certain decomposing functionality to support the segmentation of concepts. In patterns like `12.35` and `3.567.123` it helps to distinguish between higher and lower measurement units. In abbreviation patterns like `e.g.` the (first) period helps to identify the related full words. Periods within URL addresses like `www.google.com` also decompose different (semantic) concepts on a syntactical level. Motivated by Guo [116, 117] and Mikheev [180] the document itself can be used to disambiguate part of such tokens. In contrast, a period followed by a blank terminates the conceptualization process of a semantic unit, which might be the realization of an abbreviation or the end of the whole sentence. In the pattern `e.g.` the second period has two functions: It decomposes one word of an abbreviation and it terminates the conceptualization of the whole abbreviated phrase `exempli gratia` because of the following blank.

Difficulties arise if abbreviations and regular words are overlapping, like `Wash.` (Washington) and `wash`. As a consequence dictionary lookup of the word `wash` (after punctuation mark splitting and case insensitivity) to identify non-abbreviations fails. Additionally, abbreviations may simultaneously denote sentence ends (like `etc.`).

Treating these abbreviations the same way as within sentences results in missing sentence ending periods. Thus, sentence borders are not correctly detected which may lead to problems is later processing steps (e.g., sentence-based tagging). In German, for example, one can define the period as a sentence end if the following conditions are met:

1. The sentence-ending token contains the period as its last character.
2. The sentence-ending token without the period is at least two characters long (no German word consists of a single character).
3. The sentence-ending token without the period must not be a number.

4. The sentence-ending token is the last token of the text, or the token which follows is a number or a string starting with a capital letter.

From this definition it is clear that finding sentence borders requires much knowledge about a language. To sum up, the period has the following coding functions:

- It decodes the termination of a sentence and functions as a delimiter between sentences together with spacings (blank, tabulator) and new lines (carriage return, line feed).
- In the domain of word dependent conceptualization it decodes the termination of semantic word concepts.
- It decodes the reduction of full words and phrases for the purpose of abbreviation.
- In the domain of formatting, it separates units for a better readability. After a number it implies an instantiation of an arithmetic concept.

4.3.2 Abbreviation and Acronym Detection

Abbreviations mainly consist of letters separated by periods where each period terminates a conceptualization. In contrast, acronyms consist of an arbitrary number of letters without any separation. They define a compound of related concepts that are reduced for the sake of easier comprehension by hiding the internal syntactic structure.

Using list comparison, there may always be the problem of huge amounts of listed items which are in most cases still incomplete. Abbreviations may be written with or without periods, making it even harder to detect sentence borders properly. They occur within titles, month names, jobs, institutions, country codes, and common words (see Table 4.2). Also, the list of used acronyms (often used in emails or online chats) is growing fast, and keeping track of all of them is quite impossible.

Table 4.2: Examples of abbreviations and acronyms

Category	Example
titles	`Dr., Prof., Dipl.-Ing., NN., John Doe, Jr.`*
month names	`Jan., Sept., Dec.`
jobs	`Off., Lt., Capt.`
institutions	`CIA, UNO, Microtest Inc.`
countries and locations	`AUT, CH, SLO, U.S., St. Germain`
names	`M. Hassler, Ali G.`
measures and metrics	`yd., yrs., m.p.h., sec., lbs., kg, l, db, AE`
common	`Mr., etc., aka., i.e., versch., fig., Dept.`
acronyms	`AEG, IBM, ISYS, GIS`
texting abbreviations	`LOL, MfG, CU, ACK`

* MUC6 example: Guidlines that pertain only to person Guidelines (`http://www.cs.nyu.edu/cs/faculty/grishman/NEtask20.book_9.html`, 08.09.2006)

A promising approach that may work is to use a dictionary containing common abbreviations combined with a list of domain specific abbreviations. Further, language and application dependent rules identifying special formats likely to match abbreviations can be used to improve the results. In combination with a full form lexicon the number of misinterpreted sentence endings can be additionally reduced.

4.3.3 Numbers

Generally numbers are not good index terms because of their vagueness and ambiguity [21]. But they can also be clearly of importance and should be at least recognized during text processing. Numbers may be used within written texts with punctuation marks like periods for large numbers or the commas for decimal numbers. In addition their formatting is language dependent, as the number `123,456.78` in English texts becomes `123 456,78` in French texts. The slash (e.g., `1/2`) is used to express fractions of numbers. Sometimes blanks are used for formatting, depending on the language and individual styles of an author. Numbers are used in postal codes, date and time formats, enumerations, abbreviations of their full forms, currencies, temperatures, metric measurements, or simply as counter. Table 4.3 gives some examples of number usages in texts. These patterns can be treated by

Table 4.3: Examples of number occurrences in texts

Category	Example
starting words	`3rd, 5th, 14tägig, 100fach`
within words	`F-117A bomber, H2O`
ending words	`vitamin B12, ISBN:123-1433-234`
with special characters	`20% VAT, volume #4`
composite nouns	`20-dollar bills`
names	`OS/2, PS/2`
numbers and fractions	`12.345,60, 5 1/4 points, 1.5 times faster`
times	`1:30p.m., on 14.may, 510B.C.`
dates	`2005/June, 01-05-2005, 1875/12/1, 01.01.99, April 05`
periods	`1970-80, summer term 2004/05`
currencies	`$100,000, US$42,1 millions,`
measures and metrics	`12kg, 127lbs, 4sec,`

rules defining the format (covered in the next subsections). An advanced lexical analysis procedure might also perform date and number normalizations to uniform formats [21, pp. 166].

4.3.4 Special Formats

Special formats like sixteen digits in a row separated by blanks or hyphens may identify credit card numbers. Such patterns can be covered by special rules. For example, phone numbers are written in quite heterogenous forms as shown in Table 4.4. They may contain spaces, periods, hyphens, brackets, and slashes to group digits in various forms [173, pp. 130].

Table 4.4: Examples of phone number formats

Phone numbers		
(43463)12345	0043-463-12345	+43 463 12345[a]
+43(0)463 12345[a]	+43/463 12345[a]	++43-463-12345
43-463-12345	+43/(0)463/12345	0463 2700 3531[a]
(44.171) 830 1007[a]	(202) 522-2230[a]	1-925-225-3000
212.995.5402[a]	+411/284 3797[a]	+49 69 136-2 98 05[a]

[a] these format cannot be identified on single-token level

Other formats like email addresses containing the `@` character, web addresses tending to start with www, or URL definitions including a protocol as `ftp://` have to be considered in texts as well (see Table 4.5).

Table 4.5: Examples of special formats

Special token formats	
john@smith.com	http://www.fbi.org
Object::Name	x.id
ID:F175-23-ak	c:\photo\123.jpg
IP:192.168.1.51	/home/marcus/repository.txt

4.3.5 Apostrophes

Apostrophes pose a set of difficulties in finding token borders and their treatment during tokenization. They are not only used for enclosing direct speech, but also for encoding contractions and concatenations (through elision), clitics, or indication of a plurals possessive (see Table 4.6) [173, pp. 126].

Table 4.6: Examples of apostrophe usage

Category	Example
direct speech	He said: 'It is really difficult!' He says: "Give me that coke, Mr. Smith." 'What do you mean?'
elision	you're, it's, won't, l'addition, c'est la vie, presqu'le
clitics	the man's car, the cat's food
plurals possessive	the boys' hats, the cats' food
quoted proper nouns	'Water Rats', 'The Round Table'
within names	O'Reilley, c't magazine, McDonald's, Alzheimer's[a]

[a] MUC6 example: Miscellaneous types of proper names from MUC6 Guidelines (http://www.cs.nyu.edu/cs/faculty/grishman/NEtask20.book_9.html, 08.09.2006)

Apostrophe treatment can be seen as a two level process. It presupposes simple tokenization including separation of token candidates (not including a delimiting character) and basic punctuation mark separation in a first place. Afterwards, linguistically motivated apostrophe treatment follows. The central issue is whether strings containing apostrophes should be separated or not. This issue cannot be resolved without incorporating general decision guidance. In a first rough approximation the following basic guidelines for apostrophe treatment are proposed:

- Apostrophes at the start and end of tokens (strings without delimiting characters) are considered as quotes and separated.
- Apostrophes occurring within tokens are candidates for further linguistically motivated splitting.
- Apostrophes that cannot be treated in both ways (e.g., no splitting is possible) are specifically marked and left for further treatment.

There are several ways to decompose aggregated elision patterns like `isn't` in which the `isn` is the result of a phonological motivated contraction process. It is an open issue whether those tokens should be (1) treated as a single token in form of `isn't`, (2) split up into two tokens `isn` and `t`, or (3) substituted by their full forms `is` and `not`.

The first strategy (1), which is not favored, does not treat apostrophes at all. The processing is left open to the next analysis step. The second strategy (2) treats the apostrophe as a separator and decomposes the involved elements without any linguistic interpretation. The third strategy (3), which is based on (language-dependent) logical and semantical analysis (decomposition rules), is preferred for the purpose of this work. It combines the first separation step with a second substitution step via morphosyntactically motivated linguistic rules. For instance, `n't` is interpreted as the negation operator `not` and thus gets isolated and substituted by its full form. In the case of `isn't`, this results in two separate words `is` and `not`. One has to note that there are also exceptions to this processing in English, as the tokens `won't` and `can't` cannot be treated in this manner.

To achieve optimal results the treatment of strings containing apostrophe has to be done on two levels. First it has to be compared to a (countable) list of exceptions (like `won't`) with their full strings (like `will not`). Matching strings are substituted by their listed full forms. If no match occurs general strategies are applied (like strings ending with `n't` are split up into the string before the `n't` and the substitution `not`).

The complexity of the problem confirms the hypothesis that proper apostrophe treatment has to consider the linguistic context. Token isolation in this sense minimizes the error-proneness of subsequent disambiguation processes operating on the remaining apostrophe containing strings.

4.3.6 Hyphenations

Hyphenation is another relevant issue in the context of multi-level tokenization [107]. Manning and Schütze distinguish three different types of hyphenation [173, pp. 127]:

1. Hyphenation is used for text justification, where words at line breaks are split up for the sake of alignment. Finding such hyphens as the last character of a word at the end of a line, getting rid of the hyphen, and reconcatenating it with the first word of the next line is an easy task. But it leads to haplology in the ambiguous case of words containing natural hyphens. In electronic texts such hyphens are usually not present.

2. Some words contain natural hyphens that are part of the word itself. This includes lexical hyphens inserted before or after small word formatives (e.g., `re-concatenate`).

3. Hyphens are used to indicate the correct grouping of words (e.g., `state-of-the-art`). Those tokens may be split up into their components to allow a syntactic analysis [21]. Because they are very common in many corpora, the size of vocabulary increases considerably if they are not split up (e.g., `information-based`).

Hyphens can also be used as an autonomous dash to separate semantic units of a sentence, for listing items, as a minus sign (in formulas), for the sake of abbreviation, and much more (see Table 4.7).

Table 4.7: Examples of hyphenation usage

Category	Example
(a) line breaks	`at-tach, unpro-nounced, analo-gy`
(b) single words	`e-mail, co-author, A-1-plus, so-called, pro-Arab, B-52`

(c) grouping of words	`aluminium-export, text-based, 15-year-old,`
	`ring-around-the-rose, science-fiction, U.S.-Japan trade`

(d) dashes	`It works - as I supposed - fine.`

(e) itemizations	`- easy`
	`- medium`
	`- hard`

(f) minus sign	`average of -25 degrees centigrade, obviously 3-2=1 is true`
(g) abbreviations	`the pre- and postconditions are`
(h) names	`Jean-Claude van Dame, F. Gregory Fitz-Gerald`

A particular problem in this area is the inconsistent use of hyphens [173, pp. 128]. There exist many examples of different forms and writing styles of the same terms like `co-operate/cooperate`, `data-base/database`, or `hyper-text/hypertext`. A careful editor surely would filter out this inconsistencies, but in practice most of the texts will contain such forms. To cope with these difficulties, the approach in this work treats hyphenation on different levels of tokenization:

- On the first level (simple tokenization) hyphens decoding token- and line-endings are marked as potential candidates for reconcatenation on the second level of tokenization. Hyphens surrounded by blanks (examples (d) and (e) in Table 4.7) are also marked and treated as separate tokens on the first level. All other hyphens within strings are treated as valid characters and are not processed any further.

- On the second level tokens marked for reconcatenation are substituted by their concatenated form. For instance, the term `hyphenation` may be hyphenated as the three token sequence `hyphen- + EndOfLine + tion`. Therefore, list comparisons (e.g., last token is not a valid term, concatenated string is a valid term) and rule-based strategies (e.g., `[term]- + [EndOfLine] + ness` → `[term]ness`) are applied. Tokens marked as autonomous hyphens are further interpreted as sentence markers ((d) in Table 4.7), implicit logical items ((e) in Table 4.7), or part of formulas ((f) in Table 4.7).

4.3.7 Slashes and Other Special Characters

In addition to apostrophes and hyphens, slashes, backslashes, and other special characters are frequently used within texts. Like before, these patterns may also be candidates for splitting – with the same drawbacks of misinterpretation. Some examples are depicted in Table 4.8.

Table 4.8: Examples of other special characters in tokens

Category	Example
names	`OS/2, PS/3000, Micro$oft, AT&T, Horvarth&Sons`
abbreviations	`w/o`
alternatives	`this element/container, the pre/postcondition`

The next section introduces Extended Tokenization, a rule-based approach that tackles these natural language oddities.

4.4 Extended Tokenization

Researchers from the text mining, information retrieval, and computational linguistics domains agree that tokenization is the first step of any kind of natural language text processing [255, 75, 107, 116, 117, 24]. Good surveys about tokenization techniques are provided by Frakes, and Baeza-Yates [77], Baeza-Yates and Ribeiro-Neto [21, pp. 165–167], and Manning and Schütze [173, pp. 124–136]. But only very few reflect tokenization as a task of multi-language text processing with far-reaching impact [98, 260]. This involves language-related knowledge about linguistically motivated and domain specific patterns on many levels of linguistic analysis (i.e., sentence border disambiguation, composite noun identification, abbreviation handling) [141, 217, 216].

The major goal of this early (pre-linguistic) task is to convert a stream of characters into a stream of processing units called tokens. Beyond the computational linguistics community this job is taken for granted. Commonly it is seen as an already solved problem comprising the identification of word borders and punctuation marks separated by spaces and line breaks. Many textbooks even dispatch tokenization as relatively uninteresting, but in reality "tokenization is a non-trivial problem" [107]. From the linguistic point of view it should manage language related word dependencies, incorporate domain specific knowledge, and handle morphosyntactically relevant linguistic phenomena.

At one stage of linguistic analysis, elements of a text have to be assigned to certain syntactic classes. In order to find those elements (strings) the text (one long string) has to be divided into logical units (tokens) that can be assigned to those classes. A key issue of tokenization is recognizing sentence boundaries since most grammars consider them as semantic and logic units of treatment [107]. Especially if using online material such as newsgroups and web pages for data "...one finds all sorts of oddities like C|net ..." [173, pp. 125]. Furthermore, characters like numbers, hyphens, punctuation marks or upper- and lowercase letters make up a considerable part of a text and have to be taken into account. In [141, pp. 10] the authors outline that a simple approach is insufficient.

The main assumption is that text – the content of fragments – which is processed by the system has already been pre-filtered, e.g., whitespaces are collapsed and tags for structural or layout relevant markup (e.g., `<link>`-tags, `<emph>`-tags) are removed. Furthermore it is presupposed that contents in form of headers, separators, tables, figures, typesetting codes, and other isolated objects are not included in the analysis [173, pp. 123].

After this preprocessing of raw textual content, tokenization is the first step dealing with pure natural language texts. It separates tokens and sentences, identifies proper nouns and special formats, handles abbreviations, and performs basic (non-linguistic) text transformations (thesaurus like substitutions or format normalizations). All subsequent steps during natural language analysis are based on the result of the tokenizer. This means that early errors occurring during tokenizing lead to very poor overall performance. It is important to be conscious that the theory and methods applied here have a great impact and have to be considered carefully [21, pp. 167].

Therefore, rule-based Extended Tokenization [122] is proposed that includes all sorts of linguistic knowledge (e.g., language, grammar rules, dictionaries), domain knowledge, gazetteer knowledge, and expert knowledge. Extended Tokenization in this sense does not only separate strings into basic processing units, but also interprets and groups isolated tokens to create higher level tokens. This is done by exploiting so-called token types assigned through an elaborated machinery of heuristic and linguistically motivated rules, and – if available – minimized dictionary knowledge.

Figure 4.3 shows where Extended Tokenization is located within the text preparation and processing framework. First, raw texts are preprocessed and segmented into textual units. This step comprises cleansing and filtering (e.g., whitespace collapsing, stripping extraneous control characters) [196] and removal of all kinds of structural or layout relevant markup. Then, Extended Tokenization segments the plain text into appropriate processing units. Tokenization does not include any kind of linguistic analysis like morphological word analysis or syntactic sentence analysis. These tasks are left open to be accomplished on higher levels during natural language analysis. Subsequent tasks like tagging are applied on the tokenized output and thus should be supported as far as possible (e.g., format normalization, consistent terminology).

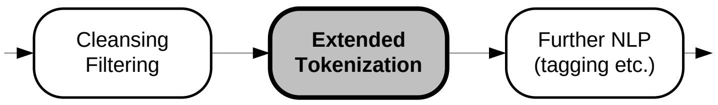


Figure 4.3: The task of text preparation and processing

The implementation prototype, JavaTok (see Section 4.4.3), provides a language independent core tokenizer that is easily adaptable for language specific needs by means of incremental rule set expansion. It operates on plain natural language texts, relying on the language-independent UTF-16 [244] character set. JavaTok supports rule-based typing of tokens using regular expression matching and methods for context dependent constraint checking. The core features of the implemented system are identification and disambiguation of all kinds of linguistic markers, detection and expansion of abbreviations, treatment of special writing formats, and typing of tokens including single-tokens and multi-tokens.

To improve the quality of textual representations, linguistically-based tokenization is a necessary step that precedes further text analysis. The complexity of subsequent processing tasks (e.g., tagging, stemming, named entity recognition, chunking, parsing, etc.) is reduced dramatically by decisions made early in the above mentioned string-interpretation domains. By mapping token types onto part-of-speech tags, the tagging process is speeded up. There is no need to identify string patterns again, because the tokenizer provides tagger relevant clues. Beyond that, the tagging results are improved because the tokenizer generates better interpretable input units (multi-tokens). This section focuses on improving the quality of standard tagging through Extended Tokenization. Thus, the overall quality of textual representation throughout all levels of analysis is enhanced.

The early identification of tokens partially covers the well known 'named entity recognition' task based on an empirically motivated categorization of proper names as defined by MUC-6 [182] and MUC-7 [183]. The Text Encoding Initiative (TEI) Guidelines [235, 234] for Electronic Text Encoding and Interchange cover such identifiers (plus abbreviations) and explain that they comprise "textual features which it is often convenient to distinguish from their surrounding text. Names, dates and numbers are likely to be of particular importance to the scholar treating a text as source for a database; distinguishing such items from the surrounding text is however equally important to the scholar primarily interested in lexis." [235, Section 6.4]

The next section provides the definitions of two token concepts. Section 4.4.2 introduces the notion of token types to support token generalizations and high level concept formulation. The procedure of assigning token types to tokens is described as a rule-based approach. The architecture and functionality of the implementation (JavaTok) is presented in Section 4.4.3. Finally, some features of JavaTok with respect to the theoretical considerations are outlined.

4.4.1 Definitions of Token Concepts

In this work two different kinds of tokens, namely single-tokens and multi-tokens are distinguished.

Single-Tokens

The simplest form of a token is the *single-token*. It is defined as a character string not containing any non-printable or delimiting characters (blank, tabulator, line feed, new line, etc.). It corresponds to the traditional concept of a token. Examples of single-tokens are words, numbers, internet addresses, and most abbreviations. See Guo [116, 117] and Mikheev [180] for more examples.

Multi-Tokens

Written texts also contain more complex language constructs that do not fit into the single-token concept. Such tokens may be specially formatted using blanks (the standard delimiter for token boundaries) that belong to semantically motivated groups of tokens. The blank is an inherent part of the token chain fixed together through interpretation (see Table 4.9).

Table 4.9: Examples of multi-tokens

Category	Example
Composite nouns	`traffic jam, rush hour, Christmas tree`
Full names	`Albert Einstein, George William Bush`
Institutions	`University of Klagenfurt, Red Cross`
Locations	`Niagara Falls, New York, Suncoast Seabird Sanctuary`
Special formats	`+43 463 12345-67, ISBN 202-546-234-0`

Tokens that contain token delimiters for formatting (e.g., blanks, tabs, new lines, line feeds) are defined as *multi-tokens*. Well known representatives are composite nouns (`summer time`), special formats (`+43 463 2700-3531`), named entities (names, locations, institutions), and idioms (formulas). Traditionally they have been identified as a sequence of atomic tokens glued together during a later processing phase - mainly using dictionary lookup. In this approach these tokens are multi-tokens through heuristic interpretation or, in other words, they are tokens through rule-based typing.

The early treatment of multi-tokens as (semantic) concepts during text processing improves the overall quality of information retrieval and document mining tasks. The representation of a text using multi-tokens leads to better intermediate results, hence structurally and (semantically) grouped tokens are treated as atomic units. If subsequent tasks do not support multi-tokens, a simple reinterpretation into standard single-tokens is on hand.

Many of the cases mentioned in Table 4.9 can be handled either by list lookup (compared to predefined strings) or by matching against regular expressions (like phone numbers). Extremely difficult to deal with are patterns including hyphenation like in `New York-New Haven railroad` [173, pp. 130]. Here, in order to avoid a misinterpretation of `York-New` as a separate hyphen-containing token, the recognition of named entities has to precede rule-based resolving of hyphens.

4.4.2 The Procedure of Token Typing

Rule-based typing of tokens has not been introduced in the literature so far. In this work typing of tokens is defined as a pre-linguistic classification process that assigns type identifiers to both, single-tokens and multi-tokens. Hence this proposal is user centered, the user himself is allowed to define his own token types for appropriate needs, where token types are simply expressed through strings. In a complex natural language processing framework token typing supports central linguistic tasks like simple part-of-speech tagging and/or basic semantic tagging. This approach partially covers named entity recognition, the classification of proper names into categories [180]. This processing step is integrated in the Extended Tokenization task because the identification of multi-tokens comprises more than just named entities.

The proposed framework for tokenization allows the definition of multi-tokens by including delimiters (like blanks) within the token recognition part. This is necessary for each variant of multi-level tokenization. The definition of token types relies on the available sources of knowledge and strongly depends on the motivation and application of token interpretation. There are

- domain knowledge: i.e., structure of an organization, knowledge about data warehouses;
- gazetteer knowledge: i.e., country names, river names;
- expert knowledge: i.e., medicine, astronomy;
- pure linguistic knowledge: i.e., morphological and syntactical rules, subject of a sentence.

The goal is to prohibit misleading separation during tokenization, providing an optimized input for further linguistic processing. It includes rule-based pattern recognition of single- and multi-tokens, grouping of tokens, and token splitting. Subtasks can be formulated as follows:

- string replacements (substitutions of single- and multi-tokens)
- abbreviation and acronym handling (recognition and substitution)

- identification of proper nouns and names (part of named entity recognition)
- application of rules on single- and multi-token level (for recognition, substitution, grouping, and splitting)

A minimal set of internal token types is used to mark end of sentences T_{eos}, punctuation marks T_{pm}, delimiters T_d, and unknown tokens T_u (darker boxes in Figure 4.4). Additional single-token types occurring in lookup lists or rule bases are dynamically incorporated and applied during the tokenization process (lighter boxes in Figure 4.4). In the figure, the numbers in parentheses count the rules that are defined for this token type (described in Chapter 5). In addition, higher level types for multi-tokens can be defined, referring

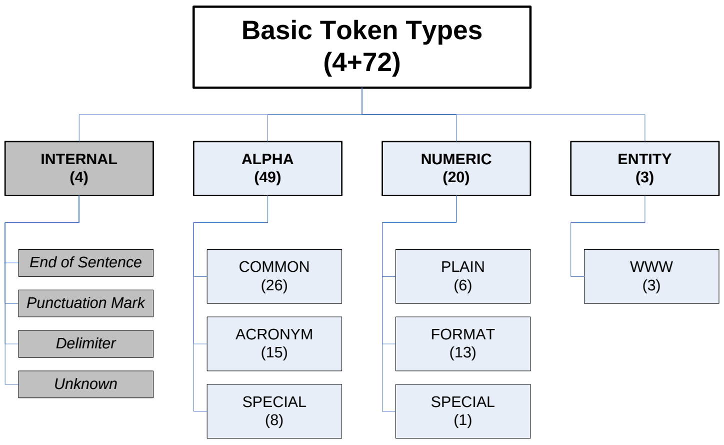


Figure 4.4: Basic token types

to predefined sequences of tokens. Using the same terminology through different natural language steps (e.g., equally named token types and part-of-speech tags), tokenization directly supports subsequent processing tasks. Otherwise, a reinterpretation or mapping step between token types and tags is needed, allowing to generalize classes of token types to a certain part-of-speech tag.

The typing process (see Algorithm 4.1) comprises two phases: The *first phase* (1-3) identifies token and sentence borders. It results in a sequence of single-tokens associated with basic token types. In a *second phase* (4-6) contextually motivated reinterpretation (retyping) of tokens is carried out. Therefore, rules for changing token types, rules for merging multiple single-tokens into a single multi-token, and rules for splitting a single multi-token into multiple single-tokens are applied recursively. The aim of the second phase is to improve the accuracy of the tokenization results achieved in the first phase.

The tokenization algorithm (Algorithm 4.1) starts with basic text segmentation, separating strings into single-tokens (step 1) using standard delimiters (blanks, tabs, new lines, line feeds). Initially, each token is assigned the unknown token type T_u. In step 2, rules assign basic token types (see Figure 4.4) to single-tokens, utilizing a classification of tokens in distinct

Algorithm 4.1 Typing of tokens

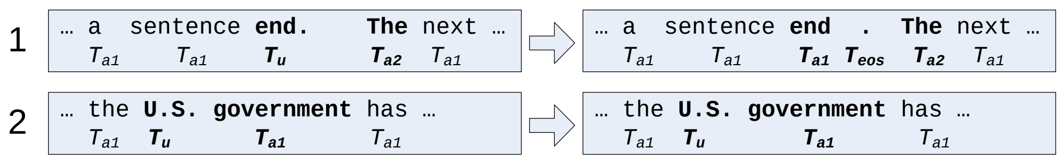
```
1: identify single-tokens
2: type single-tokens according to the basic token types
3: split sentence end markers

4: reinterpret single-token types
5: merge and split tokens recursively
6: reinterpret all token types
```

categories. This step includes assignments of the **Internal** token types. Some examples of other basic types and subtypes are

- **Alpha** T_a**:** no letters capitalized T_{a1}, first letter capitalized T_{a2}, all letters capitalized T_{a3}, mixed cases T_{a4}, etc.
- **Numeric** T_n**:** plain numbers (T_{n1}), numbers containing periods or colons T_{n2}, etc.
- **Entity** T_e**:** internet addresses T_{e1}, credit card numbers T_{e2}, etc.

The third step identifies and separates punctuation marks. Therefore, remaining tokens of unknown type T_u ending with a punctuation mark are investigated as possible sentence ends. If the complete token string does not match an entry in one of the repositories (e.g., lists of abbreviations, acronyms, regular expressions rules for single-token or multi-token typing), the last character is split and a new token is created together with its corresponding token type (see 1 in Figure 4.5). To assure the correctness of this splitting operation, a set of context-sensitive rules is applied. A token ending with a period and followed by a lower case token is not split, because the period does not mark the end of a sentence (see 2 in Figure 4.5).

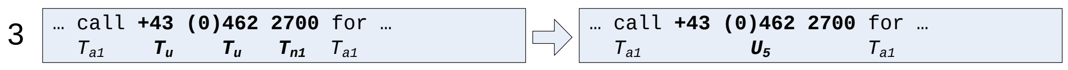

1
… a sentence **end.** **The** next …
T_{a1} T_{a1} T_u T_{a2} T_{a1}
… a sentence **end** **.** **The** next …
T_{a1} T_{a1} T_{a1} T_{eos} T_{a2} T_{a1}
2
… the **U.S.** **government** has …
T_{a1} T_u T_{a1} T_{a1}
… the **U.S.** **government** has …
T_{a1} T_u T_{a1} T_{a1}

Rule 1
IN: $t_{in,1}$.type = T_u AND $t_{in,1}$.str.endsWith('.') AND $t_{in,2}$.type = T_{a2}
OUT: $t_{out,1}$.str = $t_{in,1}$.str.substr(0,length-1) AND
$t_{out,1}$.type = getType($t_{out,1}$.str) AND
$t_{out,2}$.str = '.' AND $t_{out,2}$.type = T_{eos} AND
$t_{out,3}$.str = $t_{in,2}$.str AND $t_{out,3}$.*type* = $t_{in,2}$.type

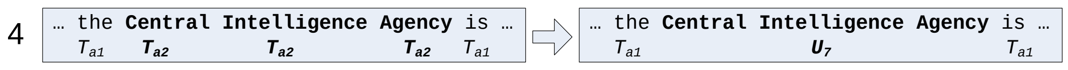
Figure 4.5: Example rule for punctuation mark splitting

A set of advanced user-defined token typing rules is then used to reinterpret, group, and split singe-tokens (steps 4–6 in Algorithm 4.1). The user is enabled to define custom rules to support domain-specific needs. These rules may also include references to the previously assigned single-token types.

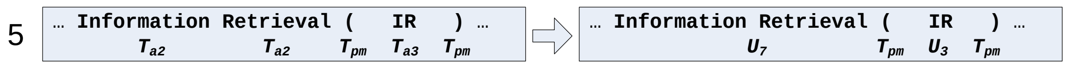
Examples of advanced user-defined types are stopwords U_1, abbreviations U_2, acronyms U_3, dates and times U_4, phone numbers U_5, email addresses U_6, and a sequence of capitalized single-tokens U_7 (in many cases extended keywords). These token types are assigned by applying two strategies: First, tokens are compared to a repository of reliable entries `(string, token type)` created either by a human expert or a (semi-)automatic machinery. If no match occurs, an ordered list of rules is applied to process the tokens and token sequences. The rules include regular expression matching of token strings (see 3 in Figure 4.6), matching of token types (see 4 in Figure 4.6), and combinations of both (see 5 in Figure 4.6).


3
… call +43 (0)462 2700 for …
T_{a1} T_u T_u T_{n1} T_{a1}
… call +43 (0)462 2700 for …
T_{a1} U_5 T_{a1}

```
Rule 2
 IN:  $t_{in,1}$.type = $T_u$ AND $t_{in,2}$.type = $T_u$ AND $t_{in,3}$.type = $T_{n1}$ AND
      ($t_{in,1}$.str ⊕ $t_{in,2}$.str ⊕ $t_{in,3}$.str).match(+[0-9]+\s\(0\)[0-9]+\s[0-9]+)
 OUT: $t_{out,1}$.str = ($t_{in,1}$.str ⊕ $t_{in,2}$.str ⊕ $t_{in,3}$.str) AND $t_{out,1}$.type = $U_5$
```


4
… the Central Intelligence Agency is …
T_{a1} T_{a2} T_{a2} T_{a2} T_{a1}
… the Central Intelligence Agency is …
T_{a1} U_7 T_{a1}

```
Rule 3
 IN:  $t_{in,1}$.type = $T_{a2}$ AND $t_{in,2}$.type = $T_{a2}$ AND $t_{in,3}$.type = $T_{a2}$
 OUT: $t_{out,1}$.str = ($t_{in,1}$.str ⊕ $t_{in,2}$.str ⊕ $t_{in,3}$.str) AND $t_{out,1}$.type = $U_7$
```


5
… Information Retrieval (IR) …
T_{a2} T_{a2} T_{pm} T_{a3} T_{pm}
… Information Retrieval (IR) …
U_7 T_{pm} U_3 T_{pm}

```
Rule 4
 IN:  $t_{in,1}$.type = $T_{a2}$ AND $t_{in,2}$.type = $T_{a2}$ AND $t_{in,3}$.type = $T_{pm}$ AND $t_{in,4}$.type = $T_{a3}$ AND
      $t_{in,5}$.type = $T_{pm}$ AND $t_{in,3}$.str.eqals(“(“) AND $t_{in,5}$.str.equals(“)“)
 OUT: $t_{out,1}$.str = ($t_{in,1}$.str ⊕ $t_{in,2}$.str) AND $t_{out,1}$.type = $U_7$ AND
      $t_{out,2}$.str = $t_{in,3}$.str AND $t_{out,2}$.type = $t_{in,3}$.type AND
      $t_{out,3}$.str = $t_{in,4}$.str AND $t_{out,3}$.type = $U_3$ AND
      $t_{out,4}$.str = $t_{in,5}$.str AND $t_{out,4}$.type = $t_{in,5}$.type
```

Figure 4.6: Tokenization rules

The examples in Figure 4.5 and Figure 4.6 outline the syntax of the tokenization rules. Each rule consists of a condition part *IN* (input sequence of typed tokens) and a consequence part *OUT* (output sequence of typed tokens). The numbered indices of tokens indicate relative token positions. The rule-based approach is based on simple and pure linguistic functional interpretation of basic token types and token strings in a given context. Example types of rules may cover morphological, syntactical, and general patterns like

- sentence border disambiguation
- multi-token identification
- special character treatment (e.g., apostrophes, slashes, ampersand etc.)
- suffix identification of well-known endings (e.g., `-ly`, `-ness`).
- identification and reconcatenation of hyphenated words at line breaks
- abbreviation treatment

4.4.3 JavaTok

This section describes the architecture of JavaTok, a freely configurable tokenizer developed in JAVA. To cope with language dependent occurrence of special characters (country specific characters like Slavic diacritics, French accents, umlauts and sharp s in German, etc.), JavaTok enables a Unicode-conform UTF-16 [243, 244] initialization and input/output processing. For the purpose of convenient higher-level tokenization the following features are necessary:

- free configuration and adaptation (character definitions, tokenization strategies)
- string replacements (abbreviation resolution, zero elimination, string and thesaurus-like substitution of multiple length)
- user-defined token type definition
- rule-based token typing (credit card numbers, phone numbers, dates, internet addresses, special IDs, . . .)
- pre-tagging functionality (based on token types)
- compound noun and proper name identification
- multi language support
- process statistics and runtime performance measurements
- multiple output formats

As presented, Extended Tokenization is defined as a rule-based process that includes basic linguistic knowledge. In case of ambiguity JavaTok does not make assumptions or guesses. Uncertain borders of tokens or sentences are not further interpreted or touched. This is because misinterpretation of tokens at an early stage leads to poor overall performance during language analysis. JavaTok also does not carry out any kind of character conversion automatically, hence other tools operating on the tokenizers' output may loose important information (such as the case of letters during tagging).

Architecture

Major aims were to support web-based processing, easy integration in existing software systems, and good overall performance. Therefore, the functionality of JavaTok can be accessed in three different ways:

- JavaTok operates as *stand-alone software* called from the command line. All initializations are specified via XML configuration files. JavaTok supports single file and batch processing, providing TXT, XML, and HTML output formats.

- The functionality of JavaTok can be included into other JAVA-based projects by simply importing JavaTok as a single *JAVA class*. Initialization and computation is done via public methods. An OutputFormatter is included supporting TXT, XML, and HTML.

- It is possible to access the tokenizer via internet as a *servlet* running on a Tomcat webserver. The online version is available at the Klagenfurt CLR web portal [123].

Figure 4.7 depicts the workflow of JavaTok. A single configuration file contains the input file, output file, parameter settings, and references to dictionaries and knowledge bases containing typing rules for single-tokens and multi-tokens. String replacements are carried out at the very beginning, eliminating undesired markup or known noise in the input texts. In a next step tokens are split according to the delimiter definitions. Subsequent single-token typing is carried out, assigning each token its basic token type (including T_u for unknown tokens). Punctuation mark splitting is only applied to tokens of unknown type. First, punctuation marks at the beginning of a token are separated and typed as T_{pm}. Then, if possible, single sentence end marks are split from the end of the tokens and typed as T_{eos}. Finally, remaining punctuation marks are isolated from the end of the token and typed as T_{pm}. After the splitting of punctuation marks multi-token types are identified. Finally, optional abbreviation substitution in both directions (expansion and abbreviation) is accomplished.

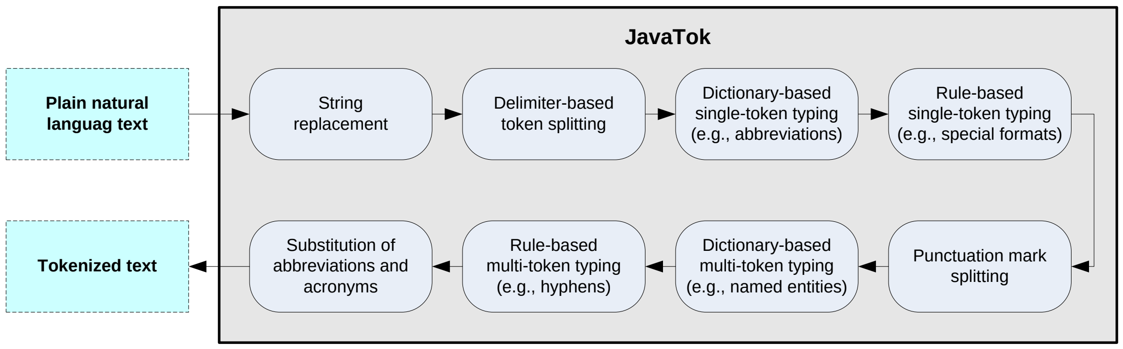

JavaTok
Plain natural languag text
String replacement
Delimiter-based token splitting
Dictionary-based single-token typing (e.g., abbreviations)
Rule-based single-token typing (e.g., special formats)
Punctuation mark splitting
Dictionary-based multi-token typing (e.g., named entities)
Rule-based multi-token typing (e.g., hyphens)
Substitution of abbreviations and acronyms
Tokenized text

Figure 4.7: The architecture of JavaTok

During all typing tasks priority is given to repository lookup. The rules for typing address token strings, token types, or both using regular expressions. The order of the tokenization rules is of great importance. The rules are applied sequentially, thus rules firing earlier prevent firing of later rules.

Sample Tokenization Outputs

In the example given in Figure 4.8 basic token types T_{a1}, T_{a2}, T_{a3}, and T_{pm} (see Section 4.4.2) are concatenated to tokens using a separating slash. Sequence of tokens creating multi-tokens are parenthesized.

User-defined token types are *ABBR* (abbreviation) and *INST* (institution). The mode describes whether single-token typing is enabled (S), whether multi-token typing is enabled

(*M*), and whether known abbreviations are replaced (*R*). A known abbreviation in the example is aka., standing for also known as. Red Cross is a known institution. RK is an unknown abbreviation standing for 'Red Cross'.

The Red Cross is aka. RK.	**S**	**M**	**R**
The Red Cross is aka. RK .			
The Red Cross is **also known as** RK .			x
The **(**Red Cross**)/INST** is aka. RK .		x	
The **(**Red Cross**)/INST** is **(also known as)/ABBR** RK .		x	x
The/T_{a2} Red/T_{a2} Cross/T_{a2} is/T_{a1} aka./ABBR RK/T_{a3} ./T_{pm}	x		
The/T_{a2} Red/T_{a2} Cross/T_{a2} is/T_{a1} **also**/T_{a1} **known**/T_{a1} **as**/T_{a1} RK/T_{a3} ./T_{pm}	x		x
The/T_{a2} **(**Red/T_{a2} Cross/T_{a2}**)/INST** is/T_{a1} aka./ABBR RK/T_{a3} ./T_{pm}	x	x	
The/T_{a2} **(**Red/T_{a2} Cross/T_{a2}**)/INST** is/T_{a1} **(also**/T_{a1} **known**/T_{a1} **as**/T_{a1}**)/ABBR** RK/T_{a3} ./T_{pm}	x	x	x

Figure 4.8: Sample JavaTok outputs

Tagging Optimization by JavaTok

Figure 4.9 shows the effect of Extended Tokenization on state-of-the-art tagging outputs. For evaluation purposes three freely available taggers are used: *QTag* [204] developed at the University of Birmingham, *POStaggerME* [192] available as part of the OpenNLP package provided by SourceForge, and *Stanford POS Tagger* [231] implemented by the Stanford Natural Language Processing Group.

The initial input sentence at the top of the figure comprises some of the discussed delimitation problems that motivate token typing and multi-token concepts. Cell (1) contains QTag outputs, cell (2) POStaggerME outputs, and cell (3) Stanford POS Tagger outputs. Lines 1a), 2a) and 3a) refer to standard tagging outputs after rudimentary tokenization, whereas lines 1b), 2b) and 3b) show the improved tagger outputs including preceded Extended Tokenization done by JavaTok. In the figure, underlines mark the tokens identified which served as input for the tagger. Tags assigned are located below the corresponding tokens, including *NN* (noun), *VB* (base verb), *VBG* (infinitive verb), *VBZ* (present tense verb, 3rd person singlular), *DOZ* (does), *XNOT* (negative marker), *JJ* (adjective), *IN* (preposition), *RB* (adverb), *CD* (number), *SYM* (symbol), *LS* (single letter), *FW* (foreign word), " (quotation mark), : (punctuation marks), and . (period). Tags written in bold are changed through JavaTok-specific groupings and/or splittings, thus resulting in optimized tagging inputs. Tagging improvement is achieved in two respects:

- Direct changes of tags through empirically motivated and more adequate input units
- Indirect changes of tags through changes of linguistic contexts

```
     Straight $n x n$ mapping doesn't fit into ... [5, 7].

   a) Straight $n  x  n$ mapping doesn't   fit into ... [  5  , 7  ]  .
      JJ       "   NN NN VBG     DOZ       JJ  IN   CD  NN CD , CD NN  .
1
   b) Straight $n x n$  mapping does not   fit into ... [5, 7]        .
      JJ       "        NN      DOZ  XNOT VB  IN   CD  NN            .

   a) Straight $n  x  n$ mapping doesn't   fit into ... [  5  , 7  ]  .
      JJ       NN JJ NN NN      RB        VB  IN   :   IN CD , CD NN  .
2
   b) Straight $n x n$  mapping does not   fit into ... [5, 7]        .
      JJ       NN       NN      VBZ  RB    VB  IN   :   CD            .

   a) Straight $n  x  n$ mapping doesn't   fit into ... [   5  , 7  ] .
      JJ       NN LS FW VBG     JJ        NN  IN   :   SYM CD , CD NN .
3
   b) Straight $n x n$  mapping does not   fit into ... [5, 7]        .
      JJ       NN       NN      VBZ  RB    VB  IN   :   CD            .
```

Figure 4.9: Tagging improvements through Extended Tokenization

Operating on the output of tokenization, tagging, the subsequent step of natural language processing, is described in the next section.

4.5 Tagging

Tagging is defined as the process of assigning syntactic categories in the form of Part-Of-Speech (POS) tags to words [173, pp. 341]. Compared to full syntactic parsing, the advantages of tagging are its stability and good computational performance. This is the reason for applying tagging before higher-level processing tasks (e.g., named entity recognition, chunk parsing) [39].

In most cases multiple syntactic categories can be assigned to a single word. Hence, the goal of tagging can be reformulated as: determine the correct (i.e., the most likely) category of a word in a given context. In other words, tagging is the disambiguation of part-of-speech [16]. In this process the set of all tags – the tagset – has a major impact on the performance of a tagger. A balance has to be found between the granularity of word categories and the accuracy of the tagger. In languages with rich morphology (e.g., German) this is extremely difficult. Two well known tagsets are those used in the Penn Treebank project[2] (for English), and the Stuttgart-Tübingen TagSet (STTS) [218] (for German).

Two groups of taggers [46, pp. 224] can be distinguished:

[2]http://www.cis.upenn.edu/~treebank (19.03.2008)

Stochastic taggers identify word categories using the probability of word category in a given context of other words and categories. Most of these taggers rely on hidden markov models (using the Viterbi algorithm [19, pp. 202–204]), decision trees, maximum entropy approaches, or support vector machines. Examples are the TreeTagger [221, 222], QTag [242], and TnT Tagger [31].

Rule-based taggers apply rules learned from a corpus to assign initial categories to the words. In a second step transformations are iteratively applied to change those initial tags to achieve better accuracy. In contrast to stochastic taggers the knowledge is encoded in a human-readable form. Thus, their performance can be increased by adding or adapting rules in the rule base. A well known example is the Brill tagger [34, 36].

Good overviews about the approaches are given by Abney [16], and Glass and Bangay [99]. Comparisons of tagging results are provided in [99, 205, 245].

In order to achieve proper tag assignments three different information sources are suggested [36]:

1. *Lexical lookup* is used to assign each term its possible tags according to the entries in a dictionary (e.g., constructed from a pre-tagged corpus). Thus, it is possible that a single word is assigned more than one category per se (e.g., `run` can be a verb or noun). Surprisingly, the accuracy of simply assigning each word its most likely tag (due to statistically ranked word tags in the dictionary) achieves up to 90% [173, pp. 344].

2. *Morphologic analysis* is capable of deciding a words' category. For instance, in English texts words ended by `-ion` (nouns), `-able` (adjectives), or `-ly` (adverbs) are easy to identify. However, there exist several exceptions to these rules (just think about `July` or `cable`). Weischedel et al. [256] and Samuelsson [215] summarize interesting experiments dedicated to the prediction of unknown word categories.

3. *Context rules* are applied to identify tags depending on their contexts. In most cases only preceding tokens are taken into account. The complexity of generating such rules increases exponentially with their window size (number of tokens they include). This is the main reason why most taggers consider n-grams: unigrams (one word context), bigrams (two words context), and trigrams (three words context). An example rule looks like $DET + ? + N \rightarrow ADJ$, which alters an unknown tag ? embraced by a determiner DET and a noun N to an adjective ADJ.

Table 4.10 depicts the general processing steps of assigning tags to words. The first step assigns each word its most likely tag according to a dictionary. Thus, the words `the` and `an` are determiners `DET`, `lady` is a noun `NN`, `is` is an auxiliary verb `AUX`, and the period defines the end of a sentence `$`. In step 2 morphological rules are applied on unidentified words not included in the dictionary. The suffixes `-ing` for verbs `VB` and `-ive` for adjectives `ADJ` are taken for granted. Finally, step 3 applies context dependent transformations on tags. Therefore, two rules $DET + ? + NN \rightarrow ADJ$ and $DET + ADJ + ? \rightarrow NN$ are applied.

Table 4.10: The process of assigning part-of-speech tags

Original:	The	nice	lady	is	buying	an	expensive	car	.
Step 1:	DET	?	NN	AUX	?	DET	?	?	$
Step 2:	DET	?	NN	AUX	VB	DET	ADJ	?	$
Step 3:	DET	ADJ	NN	AUX	VB	DET	ADJ	NN	$

However, the assignment of tags to words in real-world texts is not as simple as the example in Table 4.10 suggests. There are only very few situations where assignments are either unambiguous or can be restricted to a single tag. Furthermore, different languages require different efforts based on their morphological richness and syntactical freedom in positioning words and phrases. For instance, morphology and grammar in languages such as German and French are more complex than they are in English.

In this work tagging is exploited for the purpose of term selection. Therefore, only terms assigned to a certain subset of part-of-speech tags (i.e., nouns and verbs) are selected to create the index. All other terms (i.e., determiners, interjections, adjectives, adverbs) are simply neglected. This selection process reduces the amount of index terms and speeds up retrieval. However, inaccurate tag assignments may still lead to unwanted terms in the index. This unwanted material is further filtered using stopword lists described in the next section.

4.6 Stopword Filtering

Nearly all researchers working on natural language texts rely on a properly selected set of words and other items (e.g., alphanumeric combinations, numbers) to filter ‘meaningless’ and unwanted material. Such low level filtering lists are known as stopword lists, stoplists, or negative dictionaries. But although most retrieval systems use stoplists, there is astonishingly few literature about how to systematically generate stopword lists systematically [76, 75].

Most authors think of stopwords as words of little intrinsic meaning that occur too frequently to be useful when searching texts. Typically, these words are short (only a few characters), implement only a grammatical function, and don’t add to the meaning of the sentence [173, pp. 533–534]. Mainly these terms are articles, conjunctions, auxiliary verbs, or prepositions [46, pp. 482,491]. Some common examples of English stopwords are listed in Table 4.11.

Filtering stopwords from the set of index terms is done mainly for the reason of dimensionality reduction. Terms regarded as useless are filtered very early and do not need to be processed (e.g., stemmed) any further. According to Zipf’s law [173, pp. 533–534], this can

Table 4.11: Example for an English stopword list (57 entries) [173, pp. 533]

Special token formats								
a	also	an	and	as	at	be	but	by
can	could	do	for	from	go	have	he	her
here	his	how	i	if	in	into	it	its
my	of	on	or	our	say	she	that	the
their	there	therefore	they	this	these	those	through	to
until	we	what	when	where	which	while	who	with
would	you	your						

reduce the indexing vocabulary drastically. Thus, index construction, time, and storage costs are minimized [110, pp. 141]. Baeza-Yates [21, pp. 167–168] and Grossman [110, pp. 141] assume that up to 40% of all document terms can be regarded as stopwords.

Chu [52, pp. 8–9] extended this definition of stopwords, including other syntactic categories such as nouns and verbs. Thus, he added even infrequent, noninformative (e.g., `report`, `abstract`, `summary`), and domain and corpus specific terms (e.g., `computer`, `information retrieval`), where the latter two are also called trade words [52, pp. 47].

Although nearly all existing IR systems apply stopword filtering, there is still no standardized list agreed upon. As a consequence, these systems rely either on highly specialized hand-crafted lists, or on unreviewed lists downloaded from the internet. The drawback in both cases is that most of these lists are applied as given and remain unchanged.

Many stopword lists can be found in the web for free. Unfortunately very few attention is put on their linguistic quality, especially if used in restricted domains and application areas. Hence stoplists are used in nearly all information retrieval applications for early feature reduction, the retrieval quality highly depends on the content (and structure) of the lists. Surprisingly, the impact of the linguistic quality of those lists on the retrieval performance is often underestimated. Anyway, ignoring the effects of stopword filtering during the indexing and searching process can lead to the following problems:

- Without stopword filtering and a query that requires all terms to match, documents may be excluded because one of the stopwords did not appear in the document. Even though the stopword itself is probably irrelevant to what the user is searching for, the document may have actually been a very good result. For instance, a query `cat on a tree` does not match a document stating `cat on the tree`.

- If a search does not require all terms to match and stopwords are not filtered, any document containing at least one of the stopwords would be included in the relevant results. Since stopwords are very common, nearly all documents in the engine's index might be retrieved. As an example, the query `cat on a tree` matches all documents containing the words `a` or `the` (which can be found in nearly all documents).

- On the other hand rudimentary filtering of stopwords prohibits precise searches for query terms like `vitamin c` or `The Round Table`. As for the first example, all texts stating the term `vitamin` are returned, leading to a high number of irrelevant results. Generally, words that have different meanings in different contexts, one in which it is a stopword and an other one in which it is not, pose another problem: Confusing the noun `can` with the auxiliary verb `can` may result in serious troubles. Even more obvious, phrases consisting entirely of stopwords are hard to handle, for an example consider `to be or not to be`.

For those reasons some web engines abstain completely from stopword filtering in order to maintain high recall and to avoid results that users are unable to interpret [21, pp. 167–168].

4.6.1 The Multi-Layered Stopword Model

This work proposes that the stopword class is much more heterogenous than suggested in the literature. It is composed of three different layers, as depicted in Figure 4.10. Each layer is composed of certain groups containing terms sharing syntactic (expressed through word categories in the figure), semantic (related to common concepts), or domain-specific (related to concepts in a restricted domain) attributes.

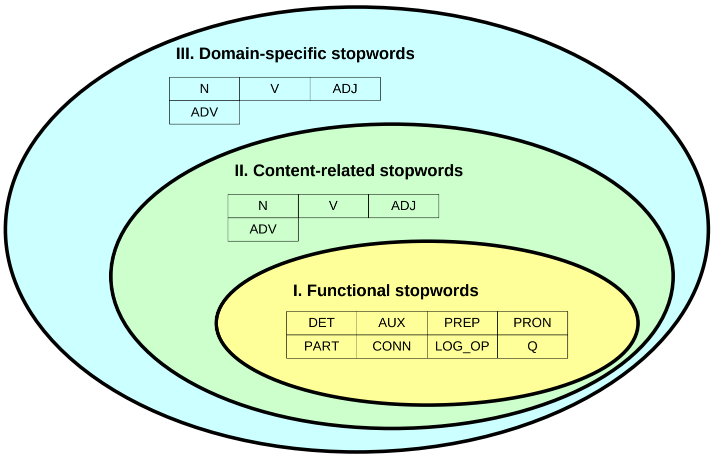


Figure 4.10: Multi-layered stopword model I

Linguistically argued, stopwords can be distinguished into three layered subsets (see Figures 4.10 and 4.11):

Functional stopwords define terms which are regarded as traditional stopwords. They belong to the class of function words (or grammatical words) which have only little lexical meaning. Normally these terms decode syntactic and morphosyntactic features (e.g., values of parameters like case, numerus, genus, tempus, modus). Generally, functional stopwords do not transport any semantics and are not bound to specific domains. Typical representatives are determiners, auxiliaries, prepositions, pronouns, particles, conjunctions, logical operators, quantors, etc.

Content-related stopwords are mainly adjectives and adverbs which decode very general semantic concepts. These stopwords do not contribute to the semantics of a text and are domain-independent. For the same reason a set of verbs (e.g., `able`, `based`, `consider`) and nouns (e.g., `abstract`, `degree`, `goal`) also belong to this category.

Domain-specific stopwords are words which refer to domain-specific entities and concepts. Normally these terms are part of the presupposed knowledge of a domain insider. Preferred candidates for this group are nouns and named entities (e.g., acronyms), verbs, adjectives, and adverbs. For domain experts, these words are taken for granted and do not contain semantically important information. For instance, terms like `computer` or `data` might not be good candidates for a common stoplist. However, both terms may be perfect stopwords when searching a huge archive about computer related topics.

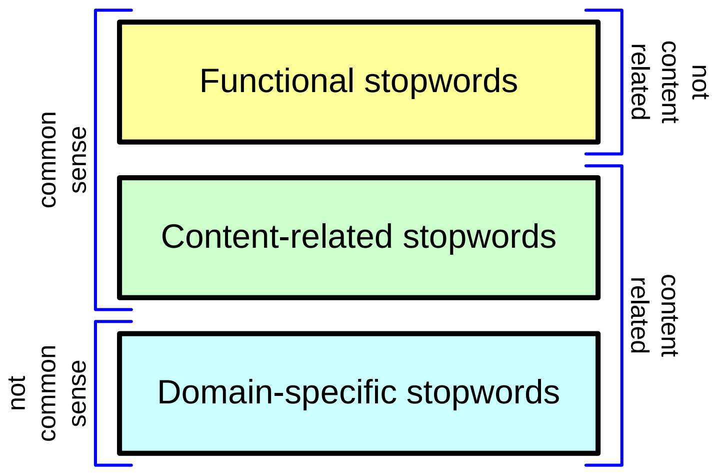


Figure 4.11: Multi-layered stopword model II

Conducting a three level stopword filtering would allow to focus on certain information retrieval and document mining tasks while improving the system's performance regarding runtime and index space. In the context of this work stopword filtering is applied in two ways:

1. Stopwords are filtered from the set of index terms, utilizing stopwords in a traditional manner. This filtering step includes stopwords of all three levels.

2. In document mining tasks, valuable resources aiding information retrieval are extracted from a corpus automatically. However, the amount of raw data extracted is inapplicable

to be applied directly and contains loads of irrelevant information. Therefore, stopword filtering is applied to exclude unwanted patterns (e.g., phrases started or ended by stopwords) from further analysis. Chapter 5 provides methods that generate appropriate dictionaries by applying stopword filtering. These dictionaries include composite nouns, named entities, formulaic speech, and full forms of acronyms.

From this point of view stopwords have to fulfill both statistic and linguistic criteria. In the sequel the process of creating and incrementally extending the list of stopwords based on statistic and heuristic methods is described.

4.6.2 The Stopword Extraction Process

In order to generate a stoplist, this work relies on the corpus described in Section 2.9.1. The result of this process is a ranked list of terms that are promising candidates. It serves as input for a linguistically-based categorization and for heuristically motivated layering of useful stopwords. The process relies on the tokenization output only, thus no part-of-speech tagging or any other linguistic processing is required.

By going through the corpus in a single pass, the output of the Extended Tokenizer is used to gather corpus based term statistics. All single-tokens that occur in the texts as 'proper' words – defined as a sequence of small letters, optionally started by a single capital letter – are regarded as potential candidates. Hence filter terms are considered as context free, no left or right neighborhood is taken into account. The token selection process itself is solely based on the token-types assigned by the Extended Tokenizer. Other token-types like numbers, acronyms, or special formats are ignored.

For each term i the term frequency n_i (the number of occurrences of term i in the collection) and the document frequency m_i (the number of documents stating term i) are computed. As this work utilizes structured documents, it is important to note that both frequencies are computed in a certain context c, resulting in $n_{i,c}$ and $m_{i,c}$. Thus, the approach can be applied to generate term statistics based on structural conditions of chapters (`/DOC/SEC`), on any kind of sections (`//SEC`), or on paragraphs (`//FRA` of type `text`). The experiments described in this chapter are conducted using the `/DOC` context.

Based on the $n_{i,c}$ and $m_{i,c}$ values, the inverse term frequency $itf_{i,c}$ and the inverse document frequency $idf_{i,c}$ are computed for each term, applying the formulae

$$itf_{i,c} = \log \frac{N_c}{n_{i,c}} \tag{4.8}$$

$$idf_{i,c} = \log \frac{M_c}{m_{i,c}} \tag{4.9}$$

where N (resp. M) is the total number of terms (resp. elements, i.e., documents, chapters, sections, paragraphs) in the whole collection. The inverse term frequency $itf_{i,c}$ is used to express the distribution of term i throughout all terms. The higher the value, the more discriminant is the term i itself. Accordingly, the inverse element frequency $idf_{i,c}$ is computed, expressing the distribution of term i through the collection. The higher the value, the rarer the term occurs in the collection.

In order to extract filter terms two parameters have to be specified:

Ranking criteria First, a ranking strategy according to the 'quality' of stopwords has to be specified. Statistically argued stopwords can be defined as extraordinary frequent words [76], thus terms having a high $n_{i,c}$ (resp. low $itf_{i,c}$) value. Alternatively, stopwords can be described as equally distributed throughout all texts, thus having a high $m_{i,c}$ (resp. low $idf_{i,c}$) value. In order to express the adequacy of a term i to become a stopword, both values $itf_{i,c}$ and $idf_{i,c}$ are combined in two different ranking formulae q_1 and q_2:

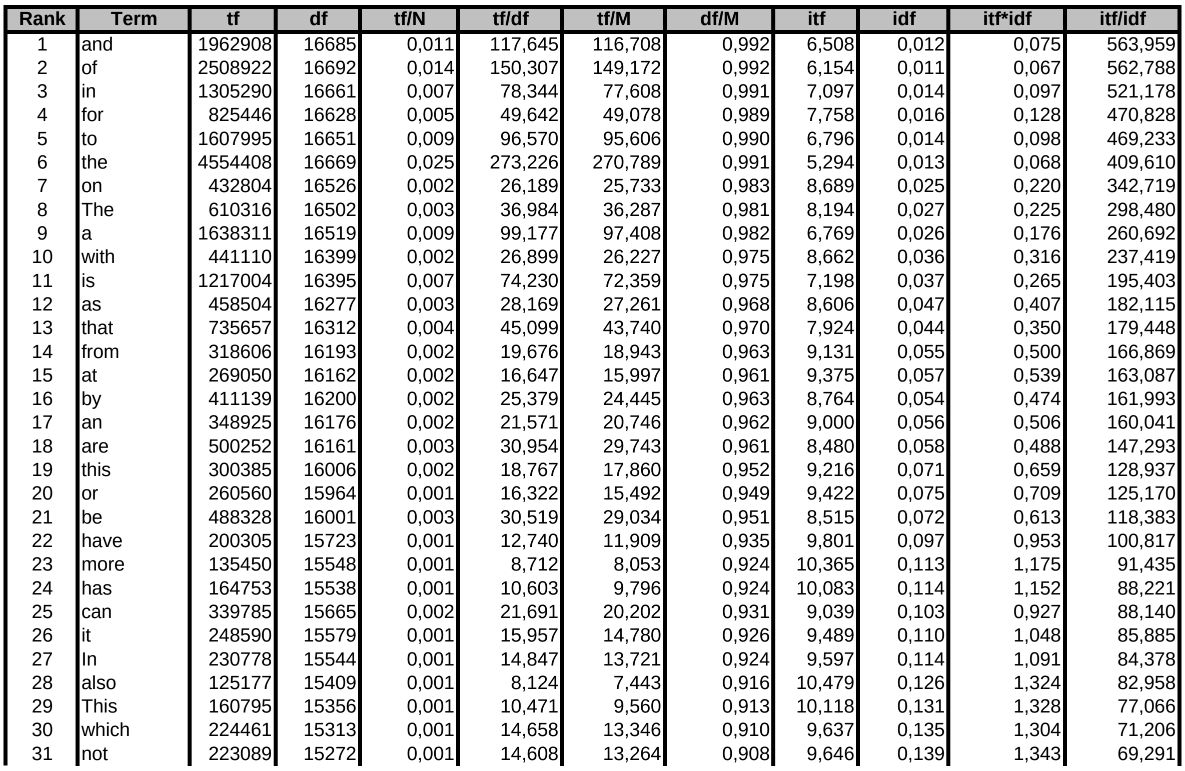
$$q_1(t_i) = itf_{i,c} \cdot idf_{i,c} \tag{4.10}$$

$$q_2(t_i) = \frac{itf_{i,c}}{idf_{i,c}} \tag{4.11}$$

Cutoff point Second, a certain cutoff point for term selection must be chosen. From the literature [21, pp. 167], the size of a general stopword list can be estimated to be up to 500 words. Using the Brown corpus ($\sim$ 1 million words), Fox [76] chose the cutoff at the minimum term frequency of 300, which was based on an empirical decision. Used for the experiments in the current work, the top 500 ranked terms are classified as stopwords.

The extracted stopwords are sorted according to the quality measures q_1 and q_2 in decreasing order. Good stopwords are reflected by high $q_1(t_i)$ and $q_2(t_i)$ values. This procedure results in the list given in Figure 4.12 (first 31 words ranked by q_2 in descending order). Note that itf and idf use different scalings: With respect to the INEX corpus, itf lies between 6,5 and 14,0 because even the most frequent term `the` does not occur in half of the cases (where $ift_{i,c}$ would be $1,0$). In contrast, idf ranges from 0,01 to 27,4 because many terms occur in more than half of the documents (where $idf_{i,c} < 1,0$). This fact explains the high impact of idf in both, q_1 and q_2. For the data shown in Figure 4.12, N_c = 178.677.541 denotes the total number of terms, and M_c = 16.819 is the total number of documents in the collection.

As the list in Figure 4.12 is case sensitive, the 500 selected entries are reduced to 445 unique terms (e.g., see `the` and `The` at ranks 6 and 8). These 445 words cover 22,55% of all



Rank	Term	tf	df	tf/N	tf/df	tf/M	df/M	itf	idf	itf*idf	itf/idf
1	and	1962908	16685	0,011	117,645	116,708	0,992	6,508	0,012	0,075	563,959
2	of	2508922	16692	0,014	150,307	149,172	0,992	6,154	0,011	0,067	562,788
3	in	1305290	16661	0,007	78,344	77,608	0,991	7,097	0,014	0,097	521,178
4	for	825446	16628	0,005	49,642	49,078	0,989	7,758	0,016	0,128	470,828
5	to	1607995	16651	0,009	96,570	95,606	0,990	6,796	0,014	0,098	469,233
6	the	4554408	16669	0,025	273,226	270,789	0,991	5,294	0,013	0,068	409,610
7	on	432804	16526	0,002	26,189	25,733	0,983	8,689	0,025	0,220	342,719
8	The	610316	16502	0,003	36,984	36,287	0,981	8,194	0,027	0,225	298,480
9	a	1638311	16519	0,009	99,177	97,408	0,982	6,769	0,026	0,176	260,692
10	with	441110	16399	0,002	26,899	26,227	0,975	8,662	0,036	0,316	237,419
11	is	1217004	16395	0,007	74,230	72,359	0,975	7,198	0,037	0,265	195,403
12	as	458504	16277	0,003	28,169	27,261	0,968	8,606	0,047	0,407	182,115
13	that	735657	16312	0,004	45,099	43,740	0,970	7,924	0,044	0,350	179,448
14	from	318606	16193	0,002	19,676	18,943	0,963	9,131	0,055	0,500	166,869
15	at	269050	16162	0,002	16,647	15,997	0,961	9,375	0,057	0,539	163,087
16	by	411139	16200	0,002	25,379	24,445	0,963	8,764	0,054	0,474	161,993
17	an	348925	16176	0,002	21,571	20,746	0,962	9,000	0,056	0,506	160,041
18	are	500252	16161	0,003	30,954	29,743	0,961	8,480	0,058	0,488	147,293
19	this	300385	16006	0,002	18,767	17,860	0,952	9,216	0,071	0,659	128,937
20	or	260560	15964	0,001	16,322	15,492	0,949	9,422	0,075	0,709	125,170
21	be	488328	16001	0,003	30,519	29,034	0,951	8,515	0,072	0,613	118,383
22	have	200305	15723	0,001	12,740	11,909	0,935	9,801	0,097	0,953	100,817
23	more	135450	15548	0,001	8,712	8,053	0,924	10,365	0,113	1,175	91,435
24	has	164753	15538	0,001	10,603	9,796	0,924	10,083	0,114	1,152	88,221
25	can	339785	15665	0,002	21,691	20,202	0,931	9,039	0,103	0,927	88,140
26	it	248590	15579	0,001	15,957	14,780	0,926	9,489	0,110	1,048	85,885
27	In	230778	15544	0,001	14,847	13,721	0,924	9,597	0,114	1,091	84,378
28	also	125177	15409	0,001	8,124	7,443	0,916	10,479	0,126	1,324	82,958
29	This	160795	15356	0,001	10,471	9,560	0,913	10,118	0,131	1,328	77,066
30	which	224461	15313	0,001	14,658	13,346	0,910	9,637	0,135	1,304	71,206
31	not	223089	15272	0,001	14,608	13,264	0,908	9,646	0,139	1,343	69,291

Figure 4.12: Top ranked INEX stopwords using q_2

words in the INEX document collection, where the minimal term frequency $n_{i,c}$ is 8271 and the minimal document frequency $M_{i,c}$ is 5342.

Figure 4.13 overviews the top 500 results applying different ranking strategies. Ranking strategies combined term frequency tf and document frequency df (resp. itf and idf), and their performance is compared to each other. The first row headed by # SW counts the number of relevant stopwords (out of the 500 extracted) compared to the final reference list. Green colored cells explicitly mark the words contained in the final list. In their work Grossman and Frieder stated that "using term frequency is a means of implementing a dynamic stop word list" [110, pp. 197]. Indeed, at the first glimpse term frequency tf based ranking looks as a good indicator.

But the results of the experiments show that this method generates many unwanted stopwords already among the top 100 ranks. Already at ranks 29, 31, and 34 frequent nouns such as `data`, `system`, and `time` occurred (see Figure 4.13). All in all, 193 of the final stopwords are identified. The (inverse) document frequency df, idf based ranking performed clearly better, including unwanted results at lower ranks 49 (`time`), 64 (`example`) and 65 (`work`). The 500 terms extracted include 251 final stopwords. The worst performance with only 67 accepted

# SW	193	251	67	219	245	262
Rank	tf	df, idf	tf/df, df/tf	tf * df	itf * idf	itf/idf
1	the	of	the	the	of	and
2	of	and	of	of	the	of
3	and	the	and	and	and	in
4	a	in	a	a	in	for
5	to	to	to	to	to	to
6	in	for	in	in	for	the
7	is	on	is	is	a	on
8	for	a	for	for	on	The
9	that	The	that	that	The	a
10	The	with	itemsets	The	is	with
11	are	is	The	are	with	is
12	be	that	are	be	that	as
13	as	as	be	as	as	that
14	with	by	itemset	with	by	from
15	on	from	quorum	on	are	at
16	by	an	as	by	from	by
17	an	at	Jul	an	an	an
18	can	are	with	can	at	are
19	we	this	on	from	be	this
20	from	be	by	this	this	or
21	this	or	submesh	we	or	be
22	at	have	we	at	can	have
23	or	can	coterie	or	have	more
24	it	it	hypernode	it	it	has
25	In	more	can	In	In	can
26	which	In	multidestination	which	has	it
27	not	has	an	not	more	In
28	have	also	supernode	have	which	also
29	data	This	from	has	also	This
30	each	which	checkpoint	data	This	which
31	system	not	minutiae	This	not	not
32	has	other	trie	all	all	other
33	This	all	this	each	other	their
34	time	such	authorizations	one	such	such
35	all	will	i	time	will	all
36	We	their	cache	will	their	will
37	one	one	node	system	one	these
38	will	these	quorums	such	these	one
39	such	than	at	more	than	than
40	two	its	or	two	its	its
41	more	use	subcube	We	use	use
42	number	into	watermark	also	into	into
43	was	but	it	its	For	but
44	set	For	data	other	but	For
45	also	two	declustering	was	two	only
46	its	only	In	these	we	they
47	used	they	which	than	only	new
48	other	new	not	For	time	two
49	For	time	n	used	they	It
50	these	we	t	use	new	some
51	than	some	query	number	each	most
52	if	It	image	only	using	time
53	only	using	algorithm	set	some	using
54	use	most	checkpoints	their	It	many
55	model	each	submeshes	if	used	first
56	using	used	Sep	using	was	when
57	information	first	k	into	between	so
58	between	when	nodes	between	when	between
59	our	between	faults	information	first	used
60	their	was	have	but	most	each
61	into	many	system	when	system	been
62	algorithm	been	x	systems	example	example
63	systems	so	p	where	been	we
64	where	example	checkpointing	example	many	was
65	when	work	each	our	if	work

Figure 4.13: Comparison of different ranking criteria

stopwords is achieved with fractions of raw term and document $\frac{tf}{df}$ and $\frac{df}{tf}$. Ranking based on $tf \cdot df$ leads to better results containing 219 final stopwords. Best results are achieved using a combination of itf and idf. The explanation for this behavior is that the high term frequency tf influences the ranking much more than the lower element frequency df, which is smoothed by the logarithm. Surprisingly, $q_1 = itf \cdot idf$ extracting 245 stopwords performs slightly worse than $q_2 = \frac{itf}{idf}$ extracting 262 stopwords. However, there is no big difference in the results compared to simple element frequency df (resp. idf) ranking, which achieved 251 of the final stopwords.

To summarize the results of the experiments, rankings including the (inverse) element frequency obviously outperform the other strategies, where the best results are tight-fitting, relying either on the raw element frequency df, $q_1(t_i)$, or $q_2(t_i)$.

The top two lists acquired by using the plain element frequency df ranking and q_2 ranking only differ with respect to 20 terms. The df list exclusively contains the terms `state`, `test`, `code`, `communication`, `points`, `respectively`, `product`, `operation`, `parallel`, `task`, `line`, `memory`, `quality`, `you`, `parameters`, `source`, `power`, `Table`, `behavior`, and `elements`. The q_2-ranked list exclusively contains `especially`, `advantage`, `providing`, `once`, `works`, `yet`, `details`, `little`, `involved`, `developing`, `likely`, `smaller`, `back`, `increasing`, `across`, `recent`, `depends`, and `ability`. From a linguistic point of view, the q_2-ranked list seems slightly better because it includes more terms fitting in the functional and content-related stopword layers. In contrast, the other strategy favors nouns and verbs. However, differences occur only at ranks lower than 337. Ranks of the first hundred entries are nearly identical, differing only in up to five positions for each stopword.

According to the multi-layered stopword model extracted stopwords are (sub)classified in the previously described layers.

4.6.3 Identification of Functional Stopwords

Starting with the q_2-ranked stopword candidates generated, a list of functional stopwords is created. This list covers most of the words included in traditional stopword lists. A classification system is applied for a systematic generation of an extended functional stopword list. The classification is based on the widely accepted schema of linguistic word categories:

Determiners (DET) are noun specifiers that express definiteness or indefiniteness of a noun phrase (e.g., `a`, `the`). In many languages, for instance German, it additionally encodes morphosyntactic information like casus, genus, and numerus (e.g., `dieser`, `eine`, `das`).

Auxiliary verbs (AUX) are verbs that accompany the head verb of the sentence, expressing grammatical distinctions like person, number, tempus, aspect (e.g., `can`, `may`, `do`, `be`, `have`).

Prepositions (PREP) are non-inflected elements that govern the case of their nominal complements (e.g., `in`, `on`, `upon`, `by`).

Pronouns (PRON) are pro-forms that substitute noun phrases. They decode values of parameters like person, genus, numerus. Common types are personal, possessive, and interrogative pronouns (e.g., `he`, `her`, `who`).

Particles (PART) are non-inflectional words that do not have any governing function. They belong to very heterogenous subclasses like verb particles (e.g., `[stand] up`, `[bring] down`), model particles (e.g., `quite`, `very`), and infinitive particles (e.g., `to`).

Connectors (CONN) are coordinating or subordinating conjunctions that establish semantic relations between parts of sentences or whole sentences (e.g., `further`, `hence`, `since`).

Logical operators (LOG_OP) establish logical connections or interpretations concerning elements in their scope (e.g., `and`, `or`, `not`).

Quantifiers (Q) express a definite or indefinite number of entities. Normally, three groups can be distinguished: fuzzy (e.g., `some`, `many`), numeral (e.g., `two`, `million`), and cardinal (e.g., `second`, `fifth`) quantors.

Further admissible candidates for functional stopword lists are interjections (e.g., `oh`, `ah`, `hey`, `man`, `uhhh`), negatives (e.g., `no`, `not`), politeness markers (e.g., `please`, `thank you`), greetings (e.g., `hello`, `goodbye`), and the existential `there`. However, these category labels are not used for categorization in this work. One might note that the categories strongly correlate with part-of-speech tags used by well known tagging systems.

The list in Table 4.12 contains 140 functional stopwords that cover 36,3% (34.299.626) of all terms in the whole INEX collection. Figure 4.14 shows the number of documents terms according to the functional stopword categories.

The subclassification of functional stopwords supports further linguistic tasks such as composite noun identification and acronym resolution (see next chapter). Further selection of certain subsets of functional stopwords strongly depends on the focus of the application. For instance, the task of comparing queries and metadata information (e.g., section titles, table captions, authors) may require only specific functional stopword categories such as DET, PREP, CONN, and LOG_OP.

4.6.4 Identification of Content-Related Stopwords

In addition to functional stopwords many other terms can be regarded as stopword candidates. For instance, commonly occurring nouns, verbs, adverbs, or adjectives are also very frequent and equally distributed throughout document collections.

Table 4.12: List of functional stopwords

Category	Terms
DET (10)	a, an, here, some, that, the, there, these, this, those
AUX (26)	are, be, become, becomes, been, being, can, cannot, could, do, does, done, had, has, have, having, is, make, may, might, must, should, was, were, will, would
PREP (29)	about, above, across, after, against, along, among, amongst, around, aside, at, before, beforehand, behind, below, beside, between, beyond, by, down, for, from, in, into, near, of, on, out, outside, per, since, through, thru, to, toward, towards, under, until, unto, up, upon, via, with, within, without, off
PRON (22)	another, anybody, anyhow, anyone, anything, anyway, anywhere, during, elsewhere, everybody, everyone, everything, everywhere, he, her, hers, herself, him, himself, his, how, i, it, its, itself, me, my, myself, nobody, none, noone, nowhere, onto, our, ours, ourselves, she, somebody, somehow, someone, something, sometime, somewhat, somewhere, such, that, their, theirs, them, themselves, they, us, we, what, when, whence, where, which, whither, who, whoever, whom, whose, why, you, your, yours, yourself, yourselves, whomever
PART (12)	almost, as, down, even, just, no, out, over, quite, rather, so, to, too, up, very, yes, off
CONN (20)	after, although, and, because, before, but, further, furthermore, hence, howbeit, if, insofar, instead, like, neither, nor, not, or, since, than, then, thence, therefore, though, thus, unless, until, whenever, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, while, nonetheless
LOG_OP (3)	and, not, or
Q (18)	all, both, each, every, few, first, five, four, many, much, often, one, second, some, third, three, two, various

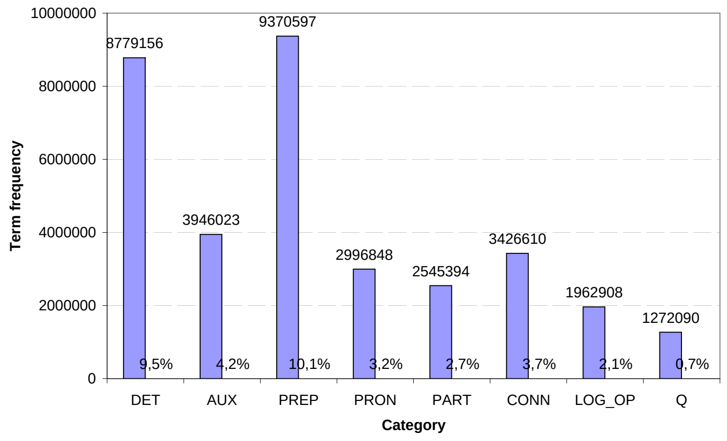


Figure 4.14: Frequency distribution of linguistic categories in functional stopwords

Terms that can be taken for granted throughout all domains (common knowledge) are regarded as content-related stopwords. Characteristically, these terms occur quite frequently and are equally distributed in documents of any document collection. Due to the generic usage, such terms do generally not add valuable information to standard texts (e.g., `use`, `accordingly`, `certain`, `problem`).

For instance, consider three different texts: one about `social and cultural effects`, another one about `effects of protuberances on the climate`, and a third one about `effects of a new indexing method`. All texts deal about certain effects. However, only few users are interested in information about general effects of any kind. Thus, the term `effect` can be considered as a content-related stopword. Content-related stopwords are classified according to the following categories:

Nouns (N) are words which refer to entities or groups of entities (persons, places, names, concepts, etc.). In many cases they bear the most relevant information of sentences, populating the verb argument structure. For the purpose of extending the stopword list those nouns which carry hardly any information without considering their context are picked. Thus, they refer to proper and common nouns that can be regarded as common knowledge (e.g., `abstract`, `field`, `example`, `conclusion`).

Verbs (V) typically encode events, actions, or processes. In a syntactical sense they constitute predicates in clauses, establishing a verb argument structure. In many theories they function as heads of sentences [72]. For the sake of this purpose verbs that do not transport much content (e.g., `use`, `apply`, `seem`, `go`) are selected.

Adjectives (ADJ) are words which normally modify nouns by specifying properties or other qualities. In a syntactical sense they function as attributes or parts of predicates (e.g., `low`, `high`, `normal`, `fast`).

Adverbs (ADV) are words which modify verbs or adjectives. Examples of stopwords in this category are `quickly`, `accordingly`, `efficiently`, etc.

By manually assigning the remaining q_2-ranked terms (excluding functional stopwords) to these categories, a list of 271 content-related stopwords is derived (see Table 4.13). The same word with different meanings is assigned to all possible categories (i.e., `run` [N,V], `study` [N,V], `potential` [N,ADJ], `focus` [N,V]).

The final list covers 8,6% (8.025.990) of all terms in the document collection. The union of the content-related and the functional stopwords covers 44,9% of all terms in the whole collection. The distribution of content-related stopword categories is given in Figure 4.15.

One way to extract content-related stopwords automatically is to use a number of document collections from different domains. On these corpora, the top ranked terms are extracted as described and functional stopwords are removed. Consequently, the intersection of the resulting lists (one list for each corpus) defines the domain-independent and content-related stopwords.

Table 4.13: List of content-related stopwords

Category	Terms
ADV (35)	along, already, also, always, any, back, currently, directly, due, easily, either, especially, even, far, finally, first, however, less, likely, more, most, much, now, often, once, only, over, rather, simply, still, together, usually, very, well, yet
ADJ (61)	additional, appropriate, available, basic, best, better, certain, common, complete, current, different, difficult, due, easy, enough, entire, full, general, good, high, higher, important, independent, initial, large, larger, last, later, least, little, long, low, lower, main, major, necessary, new, next, original, other, own, particular, possible, potential, previous, real, recent, right, same, several, significant, similar, simple, single, small, smaller, special, specific, standard, total, useful
N (74)	ability, abstract, addition, address, advantage, amount, area, areas, basis, case, cases, change, changes, cost, curricula, degree, details, difference, effect, end, example, fact, field, focus, form, future, goal, group, hand, help, individual, interest, interests, issue, issues, key, level, means, need, needs, order, others, paper, part, place, point, potential, problem, problems, project, range, result, results, section, set, sets, size, solution, space, step, study, support, terms, time, times, type, types, use, view, vitae, way, ways, work, years
V (100)	able, according, achieve, address, allow, allows, applied, apply, associated, assume, based, called, change, compared, consider, considered, consists, contains, corresponding, create, depends, describe, described, determine, effect, end, existing, find, focus, following, follows, form, found, get, give, given, gives, group, hand, help, improve, include, includes, including, increase, increasing, involved, issue, issues, know, known, let, like, limited, made, makes, making, need, needed, needs, obtain, obtained, order, part, place, point, present, presented, produce, proposed, provide, provided, provides, providing, received, reduce, related, require, required, requires, result, resulting, see, set, show, shown, shows, step, study, support, take, takes, type, use, used, uses, using, work, working, works

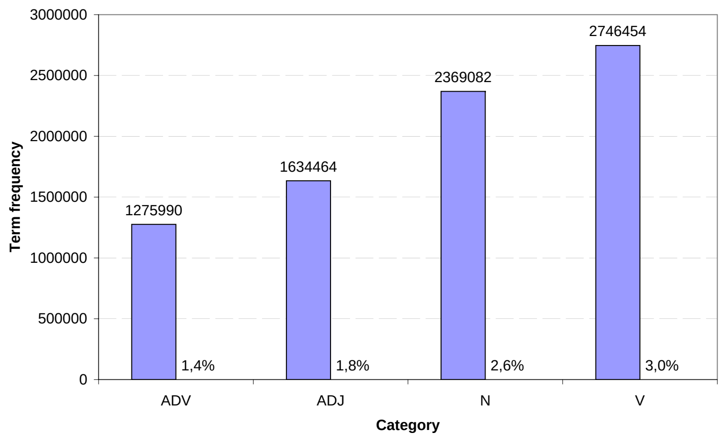


Figure 4.15: Frequency distribution of linguistic categories in content-related stopwords

4.6.5 Identification of Domain-Specific Stopwords

These stopwords are defined as terms that can be taken for granted in a given domain. In this sense they do not add information to texts in a given specific domain. This hypothesis is based on the assumption that domain experts normally presuppose a finer granulated domain vocabulary decoding their knowledge. Domain-specific stopwords are classified according to the same categories as the content-related stopwords (e.g., in the domain of computer science):

Nouns (N) like `computer`, `memory`, `model`, etc.

Verbs (V) like `perform`, `calculate`, `run`, etc.

Adjectives (ADJ) like `efficient`, `supervised`, etc.

Adverbs (ADV) like `automatically`, `effectively`, etc.

Table 4.14 shows the manually extracted domain-specific stopwords. This work relies on the INEX document collection derived from computer science literature. Hence, its application domain is computer science and information technology.

Table 4.14: List of domain-specific stopwords

Category	Terms
ADV (0)	
ADJ (6)	complex, effective, efficient, local, national, technical
N (58)	access, algorithm, algorithms, analysis, application, applications, approach, approaches, architecture, complexity, computer, control, data, department, design, development, environment, features, function, functions, hardware, implementation, information, input, institute, knowledge, management, member, method, methods, model, models, network, number, operations, performance, process, professor, program, requirements, research, science, software, structure, system, systems, technology, trans, university, user, users, components, technique, techniques, tools, value, values, version
V (27)	computing, develop, developed, developing, distributed, engineering, generate, generated, implemented, perform, performed, processing, represent, represents, sets, define, defined, designed, run, supported

The final domain-specific stopword list comprises 91 terms covering 4,7% (4.379.991) of all terms in the document collection. Merged with the functional and content-related stopwords, the complete list (445 unique stopwords) covers about 49,6% of all terms in the corpus. Figure 4.16 gives an overview of the distribution of the domain-specific stopword categories.

4.6.6 Extending the Stopword List

Providing additional knowledge about the language, the stopword list created by the steps (1) to (3) can be further extended. As an example additional terms that are (semi-)automatically identified through morphological rules are included.

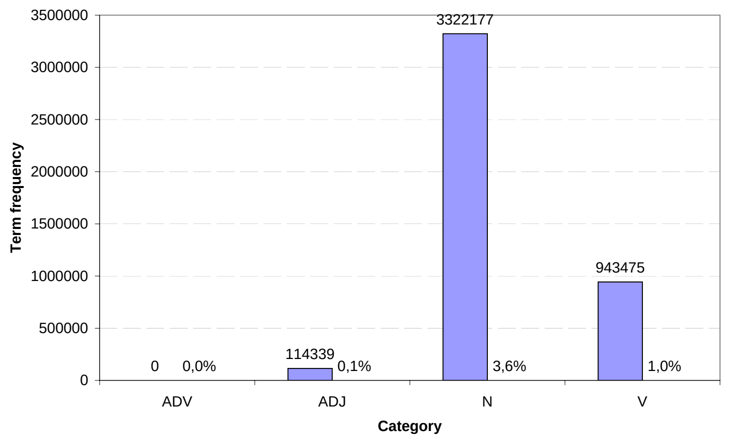


Figure 4.16: Frequency distribution of linguistic categories in domain-specific stopwords

In further experiments conducted, adverbs (`-ly`) and verbs (`-ing`, `-ed`) are selected from the INEX corpus because of their simple identification via suffix matching. Therefore, the q_2-ranked terms of the corpus are investigated. The list included 344 terms ending with `-ly` (adverb candidates), 839 terms ending with `-ing` (verb candidates), and 795 terms ending with `-ed` (verb candidates). Each of the three lists is cut after 50 entries and the terms are manually checked for their correct word category. Only five non-adverbs (`only`, `apply`, `early`, `July`, `Only`), two non-present-tense verbs (`during`, `During`), and two non-past-tense verbs (`need`, `speed`) had to be removed. The remaining terms are assigned to the set of functional F, content-related CR, or domain-specific CS stopwords according to their category. The added stopwords are given in Table 4.15.

The resulting stoplist contains 530 unique terms and covers 51,0% of all words in the document collection. The inclusion of the additional stopwords increased the coverage of all (unique) terms by 1,4%, but increased the size of the list by nearly 20% (+85 terms). This clearly shows the effect that including more terms in the stoplist at a certain size only increases the number of terms matched in the corpus marginally. The final distribution of stopword categories is shown in Figure 4.17, where a preceding F stands for functional, CR for content-related, and DS for domain-specific stopwords.

4.6.7 Coverage of the Generated Stopword List

Compared to other stopword lists the generated list seems to be well sized. As the stopwords in this work are generated from the INEX computer science corpus, a number of common terms such as `did`, `once`, or `upon` occurred quite infrequent and are not included. This problem can be handled by applying the same procedure of stopword generation to other corpora of other domains. By merging the results (besides the domain-specific stopwords), a more

Table 4.15: List of additional stopwords

Suffix	Type	Terms
ADV `-ly`	*CR* (42)	`actually, approximately, clearly, closely, completely, currently, directly, easily, especially, exactly, explicitly, extremely, Finally, frequently, fully, generally, highly, immediately, increasingly, independently, likely, necessarily, particularly, possibly, previously, primarily, probably, quickly, recently, relatively, respectively, significantly, Similarly, simply, simultaneously, slightly, specifically, successfully, typically, Unfortunately, usually, widely`
ADV `-ly`	*DS* (3)	`automatically, effectively, efficiently`
AUX `-ing`	*F* (2)	`having, doing`
V `-ing`	*CR* (32)	`according, adding, allowing, applying, beginning, being, building, changing, considering, containing, corresponding, creating, depending, existing, finding, following, including, increasing, interesting, leading, making, providing, reducing, remaining, resulting, starting, taking, underlying, understanding, using, Using, working`
V `-ing`	*DS* (14)	`computing, Computing, developing, engineering, Engineering, modeling, operating, performing, processing, Processing, programming, representing, running, testing`
V `-ed`	*CR* (36)	`achieved, added, applied, associated, based, Based, called, compared, considered, created, derived, described, detailed, determined, discussed, expected, fixed, improved, included, increased, introduced, involved, limited, needed, obtained, organized, presented, proposed, provided, published, received, reduced, related, required, selected, used`
V `-ed`	*DS* (12)	`developed, defined, designed, distributed, implemented, supported, performed, generated, represented, extended, integrated, specified`

complete stopword list could be achieved. A final intersection of all terms might be regarded as real-world 'common' stopwords.

An evaluation of the generated stopword list is performed by a coverage test comparing several other stopword lists. Therefore, nine freely available stoplists are compared to it. The stopwords presented by Fox in [76] (Fox), three standard stoplists (SL1, SL2, SL3, unfortunately without sources), the assumed Google stopwords[3] (Google), the stopwords presented by Manning in [173] (Manning), the BioPD stopwords[4] from the freeWAIS-sf[5] (BioPD), the Glasgow stopwords from The Information Retrieval Group[6] (Glasgow), and the CLEF stopwords from Smart[7] (CLEF Smart) were selected.

As said before, a more complete stopword list could be achieved by merging the stopwords generated from other corpora. Because of limited time, this step is not conducted in this work. Instead, the nine stopword lists mentioned are analyzed in detail and valid stopwords are

[3]http://www.ranks.nl/tools/stopwords.html (11.03.2008)
[4]http://www-fog.bio.unipd.it/waishelp/stoplist.html (11.03.2008)
[5]http://www.is.informatik.uni-duisburg.de/projects/freeWAIS-sf/STOPWORDS (11.03.2008)
[6]http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words (11.03.2008)
[7]http://www.unine.ch/info/clef/englishST.txt (11.03.2008)

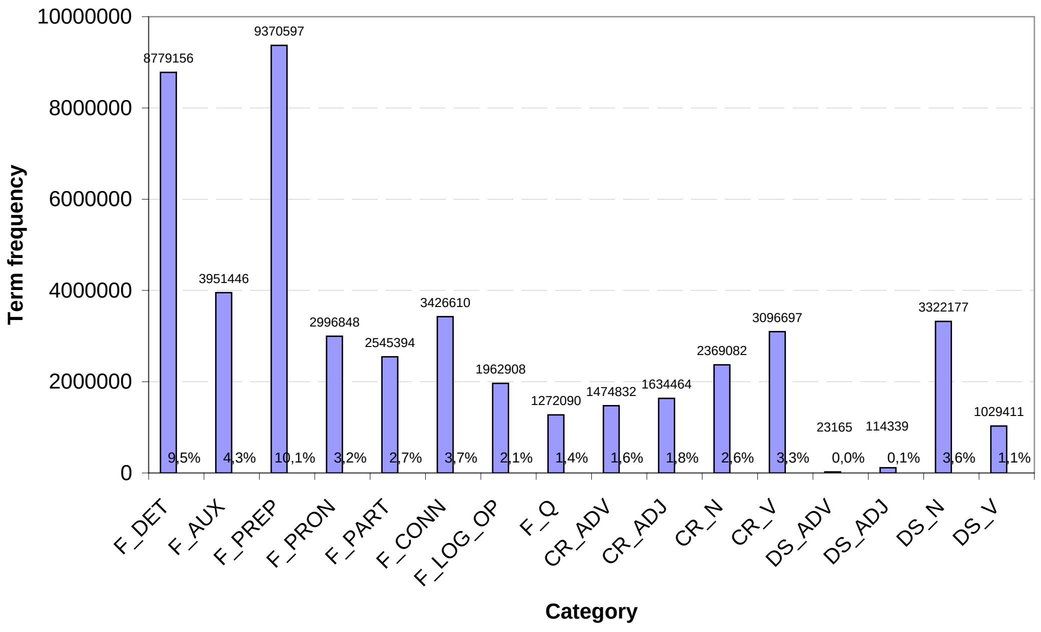


Figure 4.17: Frequency distribution of all linguistic stopword categories

extracted. According to their linguistic category, these stopwords are assigned to either the functional *F* or the content-related *C* stopword layer. This is explained by the fact that all stoplists are meant to be 'common sense'. Table 4.16 summarizes the additional stopwords added from the external stopword lists. The final generated stopword list contains 804 unique terms and is attached in Section A.1 in the Appendix.

Figure 4.18 shows the overlap of the generated list and other stopword lists. In this figure stopwords are sorted in alphabetical order, where the number at the top of each column #*SW* is the number of contained stopwords, and the number in the column headers shows the actual size of the list. One has to note that most stopword lists comprise all single letters and terms with apostrophes, hyphens, and other special characters. In the experiments such terms have been excluded by utilizing the token types assigned through Extended Tokenization. In order to achieve more accurate comparison results, stopwords of other lists that contain apostrophes (e.g., aren't, it's, they've) are removed. The stopword coverage of the final stoplist is above 77% for all lists: Fox (77,4%), SL1 (77,0%), SL2 (85,6%), SL3 (80,5%), Google (83,3%), Manning (98,2%), BioPD (88,0%), Glasgow (90,0%), and CLEF Smart (80,6%).

After stopwords are filtered the remaining set of index terms is undergoing a stemming process, described in the next subsection.

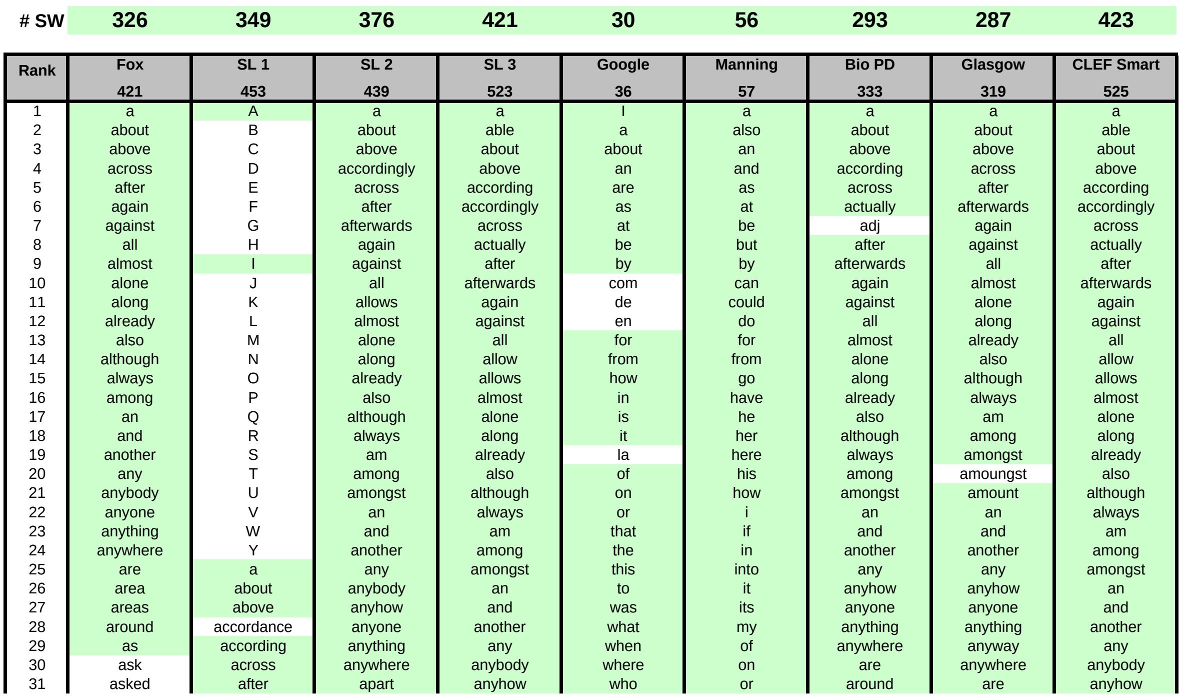
Table 4.16: Additional stopwords of external stopword lists

Category	Terms
F_AUX (6)	am, became, becoming, did, ought, shall
F_PREP (17)	against, amongst, aside, beforehand, behind, below, beside, beyond, down, near, off, onto, outside, thru, toward, towards, unto, upon
F_PRON (49)	anybody, anyhow, anyone, anything, anyway, anywhere, elsewhere, everybody, everyone, everything, everywhere, her, hers, herself, him, himself, i, me, mine, my, myself, nobody, none, noone, nowhere, ours, ourselves, she, somebody, somehow, someone, something, sometime, somewhat, somewhere, theirs, themselves, whence, whither, whoever, whom, whomever, why, you, your, yours, yourself, yourselves
F_PART (5)	almost, amongst, down, off, quite, yes
F_CONN (17)	furthermore, hence, howbeit, insofar, neither, nonetheless, nor, thence, though, unless, whenever, whereafter, whereas, whereby, wherein, whereupon, wherever
F_Q (25)	billion, eight, eighty, eleven, fifteen, fifth, fifty, forty, million, nine, ninety, secondly, sixty, seven, seventy, six, ten, third, thirty, thousand, trillion, twelve, twenty, twice, zero
C_ADV (71)	accordingly, actually, afterwards, again, apart, away, awfully, besides, certainly, clearly, consequently, definitely, differently, downwards, else, entirely, evenly, ever, formerly, hardly, hereafter, hereby, herein, hereupon, hither, hopefully, inasmuch, indeed, inward, lately, latterly, mainly, meanwhile, merely, moreover, mostly, namely, nearly, never, nevertheless, non, normally, nothing, nowhere, obviously, otherwise, overall, perhaps, preferably, presumably, really, reasonably, seriously, sometimes, soon, sure, thereafter, thereby, therein, thereof, thereto, thereupon, thorough, thoroughly, throughout, today, truly, unfortunately, unlikely, whatever
C_ADJ (41)	alone, big, brief, clear, early, except, former, forth, great, greater, greatest, highest, immediate, inner, kind, largely, latest, latter, longer, longest, main, mean, near, newer, newest, novel, old, older, oldest, regardless, sensible, serious, smallest, sorry, suitable, thick, thin, whole, young, younger, youngest
C_N (2)	ones, regards
C_V (52)	began, came, come, contain, differ, gave, gets, getting, go, goes, going, gone, got, gotten, interested, keep, keeps, kept, knew, knows, less, lets, liked, liked, likes, maybe, needing, okay, preferred, presenting, presents, put, puts, regarding, seeing, seem, seemed, seeming, seems, self, specify, specifying, sub, taken, took, unlike, want, wanted, wanting, wants, went, worked

4.7 Stemming

Stemming is one of the simplest and most successful methods of natural language processing for information retrieval [58]. Early work done by Lovins goes back to 1968 [165], already pointing out the benefits of applying stemming in information retrieval. Today, stemming has become a widely accepted processing step in nearly all retrieval systems.

Stemming itself is the process of reducing a full word form to its stem. Although quite similar to morphological processing, it is not linguistically motivated but has somewhat different goals [58]. It refers to a normalization of terms by removing affixes (prefixes and suffixes) [110, pp. 140], which is done mainly for two reasons:



# SW	326	349	376	421	30	56	293	287	423
Rank	Fox	SL 1	SL 2	SL 3	Google	Manning	Bio PD	Glasgow	CLEF Smart
	421	453	439	523	36	57	333	319	525
1	a	A	a	a	I	a	a	a	a
2	about	B	about	able	a	also	about	about	able
3	above	C	above	about	about	an	above	above	about
4	across	D	accordingly	above	an	and	according	across	above
5	after	E	across	according	are	as	across	after	according
6	again	F	after	accordingly	as	at	actually	afterwards	accordingly
7	against	G	afterwards	across	at	be	adj	again	across
8	all	H	again	actually	be	but	after	against	actually
9	almost	I	against	after	by	by	afterwards	all	after
10	alone	J	all	afterwards	com	can	again	almost	afterwards
11	along	K	allows	again	de	could	against	alone	again
12	already	L	almost	against	en	do	all	along	against
13	also	M	alone	all	for	for	almost	already	all
14	although	N	along	allow	from	from	alone	also	allow
15	always	O	already	allows	how	go	along	although	allows
16	among	P	also	almost	in	have	already	always	almost
17	an	Q	although	alone	is	he	also	am	alone
18	and	R	always	along	it	her	although	among	along
19	another	S	am	already	la	here	always	amongst	already
20	any	T	among	also	of	his	among	amoungst	also
21	anybody	U	amongst	although	on	how	amongst	amount	although
22	anyone	V	an	always	or	i	an	an	always
23	anything	W	and	am	that	if	and	and	am
24	anywhere	Y	another	among	the	in	another	another	among
25	are	a	any	amongst	this	into	any	any	amongst
26	area	about	anybody	an	to	it	anyhow	anyhow	an
27	areas	above	anyhow	and	was	its	anyone	anyone	and
28	around	accordance	anyone	another	what	my	anything	anything	another
29	as	according	anything	any	when	of	anywhere	anyway	any
30	ask	across	anywhere	anybody	where	on	are	anywhere	anybody
31	asked	after	apart	anyhow	who	or	around	are	anyhow

Figure 4.18: Final stopword coverage

- First, the vocabulary is kept as small as possible. For instance, the terms `hunt`, `hunts`, and `hunted` are reduced to the single form `hunt`.
- Second, the recall of search results is increased by retrieving identical stems of words having different spellings as complete words (e.g., another tempus, singular versus plural, comparative forms). Thus, the term `house` matches the terms `housing`, `housed`, `houses`, etc.

Besides these benefits, retrieval systems that apply stemming face some drawbacks [173, pp. 132–134]. Generally, stemmers like the Porter stemmer [203] reduce word forms by simply stripping their suffixes. This method works fine for most English words. However, such processing also leads to unwanted and even incorrect results. For instance, the rule deleting the word ending `-ly` reducing adverbs to adjectives but also reduces `July` to `Ju`. The reason for that is that the stem is computed without any deeper linguistic knowledge and analysis. In other languages with rich morphology (e.g., German) these stemmers perform very poor. Krovetz and Croft [159, 158] tried to tackle this problem using large dictionaries to ensure correct stems. However, complete lists are not likely to be available because of the maintenance costs. Another difficulty is that semantically different words (e.g., `gallery` and `gall`) are both stemmed to `gall`, leading to unwanted matches of terms in the query and in the documents. This is the reason why some web search engines do not apply stemming at all [21, pp. 168].

Generally, stemmers reduce terms to stems by accessing a list of suffixes that are stripped. The suffixes are sorted according to their length, where longer suffix matches are stripped before shorter ones. This ensures that the example rules in Equation 4.12 and Equation 4.13 reduce the term `stresses` to `stress` (and not to `stresse`).

$$sses \rightarrow ss \tag{4.12}$$

$$s \rightarrow \emptyset \tag{4.13}$$

Besides its advantages according to the vocabulary size, the reason for applying stemming is its performance and deterministic behavior, mostly implemented as finite state automata. Although it often reduces full forms very poorly, it does not greatly impact the retrieval result because both, the query and the content of documents are stemmed: the same error during indexing is done for the query, which still results in a match of both wrongly stemmed terms.

The stems serve as index terms for the textual representation. In a final step the frequencies of the stems are summed up in order to reflect the importance of each 'concept'. During retrieval, these frequencies are used to compute each stems' weight for comparison.

4.8 Summary

This chapter introduced the relevant aspects of transforming natural language text into a computable representation that is comparable to others. Therefore, the involved natural language processing steps are described and the processing chain is explained.

The steps include tokenization, tagging, stemming, and stopword filtering. The final processing result is a term frequency vector that reflects the stemmed terms and their frequency within the initial text. The generated representation is stored in the database accessible during later retrieval phases.

Chapter 5

The voyage of discovery is not in seeking new landscapes but in having new eyes.

Marcel Proust

Generation of Natural Language Resources Supporting Information Retrieval

Natural language resources consist of dictionaries and sets of language rules that improve the quality of text analysis, and thus, the quality of text representations. Incorporating them in retrieval systems increases both, computational performance and retrieval quality. This chapter presents statistical methods that extract natural language resources from a large set of documents automatically. Processing solely relies on the computationally light-weight tasks of Extended Tokenization (described in Section 4.4) and stopword filtering (described in Section 4.6). Resources generated include a set of single-token typing rules and a set of multi-token typing rules improving tokenization, a full form dictionary, an abbreviation dictionary, multi-term dictionaries for the identification of index phrases (composite nouns, named entities, and formulaic speech), and acronym dictionaries for detection and re-substitution of acronyms and full forms.

5.1 Introduction

Extended Tokenization, as discussed in Section 4.4, is applied in natural language processing mainly for index and query computation. Obviously, the quality of retrieval systems strongly relies on the quality of its index representation and on the way the query is analyzed and processed. Incorporating the generated resources in text analysis frameworks improves central linguistic tasks such as tagging, parsing, term extraction and filtering, named entity recognition, etc. The better the natural language analysis, the better the representation of textual contents. As a consequence, improvements of natural language text representation improves the quality of information retrieval tasks.

The performance of natural language processing tools depends on the underlying resources like dictionaries and knowledge bases (e.g., morphological and syntactical rules). Creating these resources is time and cost expensive, which often is the reason for trusting external sources that are freely available. Due to the high creation effort, these resources are generally tailored to certain domains and applications. Thus, different sources use different terminologies, data structures, and implicit or ambiguous semantics to describe the same concepts. Quality assessments of resources are carried out rarely. Only in few cases external resources can be applied as given, and often much efforts are needed to rework and adapt them.

In general, tokenizers preprocess texts for further analysis tasks. However, the output of a sophisticated tokenizer can also be applied to generate lists of abbreviations, acronyms, named entities, full word forms (dictionaries), and domain and application specific text patterns in an unsupervised way. Since tokenization is a light-weight process, large amounts of data can be processed efficiently. Without much effort, experts are able to filter well prepared lists and extract information in a convenient way. This section relies on the output of JavaTok – the Extended Tokenization prototype. Neither stemming nor stopword filtering, as described in the previous chapter, have been applied. The output of the tokenizer is exploited to serve information extraction and text mining tasks. Based on a large amount of example texts, domain- and application-specific resources are created in a cheap, fast, and simple way with minimal human intervention.

Figure 5.1 shows the three components that generate linguistic resources from large amounts of plain text. It is important to note that this process starts from scratch, not relying on any of the representations computed earlier. In a first step, JavaTok is applied to tokenize each text. Its output is passed to a Pattern Extractor that identifies and extracts predefined token sequences (patterns) of length b. For each pattern, the extraction process includes a left (pre-window of length a) and a right (post-window of length c) context of tokens. Identified patterns (b tokens) are sorted according to their frequency in descending order. Corresponding full windows of $a + b + c$ tokens (pre-window, pattern, post-window) are additionally stored and sorted the same way. In a final step a Rule Miner analyzes the extracted full windows of each pattern. Based on statistics, rules are proposed that identify the current pattern according to its context. Rules are sorted by their length (number of tokens addressed) and ranked according to the proportion of full window matches.

The machinery depicted in Figure 5.1 is used to extract language resources that improve information retrieval. Relying on corpus-based statistics (e.g., pattern frequency, document frequency), it is demonstrated how resources can be generated efficiently from large text corpora with minimal manual effort. The resources include:

Single-token rules Focusing on single-tokens of unknown type, rules that identify single-token types are incrementally added. These basic token-types enable subsequent extraction of more elaborated resources.

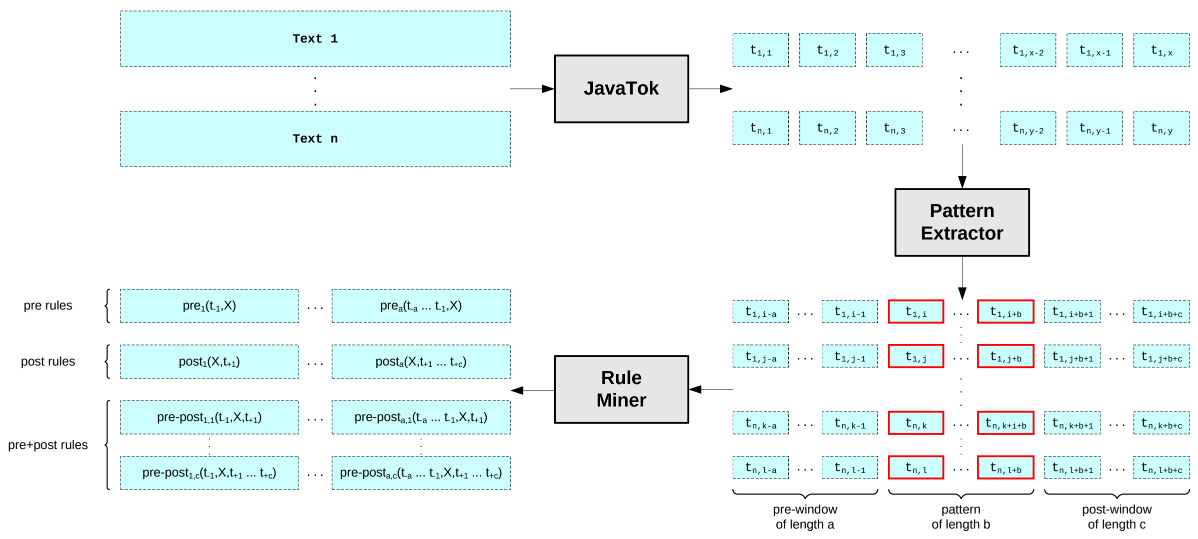

Text 1
Text n
JavaTok
$t_{1,1}$ $t_{1,2}$ $t_{1,3}$ … $t_{1,x-2}$ $t_{1,x-1}$ $t_{1,x}$
$t_{n,1}$ $t_{n,2}$ $t_{n,3}$ … $t_{n,y-2}$ $t_{n,y-1}$ $t_{n,y}$
Pattern Extractor
$t_{1,i-a}$ … $t_{1,i-1}$ $t_{1,i}$ … $t_{1,i+b}$ $t_{1,i+b+1}$ … $t_{1,i+b+c}$
$t_{1,j-a}$ … $t_{1,j-1}$ $t_{1,j}$ … $t_{1,j+b}$ $t_{1,j+b+1}$ … $t_{1,j+b+c}$
$t_{n,k-a}$ … $t_{n,k-1}$ $t_{n,k}$ … $t_{n,k+i+b}$ $t_{n,k+b+1}$ … $t_{n,k+b+c}$
$t_{n,l-a}$ … $t_{n,l-1}$ $t_{n,l}$ … $t_{n,l+b}$ $t_{n,l+b+1}$ … $t_{n,l+b+c}$
pre-window of length a
pattern of length b
post-window of length c
Rule Miner
pre rules
$pre_1(t_{-1},X)$ … $pre_a(t_{-a} ... t_{-1},X)$
post rules
$post_1(X,t_{+1})$ … $post_a(X,t_{+1} ... t_{+c})$
pre+post rules
$pre\text{-}post_{1,1}(t_{-1},X,t_{+1})$ … $pre\text{-}post_{a,1}(t_{-a} ... t_{-1},X,t_{+1})$
$pre\text{-}post_{1,c}(t_{-1},X,t_{+1} ... t_{+c})$ … $pre\text{-}post_{a,c}(t_{-a} ... t_{-1},X,t_{+1} ... t_{+c})$

Figure 5.1: Information extraction and text mining tasks

Multi-token rules Identification of complex multi-token rules based on single-tokens (strings and types) lead to better tokenization results. Especially tagging is improved by glueing conceptual related single-tokens to higher level multi-tokens.

Rule mining In order to enhance retrieval of sophisticated textual patterns, multi-token rules are identified automatically. Based on a bootstrapping mechanisms, context rules are learnt from a few known 'seed' patterns. The new rules can be used to extract further patterns that occur in the same context.

Single-term dictionaries Single-term dictionaries refer to traditional term lists, including full form lexicons, lists of abbreviations that improve the reliability of sentence border disambiguation, domain-dependent acronyms, URIs, phone numbers, and special formats (e.g., hyphenated forms such as `rule-based`, `state-of-the-art`). Extraction of specific word category dictionaries is done by focusing on token types applying regular expression matching of token strings (e.g., `-ly` ending adverbs, `-ion` and `-ings` ending nouns, `-ing` and `-ed` ending verbs).

Multi-term dictionaries Multi-term dictionaries consist of multi-tokens that serve as additional multi-term or phrase indices. These indices improve retrieval tasks by representing concepts where the order of words is significant (e.g., composite nouns, named entities, expanded acronyms).

The next section describes the general procedure of the extraction tasks. Subsequently, the different approaches generating natural language resources are presented.

5.2 Experimental Setting and Procedure

All experiments rely on English computer science texts of the INEX 2005 corpus described in Section 2.9.1. However, the proposed extraction methods can be applied to other languages and domains in the same manner. Achieved results are meant as guidelines, sketching ways to smartly extract linguistic resources on a surprisingly low processing level. The resources are generated and extended in an iterative process using JavaTok and a Pattern Extractor. Both tools are complemented by a Rule Miner that identifies statistically motivated rules based on the contexts of extracted patterns (see Figure 5.1).

In each iteration step the set of documents is tokenized using JavaTok. The output is passed to the Pattern Extractor which extracts token patterns and corresponding contexts. Patterns are defined as an arbitrary number of tokens that are specified by token strings and token types. Positive (e.g., only tokens of type `ABBR`) and negative (e.g., no token strings ending by `-ing`) pattern definitions are supported. Additionally, a list of known exceptions (negative token strings; e.g., `July` is not an adverb although ended by `-ly`) and a list of known strings (positive token strings; e.g., `wrt.` is a known abbreviation) are used to exclude previously identified patterns. If necessary, a left context (pre-window) and a right context (post-window) of arbitrary length can be included. The Pattern Extractor outputs the extracted patterns (and their contexts) according to their frequency in descending order. By investigating the top *n* outputs, patterns are efficiently identified and processed. One can either

1. assign the extracted pattern to a list of positive token strings, or
2. assign the extracted pattern to a list of negative token strings, or
3. set up a rule that identifies the pattern as a certain token type, or
4. set up a rule that ignores the pattern in the next iteration step, or
5. leave the pattern untouched.

Adding a pattern to the positive list (option 1) results in a dictionary of highly confident patterns. This dictionary is supported by positive rules that identify the pattern (option 3). The effect of a new rule can instantly be checked by extracting all tokens assigned to the new rules' token type. Similarly, options (2) and (4) exclude unwanted patterns. Each option (1)–(4) prevents the pattern from being extracted in subsequent iteration steps. Due to the iterative process, both, the lists (positive and negative token strings), and tokenization rules (positive and negative), grow quickly. Thus, the number of patterns investigated in the next iteration step is reduced efficiently. In addition to the manual identification of positive rules the Rule Miner can be applied to extract rule proposals. Statistical ranking of these rules supports the rule identification process.

5.3 Definition of Basic Token Types for Single-Tokens

Basic token types provide an abstraction of tokens that classify token strings into a set of interpretable categories. During Extended Tokenization these types are assigned to tokens in an initial step. Although token types are not interpreted by the tokenizer itself, subsequent processing, as described in this chapter, is supported by focusing on certain subsets of tokens.

Relying on the output of the Pattern Extractor, basic token types are identified manually. Therefore, single-tokens of unknown type (T_u) are investigated. Types are assigned by regular expression rules that are applied on token strings. According to the single-tokens extracted, three main categories, alphabetic ($ALPHA$), numeric ($NUMERIC$), and special ($ENTITY$) tokens are distinguished. Each category consists of further subcategories (see Figure 4.4). Within a short time of several hours a set of 72 basic token typing rules is constructed (see Tables 5.1–5.7). In the sequel the rules of each category are described.

5.3.1 Alphabetic Tokens ($ALPHA$)

Alphabetic tokens refer to 'proper words' of a text. Generally, search engines rely on these terms as index terms. Interestingly, a wide variety of different types occur frequently throughout the INEX documents. For the sake of improving postprocessing tasks, 49 specialized identification rules are identified and organized in three subcategories. $COMMON$ alphabetic tokens refer to full form words (see Table 5.1). The alphabetic subcategory $ACRONYM$ marks acronyms of various forms (see Table 5.2). Specially formatted tokens still referring to words are typed as $SPECIAL$ alphabetic tokens (see Table 5.3). The number in the table headers counts the number of rules included.

Table 5.1: *COMMON* alphabetic token types (26)

Subcategory	Regular Expression	Example
LOWER	[a-z]+	`this, house, it`
UPPER	[A-Z][a-z]+	`Vienna, Bill, Table`
UPPER_NUMBER	[A-Z][a-z]+[0-9]+	`Win32`
ELECTRONICAL_OBJECT_1	[eE](-[A-Z]?|[A-Z])[a-z]+	`e-Commerce, e-mail, eBay`
ELECTRONICAL_OBJECT_2	i(-[A-Z]?|[A-Z])[a-z]+	`iPod, i-Pod, i-pod`
MIXED	([a-z]+[A-Z]+[a-zA-Z]*|[A-Z]+[a-z]+[A-Z]+[a-zA-Z]*)	`thiS, HoUsE, iT`
HYPHEN_1	[a-z]+-[a-z]+	`red-hat`
HYPHEN_2	[a-z]+-[a-z]+(-[a-z]+)+	`state-of-the-art`
HYPHEN_3	[a-z]+-[0-9]+	`mid-1990`
HYPHEN_4	[A-Z]+-[a-z]+	`NP-hard`
HYPHEN_5	[A-Z]+-[A-Z][a-z]+	`NP-Complete`
HYPHEN_6	[A-Z]-[A-Z]?[a-z]+	`B-Spline, A-test`
HYPHEN_7	[A-Z][a-z]+-[A-Z][a-z]+	`Master-Mind, New-York`
HYPHEN_8	[A-Z][a-z]+-[a-z]+	`Master-switch`
HYPHEN_9	[A-Z][a-z]+-[A-Z][a-z]+(-[A-Z][a-z]+)+	`Master-Editor-Chief`
HYPHEN_10	[A-Z]?[a-z]+-[A-Z]?[a-z]+(-[A-Z]?[a-z]+)+	`Editor-in-Advance`
HYPHEN_3_PLURAL	[a-z]+-[0-9]+s	`mid-1990s`
HYPHEN_PREFIX	[A-Z]?[a-z]+-	`Pre-, post-`
HYPHEN_POSTFIX	-[a-z]+	`-ness`
CONTRACTION_IS_HAS	([Tt]hat’s|[Tt]here’s|[Hh]e’s|[Ss]he’s|[Ii]t’s)	`It’s, That’s, he’s`
CONTRACTION_HAD_WOULD	([A-Z]?[a-z]+|I)’d	`he’d, They’d, We’d`
CONTRACTION_CLITICS	(I|[A-Z]?[a-z]+)’(ve|ll|re|m|t)	`I’m, They’ve, We’re`
APOSTROPHE_POSS_GEN_1	[a-z]+’s	`egg’s`
APOSTROPHE_POSS_GEN_2	[A-Z][a-z]+’s	`Million’s, Dude’s`
APOSTROPHE_POSS_GEN_3	[a-z]+-[0-9]+’s	`mid-1990’s`
APOSTROPHE_POSS_GEN_4	([a-z]+[A-Z]+[a-zA-Z]*|[A-Z]+[a-z]+[A-Z]+[a-zA-Z]*)’s	`tHaT’s, McDonald’s`

Table 5.2: *ACRONYM* alphabetic token types (15)

Subcategory	Regular Expression	Example
SIMPLE	[A-Z][A-Z]+	`USA, ACM, IEEE`
NUMBER_FIRST	[0-9]+[A-Z]+	`3D, 123US`
SLASH	[A-Z]+(/[A-Z]+)+	`IEEE/ACM`
AND	[A-Z]+&[A-Z]+	`AT&T, ABC&DEFG, A&PC`
SLASH_NUMBER	[A-Z]+/[0-9]+	`PS/2, RS/6000`
MIXED_NUMBER	[A-Z]+[0-9]+[A-Z]*	`P2P, J2EE`
PLURAL_1	[A-Z]+s	`PCs, CPUs, XMLs`
PLURAL_2	[A-Z]+/[A-Z]+s	`I/Os, DA/Ps`
PLURAL_3	[A-Z]+-[A-Z]+s	`ACM-IEEEs, SP-SSs`
NAME	[A-Z]\.-?([A-Z]\.)+	`E.G.T., M.-T., A.B.-C.D.`
HYPHEN_1	[A-Z]+-[A-Z]+	`ACM-IEEE, SP-SS`
HYPHEN_2	[A-Z]+-[A-Z]+(-[A-Z]+)+	`IF-THEN-ELSE`
HYPHEN_NUMBER	[A-Z]+-[0-9]+	`F-117, B-52, US-123`
APOSTROPHE_POSS_GEN	[A-Z]+’s	`USA’s, ACM’s`
MIXED(ALPHA_MIXED)	[A-Z][A-Z]+[a-z]+	`IPsec, ITtalks, KBytes`

Table 5.3: *SPECIAL* alphabetic token types (8)

Subcategory	Regular Expression	Example
APOSTROPHE_NAME	(O|Mc|De)'[A-Z][a-z]+	`O'Conner, Mc'Donalds`
APOSTROPHE_NAME_POSS_GEN	(O|Mc|De)'[A-Z][a-z]+'s	`O'Conner's, Mc'Donalds's`
APOSTROPHE_GENERAL	[A-Z]?[a-z]*'[A-Z]?[a-z]+	`Int'l`
ALTERNATIVE_1	[a-z]+(/[a-z]+)+	`input/output`
ALTERNATIVE_2	[A-Z][a-z]+(/[A-Z][a-z]+)+	`May/June`
ALTERNATIVE_3	[A-Z]?[a-z]+\.(/[A-Z]?[a-z]+)̇+	`Jan./Feb.`
SLASH_NAME	[A-Z][a-z]+/[0-9]+	`System/6000, Ansi/200`
UNDERLINE_COMPOUND	[A-Z]?[a-z]+(_[A-Z]?[a-z]+)+	`By_the_End`

5.3.2 Numeric Tokens (*NUMERIC*)

Numeric tokens identify numbers and number-like strings which generally are not included in the set of index terms. Three different subtypes of numeric tokens are defined: plain numbers (*PLAIN*, see Table 5.4), formats that contain numbers (*FORMAT*, see Table 5.5), and special number constructs (*SPECIAL*, see Table 5.6).

Table 5.4: *PLAIN* numeric token types (6)

Subcategory	Regular Expression	Example
SIMPLE	[0-9]+	`12345`
SIMPLE_SIGNED	[\+-][0-9]+	`+512, -18`
PERIOD	[0-9]+(\.[0-9]+)+	`123.45`
COMMA	[0-9]+(,[0-9]+)+	`123,45`
PLURAL	[0-9]+s	`1990s, 60s`
APOSTROPHE_POSS_GEN	[0-9]+'s	`1990's`

Table 5.5: *FORMAT* numeric token types (13)

Subcategory	Regular Expression	Example
PERIOD_CHAPTER	[0-9]+\.[0-9]+(\.[0-9]+)+	`3.2.1`
VERSION	[0-9]+([\.-][0-9]+)+[[A-Za-z]+]	`3.2.1-5alpha, 0.3.2-1, 0-4-3`
RANGE	[0-9]+–?[0-9]+	`25-28, 11-17`
TIME_SCORE_PROPORTION	[0-9]+(:[0-9]+)+	`12:45, 10:09:99`
SLASH_NUMBER	[0-9]+/[0-9]+	`3/4, 1/2, 99/100`
CARDINAL	[0-9]+(st|nd|rd|th)	`1st, 2nd, 15th`
PERCENT	[0-9]+%	`1%, 99%, 127%`
DOLLAR_1	[0-9]+$	`123$`
DOLLAR_2	\$[0-9]+	`$123`
DATE_1	(0?[1-9]|1[0-9]|2[0-9]|3[0-1])\.(0?[1-9]|1[0-2])\.[0-9]4	`12.12.1977, 1.1.1980`
DATE_2	[0-9]4\.(0?[1-9]|1[0-2])\.(0?[1-9]|1[0-9]|2[0-9]|3[0-1])	`1977.12.12, 1980.1.1`
DATE_3	(0?[1-9]|1[0-9]|2[0-9]|3[0-1])/(0?[1-9]|1[0-2])/[0-9]4	`12/12/1977, 1/1/1980`
DATE_4	[0-9]4/(0?[1-9]|1[0-2])/(0?[1-9]|1[0-9]|2[0-9]|3[0-1])	`1977/12/12, 1980/1/1`

Table 5.6: *SPECIAL* numeric token types (1)

Subcategory	Regular Expression	Example
MEASSURE	[0-9]+-[a-z]+	`128-bit, 64-inch, 84-kg`

5.3.3 Entity Tokens (*ENTITY*)

Special token types describe application and domain specific token strings. Since the INEX corpus consists of computer science texts, a subtype *WWW* for web-related tokens is defined (see Table 5.7).

Table 5.7: *WWW* Entity Token Types (3)

Subcategory	Regular Expression	Example
EMAIL	[0-9a-zA-Z]([-\\.0-9a-zA-Z])*@([0-9a-zA-Z][-a-zA-Z]*[0-9a-zA-Z]\\.)+[a-zA-Z]2,9	`abc.def@xml.org`
URL	(http|ftp|https)://([0-9a-zA-Z])+(\\.([0-9a-zA-Z])+)+(/([0-9a-zA-Z])*)+	`http://hekkas.com`
FILE_EXT	\\.[a-zA-Z][a-zA-Z][a-zA-Z]	`.txt, .GIF, .Pdf`

5.4 Definition of Complex Token Types for Multi-Tokens

Sequences of token strings and basic token types can be further exploited to define more elaborated multi-token types. Therefore, rules that combine, split, and retype single-tokens to create multi-tokens are applied. Promising candidates consisting of multiple tokens include phone numbers (see Table 4.4), named entities, and special writing formats (e.g., credit card numbers, product identifiers). Note that some patterns (e.g., phone numbers like `+43 (0)463 2700-3511`) are identified context independently, whereas others must include their textual surrounding (left and/or right contexts). This section focuses on context independent pattern identification.

An example of a multi-token rule is given in Listing 5.1. Each rule consists of an ID (line 2), a flag that specifies whether delimiter tokens have to be considered in the token input sequence (line 3), an input sequence of tokens defined through string and type specifications using regular expressions (lines 4–17), an output sequence of (re)typed tokens eventually creating multi-tokens, and an optional description. A string manipulation language was developed to address strings and types of input tokens. For instance, $S0$ (resp. $T0$) refers to the token string S (resp. type T) of the first input token 0. Additional commands allow the extraction and recombination of substrings (e.g., `cuttail("aabcc",2)` results in the string `aab`, `cuthead("aabcc",3)` results in the string `cc`).

Rule examples identifying multi-tokens in the INEX corpus:

R-1 Rule that corrects wrong end of sentence tokens (see Listing 5.1);
e.g., `int` + *END_OF_SENTENCE* + `companies` → `int. companies`

Listing 5.1: Example rule correcting sentence ends

```
<RULE>
    <ID>R-EOS-001</ID>
    <IGNORE_DELIMS>true</IGNORE_DELIMS>
    <IN>
        <TOKEN>
            <STRING>.*</STRING>
            <TYPE>^[(TT_PM)]</TYPE>
        </TOKEN>
        <TOKEN>
            <STRING>\.</STRING>
            <TYPE>TT_EOS</TYPE>
        </TOKEN>
        <TOKEN>
            <STRING>.*</STRING>
            <TYPE>ALPHA_COMMON_LOWER</TYPE>
        </TOKEN>
    </IN>
    <OUT>
        <TOKEN>
            <STRING>$S0$.</STRING>
            <TYPE>ABBREV_CAND</TYPE>
        </TOKEN>
        <TOKEN>
            <STRING>$S2$</STRING>
            <TYPE>$T2$</TYPE>
        </TOKEN>
    </OUT>
    <DESCRIPTION>
        Rule that re-concatenates a period to the previous token if it is followed
        by a non-capitalized alphabetic token. This rule corrects wrong sentence ends
        and marks abbreviation candidates, eventually handled by subsequent rules.
    </DESCRIPTION>
</RULE>
```

R-2 Rule that re-concatenates hyphenated words at line breaks;
e.g., `auto-` + *END_OF_LINE* + `matically` → `automatically`

R-3 Rule that splits clitics and expands it;
e.g., `isn’t` → `is` + `not`

R-4 Rule that identifies international phone numbers;
e.g., `+43` + `(0)463` + `2700` + `-` + `3504` → `+43 (0)463 2700-3504`

The rules described identify patterns directly. This means that a pattern, optionally with its context(s), is specified by an expert. This allows an identification of known patterns from a text. With respect to information retrieval, the set of multi-terms identified defines the term space for content representation. In X-DOSE, this procedure identifies multi-terms that represent the multi-term contents.

However, there is the possibility to identify unknown multi-terms indirectly by specifying the context surrounding them. In these rules the multi-term itself is replaced by a wildcard. Using the Pattern Extractor to extract these context patterns, texts are mined for additional multi-terms that occur in the same context. Thus, the term space can be extended automatically based on context analysis of few known patterns by a bootstrapping-like mechanism. For instance, a small number of country names (e.g., `Austria`, `Chile`) generates a set of rules that identify countries in certain contexts (e.g., `president of [country]`). In a second step these rules are applied to identify unknown country names in the same context. The next section describes a statistical approach identifying rules automatically.

5.5 Automatic Rule Extraction

The identification of complex domain-dependent and application-specific tokenization rules is hard and time consuming even for experts. Thus, approaches that solve this task automatically are of importance. For this purpose, a Rule Miner is proposed that computes statistically-motivated tokenization rules based on extracted patterns and contexts. Single-token patterns assumed to form multi-tokens (e.g., signed numbers like `+43`) are extracted with left and right contexts. The Rule Miner analyzes the contexts and generates rules that identify the specified pattern. It is important to note that the size of the context investigated has a major impact on the computational performance, since token strings, token types, and combinations of both are considered for rule generation. Strict matching of strings and types is considered only. However, more elaborated matching (e.g., same super token type *ALPHA*, regular expressions that match token strings ended by `-ly`) seems promising.

Two experiments generating multi-token rules illustrate the usage of the Rule Miner. In a first experiment, plain number contexts with left and right window sizes of one token are analyzed. Table 5.8 shows the top ranked outputs for left, right, and both contexts included. *S* resp. *T* stands for a match of the token string resp. token type. The last column contains the percentage of contexts that are matched by the rule (out of 1.652.285 extracted patterns).

As expected, the percentage of matching contexts drops quickly. Especially larger contexts (left and right context) occur less frequent than shorter ones (either left or right context). According to a threshold (i.e., 10%), only a small fraction of extracted rules has to be investigated. Based on the top four results presented in Table 5.8c, the rule [+ *NUM_PLAIN_SIMPLE* +] that matches citations (e.g., `[3]`, `[42]`) is identified. Note that the token type *TT_PM* for punctuation marks includes the characters [and]. The final list of rules that treat plain numbers contains:

Table 5.8: Top 10 context rules for plain numbers

(a) Left context

ID	Left context	%
1	$T = TT\_PM$	37,0%
2	$T = ALPHA\_COMMON\_LOWER$	21,2%
3	$S = [$	16,1%
4	$T = ALPHA\_COMMON\_UPPER$	14,3%
5	$S =,$	13,2%
6	$T = TT\_UNKNOWN$	10,2%
7	$T = ABBR$	9,9%
8	$S = Fig.$	5,4%
9	$S = ($	5,1%
10	$T = TT\_EOS$	3,6%

(b) Right context

ID	Right context	%
1	$T = TT\_PM$	53,5%
2	$S =,$	24,7%
3	$T = ALPHA\_COMMON\_LOWER$	24,4%
4	$S =]$	16,1%
5	$S =)$	9,4%
6	$T = TT\_EOS$	9,0%
7	$S = .$	8,3%
8	$T = TT\_UNKNOWN$	5,4%
9	$T = ALPHA\_COMMON\_UPPER$	4,2%
10	$S = and$	2,8%

(c) Both contexts

ID	Left context	Right context	%
1	$T = TT\_PM$	$T = TT\_PM$	31,7%
2	$S = [$	$T = TT\_PM$	16,0%
3	$T = TT\_PM$	$S =]$	15,9%
4	$S = [$	$S =]$	15,8%
5	$T = ALPHA\_COMMON\_LOWER$	$T = ALPHA\_COMMON\_LOWER$	10,9%
6	$T = TT\_PM$	$S =,$	10,1%
7	$S =,$	$T = TT\_PM$	10,1%
8	$S =,$	$S =,$	8,9%
9	$T = ALPHA\_COMMON\_UPPER$	$T = ALPHA\_COMMON\_LOWER$	5,6%
10	$T = ALPHA\_COMMON\_UPPER$	$T = TT\_PM$	5,5%

Cross references (Table 5.8a) Numbers preceded by `Fig.` (5,4%), `Figure` (2,5%), `Section` (1,6%), `Table` (1,4%), `Step` (0,5%), `Theorem` (0,4%), `Lemma` (0,3%);
e.g., `Section 4`, `Fig. 2`, `Step 5`

Month dates (Table 5.8a) Numbers preceded by `June` (0,6%), `May` (0,6%), `July` (0,4%), `Aug.` (0,4%), `Oct.` (0,4%), `Sept.` (0,4%), `Dec.` (0,4%), `Jan.` (0,4%), `Apr.` (0,4%), `Mar.` (0,4%), `Nov.` (0,4%), `Feb.` (0,3%);
e.g., `May 24`, `Aug. 3`

Quantities (Table 5.8b) Numbers followed by `percent` (1,4%), `years` (0,3%), `bits` (0,2%), `million` (0,1%), `times` (0,1%), `MHz` (0,1%), `Mbytes` (0,1%);
e.g., `43 percent`, `2 years`, `5 times`

Citations (Table 5.8c) Numbers embraced by square brackets (15,8%);
e.g., `[2]`, `[41]`, `[111]`

A second experiment focuses on the identification of international phone numbers. Preceding country code patterns (e.g., `++43`, `+39`) are exploited to identify possible candidates. A left context of one token and a right context of five tokens are included in the analysis. The top ten results are presented in Table 5.9. As before, S resp. T denotes token strings resp. token types, and the last column is the percentage of contexts that match the rule (out of 955 extracted patterns). For better readability the token type NUM stands for $NUMERIC\_PLAIN\_SIMPLE$ (plain numbers).

Table 5.9: Top 10 context rules for international country codes

(a) Left context

ID	Left context	%
1	$T = ALPHA\_COMMON\_LOWER$	80,6%
2	$S = fax$	37,5%
3	$S = phone$	25,7%
4	$T = TT\_PM$	15,1%
5	$S = voice$	12,4%
6	$S =:$	7,5%
7	$S = ($	6,3%
8	$T = TT\_EOS$	2,6%
9	$S =;$	2,6%
10	$S = to$	1,4%

(b) Right context

ID	Right context	%
1	$T = NUM$	73,3%
2	$T = NUM, T = NUM$	64,4%
3	$T = NUM, T = NUM, T = NUM$	36,5%
4	$T = NUM, T = NUM, T = TT\_EOS$	23,1%
5	$T = NUM, T = NUM, S =;$	21,5%
6	$T = NUM, T = NUM, T = NUM, T = TT\_EOS$	19,3%
7	$T = TT\_PM$	19,3%
8	$T = NUM, T = NUM, T = NUM, S =;$	19,0%
9	$T = NUM, T = NUM, T = NUM, T = NUM$	13,5%
10	$T = TT\_PM, T = NUM$	11,8%

(c) Both contexts

ID	Left context	Right context	%
1	$T = ALPHA\_COMMON\_LOWER$	$T = NUM$	66,2%
2	$T = ALPHA\_COMMON\_LOWER$	$T = NUM, T = NUM$	57,9%
3	$T = ALPHA\_COMMON\_LOWER$	$T = NUM, T = NUM, T = NUM$	32,0%
4	$S = fax$	$T = NUM$	31,9%
5	$S = fax$	$T = NUM, T = NUM$	27,9%
6	$S = phone$	$T = NUM$	25,3%
7	$S = phone$	$T = NUM, T = NUM$	23,9%
8	$T = ALPHA\_COMMON\_LOWER$	$T = NUM, T = NUM, T = TT\_EOS$	22,2%
9	$T = ALPHA\_COMMON\_LOWER$	$T = NUM, T = NUM, S =;$	20,5%
10	$T = ALPHA\_COMMON\_LOWER$	$T = NUM, T = NUM, T = NUM, T = TT\_EOS$	17,5%

Table 5.9a shows that – in combination – 75,8% of all country code occurrences are preceded by the words `phone`, `voice`, and `fax` (optionally capitalized). Over 70% of tokens following country codes are numbers (see Table 5.9b). The experiment confirms the expectation that phone numbers are easy to identify with high confidence. An investigation of the complete Rule Miner output confirms that phone numbers are the only meaningful multi-term pattern in the INEX corpus containing a country code.

Phone numbers (Table 5.9c) International phone number preceded by `fax` (37,5%), `phone` (25,7%), `voice` (12,4%), `telephone/fax` (0,1%), `Fax` (0,1%), `Phone` (0,1%) and followed by one (73,3%), two (64,4%), three (36,5%), four (13,5%), or five (4,8%) plain numbers;
e.g., `phone +43 1023 234 32345 12`

5.6 Generating Single-Term Dictionaries

Single-term dictionaries contain single-tokens that support text analysis and text representation. Previously assigned token types and additional specifications of token strings are exploited to generate dictionaries from the corpus automatically. This section demonstrates the procedure of creating a full form dictionary and an abbreviation dictionary. The same approach can be applied to generate other domain-specific dictionaries including

- email and internet addresses
- domain-relevant acronyms
- hyphenated word forms (e.g., `state-of-the-art`, `F-117A`)
- words containing certain affixes (e.g., noun dictionaries ended by `-ion`, terms started by `re-`)
- domain-specific single-token concepts (e.g., chemical formulas like H_2SO_4)
- etc.

Creating a Full Form Dictionary

A full form dictionary refers to the set of all 'regular' words that occur in the document collection. The Pattern Extractor selects only single-token strings that consist of alphabetic lowercase characters, possibly started by a capital letter. These tokens are identified by the token types *ALPHA_COMMON_LOWER* and *ALPHA_COMMON_UPPER*. Both, left and right contexts are omitted.

The INEX collection consists of 178.677.541 single-tokens including delimiter tokens. Following the above guidelines and ignoring delimiters, a total number of 92.855.161 single-tokens is examined. Extracted token strings are converted into lowercase with cumulated term frequencies. The final list contains 171.580 unique full forms (out of 205.106 upper and lower case tokens). In this work the full form dictionary is applied for stopword extraction (described in Section 4.6, first four columns of Table 4.12) and abbreviation detection (described in the next section).

Creating an Abbreviation Dictionary

Similar to the generation of a full form dictionary, a dictionary of common abbreviations is extracted from the corpus. JavaTok skips punctuation mark splitting (see Figure 4.7) and the Pattern Extractor picks token strings ended by periods only. In order to check the correctness of the extraction, a left and right context of three words is included. Preliminary tests have shown that this window size is large enough to decide whether a pattern is an abbreviation or not. Extracted results are ranked according to their frequency in descending order.

An investigation of the extracted patterns clearly shows that a simple, rule-based abbreviation detection approach is insufficient. Table 5.10 shows incorrect abbreviations identified among the top 100-ranked results. The examples are sorted according to their frequency in descending order.

Table 5.10: Examples of incorrect abbreviation patterns

```
Urbana-Champaign., MHz., PCs., ’96., ’95., ’97., ’94., 3D., PEs., ’93., ’98., NP-complete,.
e-commerce., ’99., e-mail., ICs., FPGAs., trade-offs., fault-free., ’92., flip-flops., QoS.,
time-consuming., NP-hard., ’91., deadlock-free., CPUs., NP-Completeness., APIs., Jan.-Mar.,
Wisconsin-Madison., 1998;., LANs., run-time., URLs., cost-effective., real-time., trade-off.,
1,000., July-Sept., A.1., ’90., Mar.-Apr., PDAs., Jan.-Feb., 1999;., don’t., GHz., 2D., dB.,
MultiMedia., ?1., ’s., A.2., IPv6.
```

The generation of an abbreviation dictionary is iterative and demands human intervention for selection purpose only. The idea is based on the extraction of candidate patterns defined by token strings ended by a single period. According to Grefenstette and Tapanainen [107], a large number of non-abbreviations is excluded by using the terms of the corpus itself as a filter. Since words ended by periods possibly mark ends of sentences, a comparison to all words occurring within sentences – the full form dictionary created in the previous section – preserves an extraction.

The process starts with two empty list of abbreviations (positives) and exceptions (negatives), and an empty set of positive identification and negative exclusion rules. By skimming the top-ranked entries and their contexts, both lists and rule sets grow quickly. Applying these lists and rules in subsequent iterations, the number of correct abbreviations among the top-ranked results decreases rapidly. The procedure stops if no meaningful strings or rules are identified.

In the experiments the top $n = 100$ extracted entries of the INEX corpus are investigated. After four iterations 322 corpus-specific abbreviations (see Table 5.11) are generated efficiently. Abbreviations in the table are sorted alphabetically. The list of exceptions includes the terms listed in Table 5.10. Positive rules comprise patterns that contain periods within the string (e.g., `w.r.t.`, `i.e.`, `Ph.D.`, `Comp.Sci.`). Interestingly, capital letters separated and ended by periods (e.g., `A.`, `E.R.`, `C.P.R.`) turned out to be worthless abbreviation patterns because these decode the first name(s) of persons. In order to avoid this pattern a negative exclusion rule $([A - Z]\backslash.)+$ is added.

The extracted abbreviations are integrated in JavaTok for enhancing the sentence end disambiguation. To cope with the difficulty of incomplete lists, a combined approach of dictionary lookup and rule-based identification is proposed. A domain- and application-specific set of abbreviations is kept in a dictionary. If an abbreviation candidate is not found in the dictionary, the following regular expression rules for identification are applied:

- Abbreviations that are prefixes of full forms: e.g., `avail.`, `approx.`, `Technol.`

Table 5.11: Extracted abbreviations

```
Acad., Adv., Akad., Amer., Appl., Applic., Apr., Apt., Assn., Assoc., Aug., Automat., Av.,
Ave., B.Eng., B.Sc., B.Tech., BTech., Biol., Biomed., Bld., Blvd., Bv., Calc., Calif., Capt.,
Cdr., Chem., Cir., Co., Col., Comm., Comp., Comp.Sci., Comput., Conf., Conn., Coop., Corp.,
Crit., Ct., Ctr., D.Sc., Dec., Depart., Dept., Depts., Devel., Dipl., Dipl.-Inform.,
Dipl.-Ing., Dipl.-Math., Dipl.Ing., Discr., Dist., Div., Dr-Ing., Dr., Dr.-Ing., Dr.Eng.,
Dr.rer.nat., Drs., Elec., Eng., Engr., Eq., Eqn., Equip., Feb., Fed., Fig., Figs., Fil.,
Fla., Ft., G.v., GmbH., Govt., Hist., Img., Inc., Ind., Inf., Info., Ing., Inst., Instrum.,
Int’l., Int., Intell., Intl., Jan., Jr., Jt., Jul., Jun., Knowl., Lehrst., Lt., Ltd., M.Eng.,
M.Math., M.Phil., M.Sc., MC-Sym., MSc., Mag., Maj., Mar., Md., Mgmt., Mgt., Mr., Mrs., Ms.,
Mt., Multiconf., Nat’l., Natl’l., Nos., Nov., Numer., Oct., Oper., Orthop., Ph.D., PhD.,
Phys., Pre-Proc., Proc., Prof., Profs., Rd., Re-Eng., Rev., S.p.A., Sch., Sci., Sect., Sekr.,
Sen., Sept., Soc., Sr., St., Stn., Stud., Succ., Symp., Syst., Technol., Tel., Theor.,
Theoret., Univ., Vol., Vt., W.l.o.g., Work-Conf., a.a., a.e.w., a.k.a., a.m., a.s., a.u.,
aff., alg., algm., algms., anal., appl., appls., approx., archit., archits., arith., assoc.,
avail., b.o.d., biol., boul., c.f., c.m., calc., ccts., cf., circ., cit., combinat., commun.,
conf., constr., contrib., correl., cross-develop., d.o.f., defn., determ., diags., dimens.,
distr., distrib., dyn., e.V., e.g., edn., eds., elec., engng., environ., eq., estim., etc.,
evol., expt., expts., fl., fns., g.c.i., geom., gov., heterog., high-perform., high-resoln.,
hist., hyperb., i.c., i.e., i.i.d., inc., indust., instrum., integrat., intell., intens.,
internat., intr., intro., jns., l.h.s., lang., langs., lb., m.c., maint., mech., mfg., mgt.,
mil., mission-crit., modif., mol., n-dimens., n.b.c., n.d.d., navig., nonlin., nos., o.c.s.,
objs., op., optim., p., p.d.f., p.m., particle-syst., perform., pers., phenom., phys., pol.,
poss., pp., preconf., probab., prod., prog., propag., pt., punt., qualitat., quantitat.,
r.h.s., r.m.s., r.p.m., r.v., recogn., reconstr., refl., reliab., rer., s.d., s.t., sc.,
scatt., sci., semicond., sens., seqs., simul., simuls., single-proc., sq., std., stds.,
symp., syst., systs., t.o., techn., technol., technols., techs., topol., transl., transm.,
vol., volc., vs., w.h.p., w.l.o.g., w.r.t.
```

- Abbreviations containing uppercase letters: e.g., `GmbH.`, `BTech.`
- Abbreviations containing multiple periods: e.g., `w.r.t.`, `M.Math.`, `S.p.A.`
- Abbreviations shorter than three letters: e.g., `pp.`, `cf.`, `Av.`

If both identification methods are failing, the pattern is not treated as an abbreviation.

5.7 Generating Multi-Term Dictionaries via Concordances

One hypothesis of this work is that the identification of multi-terms is helpful not only for natural language processing but improves information retrieval related tasks. Experiments that confirm this hypothesis are conducted by Arazy and Woo [20]. Multi-terms, defined as a continuous sequence of words (adjacent tokens, n-grams), often belong together creating a semantic unit. Multi-terms can be classified into several categories: composite nouns (e.g., `information retrieval`), named entities with a minimum of two tokens (e.g., `Ford Motor Company`), formulaic speech (e.g., `catch the bus`), full forms of acronyms (e.g., `Central Intelligence Agency (CIA)`), etc.

This section shows how meaningful multi-terms can be extracted from the corpus automatically. The goal of this section is to extract meaningful multi-terms from the corpus automatically. The quality of the extracted results is improved by applying the elaborated stopword lists (described in Section 4.6) for filtering of unlikely patterns. Depending on the extraction task the different stopword layers and linguistic stopword categories are exploited.

One important issue of multi-term detection is that their length is significant. For instance, the multi-terms `information retrieval system` and `information system` are not bound to each other. In contrast, the multi-terms `world wide` and `wide web` do not occur independently. They occur only in combination with each other in `world wide web`. Thus, longer multi-terms have to be identified before shorter ones.

In the sequel three approaches are described: A first approach, aiming at composite nouns, extracts token sequences consisting of small letter tokens only. Sequences containing at least one token that is a functional stopword or that is shorter than three letters are neglected. A second approach identifies named entities. Token sequences consisting of small letters and started by a single capital letter are extracted. As before, sequences containing stopwords or tokens smaller than three letters are excluded. A third approach focuses on the extraction of token sequences that are missed by the previous composite noun extraction and named entity extraction tasks. The goal is the identification of formulaic speech and other patterns that are still useful for information retrieval.

5.7.1 Pattern Extraction Suited for Composite Nouns

This section deals with the automatic generation of non-capitalized token sequences that fall into the categories of composite nouns and non-capitalized named entities longer than a single token. The Pattern Extractor extracts sequences of two, three, four, five, six, and seven adjacent tokens of type *ALPHA_COMMON_LOWER*. Preliminary tests indicate that longer sequences are inappropriate multi-terms because these do not constitute composite nouns but arbitrary parts of sentences. During extraction, left and right contexts are not considered. Extracted multi-terms are sorted according to their frequency in decreasing order.

Multi-terms that include stopwords are ignored by the Pattern Extractor. This is done for two reasons: First, the amount of extracted patterns increases exponentially compared to the number of documents analyzed. Thus, excluding multi-terms that contain stopwords reduces the data load tremendously. The remaining multi-terms become applicable for information retrieval. Second, neglecting stopwords in token sequences leads to multi-terms that are closer related to semantic concepts than to complete phrases. Since users tend to search for concepts instead of phrases including articles and pronouns[1], this approach better fits information retrieval related tasks.

[1]The daily updated google trends website provides an insight of popular user queries: http://google.com/trends (05.05.2008)

One has to note that processing of longer multi-terms requires a considerably larger amount of memory than processing of shorter ones. Although the number of longer sequences decreases (longer sequences are more often interrupted by stopwords or tokens of other types), the number of unique multi-terms is much higher. However, the procedure can be applied on non-overlapping subsets of documents independently. A final result for the whole collection is obtained by merging the results for the sub-collections.

Experiment I - Stopword Filtering

An initial experiment evaluates the reduction factor of multi-terms by applying stopword filtering. For illustration and comparison purpose, this experiment is restricted to 3.000 INEX documents. A higher number of documents exceeds the maximum of two gigabytes of Java-addressable working memory. For each multi-term length (two, three, four, five, six, and seven tokens) several experimental runs are conducted. One run without stopword filtering, one run for each linguistic stopword category, one run for each stopword layer (functional, content-related, and domain-specific), and one run for the whole set of stopwords.

Figure 5.2 presents the results of the experiment. The different lengths of the multi-terms are color-coded. The type of stopword filtering (see Section 4.6) is indicated by the labels on the x-axis. $NONE$ means no stopword filtering, FS (resp. CR and DS) are the functional (resp. content-related and domain-specific) stopword layers, FS_SUM (resp. CR_SUM and DS_SUM) stands for the union of all FS (resp. CR and DS) stopwords, and ALL is the union of all stopwords. For instance, the figure shows that the total number of 6.117.535 multi-terms of two tokens is reduced to 1.439.427 (23,5%) by functional stopwords, 4.580.504 (74,9%) by content-related stopwords, and 5.412.582 (88,5%) by domain-specific stopwords. The union of all stopwords reduces the number of multi-terms to 614.141 (10,0%). Especially determiners DET, prepositions $PREP$, and connectors $CONN$ turn out to be excellent linguistic filter categories.

In addition to the filtering effect of independent stopword categories and stopword layers, the cumulative filtering effect of stopwords is investigated in Figure 5.3. This effect is evaluated on the same but unique set of multi-terms extracted. In the figure, the linguistic stopwords categories are successively applied for filtering from left (no filtering) to right (complete filtering using all stopwords).

Obviously, the number of shorter multi-terms is higher than the number of larger multi-terms. Consider a text consisting of four tokens: This token sequence contains at most three multi-terms of length two, two multi-terms of length three, or one multi-term of length four. Further constraints on the tokens (e.g., a certain token type as $ALPHA\_COMMON\_LOWER$) emphasize this effect. A different effect occurs by applying the unique constraint. Because there are less different combinations of a smaller number of tokens than of a larger number

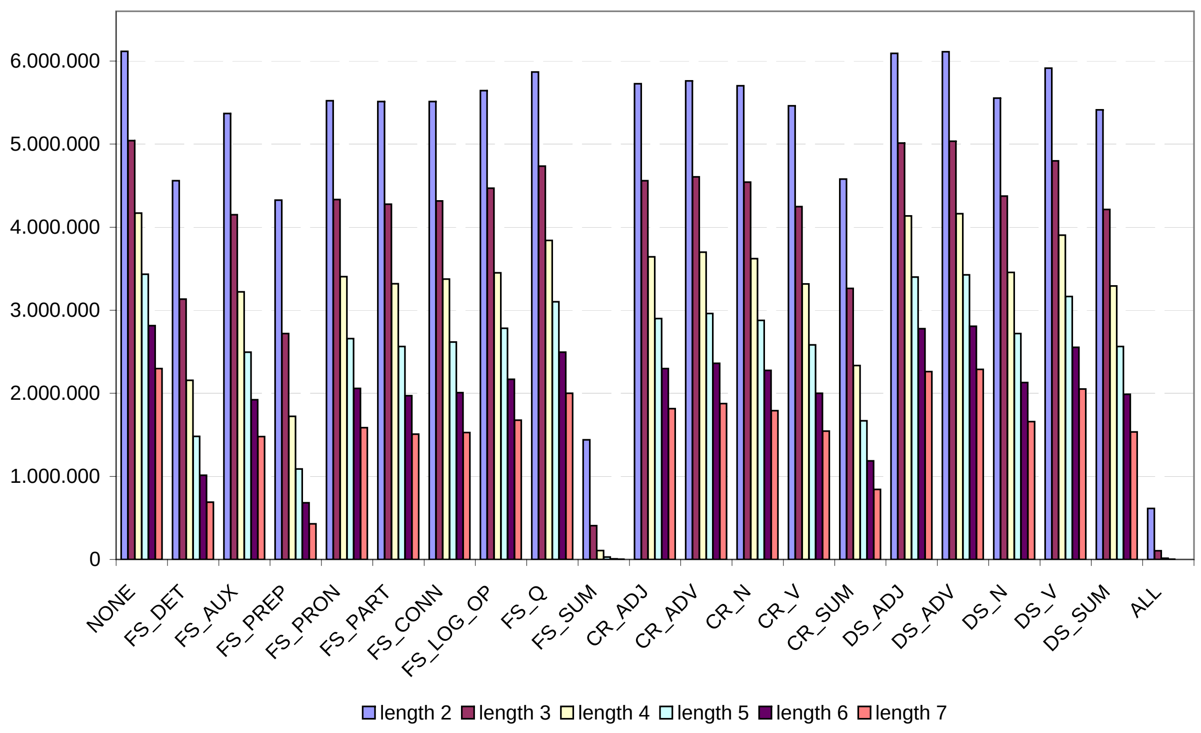


Figure 5.2: Using stopword filtering on multi-terms (color code indicates length of the multi-terms)

of tokens, shorter multi-terms are summarized to a much smaller set of unique multi-terms than longer ones. In the figure, these two effects become evident especially for the unfiltered and unique multi-terms. Multi-terms of length two occur least often (1.162.842 times), while multi-terms of length four occur most often (3.596.448 times).

As expected, longer multi-terms are filtered to a higher degree than shorter multi-terms. The overall reduction results in 340.951 (29,3%) multi-terms of length two, 92.776 (3,1%) multi-terms of length three, 15.425 (0,4%) multi-terms of length four, 2.535 (0,1%) multi-terms of length five, 506 (0,02%) multi-terms of length six, and 155 (0,01%) multi-terms of length seven.

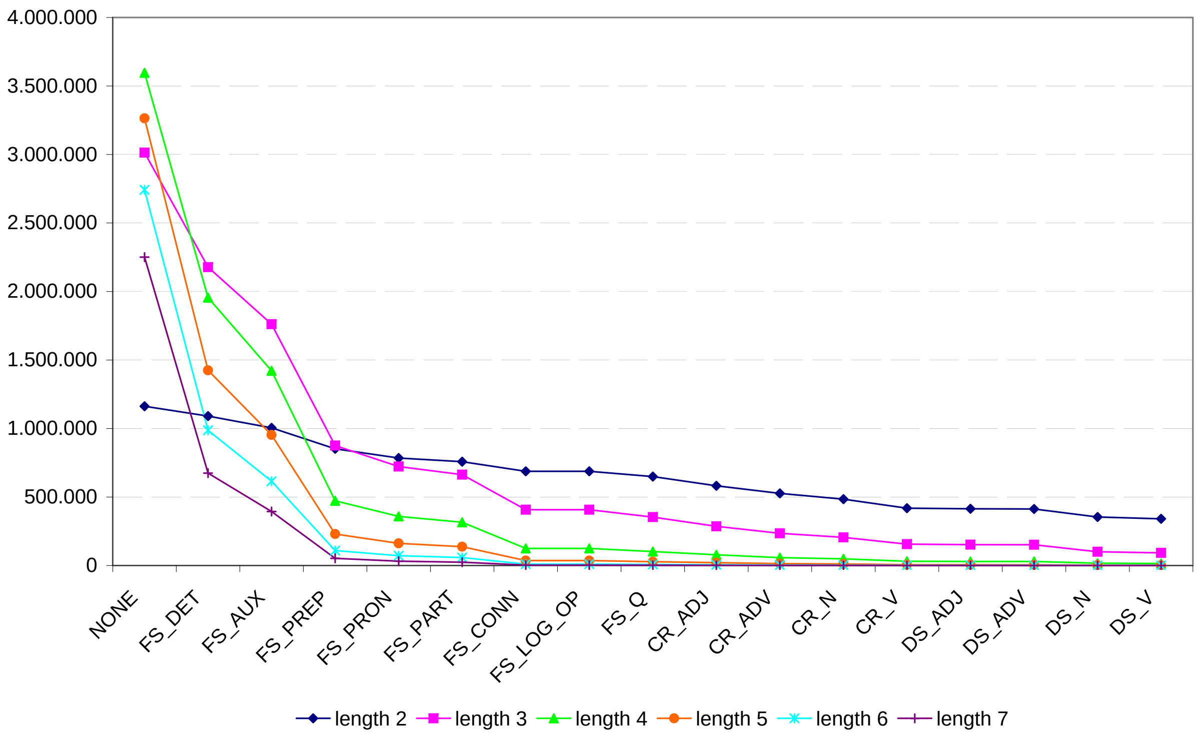


Figure 5.3: Cumulative filtering of linguistic stopword categories

Experiment II - Three Letter Constraint

An investigation of the results obtained by the first experiment shows that multi-terms frequently include terms consisting of one or two letters. Examining the full form dictionary generated in Section 5.6, a minimum length constraint of three characters per term is a good choice A second experiment repeats the previous experiment with the same settings, including the constraint of a minimal term length of three letters. Again, the 3.000 INEX documents of the first experiment are used, and all linguistic stopword categories and stopword layers are applied for filtering.

The results of the experiment are given in Figure 5.4. Compared to the first experiment, the three letter constraint reduced the number of unfiltered composite nouns to 3.827.298 (62,6%) of length two, 2.389.252 (47,4%) of length three, 1.500.294 (35,5%) of length four, 947.234 (27,4%) of length five, 595.594 (21,1%) of length six, and 372.582 (16,2%) of length seven. The figure shows that the number of multi-terms of length two is reduced to 37,3% by functional stopwords, 71,4% by content-related stopwords, and 85,6% by domain-specific stopwords, Applying all stopword layers combined, a final set of 606.169 (15,8%) length-two multi-terms remains. Compared to the results of the first experiment, this drop of reduction is explained by the pre-filtered one and two letter words, which are mostly included in the functional stopwords (e.g., `a`, `in`, `on`, `by`).

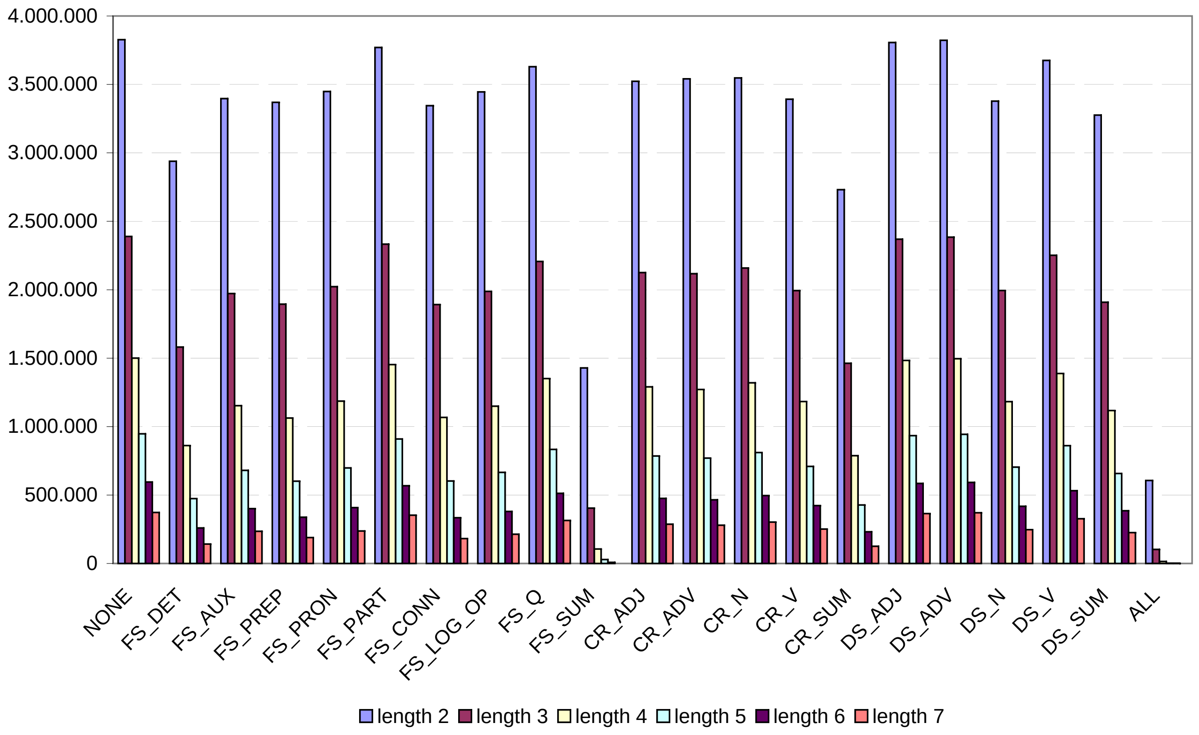


Figure 5.4: Using stopword filtering on multi-terms (minimum of three letters per term)

Figure 5.5 depicts the cumulative filter effect on the unique multi-terms extracted. In total, the three-letter filter reduces the number of cumulative filtered multi-terms to 337.746 (33,9%) of length two, 91.317 (5,3%) of length three, 14.426 (1,0%) of length four, 2.294 (0,3%) of length five, 395 (0,1%) of length six, and 78 (0,02%) of length seven. Compared to the first experiment, the three letter constraint reduces the number of stopword filtered multi-terms by 0,9% (two terms), 1,6% (three terms), 3,1% (four terms), 6,7% (five terms), 18,4% (six terms), and 44,3% (seven terms).

Experiment III - Final List

Based on the previous findings, a third experiment creates a complete dictionary of composite noun suited patterns. All documents in the INEX collection (16.819) are used. As before, different copies lengthes of two, three, four, five, six, and seven tokens are extracted. Each term must consist of at least three letters. No stopwords except content-related and domain-specific nouns (*CR_N* and *DS_N*) are allowed within extracted multi-terms. In the context

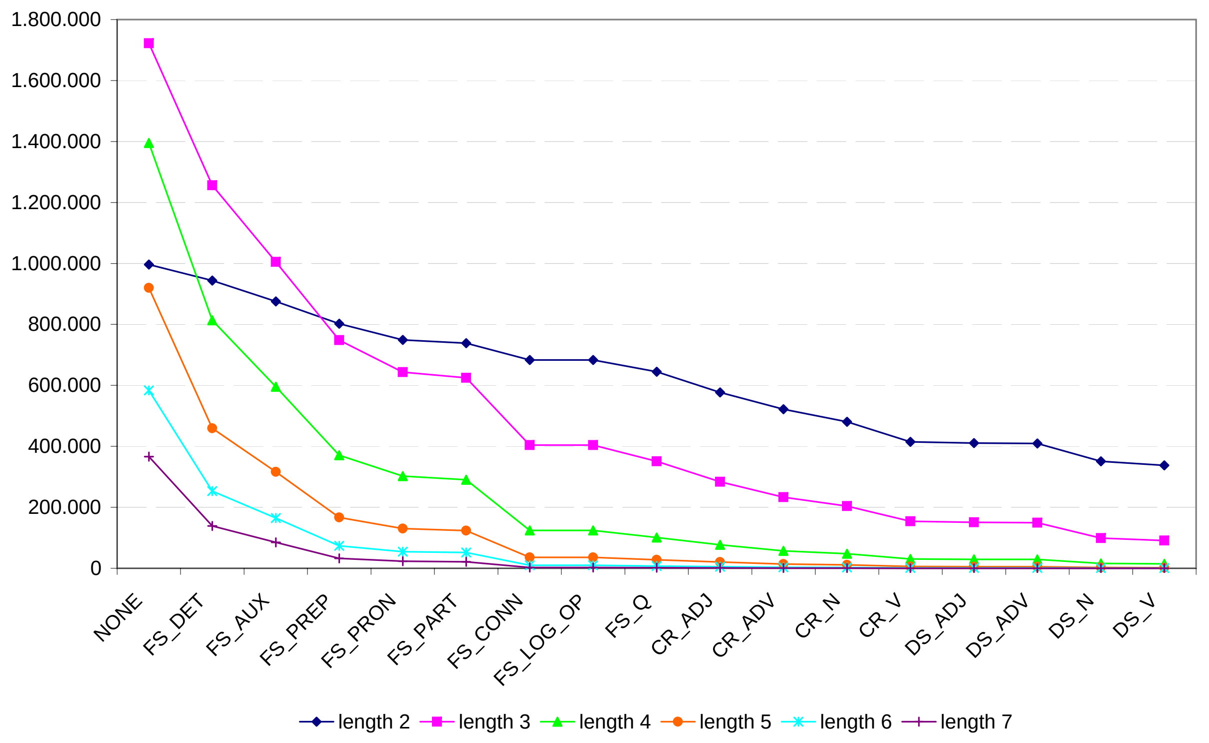


Figure 5.5: Cumulative filtering of linguistic stopword categories (minimum of three letters per term)

of extracting composite nouns the inclusion of these linguistic stopword categories seems feasible.

Table 5.12 lists the top 5 composite noun suited patterns for each length ordered by term frequency in decreasing order. In the table tf and df refers to the term frequency and document frequency of the multi-terms. The top 24 composite noun suited patterns for each length are presented in the appendix in Section A.2. The results show that term and document frequencies of longer multi-terms decrease quickly. Multi-terms of two, three, and four tokens provide appropriate index terms. Longer sequences than four terms are more related to parts of sentences or headlines than to composite nouns.

However, the number of unique composite nouns is huge compared to single-terms. In order to become available as index terms, the number of multi-terms must be downsized. Therefore, the average term frequency is used as a threshold for multi-term selection. The numbers of final multi-terms are given in Table 5.13. # denotes the number of unique multi-terms, $\varnothing\ tf$ refers to the average term frequency of all multi-terms, and $\# \geq \varnothing\ tf$ is the number of unique multi-terms that occur more often than the average. Applying the average term frequency as selection criteria turns out as an effective filter. Achieved reduction factors are 95,7% (two terms), 90,5% (three terms), 94,6% (four terms), 96,6% (five terms), 97,4% (six terms), and 98,0% (seven terms).

Table 5.12: Top 5 patterns suited for composite nouns

Multi-term	tf	df
Two terms		
execution time	6.870	1.405
electrical engineering	6.333	3.316
response time	3.803	804
source code	3.768	1.403
fault tolerance	3.758	1.024
Three terms		
digital signal processing	385	301
natural language processing	303	186
partial differential equations	302	210
cache hit ratio	282	46
directed acyclic graph	251	198
Four terms		
automatic test pattern generation	79	68
extended channel dependency graph	62	8
dynamic buffer allocation scheme	55	1
linear feedback shift register	51	38
parallel task completion time	48	1
Five terms		
handoff call channel occupancy time	19	1
submit queries concerning historical events	10	10
average message latency versus traffic	10	4
scholarly archival journals inform readers	9	9
procedurally generated partial product reduction	9	1
Six terms		
procedurally generated partial product reduction tree	9	1
row shift invariant wavelet packet transform	7	1
adaptive row shift invariant wavelet packet	7	1
vertex versus maximal clique incidence matrix	4	1
vertex versus maximal clique incidence matrices	4	1
Seven terms		
adaptive row shift invariant wavelet packet transform	7	1
systolic redundant residue arithmetic error correction circuit	3	2
wird etwas knapp bei mir sagen wir	2	1
une courbe qui remplit toute une aire	2	2
temporal strata translate temporal query language statements	2	1

Table 5.13: Number of unique patterns suited for composite nouns

Multi-term length	#	Ø tf	# $\geq$ Ø tf
two tokens	5.479.613	3,38	233.319
three tokens	1.002.764	1,35	94.798
four tokens	151.649	1,11	8.238
five tokens	23.473	1,05	795
six tokens	4.155	1,04	106
seven tokens	919	1,03	18

The composite nouns are integrated in JavaTok, enabling an efficient computation of multi-term representations. These additional representations are exploited during indexing and retrieval to improve retrieval performance.

The procedure identifying stopwords ($\frac{itf}{idf}$ ranking described in Section 4.6.2) can also be applied on multi-terms. This results in the generation of stop-phrases, which adds a fourth layer to the stopword model depicted in Figure 4.10. However, the structured document retrieval prototype developed does not apply stop-phrase filtering.

5.7.2 Pattern Extraction Suited for Named Entities

The same procedure obtaining composite nouns is applied to generate a list of named entity suited patterns. Again, sequences of two to seven tokens starting with capital letters (token type $ALPHA\_COMMON\_UPPER$) are extracted. According to the composite noun experiment, the minimum term length is three characters. Multi-terms containing stopwords except content-related and domain-specific nouns (CR_N and DS_N) are not extracted. As for the final composite noun experiment, the inclusion of nouns within named entities is plausible.

The top 5 named entity suited patterns of each length are presented in Table 5.14. tf and df refer to the term frequency and document frequency. The top 24 named entities are included in the appendix in Section A.3. The results show that top-ranked named entity patterns and composite noun patterns occur with similar frequencies. Extracted multi-terms longer than four terms tend to denote list of names instead of named entities. In some cases, the mapping of the original INEX documents onto the generic document format (see Section 3.3) led to text recurrences. It occurs if two XML components are combined in a single FRA element and explains the repetition of terms in the named entities of length five, six, and seven.

The numbers of extracted named entities are given in Table 5.15. Although uniquely named entity patterns are not as numerous as composite nouns, a reduction is still necessary for indexing and retrieval. As in the previous experiment, only multi-terms that are more frequent than the average term frequency of the actual multi-term length ($\geq Ø\ tf$) are selected.

Table 5.14: Top 5 patterns suited for named entities

Multi-term	tf	df
Two terms		
Pattern Recognition	5.914	1.575
Machine Intelligence	5.670	1.568
Artificial Intelligence	4.802	1.819
Distributed Computing	3.885	1.749
Parallel Processing	3.057	1.227
Three terms		
World Wide Web	2.650	1.505
Pattern Recognition Letters	444	287
Unified Modeling Language	394	274
Internet Engineering Task	328	284
Ad Hoc Networks	311	74
Four terms		
World Wide Web Consortium	272	230
Internet Engineering Task Force	272	237
Reader Interest Survey Indicate	204	204
Goddard Space Flight Center	166	118
Virtual Reality Modeling Language	150	136
Five terms		
Sara Reese Hedberg Sara Reese	23	23
Reese Hedberg Sara Reese Hedberg	23	23
Linda Dailey Paulson Linda Dailey	23	23
Dailey Paulson Linda Dailey Paulson	23	23
Markov Regenerative Stochastic Petri Nets	22	8
Six terms		
Sara Reese Hedberg Sara Reese Hedberg	23	23
Linda Dailey Paulson Linda Dailey Paulson	23	23
Mary Jean Harrold Mary Jean Harrold	11	11
Shari Lawrence Pfleeger Shari Lawrence Pfleeger	9	9
Khaled El Emam Khaled El Emam	9	9
Seven terms		
Mo Kim Cheng Albert Mo Kim Cheng	7	7
Albert Mo Kim Cheng Albert Mo Kim	7	7
Optimal Infinite Impulse Response Edge Detection Filters	5	5
Stochastic Petri Nets Representing Generalized Service Networks	4	4
Reversible Jump Markov Chain Monte Carlo Computation	4	4

Table 5.15: Number of unique patterns suited for named entities

Multi-term length	#	Ø tf	# $\geq$ Ø tf
two tokens	955.390	3,55	41.473
three tokens	232.245	1,98	15.276
four tokens	54.891	1,51	6.682
five tokens	9.873	1,25	872
six tokens	3.953	1,13	225
seven tokens	2.055	1,03	37

5.7.3 Pattern Extraction Suited for Formulaic Speech

A third approach aims at the extraction of multi-terms that are missed by the composite noun and named entity extraction tasks. These multi-terms are summarized as formulaic speech patterns, including not only common language patterns but other multi-terms that were excluded from previous extraction for three reasons:

1. token sequences consist of mixed token types (e.g., `United States of America`)
2. token sequences contain tokens of one or two characters (e.g., `et al`)
3. token sequences contain stopwords (e.g., `state of the art`)

From the information retrieval point of view, this allows to include stopwords in multi-term indices although stopword filtering is applied on single-terms.

Accounting for such patterns, a first experiment extracts token sequences of two to seven tokens containing both token types, *ALPHA_COMMON_LOWER* and *ALPHA_COMMON_UPPER*. The length of token strings is not accounted for and all strings are converted into lower case. Stopword checking is disabled. Only multi-terms not included in the composite noun or named entity dictionaries are extracted. Note that this extraction tasks includes composite noun or named entity patterns that are missed by the previous experiments. This is the case if patterns are used only at the beginning of sentences or as titles where the first term is capitalized.

The numbers of unique multi-terms (#) are given in Table 5.16. $\varnothing\ tf$ is the average term frequency and $\# \geq \varnothing\ tf$ the number of multi-terms with a higher term frequency than the average. However, the high numbers of unique multi-terms require further filtering to become useful for retrieval.

Table 5.16: Number of unfiltered formulaic speech

Multi-term length	#	$\varnothing\ tf$	$\# \geq \varnothing\ tf$
two tokens	48.623.745	18,20	2.671.814
three tokens	44.899.721	2,54	16.041.542
four tokens	38.341.039	1,33	26.118.414
five tokens	32.096.913	1,12	27.450.841
six tokens	26.687.256	1,06	24.745.505
seven tokens	22.088.444	1,04	21.071.165

An investigation of the unfiltered multi-terms shows that mainly functional stopwords occur frequently in the patterns (see Table 5.17). The multi-terms in the table are sorted according to their term frequencies. As before, multi-terms longer than five terms turn out to be inappropriate for indexing.

Based on the intermediate results of the initial experiment (see Tables 5.16 and 5.17), a second experiment including minimal stopword filtering is run. In order to reduce the number of patterns extracted, multi-terms started or ended by functional stopwords are excluded.

Table 5.17: Top-ranked unfiltered formulaic speech

Multi-term length	Multi-terms
two tokens	`of the`, `in the`, `to the`, `and the`, `on the`, `can be`, `for the`, `of a`, `is a`, `number of`
three tokens	`curricula vitae of`, `the university of`, `the number of`, `as well as`, `one of the`, `a set of`
four tokens	`vitae curricula vitae of`, `curricula vitae curricula vitae`, `at the university of`, `from the university of`, `his research interests include`
five tokens	`curricula vitae curricula vitae of`, `is a member of the`, `he is a member of`, `in computer science from the`
six tokens	`he is a member of the`, `computer science from the university of`, `in computer science from the university`
seven tokens	`in computer science from the university of`, `of computer science at the university of`, `in electrical engineering from the university of`

Table 5.18 lists the top 5 extracted patterns with their term frequency tf and document frequency df. The top 24 ranked formulaic speech suited patterns are given in Section A.4 in the appendix.

As explained for the named entity extraction, mapping difficulties of the INEX documents onto the generic format lead to the repetition of texts. In Table 5.18, the multi-term `curricula vitae curricula vitae` indicates that `curricula vitae` occurs as an attribute in the `<sec title="curricula vitae">` element and in the content of the element itself. During the mapping, both contents were merged into a single `FRA` element. As a consequence, the algorithm correctly extracted the three multi-terms `vitae curricula vitae`, `curricula vitae curricula`, and `curricula vitae curricula vitae` all with the same term frequency tf and document frequency df. Generally, this kind of 'redundant mentioning' is not the norm in the INEX documents. Correcting these errors manually would cost much time, which stands in no relation to the benefits.

The final numbers of multi-terms are given in Table 5.19. In contrast to the multi-terms extracted previously, only patterns occurring five times more often than the average term frequency ($\# \geq 5 \cdot \varnothing\ tf$) are extracted. This threshold is chosen for two reasons: (1) The high number of multi-terms would drop retrieval performance, and (2) the result covers a lot of less frequent multi-terms that are not relevant for information retrieval. Although the extracted patters are definitely useful for further linguistic analysis, their benefit for retrieval has to be evaluated. Extracted multi-terms longer than three terms only marginally coincide with pure concepts. However, top-ranked patterns seem to be helpful during searches. The extracted patterns further include:

Table 5.18: Top 5 patterns suited for formulaic speech

Multi-term	tf	df
Two terms		
computer science	29.743	8.233
research interests	14.241	6.737
et al	12.946	3.645
software engineering	11.901	2.927
interests include	11.382	5.923
Three terms		
vitae curricula vitae	11.048	11.048
curricula vitae curricula	11.048	11.048
research interests include	10.260	5.479
parallel and distributed	6.937	2.158
analysis and machine	5.425	1.487
Four terms		
curricula vitae curricula vitae	11.048	11.048
pattern analysis and machine	5.404	1.480
analysis and machine intelligence	5.384	1.478
science from the university	3.796	2.575
degree in computer science	3.765	2.125
Five terms		
pattern analysis and machine intelligence	5.378	1.476
computer science from the university	3.602	2.455
paper is organized as follows	2.005	2.005
computer science at the university	1.998	1.465
computer vision and pattern recognition	1.918	733
electrical engineering and computer science	1.055	756
Six terms		
professor in the department of computer	1.224	986
department of electrical and computer engineering	818	597
vitae curricula vitae of curricula vitae	550	550
curricula vitae curricula vitae of curricula	550	550
science from the university of california	542	459
professor in the department of electrical	525	457
Seven terms		
degree in computer science from the university	1.139	857
professor in the department of computer science	1.001	833
professor of computer science at the university	624	548
department of computer science at the university	580	476
curricula vitae curricula vitae of curricula vitae	550	550

Table 5.19: Number of unique patterns suited for formulaic speech

Multi-term length	#	Ø tf	$\# \geq$ Ø tf	$\# \geq 5 \cdot$ Ø tf
two tokens	1.218.870	5,66	165.344	33.529
three tokens	6.063.211	1,77	1.195.166	102.381
four tokens	9.046.078	1,26	915.462	65.591
five tokens	8.516.547	1,10	451.576	24.180
six tokens	7.100.605	1,06	252.461	11.612
seven tokens	6.057.925	1,05	168.818	6.823

- Expected writing patterns, e.g., `et al`, `point of view`, `state of the art`, `paper is organized as follows`

- Single letter combinations in titles written in spread out capital letters: e.g., `R E P O R T` (realized as a multi-term consisting of six capitalized single-letter tokens)
- Missing composite nouns in mixed case titles or acronyms, e.g., `Computer science`, `Research interests`, `National science foundation`

5.8 Acronym Extraction and Expansion

Especially in technical documents, acronyms are extensively used to reduce the length of a text, make the text more readable, and assign names to complex entities. Knowledge about the meaning of an acronym is essential to understand the whole content. In other words, misinterpretation of acronyms leads to confusion and obscurity. Within texts new acronyms are often defined on demand. Therefore, complete lists of acronyms (even for restricted domains) are not available. Acronyms have to be identified dynamically and must be interpreted according to their context (eventually only valid in the current document).

From the information retrieval point of view, acronyms provide important information about the content of a document. This is the reason for including acronyms in the index terms. However, queries that address the full form of an acronym do not match texts that solely contain the short form of the same acronym. In structured document retrieval where small portions of texts (e.g., `FRA` elements) are compared against a query, this leads to a drop of recall. On the other hand, queries addressing the short form of an acronym may retrieve in a high number of results containing the acronym but with a different meaning. Thus, retrieval precision decreases.

This section describes an automatic extraction process of acronyms and their corresponding full forms. The retrieval engine uses the extracted patterns to expand queries to include both, short and full forms of acronyms mentioned. Because queries are generally too short to apply a proper context analysis, all known full forms are added for each acronym.

An acronym dictionary is created in two steps: First, acronym patterns together with their contexts are extracted. Second, context analysis is applied for filtering incorrect full forms. Figure 5.6 depicts the two patterns investigated. The first *Pattern I* extracts acronyms that are followed by their embraced full forms. The opening and closing brackets are exploited to identify the complete full form. The second *Pattern II* identifies full forms that are followed by their embraced acronyms (reverse lookup). In contrast to the first pattern, the size of the full form (number of terms included) is not defined explicitly. Because acronyms may contain words that do not appear in the acronym string itself (e.g., `of`, `for`, `in`, `and`), a dynamic matching approach is applied.

In the experiment, two to seven letter single-tokens typed as *ALPHA_ACRONYM_SIMPLE* are investigated. The size of the pre-window (resp. post-window) is set to the number of

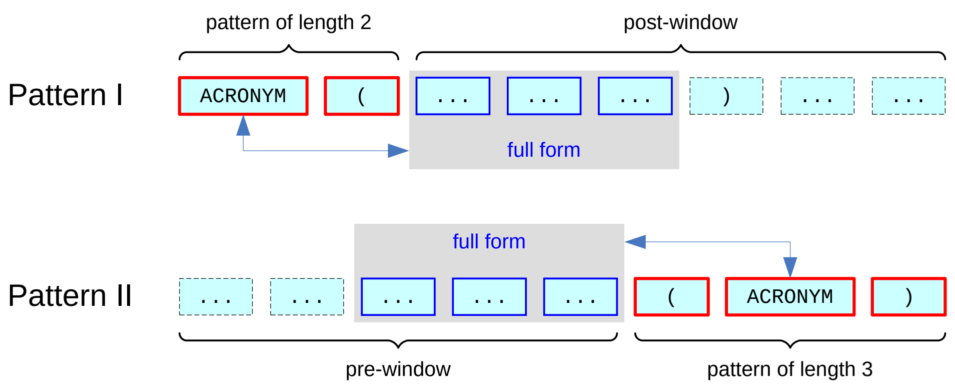


Figure 5.6: Extracted acronym patterns

characters in the acronym plus five additional single-tokens. This allows up to five context words being included which are not reflected in the acronym itself. These words are allowed to be stopwords of the functional stopword layer only.

Each extracted context (full form) is analyzed according to its conformance with the acronym (short form). Three different strategies are tested:

- **S1:** The same number of tokens than letters in the acronym is extracted.
- **S2:** The same number of tokens than letters in the acronym is extracted. Initial token-string letters must match the corresponding acronym letters.
- **S3:** The same or a higher number of tokens than letters in the acronym is extracted. Initial token-string letters must match the corresponding acronym letters. However, functional stopwords that do not correspond with acronym letters are allowed.

Table 5.20 summarizes the number of extracted acronyms according to each strategy.

Table 5.20: Number of unique extracted acronyms

	Pattern I			Pattern II			Pattern I ∪ Pattern II		
Acronym length	S1	S2	S3	S1	S2	S3	S1	S2	S3
two letters	323	206	206	5.192	2.509	2.608	5.428	2.630	2.729
three letters	1.218	1.051	1.051	15.940	7.588	8.187	16.539	8.037	8.636
four letters	533	451	451	8.879	2.695	3.621	9.197	2.942	3.868
five letters	95	68	68	2.157	372	674	2.232	422	724
six letters	29	18	18	643	43	106	669	59	122
seven letters	8	4	4	245	2	9	252	6	13

The results show that acronym patterns in the INEX documents are commonly used. *Pattern II* (e.g., `...Unified Modeling Language (UML)...`) occurs more often than *Pattern I* (e.g., `...DAG (Directed Acyclic Graph)...`). The majority of acronyms consist of three and four letters, where three letter acronyms are most frequent. But even longer acronyms of length six and seven characters are meaningful search concepts. Interestingly, the acronyms of *Pattern I* and *Pattern II* show only marginal overlap (see the last three columns in Table 5.20).

Another observation is that acronyms defined by *Pattern I* never included a stopword in their corresponding full form. This becomes evident by investigating the strategies $S2$ and $S3$, which resulted in the same acronyms (regardless their lengths).

Strategy $S1$ extracts all occurrences of a pattern. However, it includes many incorrect full forms of acronyms for both acronym patterns (see Table 5.21). The second strategy $S2$ matching first letters minimizes these errors and results in 41,1% of all patterns. Unfortunately, $S2$ excludes a lot of acronyms that include stopwords. Hence, the third strategy $S3$ relaxes the first letter matching constraint of $S2$ by allowing functional stopwords within the acronyms' full form. $S3$ results in 46,9% of all extracted acronyms.

Table 5.21: Examples of incorrectly identified acronyms applying strategy $S1$

Acronym	Full form	tf	df
	Two tokens		
EM	the expectation-maximization	42	1
TR	v i	20	1
XP	extreme programming	20	2
HA	faulty fa	12	1
MP	t comm	11	1
	Three tokens		
MIT	institute of technology	77	2
USC	of southern california	53	9
ETH	institute of technology	45	5
CNR	national research council	41	5
ISO	organization for standardization	41	2
	Four tokens		
NIST	of standards and technology	55	3
CVPR	vision and pattern recognition	35	9
RTSS	ieee real-time systems symposium	34	20
PLDI	language design and implementation	33	4
NCSA	center for supercomputing applications	33	3
	Five tokens		
KAIST	institute of science and technology	56	9
NSERC	engineering research council of canada	34	11
JETTA	testing : theory and applications	21	7
CIPIC	image processing and integrated computing	20	6
DISCA	the department of computer engineering	17	5
	Six tokens		
ASPLOS	for programming languages and operating systems	30	3
SWEBOK	the software engineering body of knowledge	14	1
TAPADS	aspects of parallel and distributed systems	11	5
LARPBS	with a reconfigurable pipelined bus system	11	1
SIGMOD	interest group on management of data	10	1
	Seven tokens		
EMMCVPR	methods in computer vision and pattern recognition	7	3
NOSSDAV	system support for digital audio and video	7	2
POSTECH	at pohang university of science and technology	5	3
IWFHRVI	sixth int'l workshop frontiers in handwriting recognition	4	1
SIGARCH	acm special interest group on computer architecture	4	2

Applying strategy $S3$, the results of both patterns are given in Tables 5.22 and 5.23.

Table 5.22: Top 5 extracted acronyms (Pattern I)

Acronym	Full form	tf	df
	Two tokens		
IP	Internet Protocol	71	1
IP	intellectual property	59	1
VR	virtual reality	58	2
AI	Artificial Intelligence	49	3
ML	maximum likelihood	45	1
	Three tokens		
NSF	National Science Foundation	207	16
ATM	asynchronous transfer mode	129	1
UML	Unified Modeling Language	96	2
PCA	principal component analysis	90	1
RDF	Resource Description Framework	87	3
	Four tokens		
IETF	Internet Engineering Task Force	85	3
VRML	Virtual Reality Modeling Language	80	2
ISCA	Int'l Symp. Computer Architecture	56	1
NIST	National Institute of Standards and Technology	54	3
VLSI	Very Large Scale Integration	50	2
	Five tokens		
DARPA	Defense Advanced Research Projects Agency	88	6
CORBA	Common Object Request Broker Architecture	58	2
KAIST	Korea Advanced Institute of Science and Technology	52	9
ICDCS	Int'l Conf. Distributed Computing Systems	20	2
CIPIC	Center for Image Processing and Integrated Computing	20	6
	Six tokens		
ASPLOS	Architectural Support for Programming Languages and Operating Systems	25	3
TAPADS	Theoretical Aspects of Parallel and Distributed Systems	11	5
SIGMOD	Special Interest Group on Management of Data	10	1
ICLASS	Illinois Computer Laboratory for Aerospace Systems and Software	9	8
PECASE	Presidential Early Career Award for Scientists and Engineers	9	3
	Seven tokens		
EMMCVPR	Energy Minimization Methods in Computer Vision and Pattern Recognition	7	3
SHOSLIF	Self-Organizing Hierarchical Optimal Subspace Learning and Inference Framework	2	1
SIGCAPH	Special Interest Group for Computers and the Physically Handicapped	1	1
INSPASS	Immigration and Naturalization Service Passenger Accelerated Service System	1	1
EMERALD	Event Monitoring Enabling Responses to Anomalous Live Disturbances	1	1

The final dictionary contains 2.729 two letter acronyms, 8.636 three letter acronyms, 3.868 four letter acronyms, 724 five letter acronyms, 122 six letter acronyms, and 13 seven letter acronyms. Tables containing the top 24 extracted acronyms of each length are attached in the appendix in Section A.5. During retrieval the acronyms extracted provide an extra multi-term index that is searched. In contrast to multi-terms of the previous section, the smaller number allows indexing of the complete set of acronyms. Note that this index also includes stopwords in special contexts and considers the order of words.

The results show clearly that both patterns investigated serve as an appropriate means for extracting acronyms with high confidence. Promising patterns left open for further

Table 5.23: Top 5 extracted acronyms (Pattern II)

Acronym	Full form	tf	df
	Two tokens		
IP	Internet protocol	19	1
EM	expectation maximization	5	1
NS	Network Simulator	5	2
CT	computed tomography	5	1
LP	longest path	4	3
	Three tokens		
PVM	Parallel Virtual Machine	25	1
ATM	asynchronous transfer mode	25	1
SQL	Structured Query Language	23	1
LRU	least recently used	21	2
CGI	common gateway interface	18	1
	Four tokens		
VLIW	very long instruction word	24	1
SNMP	Simple Network Management Protocol	15	1
SGML	standard generalized markup language	14	1
MIPS	million instructions per second	13	2
TDMA	time division multiple access	13	2
	Five tokens		
CORBA	Common Object Request Broker Architecture	23	1
ENIAC	Electronic Numerical Integrator and Computer	7	1
EDSAC	Electronic Delay Storage Automatic Calculator	4	1
TSAPI	Telephony Services Application Programming Interface	2	1
PALKA	Parallel Automatic Linguistic Knowledge Acquisition	2	1
	Six tokens		
YUPPIE	Yorktown Ultra Parallel Polymorphic Image Engine	3	1
PCMCIA	Personal Computer Memory Card International Association	3	1
EBCDIC	Extended Binary Coded Decimal Interchange Code	3	1
DOCSIS	Data Over Cable Service Interface Specification	2	1
WIDTIO	When In Doubt Throw It Out	1	1
	Seven tokens		
WYSIWYG	what you see is what you get	5	1
WYSIWYR	what you see is what you record	1	1
YCAGWYS	you can always get what you see	1	1
WYSIWYC	What You See Is What You Compute	1	1

investigation include plurals (e.g., `POWs (Prisoners of War)`), special capitalizations (e.g., `XML (eXtensibel Markup Language)`), and multiple letter acronyms (e.g., `HyREX (Hyper-media Retrieval Engine for XML)`).

5.9 Summary

This chapter described the generation of natural language resources that are relevant for information retrieval. The experiments conducted solely rely on the output of Extended Tokenization. Generated resources include typing rules for single-tokens and multi-tokens, single-term dictionaries (full forms, abbreviations), and multi-term dictionaries (composite nouns, named entities, formulaic speech, acronyms with corresponding full forms). These resources are incorporated in the indexing and retrieval processes to improve retrieval performance. Multi-term dictionaries and rules improve the output of the tokenizer and thus, the quality of content representations. The rule generation process is supported by a Rule Miner that infers rule proposals based on extracted patterns. A bootstrapping-like approach is applied to identify meaningful patterns in similar contexts automatically.

Chapter 6

Controlling complexity is the essence of computer programming.

B. Kernighan

Classification of XML Documents

With increasing amounts of available XML documents, the task of automatic knowledge discovery from the web becomes highly significant. As an appropriate machinery, classification allows to categorize documents to facilitate that task. In this chapter a classification approach for structured documents is introduced. It is based on the *k*-nearest neighborhood algorithm that relies on an edit distance measure. The originality of the work lies in combining both the content and the structure of XML documents to compute the edit distance. The approach is empirically evaluated using real-world XML collections provided by INEX.[1]

6.1 Introduction

The eXtensible Markup Language has recently emerged as a standard for developing many web applications dealing with document storage and retrieval, e.g., digital libraries. XML was mainly developed to achieve an enriched representation of documents and more retrieval flexibility when searching information. Systems concerned with such web applications are mainly augmented traditional information retrieval systems. In contrast to traditional retrieval systems that deal with flat documents, XML retrieval systems take the logical structure of documents into account. In addition to the raw text (pure content), this structure is considered as a further valuable information source for document representation. It serves to refine the search process and to improve the quality of the retrieval results. Indeed, each document is represented as a tree of XML nodes, where each node is associated with a label and a content (XML component). The goal of retrieval systems is, therefore, to retrieve only relevant components instead of the whole documents in response to the user queries. As a matter of consequence, the retrieval precision gets better.

Form the structured document point of view, it is of high importance to exploit the structure of documents in order to devise a classification machinery. This latter is a very significant mechanism in the context of XML retrieval for two reasons: (1) a user query

[1]Parts of this chapter have already been published in [30] and [121].

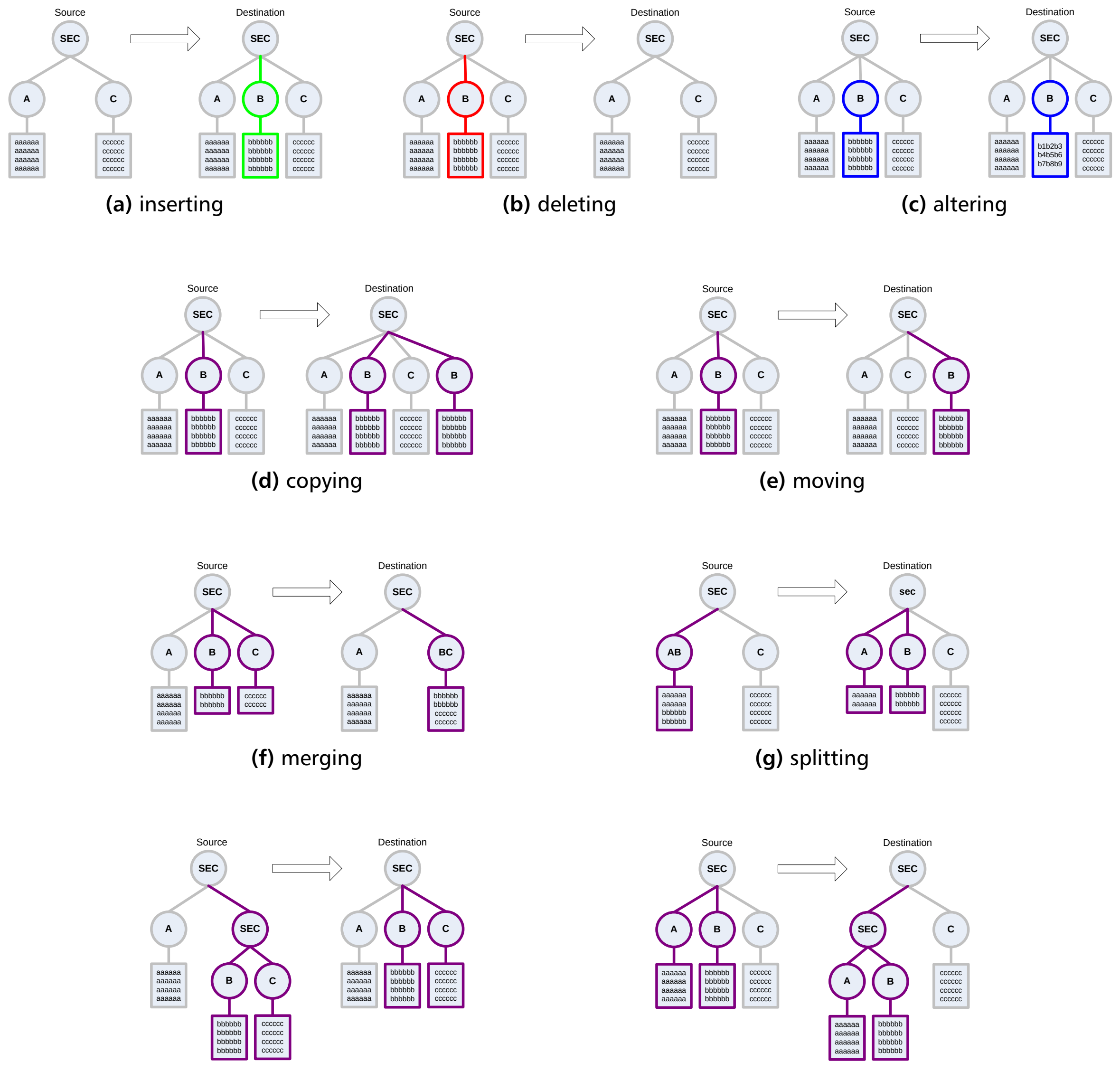
can be satisfied by means of different possible answers; closely associated documents tend to be relevant to the same requests; thus, returning all documents of classes with relevant documents is feasible, and (2) retrieval systems and web mining tools are generally operational in the same environment.

Classifying (and clustering) XML documents can be basically done in three ways: (1) using exclusively the textual contents of documents as usually done in traditional text categorization (and clustering) systems, (2) using exclusively the structure of the XML documents, and (3) using both the contents and the structure in a hybrid manner. In this work, the latter approach is followed. The aim is to discover structural and content patterns (characteristics) shared by XML documents of the same class. These patterns are mainly expressed in terms of their tags (node labels), contents, and inter-relationship.

To achieve this goal, the k-nearest neighborhood algorithm is applied for classification. The algorithm, which strongly relies on a distance measure, will be explained later. Since an XML document is represented in the form of a tree, it is straightforward to adopt an *edit distance* to measure the distance between document trees (dissimilarity measure). Basically, edit distance algorithms compute the minimum cost to transform one document tree, the *source tree*, into another document tree, the *destination tree*. Generally, existing algorithms focus on the structure only. This chapter shows how the content can be embedded into the edit distance. To the best of the author's knowledge, no previous work has used edit distance taking the content and structure of the XML tree into account.

The idea to measure the similarity between document-centric XML documents using an edit distance measure is based on the way documents are written in real world. This allows to take 'natural' document authoring techniques (see Figure 6.1) into account, applying application-specific weights. Looking at real world documents, the structure is also quite homogenous and could simply be reduced to a minimal set of different nodes (i.e., title, author, section, paragraph, table, figure, etc.). This homogenous set of XML components, which is highly recursive (i.e., chapter, section, subsection), leads to quite similar document structures of completely different content. Thus, an appropriate similarity measure plays a key role in this domain.

The rest of this chapter is organized as follows. Some of the research work dedicated to XML document classification is briefly summarized in Section 6.2. In Section 6.3, the edit distance algorithm is discussed with respect to content and structure. An alternative approach, based solely on the content of documents, is introduced in Section 6.4. Section 6.5 briefly explains the k-NN algorithm. Section 6.6 discusses the experimental evaluation of the approaches. Finally, Section 6.8 concludes this chapter.



(a) inserting **(b)** deleting **(c)** altering

(d) copying **(e)** moving

(f) merging **(g)** splitting

(h) raising **(i)** lowering

Figure 6.1: Document authoring operations

6.2 Related Work

Although classification has been widely discussed in the framework of traditional information retrieval (with flat documents), it has not yet gained much attention in the field of structured documents. In the sequel, some of the research work dedicated to XML document classification is briefly summarized.

In [64], a classification approach is proposed that aims at using the structure and the contents to classify XML documents. It relies on a generative Bayesian classifier. Here, an

XML document is represented in the conventional tree-like form of a directed acyclic graph, where each node of the graph represents a XML component and each edge represents a parental relationship between the nodes. The generative model assumes that there are two types of belief, structural and textual, which will be combined to get one single evidence about a document assignment to classes:

$$P(doc|\theta) = P(str|\theta) \times P(cont|str, \theta)$$

where θ is a set of the model parameters, *doc*, *str*, and *cont* designate document, structure, and content. The quantities $P(str|\theta)$ and $P(cont|str, \theta)$ are computed via a belief network as follows:

$$P(str|\theta) = \prod_{i=1}^{\#\ of\ tags} P(node_i|ancestor(node_i))$$

$$P(cont|str, \theta) = \prod_{i=1}^{\#\ of\ terms} P(term_i|node_j, \theta)$$

During the training phase, the parameters of the generative model are learned using the Expectation-Maximization method. During the classification phase, the classifier is used to assign unseen XML documents to the class with the highest probability $P(doc|class_k)$.

In [33], a similar work to that described in [64] is developed. It consists of exploiting structural information in semi-structured documents using a generative Bayesian Classifier. The method, however, is less general, since the assumption is that all documents have the same structure across all classes. Documents are broken down into components, each of which contains either structured data or non-structured textual data. Let d_j, c_k, s_l, t_i, designate respectively a document j, a class k, a component l, and a term i, the total probability of a document is the product of the individual component probabilities:

$$P(d_j|c_k) = \prod_{s_l \in d_j} \prod_{t_i \in s_l} P_{s_l}(t_i|c_k)^{f^j(t_{is})}$$

where $f^j(t_{is})$ indicates the word frequency of term i in the s^{th} component of the document j. The first product is over all structural components s_l that are present in the document d_j. The probability P_{s_l} is obtained for each document component. From this, it is clear that each document is represented by a set of feature vectors, one for each component, so that word frequency $f^j(t_{is})$ is maintained on a per-component basis.

In [264] a classifier, called XRules, is developed. This classifier is structure-oriented and aims at discovering a set of structural rules that define the individual classes. Basically, these rules that reflect regular structural patterns of each class are learned during the training phase. During the classification phase, given an unlabeled XML document, the set of rules pertaining to that document are used to compute the membership evidence to classes.

Hence (document) trees are a special form of graphs, approaches from the graph matching and comparison domain can also be applied [251, 265, 179, 40, 184, 229, 56, 138, 139, 100, 140]. Nearly all of these approaches rely on additional heuristics to reduce the graph isomorphism problem, which is considered to be $\mathcal{NP}$-complete [166], to a simpler problem definition. These heuristics, for the sake of comparing document-centric XML documents, are not on hand. Also, the additional freedom in the structure provided by a graph representation seems not to reflect the natural way of document authoring. Because of this and their runtime performance (high complexity) these approaches are not investigated in this work.

Most of the work in the area of XML documents classification uses *tree edit distance* to measure the distance between XML document trees. Basically, edit distance algorithms compute the minimum cost to transform one document tree into another. Each transformation (e.g., insert, delete, alter) is associated with a certain cost. In addition to the edit distance measure, other approaches apply other types of distance measures. These rely on XML peculiarities such as tags, parent-child relationships, root-leaf XPaths, arity of nodes, etc. [267, 43, 71, 198].

Many algorithms have been proposed to compute the edit distance between two trees [49, 48, 47, 54, 232, 266, 252, 227]. Most of these algorithms use dynamic programming techniques as initially described in [162]. The aim is to find the cheapest sequence of transformations (i.e., the cumulated costs), called *delta script* or *edit script* [49, 254, 253], to transform a tree T_1 into a tree T_2.

$$dist(T_1, T_2) = \min_j \{s_j\} \tag{6.1}$$

$$s_j = \sum_i^{\#\, transformations} t_i^{(j)} \tag{6.2}$$

where s_j is a script, and $t_i^{(j)}$ is the cost of a transformation i in the edit script j.

The main difference of the various edit distance measures is the set of allowed edit operations and their associated costs. Early work [225] considered *insert* and *delete* of leaf nodes, and node *relabeling* of all nodes. Extensions were then proposed to allow insertion and deletion of nodes anywhere in the tree [232, 266, 252, 227].

In general the problem of finding the edit distance between two trees is $\mathcal{NP}$-hard [171, 232, 25]. Works in [266] and [49] instead try to find an efficient algorithm for the reduced problem of ordered/binary trees, in which a left-to-right order among siblings is significant. For XML documents arising in database applications (data-centric applications), authors believe that the unordered model is more important [254]. However, in the context of XML documents originating from natural language texts (document-centric applications) one can argue for ordered trees.

Chawathe et al. [49] first applied the same edit operations and restrictions to detect changes in structured documents. In subsequent works he extended the approach to cover also move operations as basic edit operations [48]. Later he defined operations for copying and gluing of subtrees [47]. In order to accelerate edit distance calculations, Guha et al. [113] relax the problem to computationally inexpensive upper and lower bounds. Nierman and Jagadish [187] implemented their algorithm based on Chawathe's work.

Zhang and Shasha [266] provide a fast algorithm to calculate the edit distance between ordered labeled trees. The minimum costs for mapping all descendants of a node is computed in advance, using the notion of *keyroots*. Keyroots of a tree are defined as the set of all first-level children having left siblings plus the root node itself. Computing the keyroots of a tree in advance applies the concepts of *tree distance* and *forest distance*. The tree distance is calculated as the distance between two (sub-)trees without considering the context of ancestors and/or siblings. The forest distance, instead, uses the tree distance and takes ancestor and sibling relations into account. To get the minimum transformation cost for two trees, the minimum cost mapping from all keyroots amongst the children and the cost of the leftmost child (forest distance of its rightmost child) is needed. The algorithm then proceeds from the leaf nodes up to the root node in a postorder traversal.

Further methods dealing with similarity of XML documents can be found in the literature related to structural mapping, e.g., change detection [54, 253]. A good overview of existing algorithms for change detection together with their properties is given by Peters [200] and by Dalamagas et al. [60].

6.3 Tree Matching via Edit Distance

In order to apply any kind of (semi-)automatic classification mechanism on semi-structured documents, one has to define a metric that expresses the similarity of two documents first. Based on this similarity a certain distance between two documents can be defined by means of a 'dissimilarity'.

Several metrics which are quite different in terms of how this similarity is computed can be found in the literature. Mainly they can be categorized into two categories:

- **Heuristic approaches:** Mainly statistically motivated, these methods match documents by comparing common tags, parent-child relations, root-leaf XPaths, arity of nodes, etc. [267, 43, 264].

Table 6.1: Overview of change detection algorithms and properties [200]

Algorithm	Complexity	Memory	Operations	Tree type
LaDiff	$O(ne+e^2)$	linear	basic, move	ordered
MH-Diff	$O(n^2 \log n)$	-	basic, move, copy	unordered
XMLTreeDiff	$O(n^2)$	quadratic	basic	ordered
MMDiff	$O(n^2)$	quadratic	basic	ordered
XMDiff	$O(n^2)$	linear	basic	ordered
IBM's XML Diff & Merge	$-$	-	basic	-
3DM's matching algorithm	$O(n)$	-	basic, move	ordered
XyDiff	$O(n \log n)$	linear	basic, move	ordered
VM Tools	$-$	-	-	unordered
DiffXML	$O(ne+e^2)$	linear	basic, move	ordered
KF-Diff+	$O(n)$	-	basic	both
XML Diff and Patch	$-$	-	-	both
X-Diff	$O(n^2)$	quadratic	basic	unordered
DeltaXML	$-$	linear	basic	both
TreePatch	$O(ne+e^2)$	linear	basic, move	-
BioDIFF	$O(n^2)$	quadratic	basic	unordered

- **Tree-based approaches:** These approaches measure the similarity of two documents represented as labeled trees. Depending on their application, three different methods can be distinguished [26]:
  - *Tree edit distance* is defined as the minimum cost of transforming one tree into the other (see [49, 266, 227]). Each transformation (e.g., insert, delete, alter) is associated with a certain cost. The algorithm has to find the cheapest sequence of transformations known as delta script. Extensions of these approaches also operate on (rooted) graphs. Figure 6.2 shows the transformation from the source tree to the destination tree. In (a) the nodes C and E are deleted from the source tree, generating an intermediate tree (b). Afterwards, the nodes H and I are inserted into the intermediate tree. Finally, altering the nodes B to B' and D to D' results in the destination tree (c).
  - *Tree alignment* is defined as the cumulated cost for matching all nodes of a source tree T_1 to all nodes of an isomorphic destination tree T_2 [144] (disregarding node labels). The cost function operates on pairs of nodes. First, 'empty' nodes are inserted in the source and destination trees so that the trees become isomorph. Then, all pairs of nodes (n_1, n_2), $(n_1, null)$, and $(null, n_2)$ are compared and their costs are summed. Figure 6.3 illustrates the source tree (Figure 6.3a), the destination tree (Figure 6.3b), and the tree with the aligned node pairs (Figure 6.3c).
  - *Tree inclusion* is defined to determine if a source tree T_1 is included in a destination tree T_2 [155, 154]. Therefore, nodes from the destination tree are deleted until both

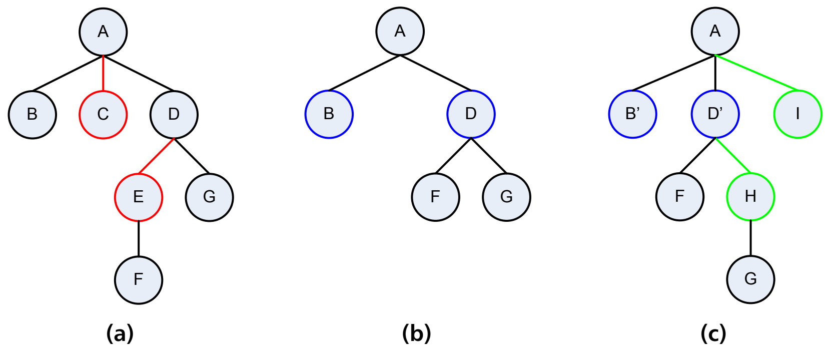


Figure 6.2: Tree Edit Distance

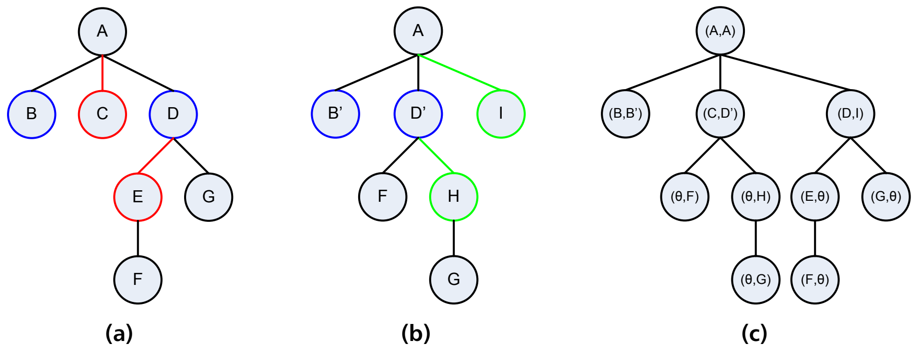


Figure 6.3: Tree Alignment

trees correspond or the number of nodes in T_1 becomes larger than the number of nodes in T_2 thus demonstrating that T_1 is no longer included in T_2. Figure 6.4 sketches the process to check whether a source tree (Figure 6.4a) is contained in a destination tree (Figure 6.4b) by successively deleting nodes (Figure 6.4c).

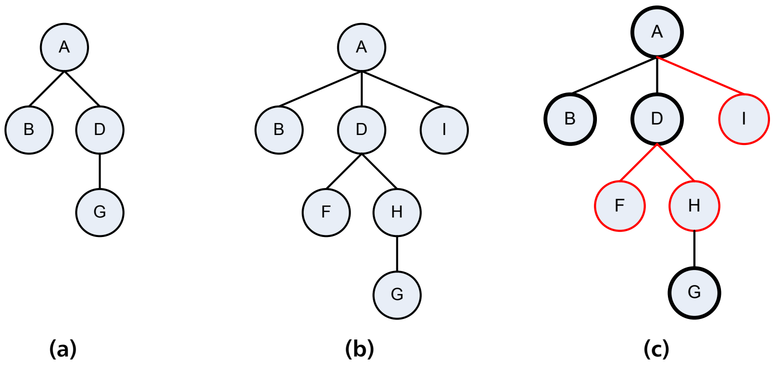


Figure 6.4: Tree inclusion

The focus of this work lies on the tree-based approach of metrics, and here especially on the tree edit distance. The algorithm proposed by Nierman and Jagadish [187], which allows to compute the edit distance between two trees using only the labels of the nodes, will be modified. Whilst the former algorithm is structure-oriented, this research aims at inducing a classifier that takes the structure and the content into account. The dynamical aspect of the algorithm is kept, but it has been improved with respect to two dimensions: (1) the algorithm will rely on simplified edit operations; (2) the algorithm will take the content match into account when computing the edit distance. Concretely, a hybrid distance measure needs to be formulated that combines structure-based and content-based distances. Each of these are described in the following sections.

6.3.1 Structure Matching

To formulate the distance, some definitions of the basic concepts are introduced.

Definition 6.1 [Tree node]: A node n of a tree T is associated with a label $\lambda(n)$, a content $\gamma(n)$, a parent node $p \in T$, and a set of child nodes $ch(n)$. $\gamma(n)$ is empty ($null$), if a node does not have a content. The parent node p of the root node of T is $null$. The child nodes of n are uniquely identified as $n^1, n^2, \ldots, n^k$, where k is the degree of n, denoted as $deg(n)$.

Definition 6.2 [Ordered Tree]: An ordered tree T is defined as a rooted tree, where a left-to-right order among the child nodes is significant. $root(T)$ denotes the root node of T. The children of T are the subtrees T^i rooted in nodes that are child nodes of $root(T)$ ($root(T)^i = root(T^i)$ and $T^1, T^2, \ldots, T^k$, $k = deg(root(T))$. Further, $|T|$ represents the total number of nodes in tree T.

Based on the aforementioned definitions, the allowed basic edit operations can be described as follows:

Definition 6.3 [Insert operation]: Given a tree T and a node $p \in T$, a node n can be inserted into T as the i^{th} child of the node p using the operation ins(T,n,p,i). The cost function of this operation is $C_{ins}(T, n, p)$.

Definition 6.4 [Delete operation]: Given a tree T and a node $n \in T$ as the i^{th} child of node $p \in T$, n can be removed using the operation del(T,n,p,i). The cost function of deletion is $C_{del}(T, n, p)$.

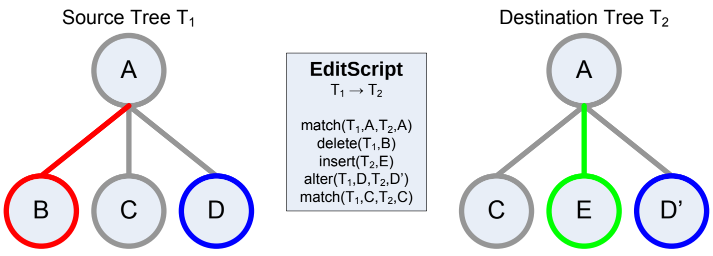
Definition 6.5 [Alter operation]: Given two nodes n_1 and n_2 whose labels are $\lambda(n_1)$ and $\lambda(n_2)$. The label of n_1 can be replaced by the label of n_2 using the operation $alt(n_1, n_2)(\equiv \lambda(n_1) \leftarrow \lambda(n_2))$. The cost function of this operation is $C_{alt}(n_1, n_2)$ defined as:

6

$$C_{alt}(n_1, n_2) = \begin{cases} 0 & \text{if } \lambda(n_1) = \lambda(n_2) \\ \beta & \text{otherwise} \end{cases} \tag{6.3}$$

Note that one can parameterize β so that altering a node lying at level l in the document tree costs $\beta(l)$. This seems reasonable for taking the depth of the tree and semantic closeness between tags into account. Renaming section marking labels such as `sec`, `ss1`, `ss2` or paragraph labels such as `p1`, `p2`, `p3` could result in cheaper costs because of their affiliation to the same (semantic) group (related labels).

Figure 6.5 illustrates the basic operations that transform the source tree T_1 into the destination tree T_2. In order to save space, no intermediate transformation steps are included. Deleted nodes colored in red are marked in T_1 only. Inserted nodes colored in green are marked in T_2 only. Altered nodes colored in blue are marked in both trees T_1 and T_2. Additionally, altered nodes are primed for better discrimination. All other nodes are left untouched. An edit script (see Definition 6.8) keeps track of all operations that transform T_1 into T_2.



EditScript
$T_1 \rightarrow T_2$

match(T₁,A,T₂,A)
delete(T₁,B)
insert(T₂,E)
alter(T₁,D,T₂,D')
match(T₁,C,T₂,C)

Figure 6.5: Basic tree edit operations and edit script

Furthermore, if $C_{ins}(T, n, p) = C_{del}(T, n, p)$, the edit distance measure becomes symmetric. One might think of unequal costs for insert and delete operations though. Thus, the similarity of two document trees $dist(T_1, T_2)$ and $dist(T_2, T_1)$ differs. Assigning a higher cost to the insert than to the delete operation may support a possible plagiarism identification. For instance, two distances $dist(T_1, T_2) = 3{,}4$ and $dist(T_2, T_1) = 1{,}2$ might indicate that T_1 and T_2 are very similar (low distance), where it is easier to generate T_1 by deleting contents from T_2 than vice versa.

These operations are applied on nodes that can be either leaf nodes or inner nodes, where an inner node is the root of a subtree. The cost of applying an operation on an inner node is recursively computed by summing up the cost of applying the respective structure modifications to its descendants. Hence, the following definitions of the cumulative costs (see Figure 6.6, nodes B and E):

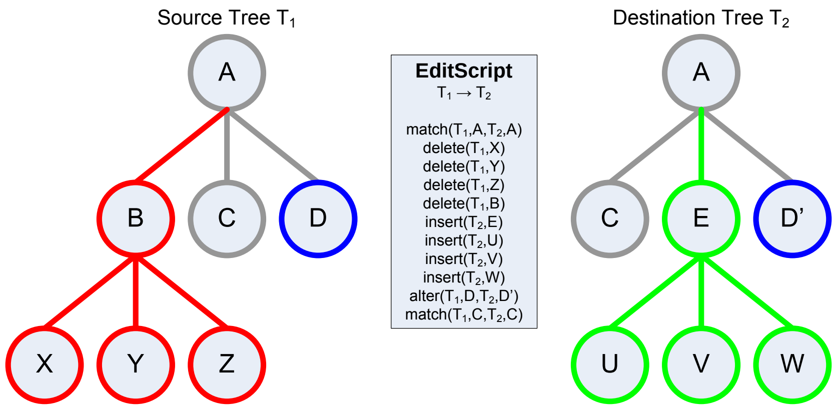
Definition 6.6 [Recursive delete]: The cumulative cost of deleting a subtree Sub from the tree T at node p is $C_{delCum}(T, Sub, p)$.

Definition 6.7 [Recursive insert]: The cumulative cost of inserting a subtree Sub into the tree T at node p is $C_{insCum}(T, Sub, p)$

Both costs C_{delCum} and C_{insCum} are bottom-up computed as follows:

$$\begin{cases} C_{delCum}(T, Sub, p) = C_{del}(T, root(Sub), p) + \sum_i C_{delCum}(T, Sub^i, root(Sub)) \\ \\ C_{insCum}(T, Sub, p) = C_{ins}(T, root(Sub), p) + \sum_i C_{insCum}(T, Sub^i, root(Sub)) \end{cases} \tag{6.4}$$

In other words, at each node of the subtree Sub of the source (resp. destination) tree T_1 (resp. T_2), the cost of the recursive delete (resp. insert) operation is calculated by summing the cost for deleting (resp. inserting) the single node $root(Sub)$ with the cumulative cost of deleting (resp. inserting) each of its children Sub^j. Obviously, the deletion of a subtree is done bottom-up, while the insertion of a subtree is done top-down.



```
match(T1,A,T2,A)
delete(T1,X)
delete(T1,Y)
delete(T1,Z)
delete(T1,B)
insert(T2,E)
insert(T2,U)
insert(T2,V)
insert(T2,W)
alter(T1,D,T2,D')
match(T1,C,T2,C)
```

Figure 6.6: Recursive tree edit operations and edit script

Having introduced some variables required, the edit distance between two trees T_1 and T_2 is computed using Algorithm 6.1. Based on dynamic programming, this algorithm constructs a $deg(root(T_1)) \times deg(root(T_2))$ matrix of distance values between the nodes of the two trees. First, the algorithm compares the root nodes of T_1 and T_2. This corresponds to an alter operation (line 5) because root nodes can neither be inserted nor deleted. Then, as seen in lines 7 and 10, the algorithm computes the distance values of inserting or deleting all nodes given the roots of two trees. These values serve to trigger the dynamic computation of cumulative costs, where each child of T_1 is compared to each child of T_2 recursively. Indeed, a cell $distMat[i][j]$ ($i > 0$ and $j > 0$) is assigned a cost that is computed using the content of its neighboring three cells:

- the content of the upper left neighbor, $distMat[i-1][j-1]$, is added to the distance between the subtrees rooted at nodes n_i and n_j (i.e., T_1^i and T_2^j) (line 15). This case corresponds to a match between node n_i and node n_j.
- the content of the left neighbor, $distMat[i][j-1]$, is added to the cost of inserting a subtree T_2^j to the source tree (line 16). This case corresponds to an insertion of a subtree rooted at node n_j.
- the content of the upper neighbor, $distMat[i-1][j]$, is added to the cost of removing a subtree T_1^i from the source tree (line 17). This case corresponds to a removal of an obsolete subtree rooted at node n_i.

The minimum cost of these three alternatives is retained and stored in $distMat[i][j]$.

Algorithm 6.1 Tree Edit Distance algorithm

```
1: procedure TreeDist($T_1, T_2$)
2:    int M = deg(root($T_1$))
3:    int N = deg(root($T_2$))
4:    int[][] distMat = new int[0..M][0..N]
5:    distMat[0][0] = $C_{alt}(root(T_1), root(T_2))$
6:    for ($j = 1$ to $N$) do
7:       distMat[0][j] = distMat[0][j-1] + $C_{insCum}(T_2, T_2^j, root(T_2))$
8:    end for
9:    for ($i = 1$ to $M$) do
10:      distMat[i][0] = distMat[i-1][0] + $C_{delCum}(T_1, T_1^i, root(T_1))$
11:   end for
12:   for ($i = 1$ to $M$) do
13:      for ($j = 1$ to $N$) do
14:         distMat[i][j] = min{
15:            distMat[i-1][j-1] + $dist(T_1^i, T_2^j)$,
16:            distMat[i][j-1] + $C_{insCum}(T_2, T_2^j, root(T_2))$,
17:            distMat[i-1][j] + $C_{delCum}(T_1, T_1^i, root(T_1))$
18:         }
19:      end for
20:   end for
21:   return distMat[M][N]
22: end procedure
```

The original algorithm proposed by Nierman and Jagadish outputs only the edit distance between two trees. There is no way to reconstruct the optimal sequence of edit operations that led to the obtained final edit distance. To overcome that, all possible edit scripts that correspond to the minimal distance between the two trees are memorized. Therefore, an edit script (see Figure 6.5, Figure 6.6, and Figure 6.7) is defined as follows:

Definition 6.8 [Edit Script]: An edit script δ is an ordered sequence of edit operations that transform a source tree T_1 into a destination tree T_2.

In general, there exist more than a single edit script that correctly transform T_1 into T_2. For instance, if the cost of an alter operation is equal to the cost of a delete and an insert operation combined, both edit scripts achieve the same transformation cost. Further, edit scripts that initially add an arbitrary number of nodes and delete the same nodes at the end also carry out correct transformations. Therefore, a minimal edit script is defined as follows:

Definition 6.9 [Minimal Edit Script]: Let $\delta^1, \delta^2, \ldots, \delta^m$ be a set of correct edit scripts transforming T_1 into T_2. A minimal edit script is then defined as:

$$\begin{aligned} Min_{k=1..m}\{\delta^k\} = \delta^i \Longleftrightarrow \forall \delta^j \mid i \neq j, \\ dist(T_1, T_2)^{\delta^i} \leq dist(T_1, T_2)^{\delta^j} \end{aligned} \tag{6.5}$$

where $dist(T_1, T_2)^{\delta^j}$ indicates the distance between T_1 and T_2 obtained after applying the script δ^j. Note that there might be more than one minimal edit script for a pair of trees.

The above algorithm is further extended to calculate the minimal edit scripts for a (minimal) edit distance. First suggestions for altering Nierman's and Jagadish's algorithm in this manner come from Barnard et al. [25]. In addition to the *distMat* storing the cumulated edit distances only, a separate entry in the matrix *scriptMat* keeps track of the possible minimal edit scripts. After the algorithm finishes, $distMat[deg(T_1)][deg(T_2)]$ contains the minimal edit distance and $scriptMat[deg(T_1)][deg(T_2)]$ the set of minimal edit scrips for the edit distance.

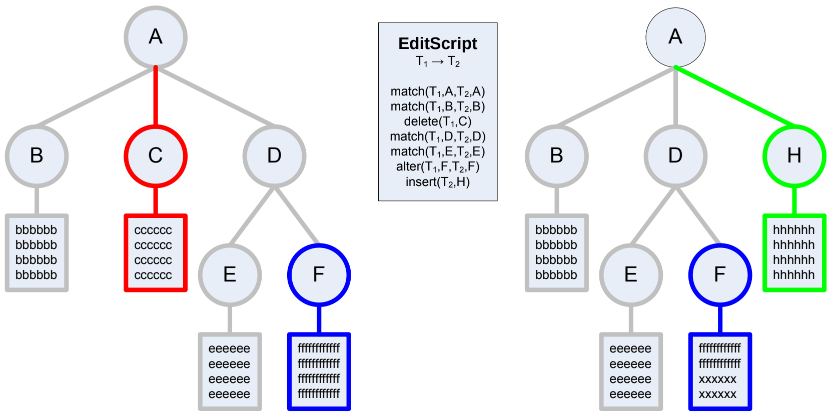
6.3.2 Content Matching

The second type of distance required is the content-based distance. To define it, this work still relies on Algorithm 6.1. However, the previously used cost functions have to be redefined in order to support the content match. Therefore, the notion of integer-valued costs is replaced by float values, allowing to use common content similarity measures.

Let $sim(\gamma(n_1), \gamma(n_2))$ be an existing similarity function that compares the contents of two nodes n_1 and n_2. This similarity can be computed by any information matching function. Assume that this similarity measure is normalized so that it takes values in the unit interval [0,1] (where $sim = 0$ means no match, $sim = 1$ indicates full match, and $sim \in]0,1[$ means partial match). Further, the following two definitions are needed:

$$sim(null, null) = 1 \tag{6.6}$$

$$sim(null, \gamma(n_2)) = sim(\gamma(n_1), null) = 0 \tag{6.7}$$


A
B
C
D
bbbbbb bbbbbb bbbbbb bbbbbb
cccccc cccccc cccccc cccccc
E
F
eeeeee eeeeee eeeeee eeeeee
ffffffffff ffffffffff ffffffffff ffffffffff
A
B
D
H
bbbbbb bbbbbb bbbbbb bbbbbb
E
F
hhhhhh hhhhhh hhhhhh hhhhhh
eeeeee eeeeee eeeeee eeeeee
ffffffffff ffffffffff xxxxxx xxxxxx

EditScript
$T_1 \rightarrow T_2$

match(T_1,A,T_2,A)
match(T_1,B,T_2,B)
delete(T_1,C)
match(T_1,D,T_2,D)
match(T_1,E,T_2,E)
alter(T_1,F,T_2,F)
insert(T_2,H)

Figure 6.7: Content-based tree edit operations and edit script

Equation 6.6 stipulates that the content-based similarity of two nodes with empty content is total (i.e., complete match), while Equation 6.7 stipulates that content-based similarity between a node with a content and another node with empty content is 0.
The cost functions for **inserting** and **deleting nodes with their contents** are defined as:

$$C_{insCon}(n) = 1 - sim(null, \gamma(n)) = 1 - 0 = 1 \tag{6.8}$$

$$C_{delCon}(n) = 1 - sim(\gamma(n), null) = 1 - 0 = 1 \tag{6.9}$$

The cost of altering the content, C_{altCon} (Equation 6.10) is the sum of the cost of changing the label $C_{alt}(n_1, n_2)$ (Equation 6.3) and the cost of altering the content, which is expressed as:

$$C_{altCon}(n_1, n_2) = C_{alt}(n_1, n_2) + \rho * (1 - sim(\gamma(n_1), \gamma(n_2))) \tag{6.10}$$

with ρ being a cost factor that scales up the dissimilarity of contents (since $sim(\gamma(n_1), \gamma(n_2)) \in [0, 1]$, $C_{alt}(n_1, n_2)$ can be larger than 1).

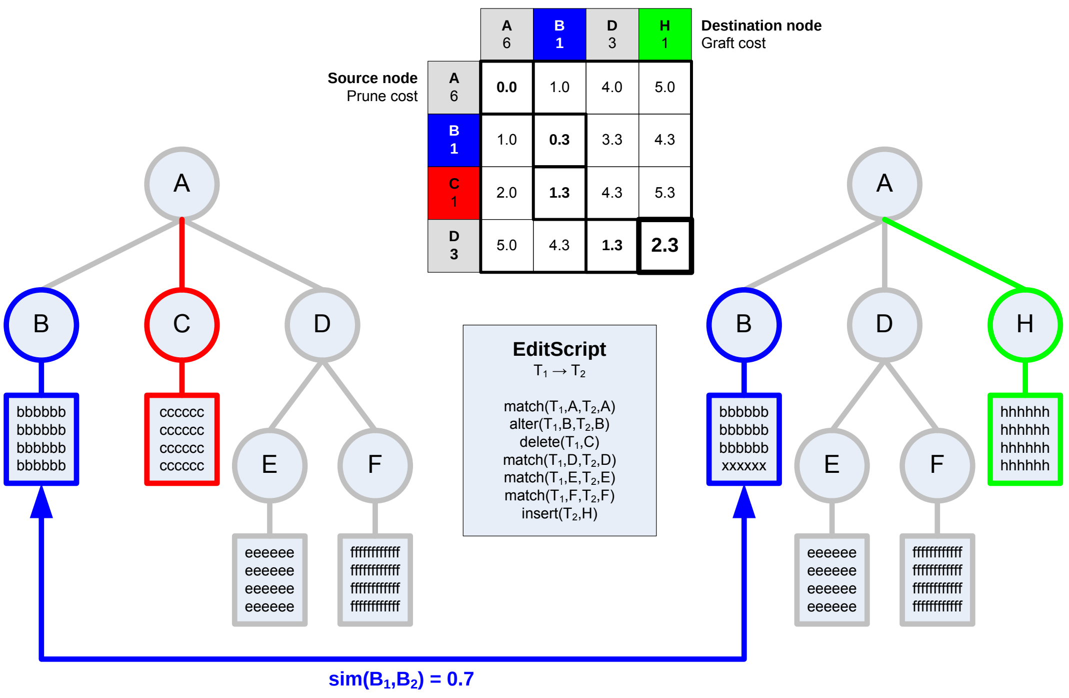
Depending on a similarity threshold α, an alter operation becomes cheaper than a delete operation and an insert operation combined. This can be reflected by a user-specified parameter, α ($0 \leq \alpha < 1$). Precisely, if $sim(\gamma(n_1), \gamma(n_2)) > \alpha$ then $C_{altCon} < (C_{delCon} + C_{insCon})$ (the alter is cheaper) and if $sim(\gamma(n_1), \gamma(n_2)) < \alpha$ then $C_{altCon} > (C_{delCon} + C_{insCon})$ (alter is more expensive); otherwise, alter has the same cost as that of both delete and insert combined. From this, the cost factor ρ is determined as follows:

$$\rho * (1 - \alpha) = C_{delCon}(n_1) + C_{insCon}(n_2) \tag{6.11}$$

leading to:

$$\rho = \frac{C_{delCon}(n_1) + C_{insCon}(n_2)}{(1 - \alpha)} \tag{6.12}$$

Now, to take the content similarity into account $C_{alt}(root(T_1), root(T_2))$ in Algorithm 6.1 (line 5) is substituted for $C_{altCon}(root(T_1), root(T_2))$. Figure 6.8 summarizes the behavior of Algorithm 6.1. $distMat[0][0]$ is initialized with the cost of altering A_1 into A_2. Row 0 is then completed by cumulating the inserting costs. Also, column 0 is completed by cumulating the pruning costs. The remaining cells are filled by the minimal cost after adding (1) the inserting costs to the neighboring left cell, (2) the deleting cost to the neighboring upper cell, or (3) the altering cost to the upper left cell. The final edit distance for transforming the source into the destination tree can be found in the bottom right corner of the distance matrix. The minimal edit script is computed in a separate $scriptMat$ matrix.


Destination node
Graft cost
Source node
Prune cost
A
B
C
D
E
F
A
B
D
E
F
H
bbbbbb bbbbbb bbbbbb bbbbbb
cccccc cccccc cccccc cccccc
eeeeee eeeeee eeeeee eeeeee
ffffffffff ffffffffff ffffffffff ffffffffff
bbbbbb bbbbbb bbbbbb xxxxxx
eeeeee eeeeee eeeeee eeeeee
ffffffffff ffffffffff ffffffffff ffffffffff
hhhhhh hhhhhh hhhhhh hhhhhh
sim(B₁,B₂) = 0.7

	A 6	B 1	D 3	H 1
A 6	**0.0**	1.0	4.0	5.0
B 1	1.0	**0.3**	3.3	4.3
C 1	2.0	**1.3**	4.3	5.3
D 3	5.0	4.3	**1.3**	**2.3**

EditScript
$T_1 \rightarrow T_2$

match(T_1,A,T_2,A)
alter(T_1,B,T_2,B)
delete(T_1,C)
match(T_1,D,T_2,D)
match(T_1,E,T_2,E)
match(T_1,F,T_2,F)
insert(T_2,H)

Figure 6.8: Tree edit distance algorithm matrix

One might note that the edit distance itself cannot be interpreted without a context, i.e., comparing two source documents with a single destination document, where one source document is more similar to the destination document than the other. Anyway, this algorithm provides a relative distance measure among XML documents based on their structure and content, which can be applied easily for classification and clustering.

In order to allow such an interpretation, a normalization of the edit distances for two trees can be defined as

$$dist_{normalized}(T_1, T_2) = \frac{dist(T_1, T_2)}{maxDist(T_1, T_2)} \tag{6.13}$$

where $maxDist(T_1, T_2) = |T_1| \cdot C_{delCon} + |T_2| \cdot C_{insCon}$ defines the maximum edit distance, calculated as deleting every node of T_1 and inserting every node of T_2. This definition allows to compare edit distances by measuring the amount of changes needed, incorporating the size of the compared documents.

6.4 Tree Matching via Content Matrix

So far, the focus has been on explicit consideration of the structure to compare documents. In the sequel a more content-oriented matching procedure is suggested that uses the structure of documents only implicitly. More expressively, the idea is to measure the similarity between documents using only nodes with contents. Indeed, every node in the source tree is compared to all nodes in the destination tree relying on a content similarity matrix (denoted *Content Matrix*), where only nodes containing content are taken into account.

A cell $contSim[i][j]$ refers to the similarity degree between the contents of the node n_i in the source tree and the node n_j in the destination tree, i.e., $contSim[i][j] = sim(\gamma(n_i), \gamma(n_j))$. One might define a two-step process to perform the comparison: If the labels of the nodes to be compared are the same, then the contents of these nodes are compared. However, the first step of this comparison can be optional at the discretion of the user via setting a flag variable ζ. If ζ is set to *true*, the similarity of nodes with non-equal labels is set to 0.

Once the content similarity matrix, *contSim*, is filled, the distance between the corresponding documents can be computed using an algorithm that traverses that matrix in a single pass. Basically, three edit operations: insert, delete, and alter are applied. During the computation, these operations are labeled either $safe$ (certain) or $unsafe$ (uncertain) as shown in Algorithm 6.2 and explained below. Basically the algorithm proceeds in two steps, (1) computing the similarities and (2) marking the nodes according to the edit operations (item numbers correspond to lines in Algorithm 6.2):

15: If $simMat[i][j] = 1$, then nodes $n_{1,i}$ and $n_{2,j}$ are both marked as $safe\_match$ with no additional cost.

16: A source node, $n_{1,i}$ having no matching destination nodes ($simMat[i][j] = 0, \forall j = 1 \ldots |dest|$) is marked as $safe\_delete$. The cumulative cost is increased by the weighed cost of $safe\_delete$.

17: A destination node, $n_{2,j}$ with no corresponding source nodes ($simMat[i][j] = 0, \forall i = 1 \ldots |source|$) is marked as $safe\_insert$. The cumulative cost is increased by the weighed cost of $safe\_insert$.

18: An unmarked source node $n_{1,i}$ is marked as $unsafe\_match$ along with one unmarked destination node $n_{2,j}$, if $n_{2,j}$ is the first node (minimal index j) fulfilling the condition $simMat[i][j] \geq \alpha$, where α is a user-specified similarity threshold.

19: Any remaining unmarked source node, $n_{1,i}$, ($simMat[i][j] < \alpha, \forall j = 1 \ldots |dest|$) are marked as $unsafe\_delete$.

20: Any remaining unmarked destination node, $n_{2,j}$, ($simMat[i][j] < \alpha, \forall i = 1 \ldots |source|$) are marked as $unsafe\_insert$.

Algorithm 6.2 Content Matrix Distance algorithm

```
1: procedure ContentDist($T_1, T_2, \zeta$)
2:   int M = $|T_1|$ /*only content nodes*/
3:   int N = $|T_2|$ /*only content nodes*/
4:   float[][] $simMat$ = new float[0..$M$][0..$N$]
5:   for ($i = 1$ to $M$) do
6:     for ($j = 1$ to $N$) do
7:       if ($\zeta = true$ and $\lambda(n_1) \neq \lambda(n_2)$) then
8:         $simMat[i][j] = 0$
9:       else
10:        $simMat[i][j] = sim(\gamma(n_{1,i}), \gamma(n_{2,j}))$
11:      end if
12:    end for
13:  end for
14:
15:  check for $safe\_match$
16:  check for $safe\_delete$
17:  check for $safe\_insert$
18:  check for $unsafe\_match$
19:  check for $unsafe\_delete$
20:  check for $unsafe\_insert$
21:
22:  return $\sum weighed\_costs$ (based on marks)
23: end procedure
```

As in the tree edit distance approach, each of the edit operations is associated with a certain cost. In addition, the labels $[safe]$ and $[unsafe]$ can be weighed. Since the distance is inversely proportional to the similarity, the transformation costs are calculated using the same formula as applied in the tree edit distance approach taking $dist(n_{1,i}, n_{2,j}) = 1,0 - contSim[i][j]$

into account. The final distance between the documents is the sum of all of the (weighed) operation's costs. Clearly, the computation of the edit script is done in linear time and space (since, there is no recursive computation).

Figure 6.9 shows how the algorithm works. All nodes containing content of the source tree (B, C, E, F, and G in Figure 6.9a) are compared to all nodes containing content of the destination tree (B, E, C', G, and C'' in Figure 6.9b). The corresponding similarity matrix $simMat$ is shown in Figure 6.9c. Additional data structures accelerate the identification of $[safe]$ matches ($rowMatch$), inserts ($colEmpty$), and deletes ($rowEmpty$). Also, $rowSim$ keeps track of the number of nodes exceeding the similarity threshold α. Figure 6.9d presents the final costs for all nodes, which are $4,1$ in sum. A special case is the multiple matches of source node C in the destination tree with C' and C''. It is resolved by matching C with the node corresponding highest, C''. As a single node is allowed to match only another single node, node C' automatically becomes marked as $[unsafe]$ insert.

Table 6.2 summarizes the described approaches, tree edit distance (TED) and content matrix (CM), and their supported document editing operations.

Table 6.2: Comparison of tree edit distance and content matrix

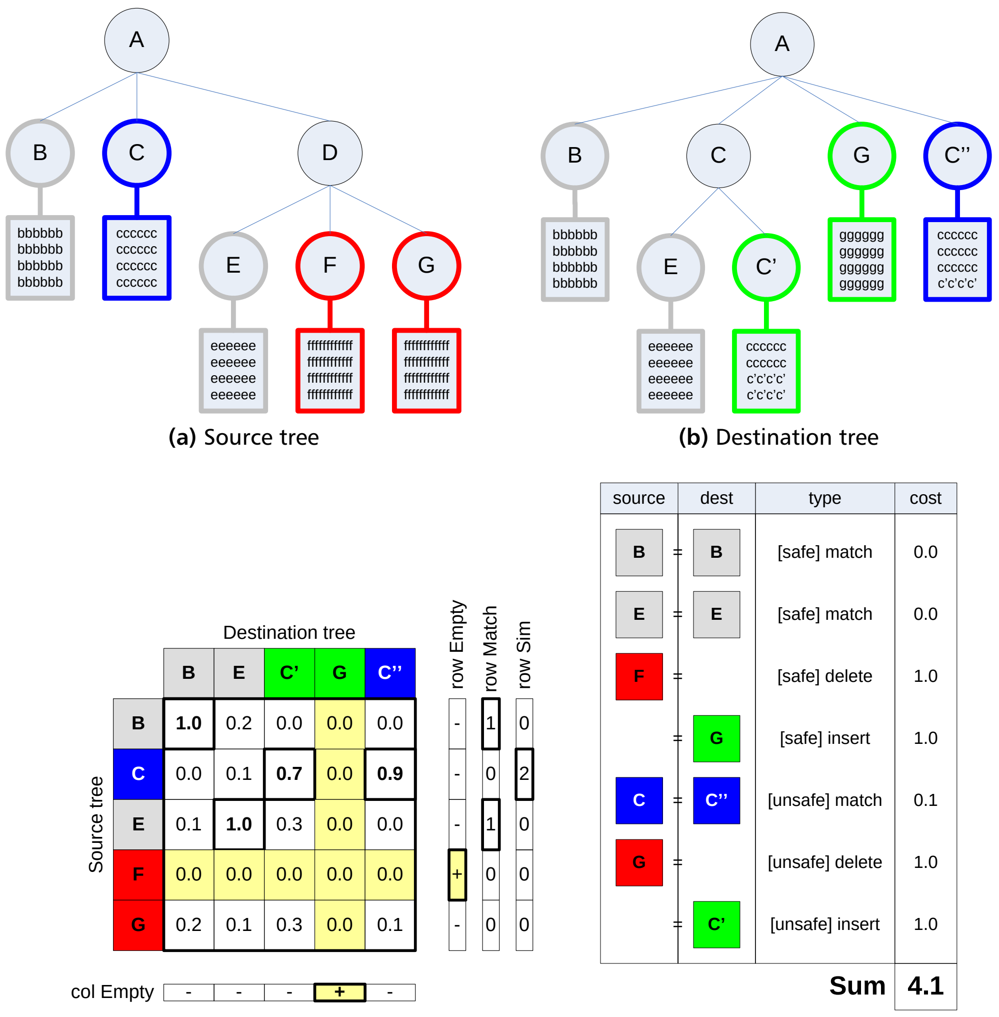
Edit operations	TED	CM
insert	+	+
delete	+	+
alter	+	+
move	edit script	+
copy	edit script	+
deepening	multiple matches	multiple matches
flattening	multiple matches	multiple matches

6.5 Overview of k-NN

The k-NN algorithm [68, 210] serves for classifying documents into pre-existing classes. It is based on the assumption that the classification of a sample is most similar to the classification of other samples that are nearby in the space. In Figure 6.10 the unknown data point is assigned to the red class, because 5 out of $k = 7$ nearest elements are assigned the red label.

Compared to other learning methods such as probabilistic classifiers, k-NN does not rely on prior probabilities. Further, it is known for its effectiveness [262]. The main computation task is to sort the training samples in order to find the k-nearest neighbors of a given query. It is then straightforward to apply the k-NN algorithm for classifying XML documents.

Its application consists of collecting the set of N training documents $X^T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where x_i are the XML documents and y_i are the class labels. This set is used as reference samples for the k-NN algorithm. To assign a label to



(a) Source tree **(b)** Destination tree

Destination tree / Source tree

	B	E	C'	G	C''	row Empty	row Match	row Sim
B	**1.0**	0.2	0.0	0.0	0.0	-	1	0
C	0.0	0.1	**0.7**	0.0	**0.9**	-	0	2
E	0.1	**1.0**	0.3	0.0	0.0	-	1	0
F	0.0	0.0	0.0	0.0	0.0	+	0	0
G	0.2	0.1	0.3	0.0	0.1	-	0	0
col Empty	-	-	-	+	-			

(c) Similarity matrix ($\alpha = 0,5$)

source	dest	type	cost
B	B	[safe] match	0.0
E	E	[safe] match	0.0
F		[safe] delete	1.0
	G	[safe] insert	1.0
C	C''	[unsafe] match	0.1
G		[unsafe] delete	1.0
	C'	[unsafe] insert	1.0
		Sum	**4.1**

(d) Cumulated costs

Figure 6.9: Content matrix match

an unlabeled document (query), *doc*, the algorithm finds the *k* documents in the reference samples (labeled documents) that are the closest to it. The label shared by the majority of these *k* nearest neighbors is assigned to the query.

However, to apply *k*-NN an appropriate value of *k* needs to be chosen, since the performance of the algorithm depends on this value.

Note that *k*-NN is a lazy learning algorithm because no model needs to be built a priori. Moreover, despite its efficiency problems, *k*-NN is known to be one of the most effective classification methods. Steps of the classification procedure via *k*-NN are summarized in Algorithm 6.3. Once the unlabeled XML document set is assigned class labels, the classification

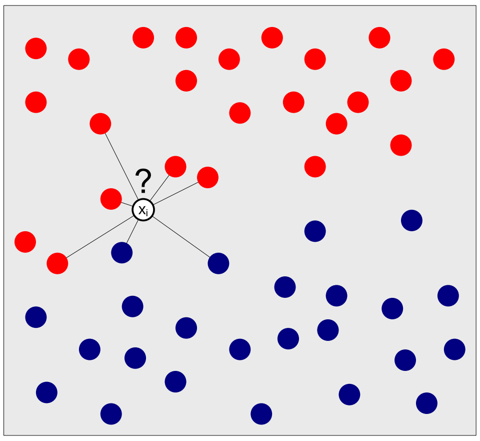


Figure 6.10: k-Nearest Neighborhood classification

Algorithm 6.3 Classification via the k-NN algorithm

```
1: $X^L$ ... labeled documents, $\mathcal{C}$ ... number of classes such that $\bigcup_j X_j = X^L$, $j = 1...\mathcal{C}$
2: $X^U$ ... unlabeled documents
3: procedure $k$-NN($k$)
4:     for (i = 1 to $|X^U|$) do  //$x_i$ ... unlabeled sample
5:         for (j = 1 to $|X^L|$) do  //$y_j$ ... labeled sample
6:             $dist[j] = sim(x_i, y_j)$
7:         end for
8:         Sort $dist[j]$ ascending
9:         Select first $k$ documents ($\in X^L$ with smallest distance)
10:        Assign $x_i$ the wining label $c_l$ that occurs most often among the $k$ documents
11:        If more than one label wins, randomly select one of them
12:    end for
13: end procedure
```

accuracy can be computed. It measures how often the algorithm's labeling decision meets the actual labels of the documents.

6.6 Evaluation

In this section, several aspects to quantify the classification quality of XML documents based on their content and structure are studied: How do different k values influence the classification? What is the impact of training size on the classification performance? How does content and structure matching perform compared to structure-only matching? All of these issues were investigated using some real-world XML collections based on the movie database

(MovieDB) [65]) which were proposed in INEX 2005 [90]. Documents of these collections are assigned to 11 classes. Basically, the XML collections are of two types:

- Structure-only (*SO*) collections containing only the structure of the XML documents. These include four collections: *m-db-s-0, m-db-s-1, m-db-s-2, m-db-s-3*, where the last three collections are noisy versions of the first one. The amount of noise and class overlap increases from the first to the last collection. The collections come in the form of two sets, a training and a testing set. The training documents of the *m-db-s* data sets are organized in 11 categories, where each category corresponds to a movie genre. Table 6.3 summarizes the number of training and testing documents of each of the collections.

Table 6.3: Structure-Only corpora

Corpus	Train/Test 10%	Train/Test 30%	Train/Test 50%	Train/Test 70%	Train/Test 100%
m-db-s-0	488/485	1.453/1.448	2.415/2.409	3.383/3.376	4.824/4.816
m-db-s-1	487/486	1.449/1.447	2.410/2.408	3.378/3.374	4.818/4.814
m-db-s-2	486/485	1.450/1.447	2.412/2.408	3.379/3.372	4.820/4.809
m-db-s-3	488/485	1.453/1.445	2.414/2.404	3.380/3.370	4.821/4.809

- Content-and-structure (*CAS*) collections consisting of the entire documents. This set contains one collection, *m-db-cs-1*, that consists of 4.825 labeled documents assigned to 11 different categories. This set is split into 2.415 training and 2.410 testing documents. Table 6.4 summarizes the number of training and testing documents of the collection.

Table 6.4: Content-and-Structure corpora

Corpus	Train/Test 10%	mccTrain/Test 20%	Train/Test 30%	Train/Test 50%	Train/Test 70%	Train/Test 100%
m-db-cs-1	247/246	488/486	730/728	1.210/1.208	1.696/1.692	2.415/2.410

To answer the questions formulated earlier, three sets of experiments are run. The first two deal with the structure-only setting, while the last one is concerned with the content-and-structure setting. A comparison of the results against some available results from other authors is highlighted at the end of this section.

In all experiments, evaluation relies on the well-known classification accuracy [224, 14, 261], that is how often the classifier's decision meets the expert's assignment. Formally it is defined as:

$$\text{Accuracy} = \frac{\#\text{ correctly classified testing docs}}{\#\text{ testing docs}} \tag{6.14}$$

In addition to the classification accuracy, recall and precision values [224] are computed for comparison. Recall, R_i, and precision, P_i, for a class i are given as follows:

$$R_i = \frac{\text{\# correctly classified testing docs in class } i}{\text{\# testing docs of class } i} \tag{6.15}$$

$$P_i = \frac{\text{\# correctly classified testing docs in class } i}{\text{\# testing docs assigned to class } i} \tag{6.16}$$

The overall recall and precision of a classification can be computed using two different methods: Micro averaging (R_{micro} and P_{micro}) sums over all individual decisions, thus gives equal weight to each assignment. Macro averaging (R_{macro} and P_{macro}), instead, sums over all classes, thus gives equal weight to each class. The formulae are given as follows:

$$R_{micro} = \frac{\sum_{i=1}^{k} R_i \cdot n_i}{n} \tag{6.17}$$

$$P_{micro} = \frac{\sum_{i=1}^{k} P_i \cdot n_i}{n} \tag{6.18}$$

$$R_{macro} = \frac{\sum_{i=1}^{k} R_i}{k} \tag{6.19}$$

$$P_{macro} = \frac{\sum_{i=1}^{k} P_i}{k} \tag{6.20}$$

n_i is the number of documents assigned to class i, n is the total number of documents, and k is the number of classes.

6.6.1 Experiment I - How Does k Affect the Accuracy?

As explained in Section 6.5, the size of the neighborhood (k) is a key parameter in the k-NN algorithm. Therefore, one aspect to look at is to check the effect of k on the accuracy. For this purpose, the values 3, 5, 7, 9, 15, and 21 are experimented. Note that, only the *SO* collections are used and only a proportion (10%) of them is selected randomly and uniformly distributed over the 11 classes to show the effect of k.

Using Algorithm 6.1 to run k-NN, and setting the required parameters to $\alpha = 0,5$ (Equation 6.12), $C_{del} = 1,0$ (Equation 6.4), $C_{ins} = 1,0$ (Equation 6.4), and $\beta = 2,0$ (Equation 6.3, $\beta \geq C_{ins} + C_{del}$ to avoid permanent node relabeling) respectively, the results displayed in Table 6.5 and Figure 6.11 are obtained.

The outcome of the experiment is a 3-fold conclusion: **(i)** as k increases, the accuracy of the algorithm monotonically decreases independently of the collection used (except for $k = 9$ in *m-db-s-1* and for $k = 5$ in *m-db-s-3*), **(ii)** the noise introduced in the *m-db-s-1/2/3* has negatively impacted the accuracy (as the amount of noise in relation to *m-db-s-0* increases, the accuracy decreases), and **(iii)** the maximum drop in the accuracy due to variation of k is

Table 6.5: Effect of k on the accuracy (Equation 6.14)

Corpus	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$	max. difference
m-db-s-0	0,922	0,920	0,918	0,903	0,892	0,889	3,3%
m-db-s-1	0,903	0,892	0,891	0,897	0,885	0,866	3,7%
m-db-s-2	0,874	0,868	0,858	0,849	0,802	0,790	8,4%
m-db-s-3	0,862	0,868	0,860	0,852	0,825	0,814	4,8%

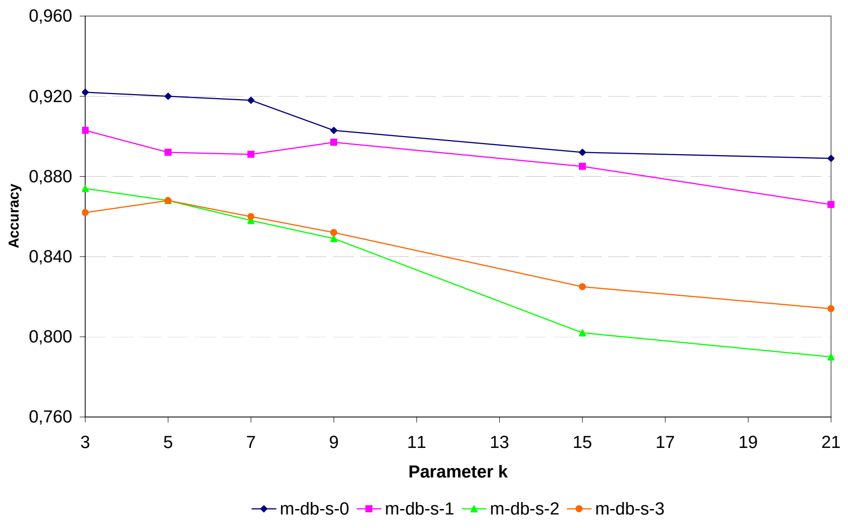


Figure 6.11: Effect of k on the accuracy

quite low, lying between 3,3% and 8,4% only (see last column of Table 6.5). Therefore, the different values of k are retained in the remaining experiments since these results do not allow to convincingly consider a particular k-value better than the others.

More interesting, the classifier provides very high accuracy results, but this remains relative to the amount of documents used in this experiment.

6.6.2 Experiment II - How Does the Training Data Affect the Accuracy?

k-NN uses a set of training samples as a basis to label unseen documents. Hence, it is clear that the size of the training set is crucial for the accuracy of the algorithm. To observe the effect of increasing the size of the training data set, the *m-db-s-0* collection is split into five ratios (10%, 30%, 50%, 70%, 100%). These ratios are randomly and uniformly selected among the whole training data so that every chunk contains all labels.

Using the same setting as described in Section 6.6.1, the results shown in Table 6.6 and Figure 6.12 are obtained. Unexpectedly, the size of the training set had no large impact on the accuracy of the classifier (see the last column of the table). There is an extremely weak trend that a larger training set improves performance. The reason might lie in the inter-document similarity, meaning that the classes are highly homogeneous. Furthermore, the accuracy remains in the same range of values (regardless of the k value) without noticeable fluctuations when increasing the size of the training data.

Table 6.6: Effect of the training data on the accuracy

Corpus	$k = 3$	$k = 5$	$k = 7$	$k = 9$	$k = 15$	$k = 21$	max. difference
10%	0,922	0,920	0,918	0,903	0,892	0,889	3,3%
30%	0,928	0,925	0,934	0,932	0,930	0,930	0,9%
50%	0,924	0,929	0,928	0,929	0,929	0,924	0,5%
70%	0,934	0,933	0,932	0,930	0,930	0,932	0,4%
100%	0,934	0,934	0,932	0,932	0,929	0,931	0,5%

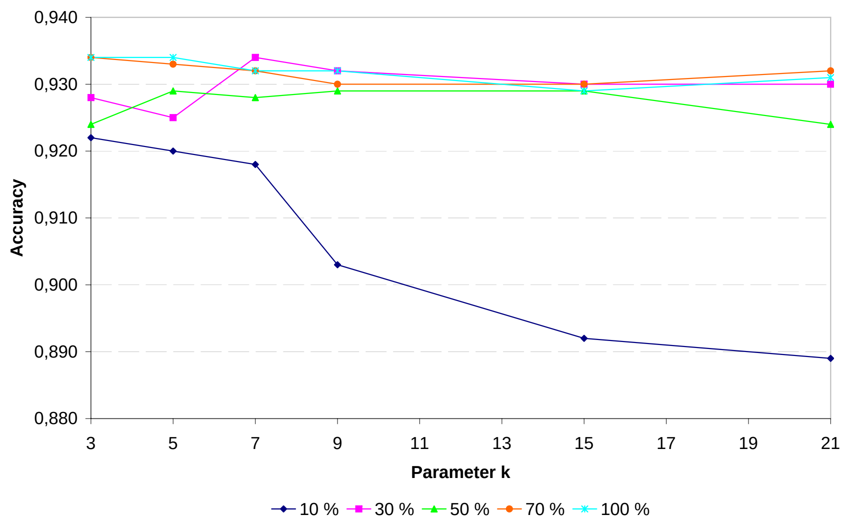


Figure 6.12: Effect of the training data on the accuracy

6.6.3 Experiment III - How Does the CAS Setting Affect the Accuracy?

To check the effectiveness of the approach taking both the content and the structure of XML documents into account, the *CAS* collection (*m-db-cs-1*) described earlier is used and five methods are applied. These methods include:

Method	Description
BM	A Boolean model [21] is applied as in traditional information retrieval, where documents are represented as a bag of words and where structure is neglected.
TED_SO*	Tree edit distance based on structure-only matching (see Section 6.3.1). Parameters are set to $\alpha = 0,5$, $\beta = 2,0$, $C_{del} = 1,0$, and $C_{ins} = 1,0$.
TED_CAS	Tree edit distance based on content-and-structure matching (see Section 6.3.2). Parameters are set to $\alpha = 0,5$, $\beta = 2,0$, $C_{del} = 1,0$, and $C_{ins} = 1,0$.
CM_match	Content matrix matching where node labels are considered during comparison (see Section 6.4).
CM_any	Content matrix matching where node labels are ignored during comparison (see Section 6.4).
TED_CM	A combination of tree edit distance based on structure-only matching (TED_SO) and content matrix matching where node labels are ignored (CM_any). Although CM_match outperforms CM_any, it was not combined with TED_CM. This was because both methods, CM_match and TED_SO, are structure-oriented, which would have given unfair weight to structural similarity. The final distance of two document trees is computed as $\alpha \cdot TED\_SO + (1-\alpha) \cdot CM\_any$, where α=0,5

* Note that the TED_SO method does not use the document contents, but it is included here for the purpose of comparison.

These methods are run on 20% of *m-db-cs-1* to obtain the results displayed in Table 6.7 and Figure 6.13.

Table 6.7: Effect of CAS on the accuracy

Approach	$k=3$	$k=5$	$k=7$	$k=9$	$k=15$	$k=21$
BooleanModel	0,327	0,352	0,352	0,331	0,335	0,313
TED_SO	0,916	0,907	0,895	0,895	0,856	0,860
TED_CAS	0,652	0,634	0,640	0,673	0,584	0,558
CM_match	0,352	0,360	0,305	0,309	0,296	0,272
CM_any	0,130	0,163	0,175	0,160	0,134	0,140
TED_SO+CM_any	0,909	0,907	0,897	0,893	0,862	0,860

Although *m-db-cs-1* and *m-db-s-0* are different form each other, the accuracy of TED_SO on both collections (when using the structure only) is nearly the same and exceeds 90% accuracy. More surprising is the fact that the BM, CM_match, and CM_any methods perform worse. Furthermore, comparing the tree-edit methods TED_CAS and TED_SO, it is worth concluding that the content deteriorates the accuracy. Indeed the difference in the accuracy

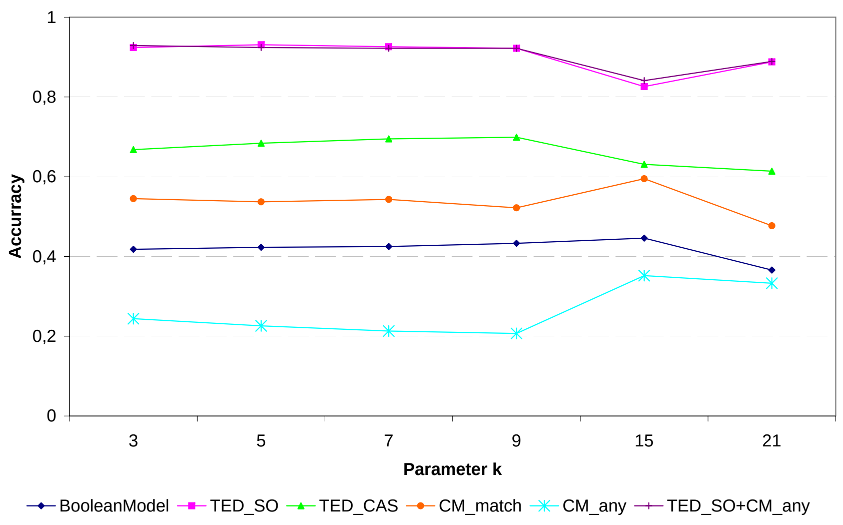


Figure 6.13: Effect of CAS on the accuracy

values of the two methods are significant: 26% ($k = 3$), 27% ($k = 5$), 25% ($k = 7$), 22% ($k = 9$), 23% ($k = 15$), and 31% ($k = 21$). This is also true when ignoring the structure entirely, as with the BM method or when using the content matrix described in Algorithm 6.2 and relying on CM_match and CM_any. The accuracy deterioration in this case is much worse. More consistent with the expectations, a combination of TED_SO and CM_any, which results in TED_CM, obtains best classification results regarding the content and the structure of document trees. It considers the content while assuring high classification accuracy of the structure-oriented method.

It is clear from these experiments that the XML documents classification is more influenced by the document's structure than by its content. However, this could be true only for particular collections where the content is relatively poor as is the case of the MovieDB.

6.6.4 Comparison

Because a range of methods is provided, it is relevant to check how these methods compare to the state-of-art methods that have been applied on the same collection. To do that, two references appearing in the INEX 2005 workshop are considered. The first is by Hagenbuchner et al. [118] who applied contextual self-organizing maps for structured data (CSOM-SD) to classify XML documents, and the second is by Candillier et al. in [43] who applied inductive decision trees (IDT).

To compare these methods against those proposed in this work, the same evaluation metrics, accuracy, recall, and precision (at the macro and micro levels) [224] as shown in

Table 6.8 have to be used. Note that just the best performance rates achieved by each method are considered.

Table 6.8: Classification Comparison for *m-db-s-0*

Approach	Accuracy	Micro Recall	Macro Recall	Micro Precision	Macro Precision
CSOM-SD	0,873	-*	-*	-*	-*
IDT	-*	0,968	0,960	-*	-*
TED_CM	0,934	0,934	0,934	0,937	0,911

* '-': means value not available

The results illustrate that TED_CM largely outperforms CSOM-SD in terms of accuracy by a rate difference of 6%. However, when considering micro and macro-recall, the IDT approach performs better than TED_CM. Unfortunately, recall without precision is not much telling. The precision values achieved by TED_CM are very encouraging especially when taking recall values into account.

6.7 Future Extensions

The algorithms for computing the edit distance of two document trees lack an intuitive applicability of standard document editing. Altering a document is generally performed by applying more complex transformations than the basic operations mentioned above [25]. Operations like inserting or deleting whole subtrees (e.g., chapters), moving of structures, and changing of contents have to be considered. Therefore, altering, merging, splitting, raising, and lowering of contents or parts of contents is of importance. Some of these operations can be modeled using combinations of the basic operations, where the main focus lies on the content of a node (i.e., merging of nodes contents).

Incorporating such additional operations in the edit distance algorithm would increase the computational complexity by far, leading to unacceptable runtime performance. According to Barnard [25] these kinds of transformation operations should be handled as a post-processing task of the minimal edit scripts. Here, the inherent order of the edit operations and proper node matching strategies can be exploited, which allows a substitution of basic operations by more elaborated ones:

- *deleteTree*(T_i, n, p): If all descendant nodes of a node are deleted, the delete operations can be replaced by a single *deleteTree* operation, where p is the parent of the deleted subtree rooted in node n in the document tree T_i.

- *insertTree*(T_i, n, p, j): If all descendant nodes of a node are inserted, the insert transformations can be replaced by a single *insertTree* operation, where p is a parent node where node n is inserted as the j-th child in the document tree T_i.

- *moveTree*$(T_i, n_{old}, p_{old}, n_{new}, p_{new}, j)$: Utilizes a combination of *deleteTree*(T_i, n_{old}, p_{old}) and *insertTree*$(T_i, n_{new}, p_{new}, j)$.
- *moveNode*(T_i, n, p, j): Moves the node n to the j-th child of node p in the document $T - i$. More elaborated move operations may include *raiseChild* and *lowerChild*.
- *moveContent*(T_i, n_1, n_2): Deletes the content of node n_2 and appends it to the content of node n_1 in the document tree T_i.

These additional operations reflect document authoring tasks better than the basic edit operations for inserting, deleting, and altering of single nodes. Different changes could be weighted differently, which allows to focus on operations at a higher level. At the same time, handling of these operations in a post processing step keeps up its processing performance.

6.8 Summary

This chapter discusses an XML classification approach based on the k-nearest neighborhood algorithm which relies on edit distance measures. The originality of the approach lies in the edit distance similarity which considers both, the content and structure of XML trees. Initial results indicate that the approach proposed is very promising on both tasks, structure-only and content-and-structure matching. One of the main findings is that the structure bears more weight than the content does. However, a combination of two methods among the proposed ones allows to tune the weight of both the content and the structure. Further empirical work using other document collections is certainly needed. Especially in the context of structured document retrieval, the benefits of automatic classification of documents or parts of documents have to be investigated.

Chapter 7

If you want truly to understand something, try to change it.

Kurt Lewin

Clustering of XML Documents

Unsupervised clustering is a means to organize documents into groups of similar documents, called clusters, automatically. These clusters can be exploited for information retrieval in two ways: restriction of search space, and improvements of retrieval performance because of inter-document similarities. Several methodologies have been applied in the context of unstructured document retrieval, including neural networks, genetic algorithms, or genetic programming. Unfortunately, there is only little work done on the adaptation of these techniques to fit the structured document retrieval paradigm. In this chapter a new approach for clustering structured documents is proposed. Clusters are represented in form of supertrees that comprise the features of all documents assigned to a cluster. The approach is evaluated with the INEX corpus used in 2005.

7.1 Introduction

Whereas document classification assigns documents to predefined classes, document clustering discovers new classes – in this context called clusters – on its own. This is a major advantage, because creating a detailed classification framework (e.g., a hierarchy of classes) consumes much time and cost. Clustering, instead, identifies relationships among documents based on their similarity to each other. The goal is to group similar documents in the same cluster. This feature can be exploited for information retrieval: Due to the cluster hypothesis [246, 69, 249] "closely associated documents tend to be relevant to the same requests". In contrast of comparing a user query to thousands of documents, a query is compared to a much lower number of document clusters. Documents lying in relevant clusters are returned to the user, who eventually adds additional constraints to refine the result. Another advantage is a browsable hierarchy of clusters. Although clusters cannot be meaningfully named automatically, it allows to restrict searches to subsets of similar documents. Thus, retrieval becomes more effective, as well as more efficient.

Besides the content of documents, the logical structure provides important information that can be exploited by a clustering machinery. This latter is a significant mechanism in the context of XML retrieval, because the number of XML components highly outnumbers the number of documents in flat retrieval systems. Therefore, reducing the number of comparisons needed to answer user queries is crucial for retrieval systems.

As with classification, clustering of XML components can be basically done in three ways: (1) using exclusively the textual content, (2) using exclusively the structure, and (3) using both, content and structure, in a hybrid manner. The previous chapter showed that classification is more influenced by the document's structure than by its content. However, applying clustering for information retrieval, the content being searched must be incorporated. Thus, the focus of this work rests on the latter approach.

Two different clustering approaches, the *k*-Means clustering algorithm [169, 199, 147] and a hierarchical clustering algorithm [97] are applied. Both approaches, which are explained later, depend on an appropriate cluster representation. Therefore, a new supertree approach is proposed. The supertree representation of a document cluster contains the combined information of all documents that are assigned. In order to compute the similarity of a document and a cluster, the supertree is searched if the document is somehow included in it. This chapter presents the formulae that are needed to calculate the similarity of an ordered or unordered document and a supertree. This similarity computation incorporates both, the content and the structure of the documents.

The remaining chapter is organized as follows. First, related work about clustering of XML documents is reviewed in Section 7.2. The proposed supertree representation is described in Section 7.3. The similarity computation of documents and supertrees are covered in Section 7.4. The two clustering approaches are presented in Section 7.5. In Section 7.6, an experimental evaluation of the proposed approaches is outlined. Finally, some remarks conclude this chapter.

7.2 Related Work

In [57], a clustering approach based on structural similarities of XML documents is devised. As in most prototype-based clustering algorithms, the problem is to find an XML cluster representative for each cluster. This prototype is an XML document that encloses the structural commonalities of documents assigned to its cluster. In other words, a cluster's prototype is a proper overlap among all the documents within that cluster. Departing from the cluster hypothesis, such a prototype-based approach is worth applying in the context of XML retrieval.

The approach relies on agglomerative hierarchical clustering. Initially, all XML documents form the set of single element cluster centers. These are iteratively compared and the pair with smallest distance is aggregated to form a new cluster with a new prototype. The aggregation process continues until all XML documents have been merged into a single cluster. The basic problems with this approach are the definition of a distance measure to perform the comparison between pairs of clusters and the aggregation operation on the prototypes of clusters having high structural similarity. As to the distance measure, the authors have used a derived distance from the Jaccard coefficient:

$$dist(d_1, d_2) = 1 - \frac{|path(d_1) \cap path(d_2)|}{\max\{|path(d_1)|,\ |path(d_2)|\}}$$

where $path(d_i)$ indicates the set of structural paths in the document d_i. The second aspect of the algorithm is the aggregation procedure which consists of three steps: matching, merging, and pruning. Matching aims at building a common subtree out of the pair of prototypes of the selected least dissimilar clusters. The merge operation aims at uniting the pair of prototypes which can also be obtained using the matching tree. Pruning aims at discarding nodes from the merge tree such that the distance between the refined merge tree and all documents gets minimal.

In [71], a clustering approach based on time series is presented. There, the structure of an XML document is represented as a time series. Each occurrence of a node label (tag) is modeled as an impulse. The clustering algorithm uses the frequencies of the Fourier transform of the resulting time series to group similar documents into clusters.

In [163], a hierarchical clustering algorithm is applied to cluster XML documents based on their structure. This algorithm, originally proposed in [114, 115] under the name ROCK, uses the Jaccard coefficient to compute the similarity between two documents in terms of overlap between their sets of elements. At the heart of the ROCK algorithm, a criterion function that relies on the Jaccard coefficient is maximized so that strongly overlapping documents will be assigned to the same cluster.

Dalamagas et al. [60] provide a nice overview on the application of edit distance in the context of XML similarity. The paper summarizes some of the known edit-distance algorithms [225, 266, 49, 47] which are dedicated to ordered trees. It also proposes a similar algorithm to that presented in [47]. This latter is then used to perform a single link hierarchical clustering of XML documents relying on the minimum spanning tree technique. There, the weights on the edges correspond to the structural distance computed by the edit-distance algorithm.

In [43], a divisive hierarchical algorithm is proposed to classify and/or cluster XML documents. The algorithm uses five sets of attributes (tags, relationship of type parent-child, relationship of type next-sibling, node positions, paths starting from the root). During the split of a cluster, the relevance of this operation is measured using a quality criterion on each attribute set. Such a criterion, called interest, is the ratio of the log-likelihood of the partition with two clusters to the same partition but only with one cluster.

Francesca et at. [78] propose a method to efficiently compute a cluster representative from a set of XML documents. Their approach includes three computation steps, matching tree computation, merging of matching trees, and pruning. Based on a nodes' name, depth level, and parent node relation, meaningful matches between two document trees are identified recursively. The process starts at the leaf nodes of the documents, identifying the set of common XML paths up to the roots. The tree that consists of the common paths is called matching tree, reflecting the common skeleton of both documents. Initially, each document forms a single cluster. Relying on an edit distance, the most similar pair of clusters is merged. The merged cluster representative is then given by the matching tree of the two initial clusters. Final cluster representatives are obtained by pruning the least frequent nodes. Therefore, each leaf node is tried to be removed from the cluster without decreasing the distance to the unpruned tree. If the distance is not dropping, the current node is pruned. The process ends if no further nodes can be removed. The work proposed focuses on structure-based clustering, thus no content is considered. Further, two documents with different root nodes (different labels) cannot be merged into a cluster representative.

7.3 Supertree Representation

This work relies on the definition of a supertree, S, to represent the center of a cluster. Thus, the supertree contains the structural information and textual content of all documents that form the cluster. This set of documents is denoted as $\mathcal{C}$. The supertree S is an augmented tree and includes information about:

- An artificial $Root'$ node to which different root nodes of individual documents from $\mathcal{C}$ are attached. Let this be called 'super-root'. This node is flagged with the number of documents in $\mathcal{C}$, denoted as $freq$.
- Document-related nodes, N, (represented as ellipses) which occur in individual documents. A document-related node, $N_k \in N$, is defined by a label $lab = \gamma(N_k)$ and a content $\lambda(N_k)$. For instance, node A' and C' in Figure 7.1 are document-related nodes in $\mathcal{S}$. In addition, such nodes are flagged with $freq$, the number of their occurrence in the set of documents, $\mathcal{C}$, that form $\mathcal{S}$. For the sake of distinction, nodes of the supertree are primed (i.e., node A' in the supertree corresponds to a node A in a document). The content of a document-related node in the supertree is obtained by means of a content

merge operation (of the representations) applied on the corresponding nodes in the documents.

- Descriptive nodes, $\mathcal{D}$, (graphically represented as rectangles) lying on the outgoing edges (downwards) specify some statistical information about the parental relationship between the interconnected nodes. Let A' be the parent node of B' in the supertree, S. The corresponding descriptive node, D_j, contains:
  - □ an ordinal number indicating that B appears as the i^{th} child of A, called *ord*.
  - □ the number of times the node B (corresponding to B') appears as the i^{th} child of A in the set of documents, $\mathcal{C}$, that form S. This number is referred to as $freq$.

  Note that the 0^{th} descriptive node indicates how many times the corresponding (parent) document-related node has no children. The descriptive node attached to the 'super-root' contains only $ord = 1^{st}$ with the same $freq$ as that of 'super-root', meaning that each XML document must have a root node.

For the sake of illustration, Figure 7.1 shows a supertree example created from five documents (as indicated by $freq = 5$ of the 'super-root'). In three documents A occurs as a root node, and the remaining two documents are rooted in B. As shown, node A appears once without children ($ord = 0^{th}$), with a first child ($ord = 1^{st}$) twice (either B or C, both without further children), and with a second child ($ord = 2^{nd}$) once (only D). In the figure, $freq = 3$ of node A' is not equal to the sum of $freq$s of the descriptive child nodes (= 4). This is explained implicitly by the definition of the descriptive nodes, where the existence of a 2^{nd} child implies the existence of a 1^{st} child.

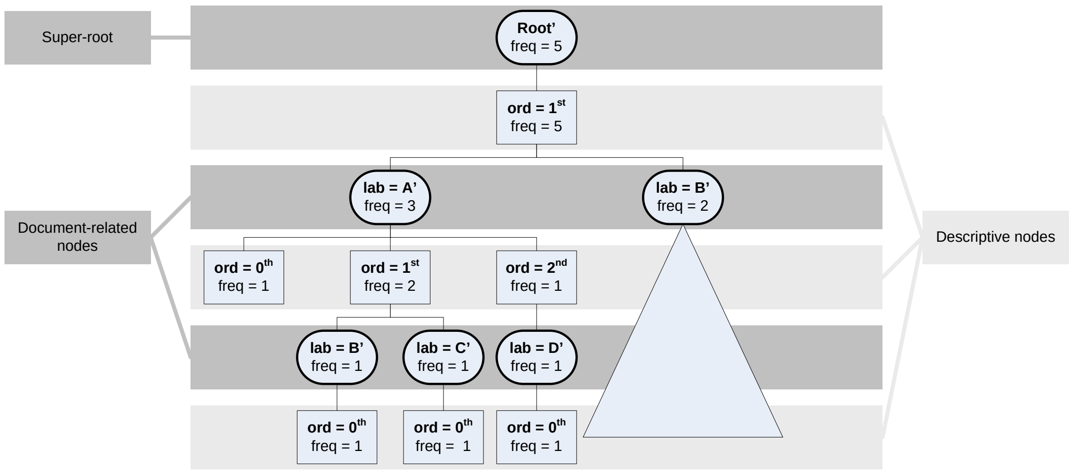


Figure 7.1: Excerpt of a supertree

The proposed supertree representation has the following features:

- XML documents with different root nodes can be represented in a single supertree.
- The order of documents merged is irrelevant.
- An individual document can be removed from the supertree provided that the content of its nodes can be recovered from the document-related node's contents of the supertree.
- Relying on merge and removal operations, supertrees can easily be restructured without the need for creating new representatives starting from scratch.
- Supertrees can be merged and/or split in order to perform a hierarchical clustering of both types agglomerative and divisive.
- Adding/removing documents from the supertree is linear in time and space. Thus, it can be computed efficiently.
- Supertrees are efficiently compared to documents and other supertrees, as described in the next section.

This work adopts the supertree representation for clustering complete documents, where supertrees are created from the XML-structured document trees. The same approach is also applicable for clustering subtrees of documents (e.g., all sections at the same level, all paragraphs, etc.).

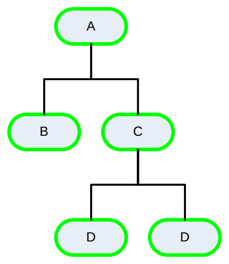
7.3.1 Creation of a Supertree

In order to build the supertree from a set of documents, the documents are incrementally added. Hereby, the order of documents added is not significant. Figures 7.2 depicts how documents d_1, d_2, and d_3 are bound to the supertree S. The $\oplus$ operation indicates the merge of a new document d with an existing supertree S. The labels of document-related nodes and ordinal numbers of descriptive nodes are written in bold letters. The number in brackets refers to $freq$.

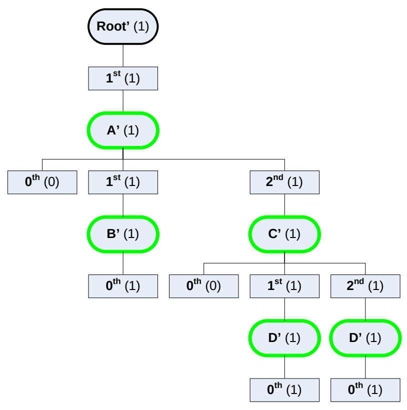
7.3.2 Merging of Supertrees

Supertrees contain pure statistical information about the structure and the content of a set of documents. Thus, not only additional documents but also other supertrees can be integrated in an existing supertree. This is the basic requirement for applying hierarchical clustering, which is described in Section 7.5.2. Clustering of documents or parts thereof partitions the set of input XML trees into disjunct clusters. Thus, the assumption that documents are included exclusively in one of the two supertree representations is valid. This fact allows to define the merging process as a cumulation of structural statistics and content merges.

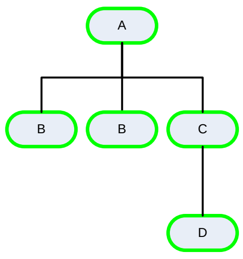
Merging one supertree ST_1 with another supertree ST_2 is done by a single preorder traversal of the second supertree ST_2. No difference between document-related nodes and descriptive nodes is made. Each node $N_{2,j}$ of ST_2 is compared to each node $N_{1,i}$ of ST_1. If



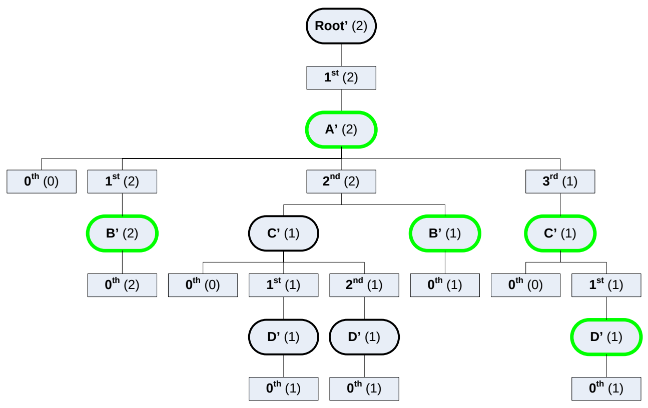
(a) Document d_1



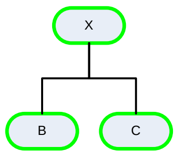
(b) Supertree $S = d_1$



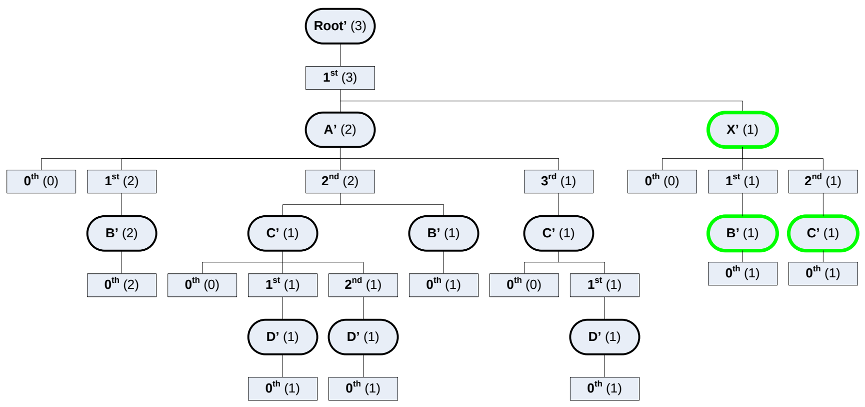
(c) Document d_2



(d) Supertree $S = d_1 \oplus d_2$



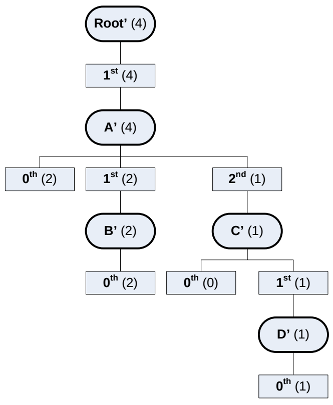
(e) Document d_3



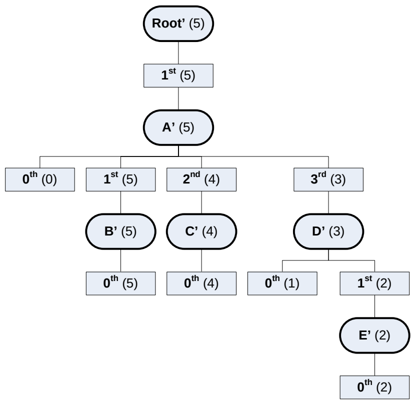
(f) Supertree $S = d_1 \oplus d_2 \oplus d_3$

Figure 7.2: Supertree creation

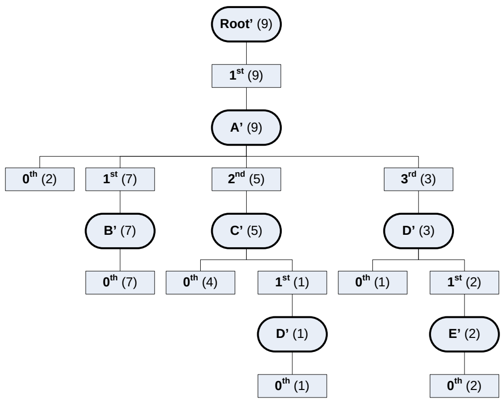
the node matches, the *freq* of $N_{1,i}$ is incremented by the *freq* of $N_{2,j}$. Otherwise, $N_{2,j}$ is added to the parent node of $N_{1,i}$ as a child node. Note that the order of child nodes of document-related nodes (descriptive nodes) is maintained. The same procedure is applied recursively to the child nodes. As the creation of a supertree from documents, the merging procedure is unaffected by the order of the merged supertrees. Figure 7.3 illustrates the merging process of two supertrees.



(a) Supertree ST_1



(b) Supertree ST_2



(c) Merged supertree $ST_1 \oplus ST_2$

Figure 7.3: Merging two supertrees

7.4 Similarity Computation

Supertree representations allow to compare a single document to a group of documents represented as a single supertree. Further, two supertrees can also be compared to each other. This section provides the formulae that compute the similarity of (1) an ordered document and a supertree, (2) an unordered document and a supertree, and (3) two supertrees.

7.4.1 Comparing Ordered Document Trees and Supertrees

In order to compare an ordered document to a supertree, the first similarity to be defined is between a document-related node A' in the supertree and a regular node B in the document.

Therefore, two similarities – one for the label and one for the content – have to be defined. A *labelSim* function is used to define the similarity among labels, returning values between 0 and 1. For instance, a simple function may return either 1 (equal labels) or 0 (unequal labels). One can imagine more elaborated similarity functions for labels that incorporate dictionaries and thesauri performing a conceptual matching. For instance, the similarity of a `section` label and a `subsection` label may be defined as 0,75. The similarity *contentSim* between the content of two nodes can be measured using standard comparison methods like the Jaccard coefficient of the boolean model [21, pp. 25ff] or the cosine matching of the vector space model (see Section 2.3.3).

In this work, the boolean model is used to compute content similarities of supertree nodes and document nodes. Formally, label similarity and content similarity are given as follows:

$$labelSim(\lambda(A'), \lambda(B)) = \begin{cases} 1 & \text{if } \lambda(A') = \lambda(B) \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

$$contentSim(\gamma(A'), \gamma(B)) = \frac{|\gamma(A') \cap \gamma(B)|}{|\gamma(A') \cup \gamma(B)|} \tag{7.2}$$

In the equations $\lambda(N)$ denotes the label and $\gamma(N)$ the content of node N.

Since parental relationships are established among document nodes, these only exist among document-related nodes of the supertree. It is worth recalling that descriptive nodes are simply artificial nodes and hence are not related to any structural relationship in the documents.

Besides the defined *labelSim*, the computation of the structural similarity of a document-related supertree node A' and a document node A incorporates two additional parameters, a branching factor $w_{branch}(A')$ and a labeling factor $w_{label}(A')$. Formally, these factors are defined as follows:

Relative branching factor $w_{branch}(A')$ The frequency of the descriptive node $[i^{th}]$ on the edge between A' and P', divided by the frequency of P'. P' denotes the document-related parent node of the document-related node A'.

$$w_{branch}(A') = \frac{[i^{th}].freq}{P'.freq} \tag{7.3}$$

Relative labeling factor $w_{label}(A')$ The frequency of A' having the same label as A, divided by the frequency of the descriptive node $[i^{th}]$ on the edge between A' and P'

$$w_{label}(A') = \frac{A'.freq}{[i^{th}].freq} \tag{7.4}$$

Figure 7.4 illustrates the way both factors are derived from the supertree. The structural

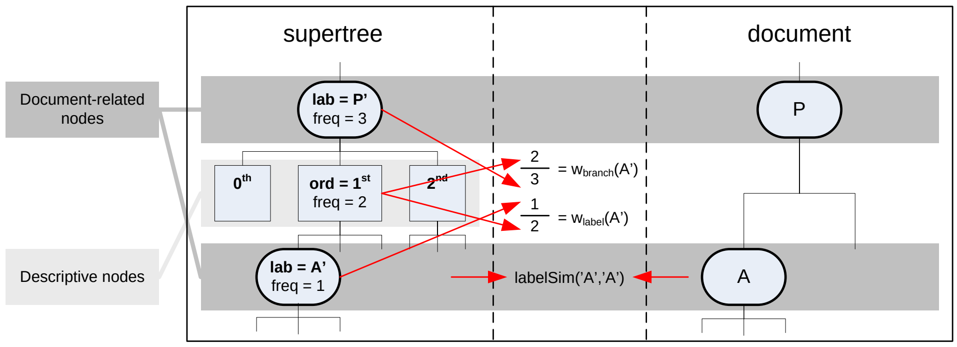


Figure 7.4: Parameters of structural similarity

similarity $structSim(A', A)$ of a supertree node A' and a document node A is defined by Equation 7.5.

$$structSim(A', A) = w_{branch}(A') \cdot w_{label}(A') \cdot labelSim(\lambda(A'), \lambda(A)) \tag{7.5}$$

The final node similarity of a supertree node and a document node is defined in Equation 7.6. A parameter α_{struct} $(0 \leq \alpha_{struct} \leq 1)$ allows to tune the impact of the structural similarity and the content similarity.

$$nodeSim(A', A) = \alpha_{struct} \cdot structSim(A', A) + (1 - \alpha_{struct}) \cdot contentSim(\gamma(A'), \gamma(A)) \tag{7.6}$$

A document and a supertree are compared by matching the root node of the document tree and the root node of the supertree. Note that it is not checked whether a supertree includes the document tree as a subtree. The similarity function *sim* combines the node similarity of the root nodes and the recursively computed node similarities of the child nodes *childSim*. A parameter β_{parent} $(0 \leq \beta_{parent} \leq 1)$ controls the influence of the parents' node similarity and the averaged childrens' node similarity. Formally, the overall similarity of a supertree rooted in A' and a document rooted in A is given by $sim(A', A)$:

$$sim(A', A) = \beta_{parent} \cdot simNode(A', A) + (1 - \beta_{parent}) \cdot childSim(A', A) \tag{7.7}$$

$$childSim(B', B) = \frac{1}{|B|} \cdot \sum_{i=1}^{|B|} sim(B'_i, B_i) \tag{7.8}$$

where $|B|$ is the number of children of node B, and B'_i (resp. B_i) denotes the i-th child of B' (resp. B). Recall that the primed nodes belong to the supertree. Averaging the similarity of the child nodes in $childSim(B', B)$ is justified because weighing is implicitly applied in the recursive computation of $sim(A', A)$.

Starting at the root of the document tree, each document node is compared to an equivalent document-related node in the supertree. If a document node does not correspond to any of the document-related nodes in the supertree, its similarity is defined as 0. Accordingly, the similarity *childSim* of a childless document node (leaf node) and a document-related supertree node that always contains child nodes (leftmost descriptive child node $[0^{th}]$ has $freq = 0$) is defined as 0.

Figure 7.5 shows the four major cases of comparing child nodes. In all subfigures the branching factor w_{branch} and the labeling factor w_{label} remain the same. In the example $w_{branch} = \frac{5}{7}$ means that 5 out of 7 supertree parent nodes of A' contain a 3^{rd} child node. $w_{label} = \frac{4}{5}$ reflects that 4 times out of 5 the 3^{rd} child node is A. Based on *labelSim*, *contentSim*, and $\alpha_{struct} = 0,5$, the node similarity of node A' and A is computed using Equation 7.6. For better illustration the constant $nodeSim(A', A)$ is assumed to be $0,8$ ($0,786 = \frac{1}{2} \cdot \frac{5}{7} \cdot \frac{4}{5} \cdot 1,0 + \frac{1}{2} \cdot 1,0 = \frac{11}{14}$). Further, β_{parent} is set to $0,5$.

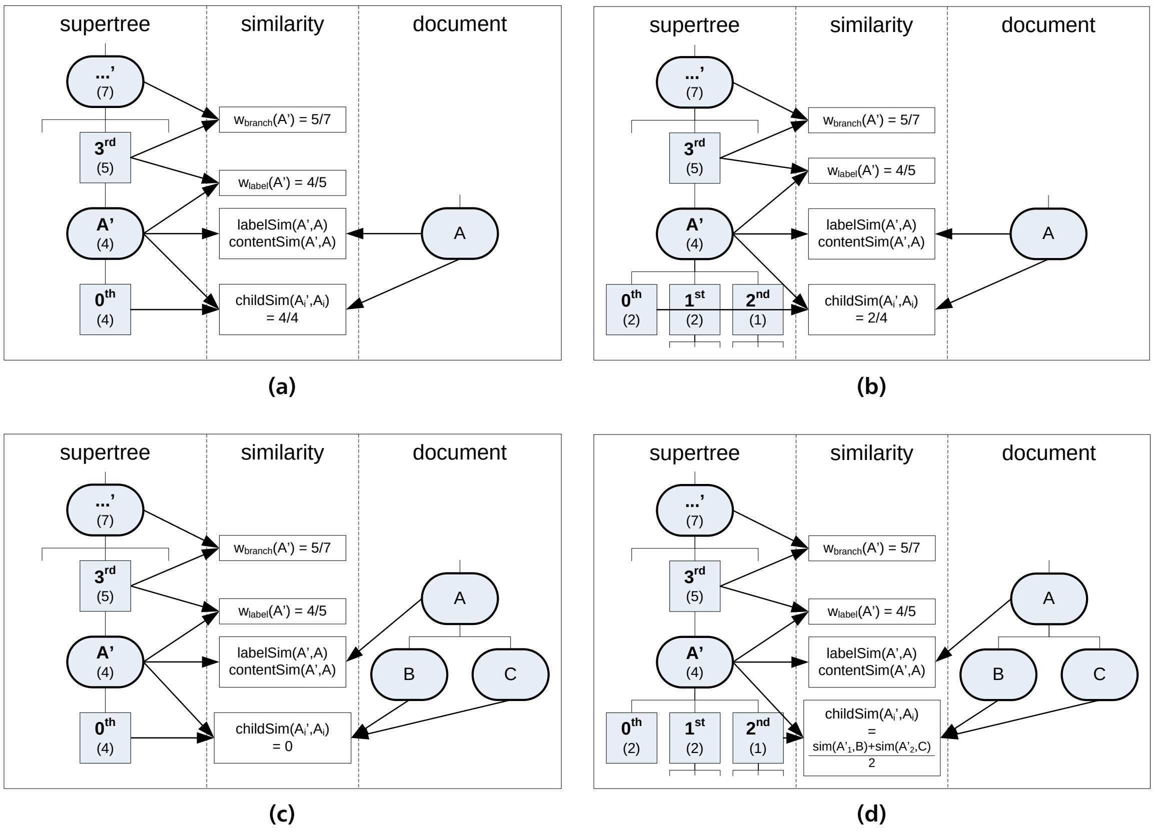

supertree
similarity
document
...' (7)
3rd (5)
A' (4)
0th (4)
0th (2)
1st (2)
2nd (1)
$w_{branch}(A') = 5/7$
$w_{label}(A') = 4/5$
labelSim(A',A) contentSim(A',A)
childSim(A$_i$',A$_i$) = 4/4
childSim(A$_i$',A$_i$) = 2/4
childSim(A$_i$',A$_i$) = 0
childSim(A$_i$',A$_i$) = (sim(A'$_1$,B)+sim(A'$_2$,C))/2
A
B
C
(a)
(b)
(c)
(d)

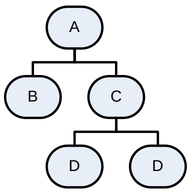
Figure 7.5: Comparing document nodes to supertree nodes

In Figure 7.5a, a document leaf node A is compared to a supertree node A' without child nodes. The overall similarity of the two subtrees is $sim(A', A) = \frac{1}{2} \cdot 0,8 + \frac{1}{2} \cdot \frac{4}{4} = 0,9$.

The similarity of child nodes A_i (no children) is 1,0 because 4 out of 4 A'_i nodes have no children. In the next Figure 7.5b a document leaf node A without children is compared to a supertree node A' that does contain child nodes. The similarity of the two trees is $sim(A', A) = \frac{1}{2} \cdot 0,8 + \frac{1}{2} \cdot \frac{2}{4} = 0,65$. If supertree node A' would never come without child nodes, $childSim(A', A)$ would be zero. The opposite case is given in Figure 7.5c where a document node with child nodes is compared to a supertree node A' without child nodes. The similarity between the two trees is $sim(A', A) = \frac{1}{2} \cdot 0,8 + \frac{1}{2} \cdot 0,0 = 0,4$. Here, $childSim(A', A) = 0,0$ because A' never occurs with child nodes. The general case is depicted in Figure 7.5d. A document node with child nodes is compared to a supertree node A' with child nodes. The similarity between the two trees is given by $sim(A', A) = \frac{1}{2} \cdot 0,8 + \frac{1}{2} \cdot (\frac{1}{2} \cdot sim(A'_1, B) + sim(A'_2, C))$.

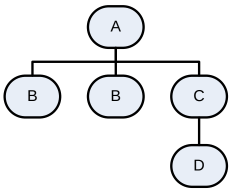
Figure 7.6 gives the similarity values of five different document trees and a supertree that is built from the first three documents (see Figure 7.2f). Thus, the first three documents are fully included in the supertree (Figures 7.6a, 7.6b, and 7.6c), the fourth document is partially included (Figure 7.6d), and the fifth document is not included in the supertree (Figure 7.6e). The similarities of all documents are quite high because (1) child nodes are equally important as their root node and (2) contents of identically labeled nodes fully match (see Figure 7.6e). This result can be varied by using other values for α_{struct} and β_{label}.

One feature of the proposed similarity method is that the depth level of a node is implicitly considered during similarity computation. Figure 7.7a shows an example supertree built from a single document. The next two figures, Figure 7.7b and Figure 7.7c, show two similar documents where only a single node has been changed on different levels. From the similarity values one can see that the document in Figure 7.7c is more similar to the document in Figure 7.7a than the document in Figure 7.7b because the mismatching node X is nested at a deeper level in the document structure.



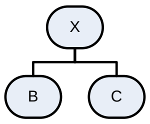
```
sim(A′,A) = 0.5*[0.5*(1.0*0.67*1.0) + 0.5*1.0] + 0.5*0.94 = 0.885
sim(B′,B) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
sim(C′,C) = 0.5*[0.5*(1.0*0.5 *1.0) + 0.5*1.0] + 0.5*1.0  = 0.875
sim(D′,D) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
sim(D′,D) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
```

(a)



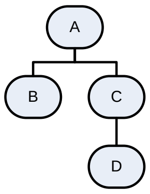
```
sim(A′,A) = 0.5*[0.5*(1.0*0.67*1.0) + 0.5*1.0] + 0.5*0.92 = 0.875
sim(B′,B) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
sim(B′,B) = 0.5*[0.5*(1.0*0.5 *1.0) + 0.5*1.0] + 0.5*1.0  = 0.875
sim(C′,C) = 0.5*[0.5*(0.5*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 0.875
sim(D′,D) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
```

(b)



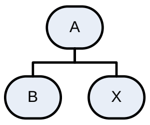
```
sim(X′,X) = 0.5*[0.5*(1.0*0.34*1.0) + 0.5*1.0] + 0.5*1.0 = 0.833
sim(B′,B) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0 = 1.0
sim(C′,C) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0 = 1.0
```

(c)



```
sim(A′,A) = 0.5*[0.5*(1.0*0.67*1.0) + 0.5*1.0] + 0.5*0.94 = 0.885
sim(B′,B) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
sim(C′,C) = 0.5*[0.5*(1.0*0.5 *1.0) + 0.5*1.0] + 0.5*1.0  = 0.875
sim(D′,D) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0  = 1.0
```

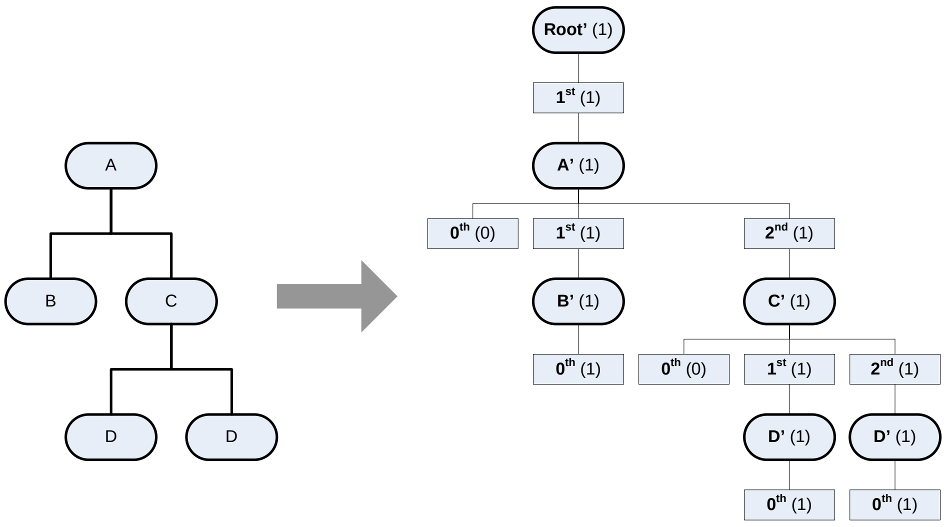
(d)



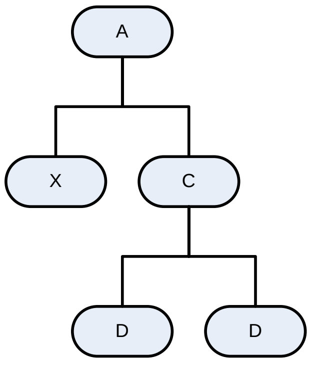
```
sim(A′,A) = 0.5*[0.5*(1.0*0.67*1.0) + 0.5*1.0] + 0.5*0.5 = 0.667
sim(B′,B) = 0.5*[0.5*(1.0*1.0 *1.0) + 0.5*1.0] + 0.5*1.0 = 1.0
sim(C′,X) = 0.5*[0.5*(1.0*0.0 *0.0) + 0.5*0.0] + 0.5*0.0 = 0.0
```

(e)

Figure 7.6: Similarities of documents and the supertree constructed in Figure 7.2f

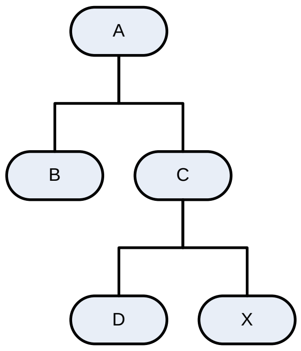


(a) Supertree of a single document



```
sim(A’,A) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*0.5 = 0.75
sim(B’,X) = 0.5*[0.5*(1.0*0.0*0.0) + 0.5*0.0] + 0.5*0.0 = 0.0
sim(C’,C) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*1.0 = 1.0
sim(D’,D) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*1.0 = 1.0
sim(D’,D) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*1.0 = 1.0
```

(b) Document d_1



```
sim(A’,A) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*0.875 = 0.9375
sim(C’,C) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*0.5   = 0.75
sim(D’,D) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*1.0   = 1.0
sim(D’,X) = 0.5*[0.5*(1.0*0.0*0.0) + 0.5*0.0] + 0.5*0.0   = 0.0
sim(B’,B) = 0.5*[0.5*(1.0*1.0*1.0) + 0.5*1.0] + 0.5*1.0   = 1.0
```

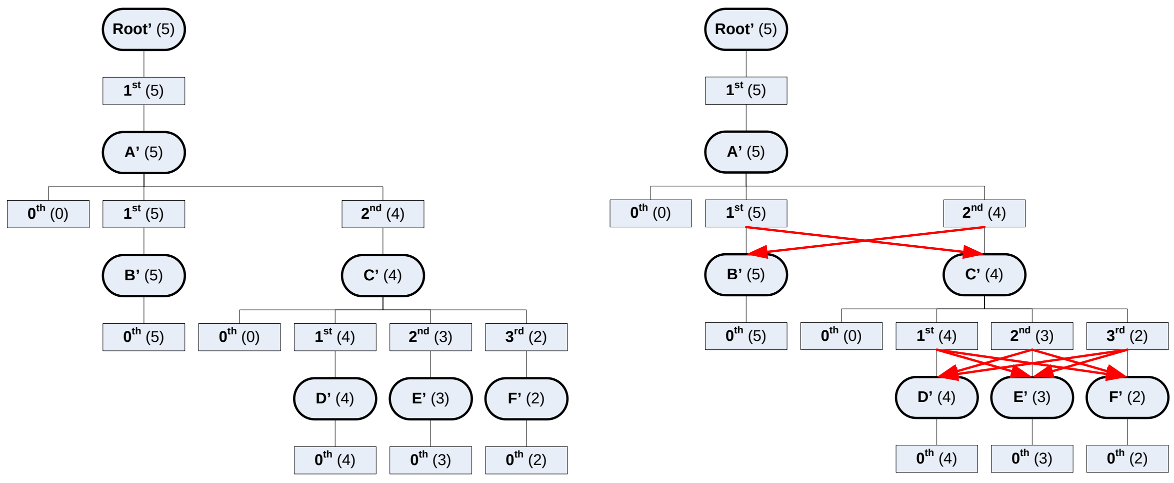
(c) Document d_2

Figure 7.7: Depth considerations of comparing documents and a supertree

7.4.2 Comparing Unordered Document Trees and Supertrees

The previous section was concerned with comparing ordered documents to an ordered supertree. From a document-centric XML point of view, this assumption seems to be valid. However, the supertree approach proposed is simply extended to handle unsorted trees. Therefore, additional edges are inserted in the supertree that maintain correct parent-child relations. In Figure 7.8, an ordered supertree (Figure 7.8a) and an equivalent unordered supertree (Figure 7.8b) are depicted. In the unordered supertree, inserted edges are highlighted in red color.

The effect of the new edges is that all document-related child nodes B' and C' of a parent document-related node A' are allowed in any position (i.e., become children of all descriptive nodes). This allows both nodes B' and C' being matched as the first and/or as the second child of A' in the figure. The descriptive nodes are maintained for computing the branching factors of the supertree structure. Most important, the supertree does not have to be recalculated explicitly. The only parameter that needs to be changed in the formulae is the labeling factor w_{label}, which is redefined to the constant value of 1,0. The reason for this is that the frequencies in the descriptive nodes on the edges between child nodes and parent nodes are independent of each other and must not be mixed up. Thus, the relative labeling frequency would become incorrect.



(a) Ordered supertree **(b)** Unordered supertree

Figure 7.8: Ordered vs. unordered supertree

7.4.3 Comparing Supertrees

Comparing two supertrees differs from comparing a supertree and a document. The reason for this difference is that two types of nodes, descriptive nodes and document-related nodes, have to be compared during a single recursion step. This is because descriptive node similarities and document-related node similarities incorporate a set of different parameters.

The similarity computation starts at the root nodes of the supertrees ST_1 and ST_2 using the Equations 7.11–7.19. In the equations, $labelSim$, $contentSim$, α_{struct}, and β_{parent} remain as defined in Section 7.4.1. In order to compare different relative frequencies, both the branching factor and the labeling factor need to be redefined. The branching factor w_{branch} is now computed for two descriptive nodes A'_{descr} and B'_{descr}, while the labeling factor w_{label} is computed for two document-related nodes C_{docrel} and D_{docrel}. The new formulae are given as follows:

$$w_{branch}(A'_{descr}, B'_{descr}) = \frac{A'_{descr}.parent().freq}{A'_{descr}.parent().freq + B'_{descr}.parent().freq} \tag{7.9}$$

$$w_{label}(C'_{docrel}, D'_{docrel}) = \frac{C'_{docrel}.parent().freq}{C'_{docrel}.parent().freq + D'_{docrel}.parent().freq} \tag{7.10}$$

where $A'.parent()$ denotes the parent node of A' (either a document-related node or a descriptive node). This redefinition considers the frequencies of both supertree parent nodes to compute the relative frequency. Thus, the factors become more meaningful and better interpretable.

The similarity of two document-related nodes A' (in supertree ST_1) and B' (in supertree ST_2) combines the similarity of the current document-related node $nodeSim$ and the averaged similarity of the descriptive child nodes $avgDescrSim$. Figure 7.9 explains the scope of the main parameters in the Equations 7.11 to 7.19.

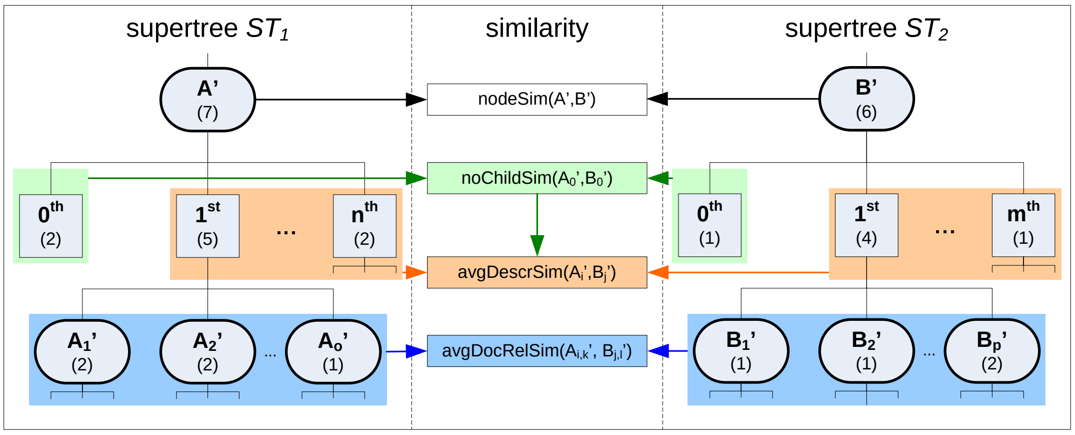


Figure 7.9: Comparison of two document-related supertree nodes

The similarity of two document-oriented supertree nodes A' and B' is given by the following formulae:

$$\begin{aligned}
sim(A',B') &= \beta_{parent} \cdot nodeSim(A',B') + (1-\beta_{parent}) avgDescrSim(A',B') && (7.11)\\
nodeSim(A',B') &= \alpha_{struct} \cdot labelSim(\lambda(A'),\lambda(B')) + (1-\alpha_{struct}) \cdot contentSim(\gamma(A'),\gamma(B')) && (7.12)\\
avgDescrSim(C',D') &= \frac{1}{max(|C'|,|D'|)} \cdot [(\sum_{i=1}^{max(|C'|,|D'|)} descrSim(C'_i,D'_i)) + noChildSim(C'_0,D'_0)] && (7.13)\\
noChildSim(E',F') &= \begin{cases} 1 & \text{if } w_{branch}(E',F') = 0 \wedge w_{branch}(F',E') = 0 \\ 0 & \text{if } w_{branch}(E',F') = 0 \veebar w_{branch}(F',E') = 0 \\ \frac{\min(w_{branch}(E',F'),w_{branch}(F',E'))}{\max(w_{branch}(E',F'),w_{branch}(F',E'))} & \text{otherwise} \end{cases} && (7.14)
\end{aligned}$$

The similarity of a descriptive node *descrSim* includes the relative branching factor as well as the averaged similarity of the document-related child nodes *avgDocRelSim*. Missing descriptive nodes are defined as unequal (Equation 7.15).

$$\begin{aligned}
descrSim(E',null) &= descrSim(null,F') = 0 && (7.15)\\
descrSim(E',F') &= \frac{\min(w_{branch}(E',F'),w_{branch}(F',E'))}{\max(w_{branch}(E',F'),w_{branch}(F',E'))} \cdot avgDocRelSim(E',F') && (7.16)\\
avgDocRelSim(E',F') &= \frac{1}{max(|E'|,|F'|)} \cdot \sum_{i=1}^{max(|E'|,|F'|} docRelSim(E'_i,F'_i) && (7.17)
\end{aligned}$$

The recursion takes place in the similarity computation of the document-related node *docRelSim*, which combines the relative labeling factor and the recursive node similarity. As for the descriptive nodes, missing document-related nodes are also defined as unequal (Equation 7.18).

$$\begin{aligned}
docRelSim(E',null) &= docRelSim(null,F') = 0 && (7.18)\\
docRelSim(E',F') &= \frac{\min(w_{label}(E',F'),w_{label}(F',E'))}{\max(w_{label}(E',F'),w_{label}(F',E'))} \cdot sim(E',F') && (7.19)
\end{aligned}$$

7.5 Clustering Approaches

Two different clustering techniques are used in this work. First, k-Means is applied to partition a set of XML trees into flat clusters. However, for the sake of improving information retrieval tasks, a second approach based on hierarchical clustering is implemented. Both clustering approaches operate on supertree representations presented in the previous sections. The results using an INEX document collection dedicated to evaluate clustering performance are given in the next section.

Algorithm 7.1 Clustering via the *k*-Means algorithm

```
1:  $X = \{x_1 \ldots x_m\}$ //unlabeled documents
2:  $C = \{c_1 \ldots c_n\}$ //cluster centers
3:  procedure $k$-MEANS($k$)
4:      initialize $C$ (randomly select $k$ documents of $X$)
5:      repeat
6:          for (i = 1 to $|X|$) do
7:              assign $x_i$ to closest $c_j$
8:          end for
9:          $distortion = \sum_i \sum_j dist^2(c_i, x_j)$
10:         recalculate cluster centers $C'$
11:     until ($C = C'$ || $distortion \leq \epsilon$ || maximum number of iterations)
12: end procedure
```

7.5.1 Overview of *k*-Means

The *k*-Means algorithm [169, 199, 147] is one of the simplest unsupervised learning algorithms that solves the problem of partitioning a set of unknown data points into an a priori fixed number of *k* disjunct clusters. The main idea is to find *k* clusters such that the error of all data point assignments becomes minimal, satisfying two properties: (1) Each cluster is represented by an (artificial) cluster center called centroid, which is the mean of all data points in that cluster, and (2) each data point is assigned to that cluster whose center it is closest to.

The algorithm (see Algorithm 7.1) starts with randomly selecting *k* data points as initial cluster centers. In the context of this work, a cluster center is represented as an ordered supertree that is created from all document trees the cluster contains. Initial clusters are supertrees built from a single document tree only. Then, it proceeds in iterations while keeping track of the centroids. Each iteration tries to improve the quality of clustering. Therefore, all data points are assigned to their closest centroid. The sum of the square distances (distortion) of all data points to their assigned cluster centers is taken as objective function, which is tried to be minimized. Afterwards, *k* new cluster centers are computed based on their assigned data points, establishing the loop. The algorithm terminates if (1) the cluster centers (assignments) of subsequent iterations do not change, (2) the distortion *distortion* drops below a certain threshold ϵ, or (3) the maximum number of iterations is reached.

7.5.2 Overview of Hierarchical Clustering

In contrast to flat clusters as created by *k*-Means, hierarchical clustering [97] is a method that creates a hierarchy of clusters (called dendogram). Individual data points define the leaf nodes of the cluster hierarchy. A single top cluster that contains every data point defines the root node. Figure 7.10 shows the procedure of hierarchical clustering graphically.

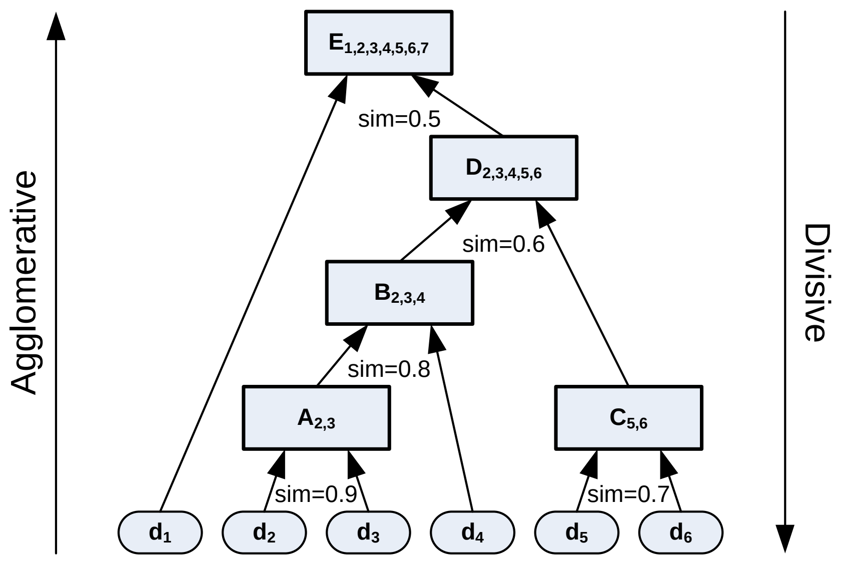


Figure 7.10: Hierarchical clustering

There are two possibilities to create a cluster hierarchy: agglomerative (bottom-up) clustering starts by defining each data point as a separate cluster containing only one element and merges the two most similar (closest) clusters until a single top cluster remains; divisive (top-down) clustering starts with a single top cluster that contains all data points and breaks up the least similar (farthest) data point until only single elements remain. The outer arrows in Figure 7.10 indicate the agglomerative and divisive clustering approach. Applying agglomerative clustering, the order of clusters created is given by the similarity values beneath the clusters. First, cluster A is created due to $sim(d_2, d_3) = 0,9$. Subsequently, clusters B ($sim(A, d_4) = 0,8$) to E ($sim(d_1, E) = 0,5$) are formed. Clusters that consist of another cluster and a single data point are called 'mixed clusters' (e.g., clusters B and E in the figure). Additional constraints or stopping criteria (e.g., the final number of clusters, a minimal closeness of clusters to get merged) can be introduced to govern the clustering process. The advantage of hierarchical clustering is that the dendogram can be cut at a certain height (a certain number of top clusters). Thus, precision can be tuned by choosing either a larger number of smaller clusters or a smaller number of larger clusters.

According to [97], there are three different methods to determine the closeness of two clusters:

Single-linkage the least distant pair of data points defines the distance of two clusters (as in Figure 7.10);

Complete-linkage the most distant pair of data points defines the distance of two clusters;

Average-linkage the average distance of all pairs of data points (distance of the two cluster centers) defines the distance of two clusters;

This work applies an agglomerative clustering approach based on average-linkage. Supertrees (see Section 7.3) provide the representations of cluster centers and supertree comparison (see Section 7.4.3) defines their similarity. To test the efficiency of the approach, a prototype was implemented that enables the evaluation of different supertree and clustering settings.

7.6 Evaluation

As for the XML document classification described in the previous chapter, several aspects of clustering XML documents are investigated in this section. The evaluation is conducted on a set of real-world XML collections based on the movie database (MovieDB) [65]) proposed in INEX'05 [90]. The goal is to find optimal parameter settings that are applied for retrieval-related clustering tasks. The key questions can be formulated as follows:

1. How does the number of clusters influence the clustering?
2. What are the optimal parameter settings?
3. How does the clustering performance depend on the size of the training set?
4. How does content-and-structure based clustering perform compared to pure structure based clustering?

The INEX collections used for clustering do not contain the same documents as the collections used for classification. Also, the clusters do not correspond to the categories. INEX provides human-defined clusters and cluster assignments for the collections, which allow performance comparisons of different approaches and systems. For better understanding of this section, the term 'classes' refers to the optimal clusters of a collection, while the term 'clusters' refers to the system-created clusters. Again, the XML collections are of two types:

- Structure-only (*SO*) collections containing only the structure of the XML documents. These include four collections: *m-db-s-0*, *m-db-s-1*, *m-db-s-2*, *m-db-s-3*, where the last three collections are noisy versions of the first one. The amount of noise and class overlap increases from *m-db-s-0* to *m-db-s-3*. The collections come in the form of two sets, a training and a testing set. The training documents of the *m-db-s* data sets are organized in 11 classes, where each class corresponds to a movie genre. Table 7.1 summarizes the number of training and testing documents of each collection. Training documents are selected randomly and uniformly from the classes.

Table 7.1: Structure-Only corpora

Corpus	Train/Test 10%	Train/Test 30%	Train/Test 50%	Train/Test 70%	Train/Test 100%
m-db-s-0	488/485	1.453/1.448	2.415/2.409	3.383/3.376	4.824/4.816
m-db-s-1	487/486	1.449/1.447	2.410/2.408	3.378/3.374	4.818/4.814
m-db-s-2	486/485	1.450/1.447	2.412/2.408	3.379/3.372	4.820/4.810
m-db-s-3	488/485	1.453/1.445	2.414/2.404	3.380/3.370	4.821/4.810

- Content-and-structure (*CAS*) collections consisting of the entire documents. The structure of the *CAS* documents is more heterogenous than that of the *SO* documents. Thus, content information is expected to improve clustering performance. Both collections, *m-db-cs-1* and *m-db-cs-2*, consist of 4.825 labeled documents assigned to 11 different classes. The sets are split into 2.415 training and 2.410 testing documents. Table 7.2 summarizes the number of training and testing documents of the two collections.

Table 7.2: Content-and-Structure corpora

Corpus	Train/Test 10%	Train/Test 30%	Train/Test 50%	Train/Test 70%	Train/Test 100%
m-db-cs-1	247/246	730/728	1.210/1.208	1.696/1.692	2.415/2.410
m-db-cs-2	247/246	730/728	1.210/1.208	1.696/1.692	2.415/2.410

Four sets of experiments aim at answering the questions stated earlier. The first three deal with the structure-only setting, while the last one is concerned with the content-and-structure setting. A comparison of the results against some available results from other authors is highlighted at the end of this section.

7.6.1 Measures

Results achieved are evaluated using two evaluation measures, namely purity and entropy [230, 268, 74]. Purity describes the homogeneity of a cluster c_i. In general, the higher the purity the better a system works. The $[0,1]$ purity of a cluster c_i is defined as

$$p(c_i) = \frac{\text{number of documents of the majority class of cluster } c_i}{\text{number of documents in the cluster } c_i} = \frac{1}{n_i} \cdot \max_j(n_i^j) \qquad (7.20)$$

where n_i is the number of elements in cluster c_i, and n_i^j is the number of elements of class j assigned to cluster i.

In contrast to purity, entropy measures the quality of a cluster in terms of its disorder. In general, the lower the entropy, the better a system performs. Formally, the $[0,1]$ entropy of a cluster c_i is defined as

$$e(c_i) = -\frac{1}{\ln l} \cdot \sum_{j=1}^{l} \frac{n_i^j}{n_i} \ln \frac{n_i^j}{n_i} \tag{7.21}$$

where l is the total number of classes in the dataset. Thus, entropy is more comprehensive than purity because it considers the entire collection [230].

The overall purity and entropy of all clusters is computed using two kinds of averages: unweighed and weighed.

Unweighed macro averaging evaluates on the cluster level. Each cluster is given equal weight. The overall purity and entropy are computed as the arithmetic averages:

$$p_{macro} = \frac{\sum_{i=1}^{k} p(c_i)}{k} \tag{7.22}$$

$$e_{macro} = \frac{\sum_{i=1}^{k} e(c_i)}{k} \tag{7.23}$$

where k is the number of clusters.

Weighed micro averaging evaluates on the document level. Each cluster is assigned a weight proportional to the size of the cluster. Thus, p_{micro} and e_{micro} give equal weight to each document:

$$p_{micro} = \frac{\sum_{i=1}^{k} p(c_i) \cdot n_i}{n} \tag{7.24}$$

$$e_{micro} = \frac{\sum_{i=1}^{k} e(c_i) \cdot n_i}{n} \tag{7.25}$$

$$\tag{7.26}$$

where k is the number of clusters and n is the total number of clustered elements.

Purity (for clustering) defines the percentage of documents associated with the majority classes of all documents, whereas accuracy (for classification) is the percentage of correctly assigned documents to all documents. In case of comparing classification and clustering, macro-averaged purity and accuracy are both equal.

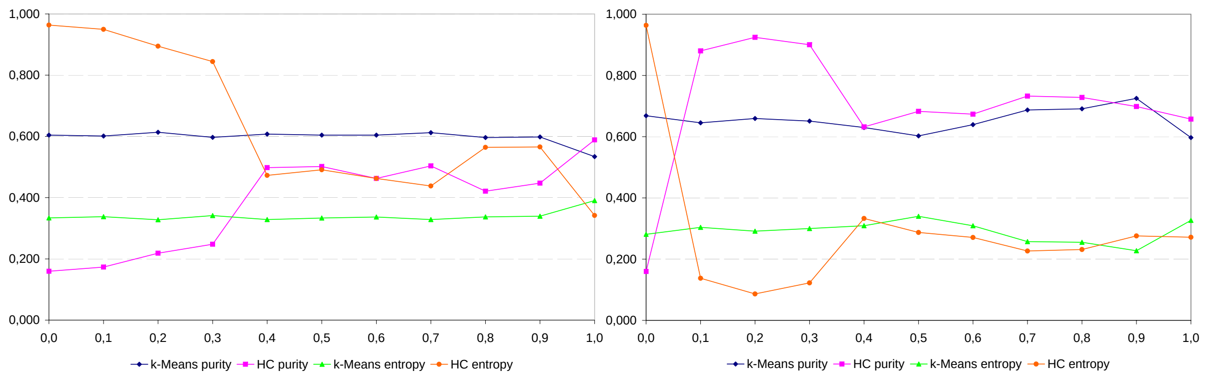
7.6.2 Experiment I - How Does the Parameter β_{parent} Affect the Purity and Entropy?

In a first experiment, the parameter β_{parent} which controls the importance of the parent node and child nodes similarities is tested. Using the *SO* corpus *m-db-s-0*, different values for both clustering approaches, k-Means and hierarchical clustering, are evaluated. Since the corpus is structure-only, α_{struct} is set to $1,0$. The number of clusters is fixed at 11, which is the number of original classes.

The results of k-Means and hierarchical clustering HC are given in Table 7.3 and Figure 7.11. For the k-Means approach, all measures are averaged over ten runs.

Table 7.3: Evaluation results for the parameter β_{parent}

	k-Means				HC			
β_{parent}	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
0,0	0,604	0,668	0,334	0,281	0,160	0,160	0,964	0,964
0,1	0,601	0,645	0,338	0,304	0,173	0,880	0,950	0,138
0,2	0,613	0,659	0,328	0,291	0,219	0,925	0,895	0,086
0,3	0,597	0,651	0,342	0,300	0,248	0,900	0,844	0,123
0,4	0,608	0,630	0,328	0,309	0,498	0,632	0,473	0,333
0,5	0,604	0,602	0,333	0,340	0,502	0,683	0,491	0,287
0,6	0,604	0,639	0,337	0,309	0,463	0,674	0,463	0,271
0,7	0,612	0,687	0,328	0,257	0,504	0,733	0,438	0,227
0,8	0,596	0,691	0,337	0,255	0,421	0,728	0,564	0,231
0,9	0,598	0,725	0,339	0,228	0,447	0,699	0,566	0,276
1,0	0,534	0,597	0,390	0,327	0,588	0,657	0,342	0,272



(a) Micro averaging **(b)** Macro averaging

Figure 7.11: Effect of β_{parent} on the purity and entropy (m-db-s-0)

A parameter setting of $\beta_{parent} = 1,0$ leads to very high similarities of clusters that contain documents with the same root node. Contrary, $\beta_{parent} = 0,0$ puts all weight to the similarity of the leaf nodes. Interestingly, the same parameter setting performs different in the two approaches.

k-Means achieves best results for $\beta_{parent} = 0,2$ (micro averaging) and $\beta_{parent} = 0,9$ (macro averaging). Generally, purity and entropy are not too strongly influenced by β_{parent}. Values differ by only 8,0%-points for micro averaging and 12,8%-points for macro averaging.

Hierarchical clustering is influenced strongly by β_{parent}. Best results are computed for $\beta_{parent} = 1,0$ (micro averaging) and $\beta_{parent} = 0,2$ (macro averaging). In contrast to k-Means, values differ by 42,9%-points for micro averaging and 76,5%-points for macro averaging. Very high macro (e.g., 92,5%) values occur with low micro purity values (e.g., 21,9%). This effect is explained by many small clusters with high purity and few large clusters with low purity.

In further experiments both approaches stick to the parameter that gives best results using macro averaging.

7.6.3 Experiment II - How Does k Affect the Purity and Entropy?

The second experiment evaluates the clustering performance for different numbers of clusters. k-Means and hierarchical clustering are applied on the *SO* corpus *m-db-s-0*. The different k values tested are 3, 5, 7, 9, 11 (number of intended clusters), 13, 15, and 21. α_{struct} is set to $1,0$ (only structure is considered). The parameter β_{parent} is fixed at $0,9$ for k-Means and $0,2$ for hierarchical clustering.

All experimental runs of the previous experiment are recalculated. Since k-Means obtains all values by averaging over ten runs, the result for $k = 11$ slightly differs. In contrast, *HC* measures remain the same for $k = 11$. The results are presented in Table 7.4. Figure 7.12 compares the purity and entropy values computed.

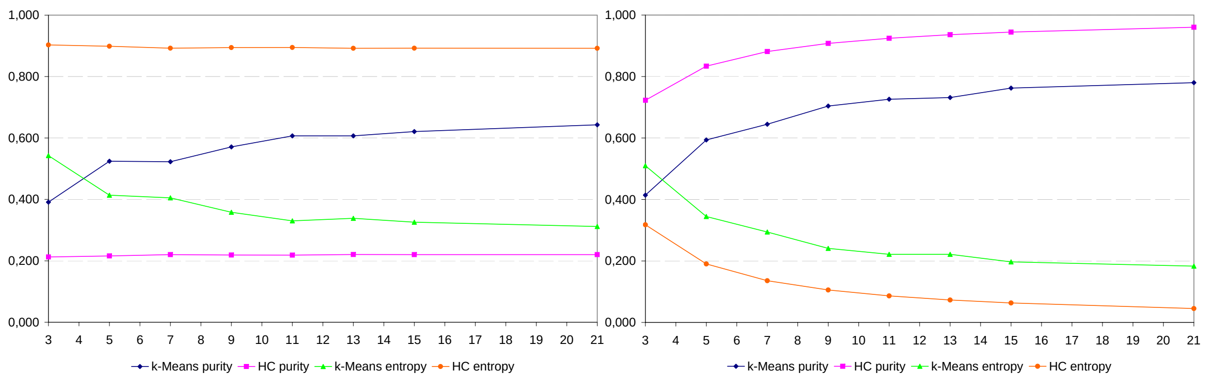
Table 7.4: Evaluation results for the parameter k

	k-Means				*HC*			
β_{parent}	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
3	0,391	0,414	0,543	0,510	0,213	0,723	0,903	0,318
5	0,524	0,594	0,414	0,345	0,216	0,834	0,899	0,190
7	0,523	0,645	0,405	0,294	0,220	0,881	0,893	0,136
9	0,571	0,704	0,358	0,241	0,219	0,908	0,894	0,106
11	0,607	0,726	0,330	0,222	0,219	0,925	0,895	0,086
13	0,607	0,732	0,338	0,222	0,221	0,936	0,892	0,073
15	0,621	0,762	0,326	0,197	0,220	0,945	0,893	0,063
21	0,643	0,780	0,312	0,183	0,220	0,960	0,892	0,045

The charts for both approaches clearly show an improvement for higher numbers of clusters. The performance of k-Means is more influenced than the performance of hierarchical clustering.

In k-Means, purity differs by 25,2%-points (micro averaging) and 36,6%-points (macro averaging). Applying *HC*, purity values are less affected, differing by only for 0,8%-points (micro averaging) and 23,8%-points (macro averaging).

Irrespective of the results achieved in this experiment, the number of clusters in subsequent experiments is set to 11. The reason for this decision is that other researchers evaluate their results with this number too. A comparison of the final results is carried out in Section 7.6.6.



(a) Micro averaging **(b)** Macro averaging

Figure 7.12: Effect of k on the purity and entropy (m-db-s-0)

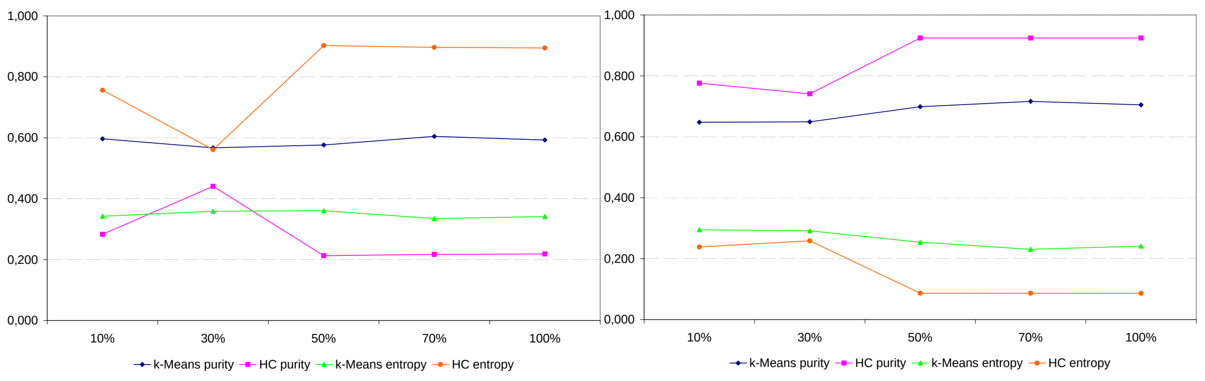
7.6.4 Experiment III - How Does the Training Data Affect the Purity and Entropy?

A third experiment evaluates the impact of the training size on the clustering performance. Normally, the entire set of training samples is taken to create clusters. Hence, it is clear that the size of the training set is crucial for the clustering algorithm. To observe the effect of increasing the size of the training data set, the *m-db-s-0* collection is split into five ratios (10%, 30%, 50%, 70%, 100%). These ratios are randomly and uniformly selected among the whole training data so that every chunk contains documents of all intended clusters. For k-Means, parameter settings are $\beta_{parent} = 0,9$ and $\alpha_{struct} = 1,0$. Hierarchical clustering is conducted with $\beta_{parent} = 0,2$ and $\alpha_{struct} = 1,0$. The results obtained are shown in Table 7.5 and Figure 7.13.

Table 7.5: Evaluation results for the training size parameter

	k-Means				*HC*			
Size	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
10%	0,597	0,648	0,342	0,295	0,283	0,776	0,756	0,238
30%	0,567	0,649	0,359	0,291	0,441	0,741	0,561	0,259
50%	0,576	0,699	0,360	0,254	0,213	0,924	0,903	0,087
70%	0,605	0,717	0,335	0,231	0,217	0,924	0,897	0,087
100%	0,593	0,705	0,342	0,241	0,219	0,925	0,895	0,086

Similar to the classification result, the training size had not much impact on the k-Means clustering. In contrast, performance of the hierarchical clustering is influenced. Unexpectedly, top micro averaged results are achieved with 30% of the training data. This might come from the selected documents. However, macro-averaging results become better the more training data is included.



(a) Micro averaging **(b)** Macro averaging

Figure 7.13: Effect of the training data on the purity and entropy (m-db-s-0)

Based on the results, it might be reasonable to use half of the available training data for training only. In the context of applying clustering for information retrieval, the whole set of documents has to be used anyway.

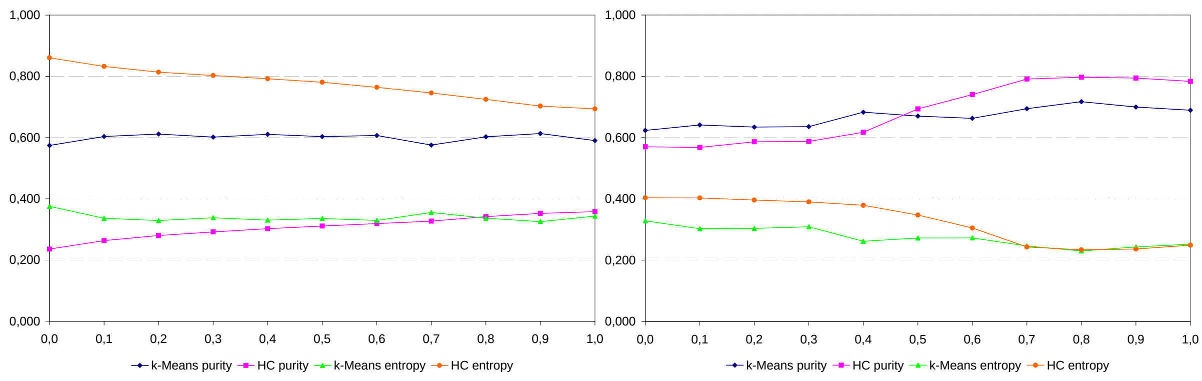
7.6.5 Experiment IV - How Does CAS Setting Affect the Purity and Entropy?

This experiment checks the effectiveness of the proposed clustering approaches on both, the content and the structure of XML documents. Therefore, the *CAS* collections *m-db-cs-1* and *m-db-cs-2* are used. The goal is to find an appropriate α_{struct} setting for each approach. According to the last experiments, β_{parent} is set to $0,9$ for k-Means and $0,2$ for *HC*. The results are displayed in Tables 7.6 and 7.7, with the corresponding Figures 7.14 and 7.15.

Table 7.6: Evaluation results for the α_{struct} parameter (m-db-cs-1)

	k-Means				*HC*			
α_{struct}	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
0,0	0,574	0,623	0,375	0,328	0,236	0,784	0,861	0,249
0,1	0,603	0,641	0,336	0,303	0,263	0,794	0,833	0,236
0,2	0,612	0,634	0,329	0,304	0,280	0,797	0,814	0,234
0,3	0,601	0,636	0,338	0,309	0,292	0,791	0,803	0,243
0,4	0,611	0,683	0,330	0,262	0,302	0,740	0,792	0,305
0,5	0,603	0,670	0,336	0,272	0,311	0,694	0,781	0,347
0,6	0,607	0,663	0,329	0,273	0,319	0,617	0,764	0,379
0,7	0,576	0,694	0,355	0,246	0,327	0,588	0,746	0,390
0,8	0,603	0,717	0,337	0,230	0,342	0,586	0,725	0,396
0,9	0,613	0,700	0,325	0,243	0,352	0,568	0,703	0,403
1,0	0,590	0,689	0,343	0,252	0,358	0,570	0,694	0,404

Although *m-db-cs-1* and *m-db-s-0* are different form each other, the performance of the approaches on both collections (when using the structure only) is nearly the same. k-Means is



(a) Micro averaging **(b)** Macro averaging

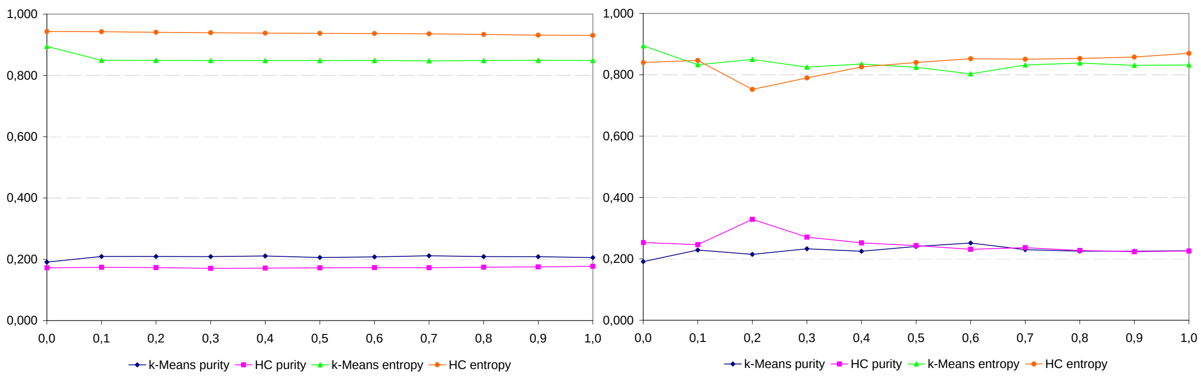
Figure 7.14: Effect of CAS on the purity and entropy (m-db-cs-1)

nearly unaffected by higher weights put on the structure (difference of 3,9%-points). Only macro averaging shows a slight tendency that structure helps more than the content (ranging 9,4%). For *HC*, micro averaging shows that including structure linearly increases clustering performance of *HC*: The more weight is put on the structure, the higher the performance (difference of 12,2%-points). The same effect turns out applying macro averaging (difference of 22,9%-points).

Note that a setting of $\alpha_{struct} = 1,0$ uses the structure of documents only, while $\alpha_{struct} = 0,0$ considers the content only. The classification evaluation showed that pure structure-based approaches clearly outperform content-based ones. Interestingly, structure-only and content-only approaches nearly obtain the same results during clustering. Best results are achieved by combining structure and content information. From this point of view, clustering seems better suited for content-based retrieval tasks than classification.

Table 7.7: Evaluation results for the α_{struct} parameter (m-db-cs-2)

	k-Means				*HC*			
α_{struct}	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
0,0	0,191	0,191	0,895	0,894	0,172	0,253	0,943	0,840
0,1	0,209	0,229	0,850	0,833	0,174	0,246	0,943	0,847
0,2	0,209	0,215	0,849	0,850	0,173	0,329	0,941	0,752
0,3	0,209	0,233	0,849	0,825	0,171	0,271	0,939	0,790
0,4	0,211	0,225	0,848	0,835	0,171	0,252	0,938	0,826
0,5	0,206	0,240	0,848	0,825	0,172	0,243	0,938	0,840
0,6	0,208	0,252	0,849	0,803	0,173	0,231	0,937	0,852
0,7	0,211	0,230	0,847	0,832	0,173	0,236	0,936	0,851
0,8	0,209	0,225	0,849	0,838	0,174	0,227	0,934	0,853
0,9	0,209	0,225	0,849	0,831	0,175	0,223	0,931	0,858
1,0	0,205	0,226	0,849	0,832	0,177	0,226	0,931	0,870



(a) Micro averaging **(b)** Macro averaging

Figure 7.15: Effect of CAS on the purity and entropy (m-db-cs-2)

On the second *CAS* collection *m-db-cs-2*, both approaches perform poor. The structural parameter α_{struct} becomes irrelevant. The reason for this behavior is the document collection itself, which seems unsuited for the evaluation of clustering approaches. This is because the documents are too similar in both regards, their structure and their content. Other INEX'05 participants came to a similar conclusion about this document collection.

Hand in hand with the findings for classification performance, the content turns out to be less important than the structure for clustering. Best results are achieved by α_{struct} values between $0,8$ and $1,0$. However, this could be true only for particular collections where the content is relatively poor as is the case of the MovieDB.

7.6.6 Comparison

A final experiment compares the approach of this work to other state-of-art methods that have been applied on the same collection. Two other approaches presented at the INEX 2005 workshop are considered. A first work by Nayak and Xu [186] applied the XCLS clustering algorithm based on the LevelSim global criterion function. The second work was done by Vercoustre et at. [247], who represent documents by sets of their subpaths and apply the dynamic clouds clustering algorithm. Unfortunately, no comparable results are available that take structure and content into account. Thus, the evaluation is carried out only for the *SO* corpora.

The four *SO* corpora (*m-db-s-0*, *m-db-s-1*, *m-db-s-2*, and *m-db-s-3*) are used for evaluation. The two top parameter settings, one maximizing micro averaging and the other one maximizing macro averaging, are chosen for k-Means ($\beta_{parent} = \{0,2;0,9\}$) and for hierarchical clustering ($\beta_{parent} = \{0,2;1,0\}$) *HC*. Since all corpora contain structure only, α_{struct} is set to $1,0$. The results are presented in Table 7.8

Table 7.8: Comparison of the INEX 2005 evaluation results I

	k-Means				HC			
Corpus	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
m-db-s-0 [a]	0,602	0,638	0,337	0,309	0,219	0,925	0,895	0,086
m-db-s-0 [b]	0,598	0,721	0,339	0,229	0,588	0,657	0,342	0,272
m-db-s-1 [a]	0,621	0,650	0,325	0,307	0,236	0,925	0,867	0,085
m-db-s-1 [b]	0,566	0,723	0,372	0,227	0,588	0,657	0,342	0,272
m-db-s-2 [a]	0,640	0,662	0,308	0,289	0,331	0,751	0,753	0,276
m-db-s-2 [b]	0,605	0,719	0,336	0,226	0,589	0,658	0,342	0,271
m-db-s-3 [a]	0,616	0,655	0,322	0,291	0,400	0,751	0,634	0,237
m-db-s-3 [b]	0,609	0,730	0,331	0,216	0,588	0,657	0,342	0,272

[a] $\beta_{parent} = 0,2$ (k-Means), $\beta_{parent} = 0,2$ (HC)
[b] $\beta_{parent} = 0,9$ (k-Means), $\beta_{parent} = 1,0$ (HC)

Table 7.9 summarizes the results obtained by Nayak and Xu, and Vercoustre et al. According to the micro averaging results, the hierarchical clustering approach HC is least effective. k-Means performs slightly better than the approach presented by Nayak and Xu. The approach of Vercoustre et al. clearly outperforms all other approaches.

Table 7.9: Comparison of the INEX 2005 evaluation results II

	Nayak and Xu [186]				Vercoustre et al. [247]			
Corpus	p_{micro}	p_{macro}	e_{micro}	e_{macro}	p_{micro}	p_{macro}	e_{micro}	e_{macro}
m-db-s-0	0,604	0,794	0,325	0,171	0,732	0,841	0,203	0,136
m-db-s-0	0,589	0,786	0,334	0,174	0,732	0,841	0,203	0,136
m-db-s-1	0,603	0,682	0,331	0,268	0,688	0,804	0,326	0,226
m-db-s-1	0,596	0,769	0,335	0,203	0,707	0,835	0,256	0,144
m-db-s-2	0,590	0,720	0,335	0,249	0,688	0,758	0,296	0,209
m-db-s-2	0,592	0,728	0,335	0,241	0,458	0,501	0,487	0,446
m-db-s-3	0,592	0,756	0,340	0,197	0,623	0,714	0,316	0,238
m-db-s-3	0,589	0,707	0,340	0,224	0,553	0,636	0,527	0,438

Applying macro averaging, HC achieves best results. Vercoustre et al. obtain the second best results. Nayak and Xu obtain slightly better results than k-Means. In contrast to the other approaches, the noise introduced in the collections had no effect on the performance using both, k-Means and HC clustering.

7.7 Conclusion

This chapter presented two XML clustering approaches: k-Means and a hierarchical clustering algorithm. Both approaches operate on supertree representations, which subsume all structure and content features of a set of XML documents. Supertrees are created in a simple and fast manner, where the order of documents included in a supertree is irrelevant. The representation allows fast computation of similarity of ordered and unordered documents. Further, supertrees can be merged to support hierarchical clustering algorithms. Both approaches are evaluated using real world data. A comparison to other approaches on the same data shows promising results. In contrast to classification, inclusion of a document's content improved clustering performance. Thus, information retrieval may benefit from content-aware clustering. From the information retrieval point of view, hierarchical clustering seems to be a promising candidate.

Chapter 8

The mind is not a vessel to be filled, but a fire to be ignited.

Plutarch

Overview of the X-DOSE System

Based on the facets discussed in the previous chapters, this chapter describes the system implemented that covers those aspects. X-DOSE, the XML-Document Oriented Search Engine, is based on a client-server architecture. A server processes incoming indexing and retrieval requests. Query results are sent to the client which displays the document components retrieved. Single results can be selected and displayed in a document view which highlights relevant components in their contexts. By browsing the document tree, users are able to extend or refine initial queries by adding new constraints. A classification machinery supports the user in creating groups of related elements. Both, the architecture of the server and the client are described in detail. Before explaining the system proposed, existing retrieval systems operating on structured documents are reviewed in Section 8.1.

8.1 Related Work

8.1.1 HyREX

HyREX [81, 18, 102, 87], the Hyper-media Retrieval Engine for XML, is a system developed by Gövert and Fuhr at the University of Dortmund. It relies on the XIRQL [86] query language, which extends the XPath subset of XQuery by introducing features for term weighing, relevance-oriented searching, data types and vague predicates, and semantic relativism. During query calculation, XIRQL queries are first translated into a path algebra, which is optimized and processed by the database engine. HyREX aims at retrieving the most specific component of an XML document according to Chiaramella's model for multimedia information retrieval [51]. To achieve that, non-overlapping sets of elements are a priori defined as index nodes (see Figure 8.1). All index nodes are retrievable units for queries. From the representation point of view, indexing nodes are considered as individual documents (atomic units) that are retrievable units for queries. Term weights are computed applying the standard vector space model. During retrieval the system matches the query with each of these index nodes and ranks them according to their relevance.

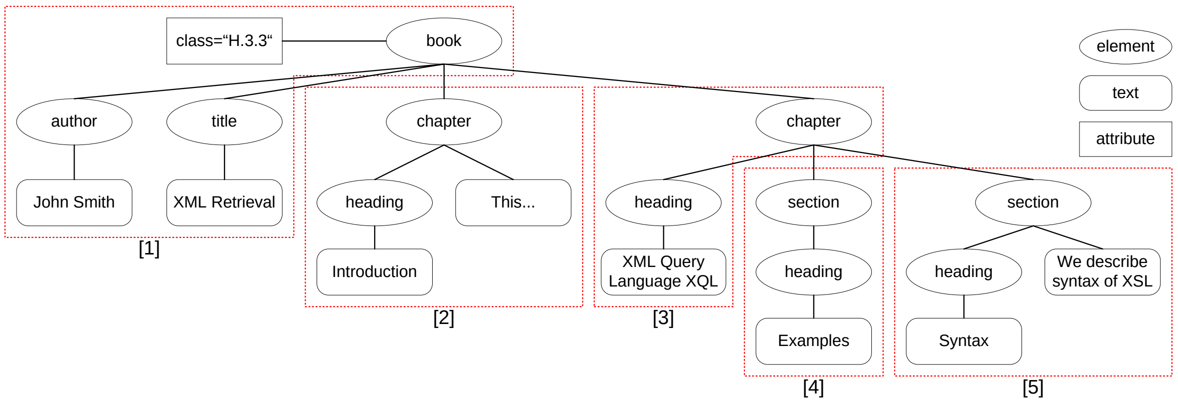


Figure 8.1: XML document tree and corresponding indexing objects [85]

The content of an XML document is restricted to the leaf nodes of the document tree. It is, therefore, necessary to infer representations of inner index nodes based on the leaf nodes. This is done bottom-up and recursively, such that the representation of each intermediate node is based on the representations of its descendant node, relying on *event keys* and *event expressions* [94]. Basically, the weight of terms of an ascendent node is computed using the so-called *inclusion-exclusion* mechanism [27, pp. 20]. Assuming term independency, the merge (join) probability of a term lying in two nodes is computed using a modified version of the traditional formula:

$$P(asc \vee (f \cdot desc)) = P(asc) + P(desc) \cdot f - P(asc) \cdot P(desc) \cdot f \tag{8.1}$$

where *asc* and *desc* indicate a node and its descendant node containing a term t, while $f \in [0,1]$ is an augmentation factor used to moderate the weights propagated upwards to the ascendant node. Different index nodes may be assigned different augmentation factors. In their experiments, Gövert and Fuhr found that augmentation weights chosen from $[0,3;0,7]$ achieved best results. Systematic methods like Amanti's approach of divergence from randomness [17] can be used to identify proper augmentation factors automatically. However, those augmentation factors are pre-estimated using empirical data, which makes weight computation dependent on the corpus. Thus, these values cannot be claimed valid with other data.

Further enhancements of HyREX include bilingual information retrieval for English and German [101], result presentation issues [109, 108, 87], and performance improvements (e.g., index compression) [80, 79]. General information about HyREX is also available via the Internet[1].

8.1.2 HySpirit

HySpirit [207, 208, 95, 152, 209], the HYpermedia System with Probabilistic Inference for the Retrieval of InformaTion, is a scalable hypermedia framework developed by Rölleke et al. The system is based on a probabilistic relational algebra. Based on a probabilistic relational algebra, the system processes large scale retrieval tasks on distributed databases in parallel. In this context, the notion of a local representation of a document refers to the representation of a given element within a sub-collection (independent database), whereas a global representation describes the collection as a whole (all databases). Each element in a document tree is regarded as an atomic unit (separate document). Thus, a level in the hierarchy corresponds a collection of atomic units.

A kind of $tf \cdot idf$ formula is adapted to compute term weights like in traditional information retrieval. Since the concept of a document is not clear, term frequency tf and inverse document frequency idf are interpreted on the basis of XML elements at a certain level in the collection's hierarchy. To do that, an aggregation function is used to combine the idf values of different sub-collection elements using the formula:

$$idf(term_j) = \log \frac{\sum\limits_{i=1}^{n} N_i}{\sum\limits_{i=1}^{n} n_{i,term_j}} \tag{8.2}$$

where N_i is the number of direct subdocuments (child elements) in the collection, and $n_{i,term_j}$ is the number of documents in that collection referring to $term_i$. During retrieval, the mechanism of augmentation is applied to represent the content of elements made up of the contents of their sub-contexts. Target elements of queries, the elements retrieved, are processed by a post-retrieval filtering task.

Database pre-selection based on a cost-function and a content-based measure estimates the efficiency of the sub-collections and speeds up the overall data access. Term selection (stopwords) as well as context selection (stop-contexts) are used to improve indexing and retrieval. Based on database accessibility and access costs, two different strategies combine the query and database dimension in distributed environments (parallel query computation):

- For each query, the system retrieves from the set of databases.
- For each database, the system runs the set of queries.

[1]http://www.is.informatik.uni-duisburg.de/projects/hyrex/ (15.08.2008)

8.1.3 JuruXML

Juru [44] is a full-text information retrieval system developed by Carmel et al. at the IBM Research Lab in Haifa. Juru is implemented in Java and operates on an inverted list index. Novel pruning methods and compression techniques are applied, reducing the size of the index significantly while maintaining high precision.

In 2002, Mass et al. [176] extended Juru by means of XML retrieval features to JuruXML. Their approach is based on querying XML documents via pieces of XML documents, so-called XML fragments, rather than inventing a new query language. This differentiates JuruXML from other systems. Query results returned contain not only perfect matches but matches that are 'close enough', according to some relevance measure. Most of the logic is put to the ranking mechanism, targeting intuitive querying and user friendliness.

Within XML fragments, elements, attributes, and contents can be augmented by prefixes for mandatory, prohibited, and phrase terms. Optionally, a list of target elements to be returned can be included. For better relevance ranking and approximate matching, an extension of the vector space model described in [45] is proposed. In contrast to global term weights, the content of an XML component is expressed by pairs of terms t_i and contexts c_j, where the context refers to the XPath of the component (e.g., `/article[1]/abstract[1]`). A document is considered relevant if at least one path (and its content) of the document matches at least one path (and its content) of the XML fragment (query). The match of paths cr (document and query paths) is expressed by the longest common subsequence. The similarity between a document d and a query q is given by

$$sim(d,q) = \frac{\sum_{(t_i,c_j)\in q} \sum_{(t_i,c_k)\in d} w_d(t_i,c_k) \cdot w_q(t_i,c_j) \cdot cr(c_j,c_k)}{||d|| \cdot ||q||} \tag{8.3}$$

$w_d(t,c)$ and $w_q(t,c)$ denote the weight of term t in context c in the document d and the query q, $cr(c_j,c_k)$ is the similarity of the contexts c_j and c_k, and $||d||$ (resp. $||q||$) denotes the number of contexts in d (resp. q). To this stage, JuruXML takes term statistics only on the document level. Thus, matching and ranking are performed for complete documents only.

8.1.4 XXL Search Engine

Theobald and Weikum describe the XXL Search Engine in [236, 237]. It is based on the XXL (fleXible XML search Language) query language, combining standard $tf \cdot idf$ weighing and ontology searching. The index structure consists of three layers: an element path index EPI, an element content index ECI, and an ontology index OI.

In a first step, XXL queries are decomposed and represented as a graph. After that, the order of the subquery evaluation is chosen. Then, the order of the internal path expressions for each subquery is decided. Subqueries are evaluated making best usage of the three indices. Finally, intermediate subquery results are composed into a global result.

8.1.5 K2 Search Engine from Verity

Tong [238] presented an approach for XML retrieval using K2, an enterprise-class document retrieval platform from Verity, Inc.[2] Although the system does not have an explicit representation for the document structure, 'zone indexing' is applied to distinguish different regions within a document.

Queries are formulated in the Verity Query Language (VQL), a rich and expressive query language that supports standard keyword operators, boolean operators, and weighing. Since the system does not allow to return pointers to specific document elements, a path reporting strategy was adopted that retrieves either the first or the smallest XML element found. An evaluation at INEX 2002 showed that further investigation and revision is needed. The performance of the K2 search engine was clearly lower than that of other systems addressing the document structure explicitly.

8.1.6 Cheshire II

Cheshire II [161] is an XML retrieval system proposed by Larson, formerly developed as an online library catalog system. It is based on a client-server architecture with an embedded database engine. Features include multiple, scriptable clients, and relevance feedback.

Each node in the XML tree is assigned an index data structure (B-TREE), an extraction type (e.g., KEYWORD, EXACTKEY, DATE), and its type of normalization (e.g., STEM, NONE). Queries are represented applying stopword removal, stemming, and word replacements exploiting a wordnet dictionary and thesaurus. Logistic regression is used to match, rank, and retrieve document components of any granularity.

The probability of relevance R given a query Q and a document D is expressed by $P(R|Q,D)$. Since Cheshire II supports probabilistic and boolean searches, a kind of 'Fusion Search' merges the subsets retrieved from different searches to a global result, defined as

$$P(R|Q,D) = P(R|Q_{bool},D) \cdot P(R|Q_{prob},D) \tag{8.4}$$

where $P(R|Q_{bool},D)$ is the probability estimate from the probabilistic portion of the search, and $P(R|Q_{prob},D)$ is the estimate from the boolean operator.

[2]http://www.verity.com (15.08.2008)

8.1.7 PADRE

The PADRE (Parallel Document Retrieval Engine) system [126, 136] was developed by Hawking at the Australian National University of Canberra in 1994. It was the first text retrieval system that implemented indexing, query processing, ranking, and retrieval of text documents on parallel and distributed systems for terabyte-scale collections. PADRE views document retrieval as an inherently parallel problem, since a document collection can be divided into N sub-collections searched independently. The only exception to this are global term statistics (i.e., inverse document frequency) and ranking, discussed by Bailey and Hawking [22].

Mechanisms for load balancing and automatic document distribution improve response time considerably. Further performance improvements are achieved by single-pass scanning of multiple alternate patterns, fast indexing methods, and decreased collection load times [133, 131]. TREC-5 experiments on automatic query generation, distance-based relevance scoring, server selection, and result merging are described in [135]. A multi-user, time-sharing implementation developed in 1996 applied natural language processing for automatic query generation to prose specification texts [133, 132]. Further developments led to WWW capabilities [132] and retrieval of OCR-scanned texts [128]. The initial system already allowed restricted searches on marked sections like `authorname` or `title` [127, 133]. Based on compressed inverted file indices, former versions of PADRE like PADRE97 [130, 129] and PADRE99 [134] outperformed many other state-of-the-art retrieval systems. Today, PADRE is the core of CSIRO's (Commonwealth Scientific & Industrial Research Organisation[3]) Panoptic Enterprise Search Engine[4].

Vercoustre et al. [248] extended PADRE for XML retrieval. In their work, they added additional database techniques to the underlying text retrieval technology. The definition of meaningful retrieval components is implemented as a pre-indexing task. Before indexing, XML documents are split into retrievable units. Both, the complete documents and the split document components are treated as atomic units.

During retrieval, queries are first translated into PADRE syntax. Terms lying in specific XML components are mapped onto corresponding PADRE fields, using a set of metadata classes. This avoids semantic misinterpretation of same-named relational data fields. Dates and texts are defined as basic types. Progressive weakening of query constraints supports retrieval of XML components not fulfilling all conditions. Results returned by PADRE are sent to an extractor unit. According to the query constraints, the extractor re-ranks the PADRE results and presents them to the user.

Currently, an implementation of the system is in productive operation at the Australian National University in Canberra. It is used for email and intranet retrieval, and demonstrates

[3]http://www.csiro.au (24.08.2004)
[4]http://www.panopticsearch.com (24.08.2004)

fast indexing and query processing on an inexpensive hardware. Additional information about PADRE is available via the Internet[5,6,7].

8.2 Architecture of X-DOSE

Taking the aspects discussed in Chapters 3 to 7 into account, the XML-Document Oriented Search Engine (X-DOSE) was developed. The focus has been put on the natural language text representation, and the retrieval improvements achieved by clustering of XML components. X-DOSE is fully implemented in Java and consists of three modular subsystems:

- An external database server stores the documents and their corresponding representations in a relational database.
- The server processes index, query, and classification requests that are transmitted via the Remote Method Invocation (RMI) protocol.
- A client starts index requests, aids the query formulation, enables classification of arbitrary components, and displays the results to the user.

An overview of this architecture is given in Figure 8.2. Triggered by an index request, the server obtains the document from the internet and stores it in the database. During querying, the information in the database is used to retrieve relevant document components. The results are transferred to the client, which displays them to the user. Additionally, links referring to the original source document are provided.

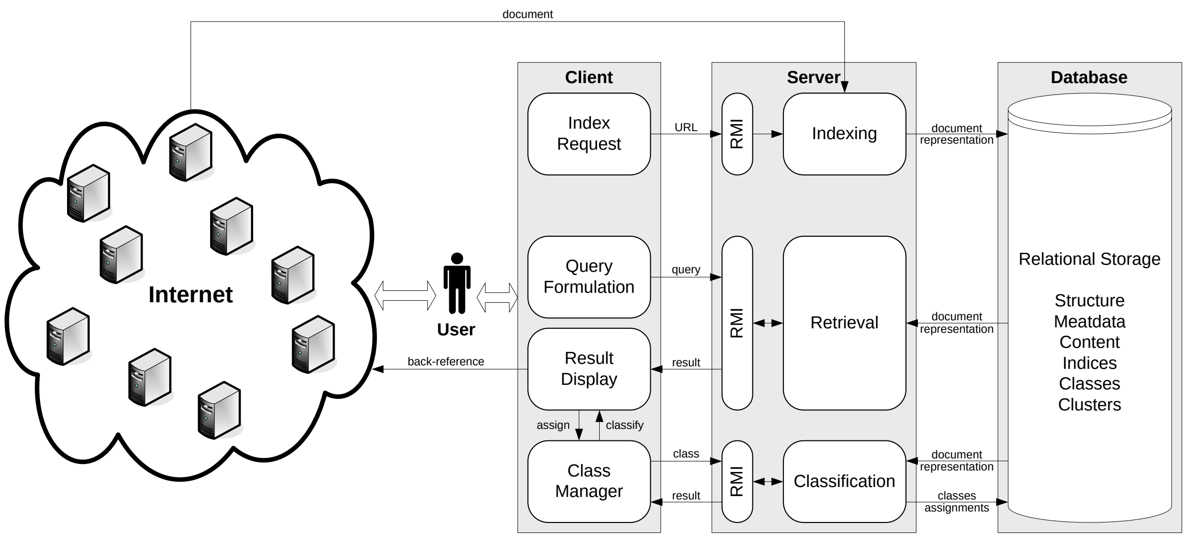


Figure 8.2: Architecture of X-DOSE

[5]http://www.csiro.au (15.08.2008)
[6]http://www.panopticsearch.com (15.08.2008)
[7]http://deneb.soi.city.ac.uk/ andym/PADRE/tarpubs.html (15.08.2008)

In the sequel, the client and its tasks, index requesting, query formulation, result presentation, and class management are described in Section 8.3. Details about the architecture of the server and its corresponding tasks, indexing, retrieval (including clustering), and classification are presented in Section 8.4. Final remarks and future extensions conclude the chapter.

8.3 The Client

The client provides the user with a graphical interface and communicates with the server. An overview of the architecture is given in Figure 8.3. This section describes the main tasks of the client: indexing, querying, displaying of results, and class management.

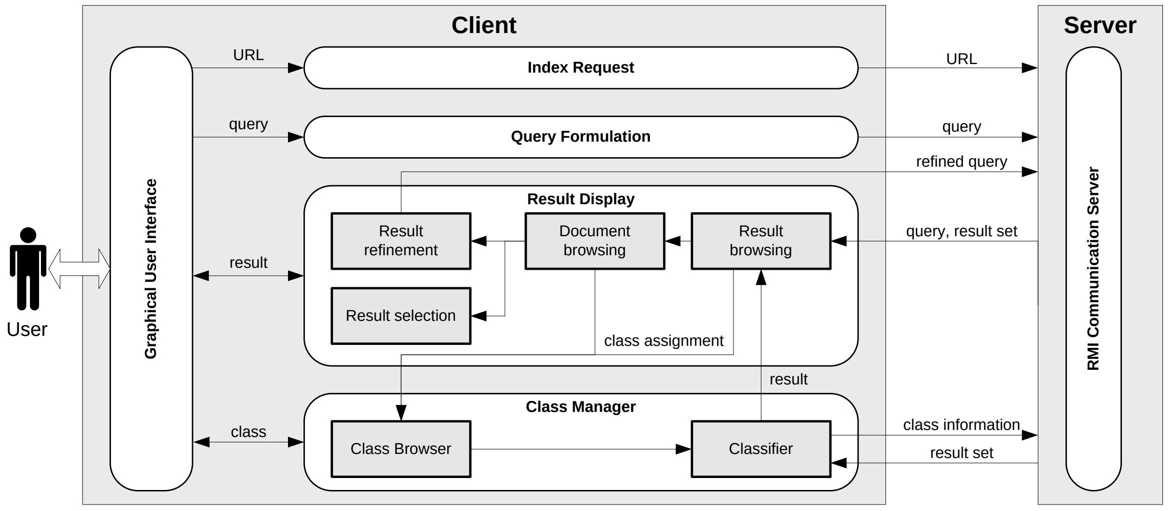


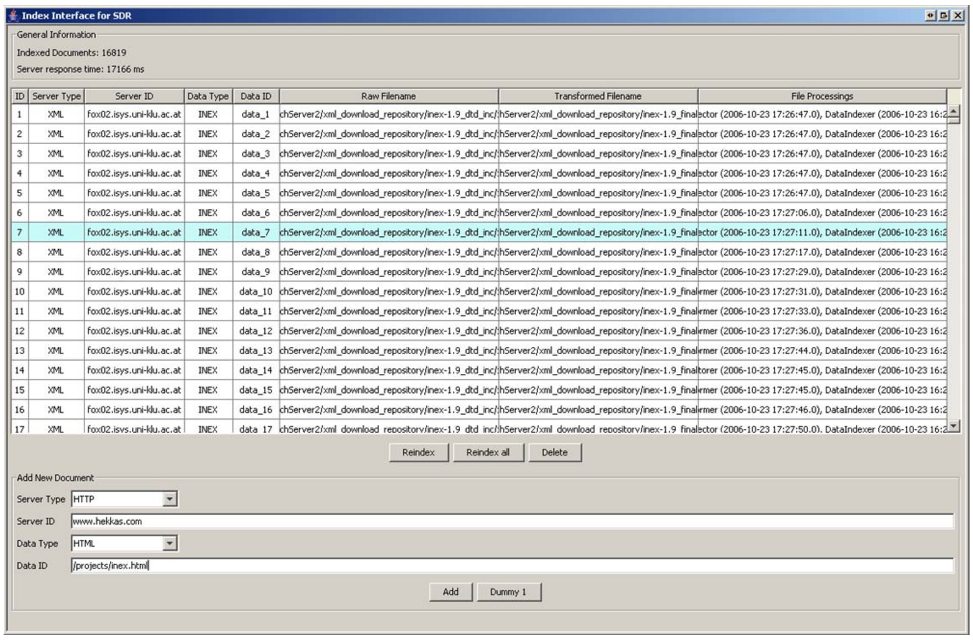
Figure 8.3: Architecture of the client

8.3.1 Index Request

Proper management of the documents indexed is essential to maintain a consistent information pool. Figure 8.4 shows the user interface of the system handling the indexing objects. New documents of different sources (e.g., local files and directories, HTTP, FTP, HIS, etc.) and of various types (e.g., TXT, XML, HTML, PDF, etc.) can be added to the index. Existing documents may be reindexed and obsolete documents removed.

8.3.2 Query Formulation

The client supports three different types of queries: simple *keyword queries*, *free text queries*, and complex *XOR queries*. According to the query type, the client offers three different query interfaces.



General Information

Indexed Documents: 16819

Server response time: 17166 ms

ID	Server Type	Server ID	Data Type	Data ID	Raw Filename	Transformed Filename	File Processings
1	XML	fox02.isys.uni-klu.ac.at	INEX	data_1	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:26:47.0), DataIndexer (2006-10-23 16:2
2	XML	fox02.isys.uni-klu.ac.at	INEX	data_2	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:26:47.0), DataIndexer (2006-10-23 16:2
3	XML	fox02.isys.uni-klu.ac.at	INEX	data_3	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:26:47.0), DataIndexer (2006-10-23 16:2
4	XML	fox02.isys.uni-klu.ac.at	INEX	data_4	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:26:47.0), DataIndexer (2006-10-23 16:2
5	XML	fox02.isys.uni-klu.ac.at	INEX	data_5	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:26:47.0), DataIndexer (2006-10-23 16:2
6	XML	fox02.isys.uni-klu.ac.at	INEX	data_6	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:27:06.0), DataIndexer (2006-10-23 16:2
7	XML	fox02.isys.uni-klu.ac.at	INEX	data_7	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:27:11.0), DataIndexer (2006-10-23 16:2
8	XML	fox02.isys.uni-klu.ac.at	INEX	data_8	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:27:17.0), DataIndexer (2006-10-23 16:2
9	XML	fox02.isys.uni-klu.ac.at	INEX	data_9	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:27:29.0), DataIndexer (2006-10-23 16:2
10	XML	fox02.isys.uni-klu.ac.at	INEX	data_10	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	rmer (2006-10-23 17:27:31.0), DataIndexer (2006-10-23 16:2
11	XML	fox02.isys.uni-klu.ac.at	INEX	data_11	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	rmer (2006-10-23 17:27:33.0), DataIndexer (2006-10-23 16:2
12	XML	fox02.isys.uni-klu.ac.at	INEX	data_12	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	rmer (2006-10-23 17:27:36.0), DataIndexer (2006-10-23 16:2
13	XML	fox02.isys.uni-klu.ac.at	INEX	data_13	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	rmer (2006-10-23 17:27:44.0), DataIndexer (2006-10-23 16:2
14	XML	fox02.isys.uni-klu.ac.at	INEX	data_14	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	torer (2006-10-23 17:27:45.0), DataIndexer (2006-10-23 16:2
15	XML	fox02.isys.uni-klu.ac.at	INEX	data_15	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	rmer (2006-10-23 17:27:45.0), DataIndexer (2006-10-23 16:2
16	XML	fox02.isys.uni-klu.ac.at	INEX	data_16	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	rmer (2006-10-23 17:27:46.0), DataIndexer (2006-10-23 16:2
17	XML	fox02.isys.uni-klu.ac.at	INEX	data_17	chServer2/xml_download_repository/inex-1.9_dtd_inc/	hServer2/xml_download_repository/inex-1.9_final	ector (2006-10-23 17:27:50.0), DataIndexer (2006-10-23 16:2

Add New Document

Server Type	HTTP
Server ID	www.hekkas.com
Data Type	HTML
Data ID	/projects/inex.html

Figure 8.4: Index interface of the client

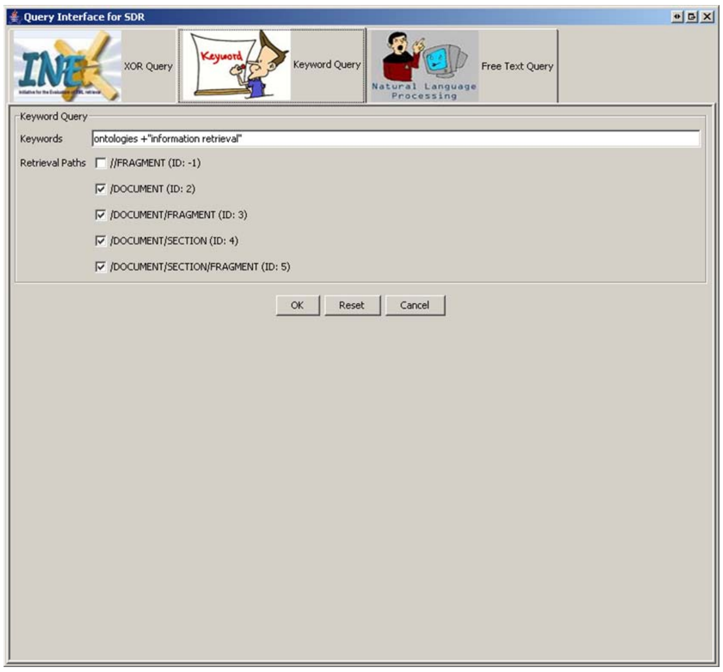
Keyword queries (see Figures 8.5) are considered as a set of search terms (google-like interface). Optionally, document components searched (and retrieved) can be stated to restrict the results to certain XML components. In contrast, free text queries (see Figures 8.6) are formulated as a complete piece of text (e.g., a paragraph). The system retrieves those components that are most similar to the content provided. Both of these query types are translated internally into XOR queries. Thus, this section concentrates on the description of this type of queries. XOR queries (see Figure 8.7) pose the most complex type of queries. These queries allow searches of metadata, structure, and content information. Several parameters described below provide the possibility to tune the query results computed.

The XOR query language was already described in Section 2.3.5 and Section 2.7. These queries are composed of a sequence of subqueries, where each subquery result serves as a filter for subsequent subqueries. Consider the XOR example given in Listing 8.1:

Listing 8.1: XOR query example

```
/DOC[meta(.,documentMeta.title like '\%Computer\%')]
//SEC[(about(.,retrieval) AND about(./FRA,information))]
/FRA[about(.,processing data storage instructions logic)]
```

For better readability, each subquery is written in a separate line. The first subquery retrieves documents with titles containing the term `computer`. Within those documents, the second subquery refines the search to sections at any level (e.g., subsection, subsubsection, etc.) that

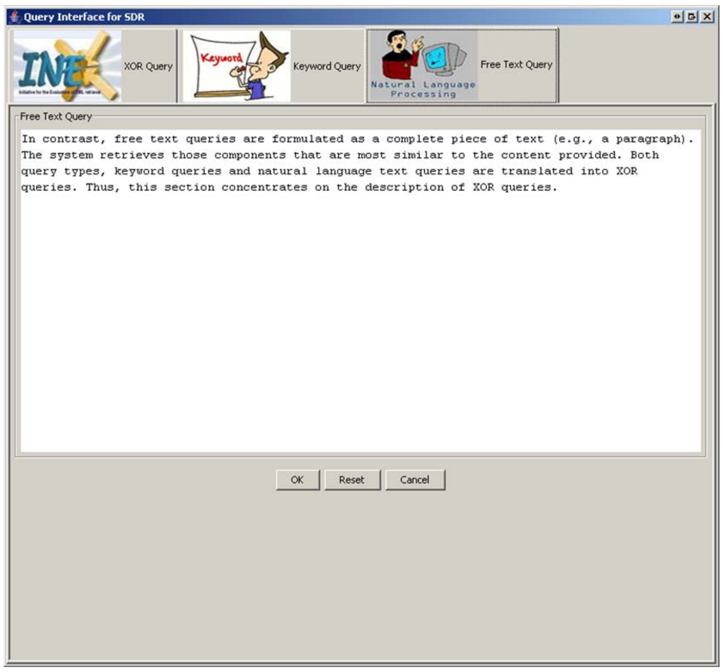


Keyword Query

Keywords: ontologies +"information retrieval"

Retrieval Paths:
- [ ] //FRAGMENT (ID: -1)
- [✓] /DOCUMENT (ID: 2)
- [✓] /DOCUMENT/FRAGMENT (ID: 3)
- [✓] /DOCUMENT/SECTION (ID: 4)
- [✓] /DOCUMENT/SECTION/FRAGMENT (ID: 5)

Figure 8.5: Keyword query interface of the client

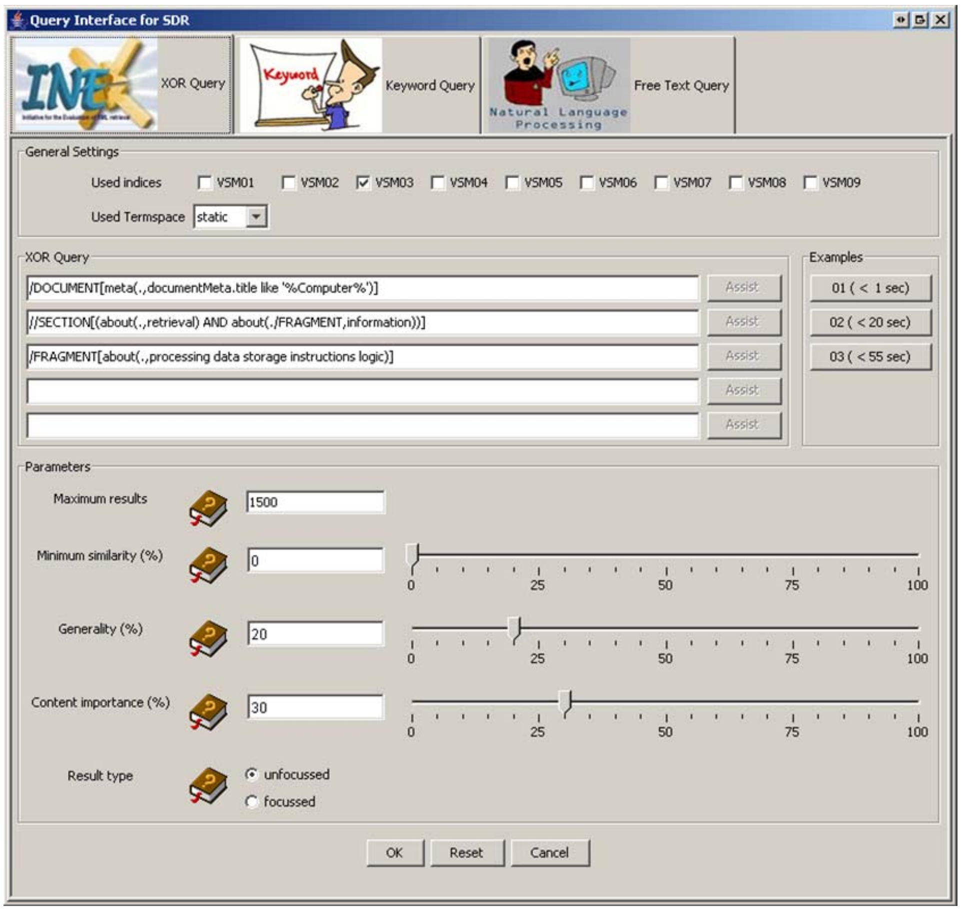


Free Text Query

```
In contrast, free text queries are formulated as a complete piece of text (e.g., a paragraph).
The system retrieves those components that are most similar to the content provided. Both
query types, keyword queries and natural language text queries are translated into XOR
queries. Thus, this section concentrates on the description of XOR queries.
```

Figure 8.6: Free text query interface of the client

include the term `retrieval` and further contain a fragment that deals about `information`. Out of these sections, the third subquery retrieves all fragments that contain any of the terms


Query Interface for SDR

General Settings

Used indices [ ] VSM01 [ ] VSM02 [✓ VSM03] [ ] VSM04 [ ] VSM05 [ ] VSM06 [ ] VSM07 [ ] VSM08 [ ] VSM09

Used Termspace: static

XOR Query

```
/DOCUMENT[meta(.,documentMeta.title like '%Computer%')]
//SECTION[(about(.,retrieval) AND about(./FRAGMENT,information))]
/FRAGMENT[about(.,processing data storage instructions logic)]
```

Examples: 01 (< 1 sec) · 02 (< 20 sec) · 03 (< 55 sec)

Parameters

Maximum results: 1500

Minimum similarity (%): 0

Generality (%): 20

Content importance (%): 30

Result type: [● unfocussed] ○ focussed

Figure 8.7: XOR query interface of the client

`processing`, `data`, `storage`, `instructions`, or `logic`. This set of final fragments is returned to the user as the result of the query. Thus, the retrieval result is defined by the last subquery component.

As indicated by the second subquery in the example, two different aspects have to be distinguished: components searched (search path, support elements) and components retrieved (retrieval path, target elements). The search path specifies the components that are matched against the current query. In contrast, the retrieval path specifies the components that are returned to the user. In most cases, these two paths are equal (e.g., `/FRA[about(.,processing data storage instructions logic)]`. However, the second subquery `//SEC[about(.,retrieval) AND about(./FRA,information)]` is an example where the retrieval path `//SEC` is a subset of the search path `//SEC/FRA`. Note that the search path is always the same or more specific than the retrieval path.

In order to avoid long and confusing single-line queries, query formulation is done by chaining XOR subqueries in the client interface. This concept is closely related to the natural way of questioning, where a query is successively refined by introducing additional constraints (subqueries). Each subquery result operates as a strict filter, allowing only elements of the same or smaller granularity to be retrieved. This improves the performance without skipping relevant components. Furthermore, the chains are used to reweigh elements regarding to a user-defined generality parameter.

The system processes an extended XOR query syntax that supports different kinds of matching. An `about(path,keywords)` predicate matches all keywords against a component's content. Table 8.1 summarizes the modifiers to express different semantics of keywords. `+` (MUST) and `-` (MUST NOT) indicate whether a term must or must not be present in a component. More complex is the treatment of quoted keywords. For instance, `books` and `"books"` have to be treated differently. While `books` is stemmed and matches the term `book` in documents as well, `"books"` searches for contents that explicitly contain the term as it is (a plural noun) in the full text. No kind of linguistic transformation is applied on these terms. Quoted multi-terms are particulary difficult to process. Consider `"red cars"`. The single-term `red` is an adjective. Thus, it may not be included in the index. In another context (e.g., "`Red Cross`"), it is part of a proper noun and, therefore, may exist in the index. The approach proposed treats quoted keywords in two steps: First, all terms are treated as unquoted and their similarity is computed using the standard vector space model of single-term and multi-term indices. Second, instead of searching the indices, a string matching strategy is applied on the full text of the component to re-rank the results. Combinations of `+/-` and quoted expressions are treated as if all single-terms within the quotes are separately marked by `+/-` and an initial result set is computed. This result is reduced to those components that exactly contain the quoted expression (using string matching).

Table 8.1: Different semantics of keywords

Keywords	Semantics
`information retrieval techniques`	All terms optional
`+information +retrieval -techniques`	`information` and `retrieval` must occur, `techniques` must not occur
`"information retrieval" techniques`	`"information retrieval"` optionally occurs as multi-term, all terms are optional single-terms
`+"information retrieval" techniques`	`"information retrieval"` must occur as multi-term, `information` and `retrieval` must both occur as single-terms, `techniques` is optional
`+"running"`	`"running"` must occur unstemmed in the text (string matching)

Similar to the `about()` predicate for searching contents, an extra `meta(path,condition)` search predicate is included for metadata searches. It allows, for example, to efficiently deal with queries like: "return all documents written by Einstein" using the command `//DOC[meta(.,author like '%Einstein%')]`. The result for this query is computed by a single SQL select statement and, thus, is answered by the database engine without further processing.

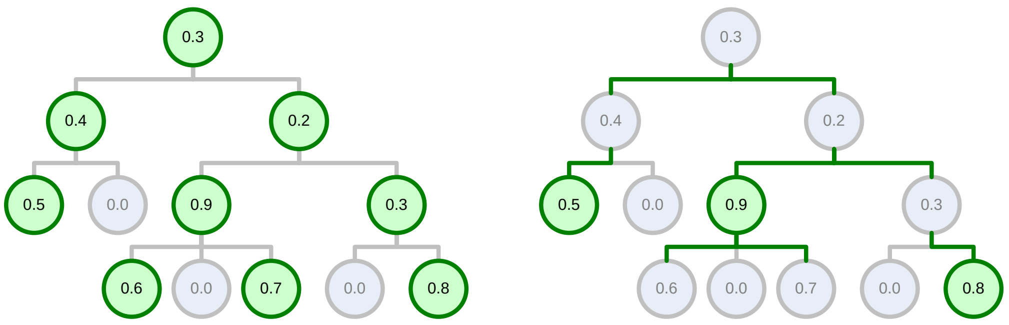
In addition to the XOR subquery chains, several query parameters can be specified by the user (see Figure 8.7):

- **Maximum results (*maxRes*):** Defines the maximum number of results returned. Its values range from 1 to $MAXINT$.
- **Minimum similarity (*minSim*):** Defines the minimum similarity of results returned $\in [0, 1]$, truncating the list of results below a given similarity threshold.
- **Content importance (*ci*):** Defines the influence of the content similarity and the metadata similarity on the calculation of the retrieval status value (*rsv*, element relevance). This parameter ranges from $0,0$ (only *simMeta*) to $1,0$ (only *simCont*). The final *rsv* (ranking criteria) is computed as $rsv = simCont * ci + simMeta * (1,0 - ci)$.
- **Generality factor (*gf*):** This parameter ($\in [0, 1]$) influences the retrieval granularity. The higher the parameter, the more weight is put on the previous subquery relevance. Let $q_{1..n}$ be a chain of n subqueries. Then, *rsv* is recursively computed as $rsv_i = rsv_{i-1} * gf + rsv_i * (1,0 - gf)$.
- **Result type (*rt*):** Defines which kind of results is obtained: focused or unfocused. Unfocused retrieval returns all relevant components, including multiple results of the same document that are in a structural relationship (e.g., the section and its child fragment) to each other. In contrast, focused retrieval only returns the most relevant component of a branch in the document tree. Overlapping elements in the result set are discarded. Note that focused retrieval may also retrieve multiple components of the same document. This strategy reduces the number of elements returned drastically. Figure 8.8 shows the results of the same document according to the result type. Green encircled components are returned. The number in the nodes indicate their relevance (*rsv*).

8.3.3 Result Display

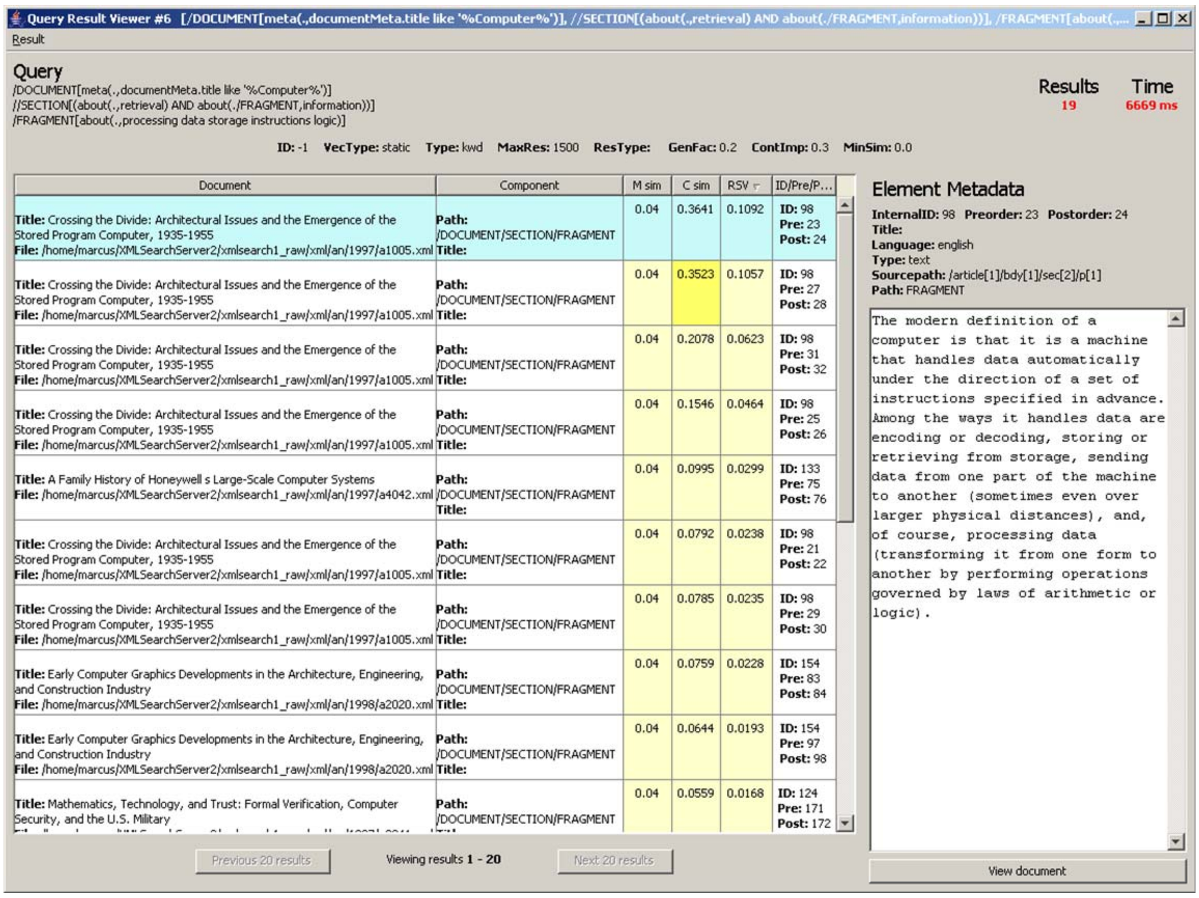
After a query is computed by the server, the list of results is returned to the client. The results are ranked according to their retrieval status value in decreasing order. Besides metadata similarity *metaSim* and content similarity *contSim*, the retrieval status value further combines the similarities of chained subqueries (described in the previous section). Figure 8.9 shows the interface for browsing the results.

If the user selects a particular result (e.g., a relevant subsection), the system displays the whole document in an explorer-like view (see Figure 8.10). Relevant components in



(a) Unfocused retrieval **(b)** Focused retrieval

Figure 8.8: Result types



Query Result Viewer #6 [/DOCUMENT[meta(.,documentMeta.title like '%Computer%')], //SECTION[(about(.,retrieval) AND about(./FRAGMENT,information))], /FRAGMENT[about(.,...

Result

Query
/DOCUMENT[meta(.,documentMeta.title like '%Computer%')]
//SECTION[(about(.,retrieval) AND about(./FRAGMENT,information))]
/FRAGMENT[about(.,processing data storage instructions logic)]

Results 19 Time 6669 ms

ID: -1 **VecType:** static **Type:** kwd **MaxRes:** 1500 **ResType:** **GenFac:** 0.2 **ContImp:** 0.3 **MinSim:** 0.0

Document	Component	M sim	C sim	RSV	ID/Pre/P...
Title: Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955 **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.3641	0.1092	**ID:** 98 **Pre:** 23 **Post:** 24
Title: Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955 **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.3523	0.1057	**ID:** 98 **Pre:** 27 **Post:** 28
Title: Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955 **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.2078	0.0623	**ID:** 98 **Pre:** 31 **Post:** 32
Title: Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955 **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.1546	0.0464	**ID:** 98 **Pre:** 25 **Post:** 26
Title: A Family History of Honeywell s Large-Scale Computer Systems **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a4042.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.0995	0.0299	**ID:** 133 **Pre:** 75 **Post:** 76
Title: Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955 **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.0792	0.0238	**ID:** 98 **Pre:** 21 **Post:** 22
Title: Crossing the Divide: Architectural Issues and the Emergence of the Stored Program Computer, 1935-1955 **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.0785	0.0235	**ID:** 98 **Pre:** 29 **Post:** 30
Title: Early Computer Graphics Developments in the Architecture, Engineering, and Construction Industry **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1998/a2020.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.0759	0.0228	**ID:** 154 **Pre:** 83 **Post:** 84
Title: Early Computer Graphics Developments in the Architecture, Engineering, and Construction Industry **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1998/a2020.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.04	0.0644	0.0193	**ID:** 154 **Pre:** 97 **Post:** 98
Title: Mathematics, Technology, and Trust: Formal Verification, Computer Security, and the U.S. Military	**Path:** /DOCUMENT/SECTION/FRAGMENT	0.04	0.0559	0.0168	**ID:** 124 **Pre:** 171 **Post:** 172

Previous 20 results Viewing results **1 - 20** Next 20 results

Element Metadata
InternalID: 98 **Preorder:** 23 **Postorder:** 24
Title:
Language: english
Type: text
Sourcepath: /article[1]/bdy[1]/sec[2]/p[1]
Path: FRAGMENT

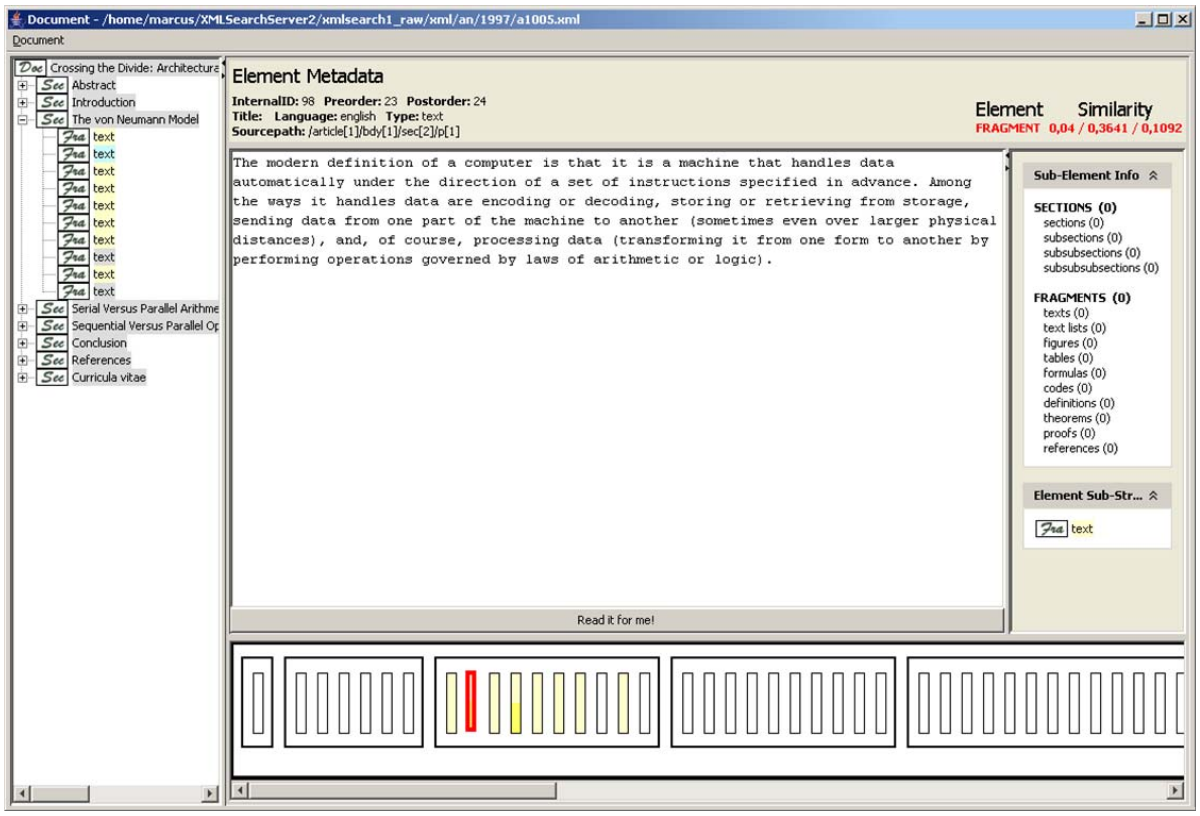
```
The modern definition of a
computer is that it is a machine
that handles data automatically
under the direction of a set of
instructions specified in advance.
Among the ways it handles data are
encoding or decoding, storing or
retrieving from storage, sending
data from one part of the machine
to another (sometimes even over
larger physical distances), and,
of course, processing data
(transforming it from one form to
another by performing operations
governed by laws of arithmetic or
logic).
```

View document

Figure 8.9: Result display of the client

the document are highlighted using different colors reflecting the degree of similarity of the matched elements. The document's structure is presented as an expandable tree, where the selected element is expanded and focused. A graphical representation at the bottom of

Figure 8.10 assists the user in understanding the result's granularity and substructure, similar to the TextBars presented in Section 2.8. Besides outlining the document structure, it can be used alternatively to navigate through the document. The actual component selected is highlighted using a red border. Each component is filled with up to two different colors that reflect its metadata (top half) and content (bottom half) similarity. In the figure, two different kinds of yellow are used. Having similarity values available on the screen in both, the table of contents and the graphical representation, the document can be browsed efficiently.


Document - /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a1005.xml

Document

Doc Crossing the Divide: Architectura
Sec Abstract
Sec Introduction
Sec The von Neumann Model
Fra text
Fra text
Fra text
Fra text
Fra text
Fra text
Fra text
Fra text
Fra text
Fra text
Sec Serial Versus Parallel Arithme
Sec Sequential Versus Parallel Op
Sec Conclusion
Sec References
Sec Curricula vitae

Element Metadata

InternalID: 98 **Preorder:** 23 **Postorder:** 24
Title: **Language:** english **Type:** text
Sourcepath: /article[1]/bdy[1]/sec[2]/p[1]

Element Similarity
FRAGMENT 0,04 / 0,3641 / 0,1092

```
The modern definition of a computer is that it is a machine that handles data
automatically under the direction of a set of instructions specified in advance. Among
the ways it handles data are encoding or decoding, storing or retrieving from storage,
sending data from one part of the machine to another (sometimes even over larger physical
distances), and, of course, processing data (transforming it from one form to another by
performing operations governed by laws of arithmetic or logic).
```

Read it for me!

Sub-Element Info

SECTIONS (0)
sections (0)
subsections (0)
subsubsections (0)
subsubsubsections (0)

FRAGMENTS (0)
texts (0)
text lists (0)
figures (0)
tables (0)
formulas (0)
codes (0)
definitions (0)
theorems (0)
proofs (0)
references (0)

Element Sub-Str...

Fra text

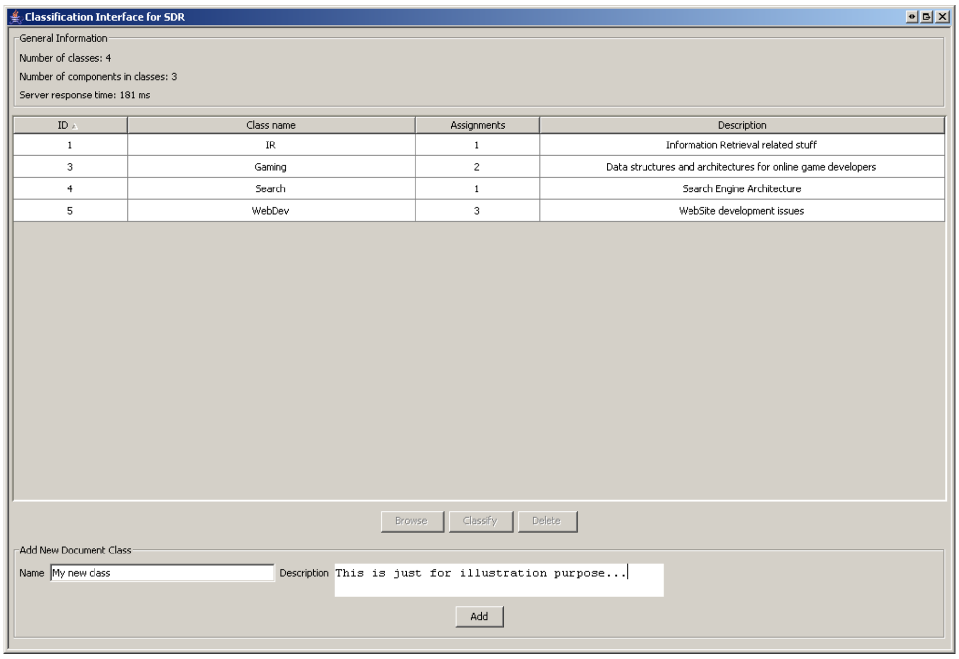
Figure 8.10: Result document browser of the client

In most cases a final search result is achieved through iterative refinement of the query, where the number of results is reduced stepwise by adding new information to the query. To enable such a feature, the user may include a list of preliminary results in a query. If such a result is set, it acts as a strict filter during query processing, avoiding a computation starting from scratch. The system offers the possibility for the user to save (partial) results, R, in response to her/his query Q at a given working session. In future sessions, the user might refine R via a refined query Q'. For this purpose, the query Q' along with the result set R can be introduced to the system. If so, R is the sub-collection of elements to be searched instead of the whole collection, and consequently a refined result set R' is returned.

The information presented by the client solely comes from the documents stored in the database. Documents found highly relevant to the query can be opened in an external application (e.g., a web browser or specific file browser). Therefore, the reference stored in the `sourcepath` attribute in the `METADATA` block of the `DOC` element is used to acquire the original source directly.

8.3.4 Class Manager

The class manager provides methods to create and delete classes of document components of arbitrary granularity. The main management interface is shown in Figure 8.11). Details about



Classification Interface for SDR

General Information
Number of classes: 4
Number of components in classes: 3
Server response time: 181 ms

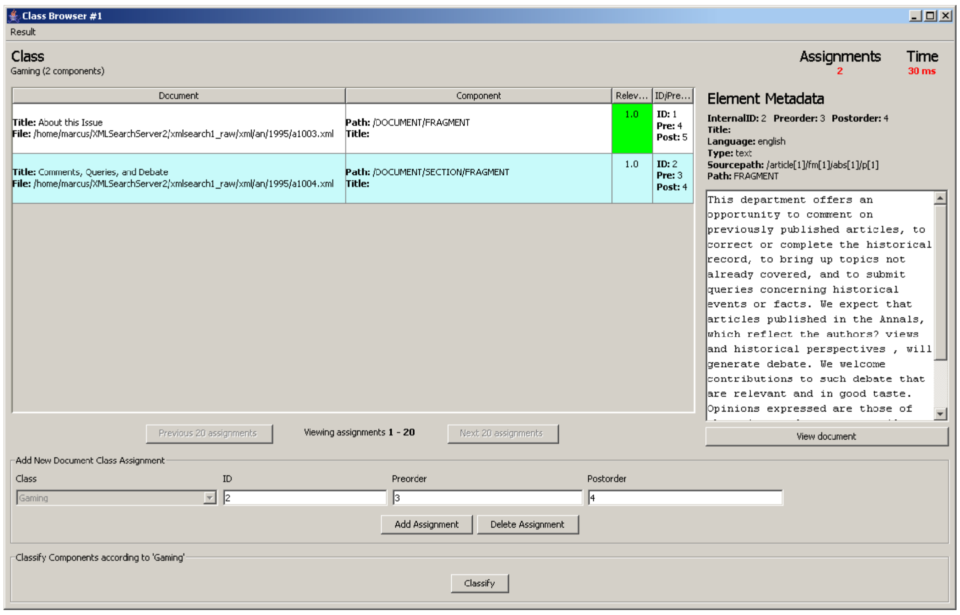
ID	Class name	Assignments	Description
1	IR	1	Information Retrieval related stuff
3	Gaming	2	Data structures and architectures for online game developers
4	Search	1	Search Engine Architecture
5	WebDev	3	WebSite development issues

Add New Document Class
Name: My new class
Description: This is just for illustration purpose...

Figure 8.11: Class manager of the client

the components assigned to a single class are shown by the class browser (see Figure 8.12).

The main concern of the classification system is, besides the compilation of related contents, the automatic identification of document components that are similar to the components of that class. A classification process matches all components known by the system to the members of the class. If the average similarity exceeds a user-defined threshold (e.g., $sim_{avg} > 0,0$), the component is added to the results, which is sorted according to the similarity computed. The result of the classification process (similar to the result of a retrieval process) is presented to the user by the interface illustrated in Figure 8.13. By manually assigning result items to a class, each client creates its own classification system. Note that



Class Browser #1

Result

Class
Gaming (2 components)

Assignments 2 **Time** 30 ms

Document	Component	Relev...	ID/Pre...
Title: About this Issue **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a1003.xml	**Path:** /DOCUMENT/FRAGMENT **Title:**	1.0	**ID:** 1 **Pre:** 4 **Post:** 5
Title: Comments, Queries, and Debate **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a1004.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	1.0	**ID:** 2 **Pre:** 3 **Post:** 4

Element Metadata
InternalID: 2 **Preorder:** 3 **Postorder:** 4
Title:
Language: english
Type: text
Sourcepath: /article[1]/fm[1]/abs[1]/p[1]
Path: FRAGMENT

```
This department offers an
opportunity to comment on
previously published articles, to
correct or complete the historical
record, to bring up topics not
already covered, and to submit
queries concerning historical
events or facts. We expect that
articles published in the Annals,
which reflect the authors? views
and historical perspectives , will
generate debate. We welcome
contributions to such debate that
are relevant and in good taste.
Opinions expressed are those of
```

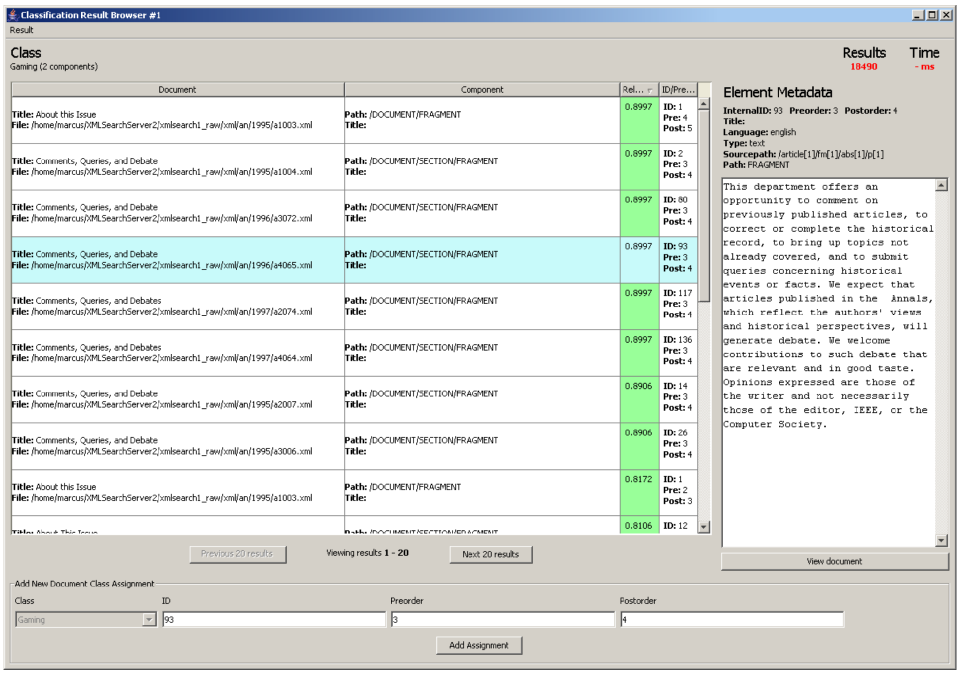
Viewing assignments **1 - 20**

Add New Document Class Assignment

Class	ID	Preorder	Postorder
Gaming	2	3	4

Classify Components according to 'Gaming'

Figure 8.12: Class browser of the client



Classification Result Browser #1

Result

Class
Gaming (2 components)

Results 18490 **Time** - ms

Document	Component	Rel...	ID/Pre...
Title: About this Issue **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a1003.xml	**Path:** /DOCUMENT/FRAGMENT **Title:**	0.8997	**ID:** 1 **Pre:** 4 **Post:** 5
Title: Comments, Queries, and Debate **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a1004.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8997	**ID:** 2 **Pre:** 3 **Post:** 4
Title: Comments, Queries, and Debate **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1996/a3072.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8997	**ID:** 80 **Pre:** 3 **Post:** 4
Title: Comments, Queries, and Debate **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1996/a4065.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8997	**ID:** 93 **Pre:** 3 **Post:** 4
Title: Comments, Queries, and Debates **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a2074.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8997	**ID:** 117 **Pre:** 3 **Post:** 4
Title: Comments, Queries, and Debates **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1997/a4064.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8997	**ID:** 136 **Pre:** 3 **Post:** 4
Title: Comments, Queries, and Debate **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a2007.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8906	**ID:** 14 **Pre:** 3 **Post:** 4
Title: Comments, Queries, and Debate **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a3006.xml	**Path:** /DOCUMENT/SECTION/FRAGMENT **Title:**	0.8906	**ID:** 26 **Pre:** 3 **Post:** 4
Title: About this Issue **File:** /home/marcus/XMLSearchServer2/xmlsearch1_raw/xml/an/1995/a1003.xml	**Path:** /DOCUMENT/FRAGMENT **Title:**	0.8172	**ID:** 1 **Pre:** 2 **Post:** 3
Title: About This Issue	**Path:** /DOCUMENT/SECTION/FRAGMENT	0.8106	**ID:** 12

Element Metadata
InternalID: 93 **Preorder:** 3 **Postorder:** 4
Title:
Language: english
Type: text
Sourcepath: /article[1]/fm[1]/abs[1]/p[1]
Path: FRAGMENT

```
This department offers an
opportunity to comment on
previously published articles, to
correct or complete the historical
record, to bring up topics not
already covered, and to submit
queries concerning historical
events or facts. We expect that
articles published in the  Annals,
which reflect the authors' views
and historical perspectives, will
generate debate. We welcome
contributions to such debate that
are relevant and in good taste.
Opinions expressed are those of
the writer and not necessarily
those of the editor, IEEE, or the
Computer Society.
```

Viewing results **1 - 20**

Add New Document Class Assignment

Class	ID	Preorder	Postorder
Gaming	93	3	4

Figure 8.13: Classification result browser of the client

the top two classification results in Figure 8.13 are the same components initially assigned to that class (Figure 8.12).

8.4 The Server

The main components of the server include an RMI communication server, a threadpool of indexing tasks, a threadpool of retrieval tasks, a threadpool of classification tasks, and a direct data request unit. Figure 8.14 shows the five components and their interconnections.

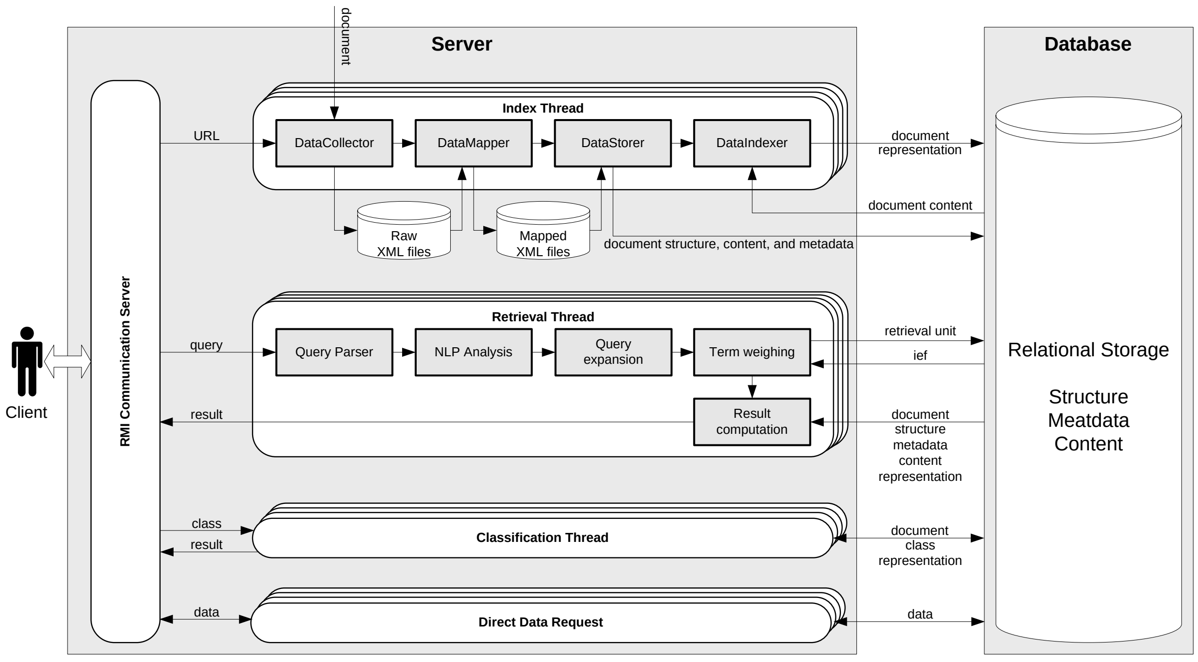


Figure 8.14: Architecture of the server

The RMI server processes incoming requests and initiates a new thread for each call. The goal of this concept is to achieve a high degree of parallelism. The maximum number of parallel threads depends on the performance of the hardware and is configured in a global configuration file. From the software architecture point of view, both index and retrieval tasks use a pipelined pattern of processing units (see Figure 8.14). For portability and tuning purposes, threads and single processing units are independently configurable via separate configuration files.

During indexing, requested documents are downloaded by the DataCollector. The complete information of the source document is kept in a temporary repository of raw XML documents. The DataMapper transforms the raw files into the generic document format and places them in another temporary repository of mapped XML documents. For each raw document format acquired by the DataCollector, an XSLT is defined for the transformation which depends on the original document source format. Afterwards, a DataStorer analyzes the transformed XML documents. Document structure, metadata, and content are stored in the relational database. Finally, a DataIndexer computes multiple representations of the XML components that contain plain natural language text.

As soon as a query is sent to the system, the query content is analyzed in the same way the DataIndexer processed the textual contents previously. If possible, the query is expanded by additional search terms that may improve retrieval results. Query terms are weighed according to the XML components addressed. Documents and document components in the database are matched against the query, and relevant results are ranked in decreasing order. The final result set is transmitted to the client. Users can refine their search within the set of results retrieved, browse relevant documents, and access original source documents by using the references stored in the database during indexing.

In order to classify documents and document components, the client provides an interface for adding and removing classes. Each of the classes contains a number of elements (either documents, sections, or fragments) assigned to that class. By comparing the components in the database to the members of a class, a similarity-ranked result set is computed. Based on the results, users may further assign elements to the class currently being investigated. Applying this process recursively, groups of closely related and user-specific contents are grown continually.

A set of general database requests necessary for the client is supported by a database abstraction layer. Such direct data requests allow fast read and write access to the database via the RMI communication server.

8.4.1 Indexing

This section overviews the indexing task including document acquisition, document mapping, document storage, and computation of the indices.

DataCollector

The task of the DataCollector is to access the documents requested and to store them as lossless XML documents in the raw XML document repository. In this repository, documents are only held temporary during the download. Supporting scalability, the DataCollector can be run in two modes, a single file mode and a collection mode.

The architecture of the DataCollector is given in Figure 8.15. The abstract DataCollector performs five processing steps. First, a data connection between the server and the source document is established, possibly including login information or connection-relevant parameters. The document is transferred as an XML tree into the main memory of the server. Optionally, filtering of irrelevant information reduces the stored data load (e.g., file owner, access rights). Data conversion is carried out to get rid of tags (e.g., HTML markup, formatting information) within textual contents. Finally, the complete XML document is stored in the raw XML document repository. Currently, the system includes a DataCollector implementation for plain text documents, XML documents, HTML documents, and Hyperwave Information Server (HIS) documents. Further extensions may include other formats such as PDF documents, MS-Word documents, or PostScript documents.

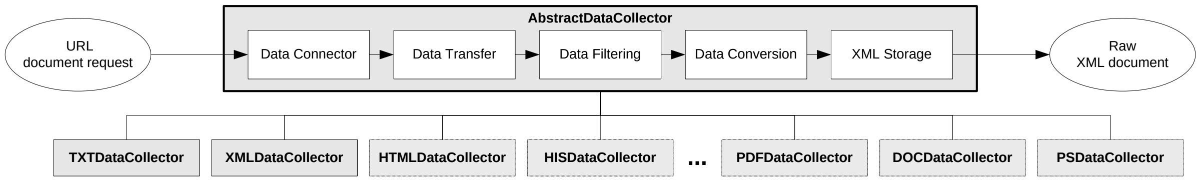


Figure 8.15: The DataCollector

DataMapper

The DataMapper transforms the raw XML documents into the generic XML document format described in Section 3.3. For each of the raw XML document formats (i.e., the source document formats), a separate XSLT stylesheet is defined for the mapping procedure. This simplifies the transformation process, requiring a single coding language (XSL) interpreted by a common XSLT engine only.

Since the INEX 2005 document collection does not account for explicit linkage within or among documents (at least it is not evaluated), the linking functionality of the generic schema is neither exploited in this section nor in the evaluation chapter.

Figure 8.16 explains the three main components of the DataMapper. An XML parser processes the XML documents and outputs them to the XSLT transformation. After transformation, the XML document conforming to the generic document format is stored in the mapped XML repository.

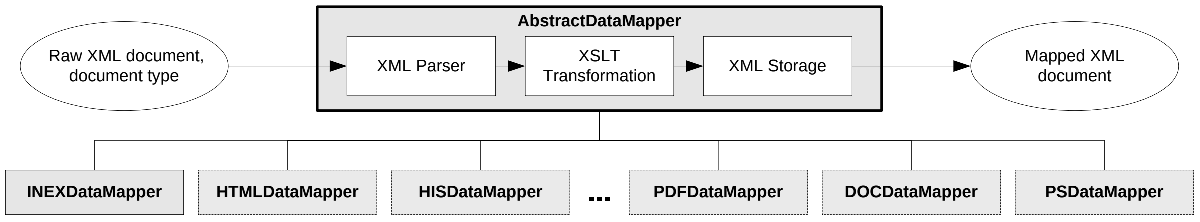


Figure 8.16: The DataMapper

DataStorer

After the documents are transformed into the generic document format, the DataStorer stores the structure, metadata, and content in the relational database. This step involves typing of metadata elements and content elements. Figure 8.17 overviews the components of the DataStorer.

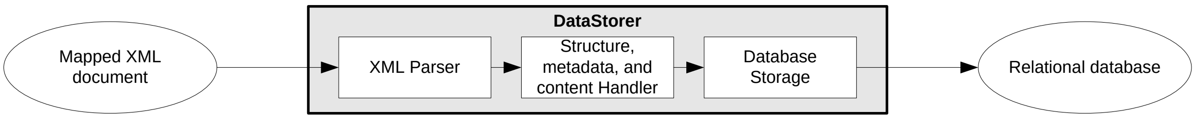


Figure 8.17: The DataStorer

A global view on the database is depicted in Figure 8.18. The tables `files` and `file_processings` keep track of the documents indexed and their processing status. The central `XMLstructure` table holds the structure of the XML documents. XML paths and XML tags are kept in separate tables. Substructure entries such as links and mathematical environments within a fragment's content are stored in the `XMLsubstructure` and `XMLsubstructureContent` tables. Metadata (`Metadata` table) and content (`XMLstructureContent` table) are associated with the document structure, where both entries are optional. In the current implementation metadata entries are considered as unstructured (flat). Different representations of the contents are stored using the vector space model representation framework (gray rectangles comprising blue contents), which is explained in the next section.

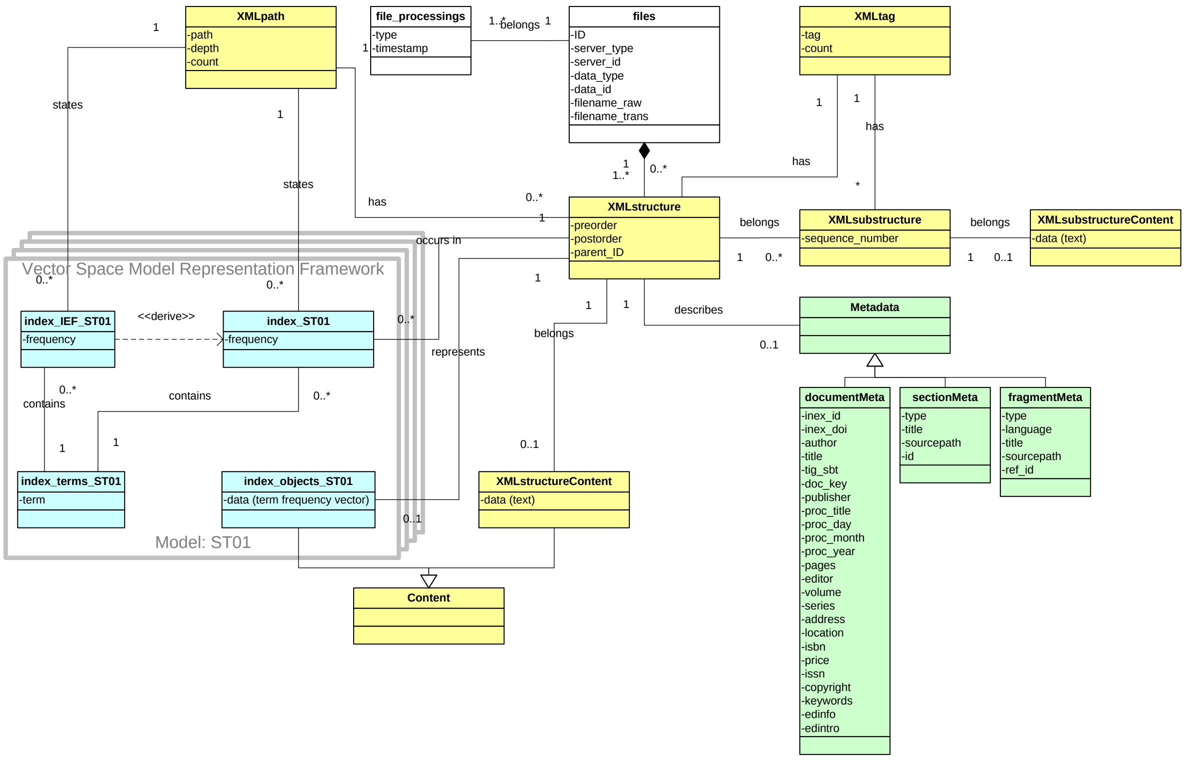


Figure 8.18: Conceptual database schema

DataIndexer

The DataIndexer computes the content representations of the fragments (FRA elements). Optionally, redundant representations of specific inner nodes (e.g., sections at a certain level) can be included to improve retrieval performance. Since the focus of this work rests on text retrieval, the current implementation focuses on multiple representations of plain texts. However, representations of images (e.g., histograms), tables, or other multimedia contents can be included.

An overview of the building blocks of the DataIndexer is given in Figure 8.19. Texts that are indexed are loaded from the database. Natural language processing is applied on the texts to achieve mutually comparable representations. This step includes tokenization, tagging, term extraction, stemming, and term frequency calculation. Besides single-terms (indices ST01 – ST12), the final term frequency vectors include a composite noun vector (index MT01), a named entity vector (index MT02), a formulaic speech vector (index MT03), and a vector of full forms of acronyms (index MT04). Taking advantage of the modularity aspect, different configurations of the natural language processing components are instantiated and selected during runtime. Thus, the system can be easily adapted to process documents in other languages.

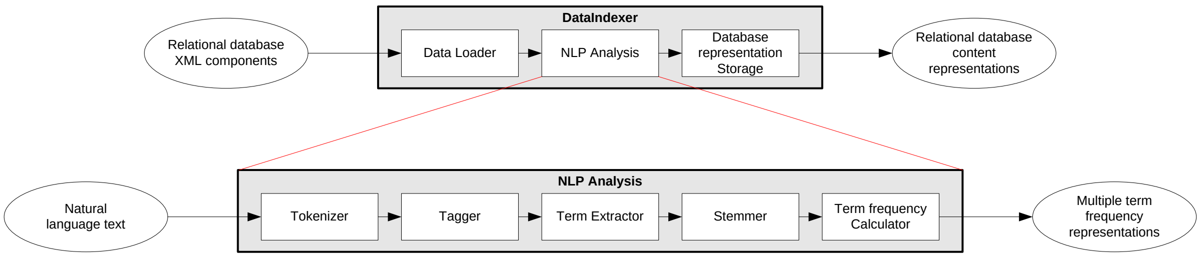

DataIndexer
Relational database XML components
Data Loader
NLP Analysis
Database representation Storage
Relational database content representations
NLP Analysis
Natural language text
Tokenizer
Tagger
Term Extractor
Stemmer
Term frequency Calculator
Multiple term frequency representations

Figure 8.19: The DataIndexer

Each representation is stored in the database tables of the corresponding vector space model representation framework (see the gray rectangles with blue content in Figure 8.18). This framework allows to maintain multiple representations of the same content. It consists of four database tables for each representation model (e.g., `ST01`). A table `index_terms_ST01` holds the terms that are used to represent the content in the `ST01` model. In `index_ST01`, terms and term frequencies are associated with structural entries of the documents. Thus, the table contains the term frequency representations of the document components. Derived from the representation table, `index_IEF_ST01` stores the number of term occurrences in a given XML path. This table is essential to compute the inverse element frequency needed for term weighing. Further, it enables dynamic term space generations as described in Section 3.5. Table updates during indexing, re-indexing, and removal of documents are immediately done. In addition to the term frequency representation table, a table `index_objects_ST01` is provided to store binary representations of document components that do not fit the term frequency representation type. Such representations include semantic concepts (no meaningful ief), figure representations (e.g., histograms), or formulas (e.g., like MathML), to name just some of them.

In order to compare the performance of different representations, several indexing models are tested. Table 8.2 lists the supported single-term indices. These indices are implemented as uncontrolled vocabulary, meaning that unknown terms extracted of new documents are added to the term space.

Table 8.2: Single-term indices maintained by the system

Index	Tokenizer	Tagger	Extractor	Stemmer	Stopword Filtering
ST01	SimpleTokenizer	-	all	-	-
ST02	OpenNLPTokenizer	-	all	-	-
ST03	JavaTok	-	all	-	-
ST04	OpenNLPTokenizer	QTag	nouns, verbs	PorterStemmer	Fox (the best)
ST05	JavaTok	QTag	nouns, verbs	PorterStemmer	Fox (the best)
ST06	JavaTok	QTag	nouns, verbs	PorterStemmer	FS, CR, DS
ST07	JavaTok	QTag	nouns, verbs, adjectives, adverbs	PorterStemmer	FS, CR, DS
ST08	JavaTok	QTag	nouns, verbs, adjectives, adverbs	-	FS, CR, DS
ST09	JavaTok	token types	valid words	PorterStemmer	FS, CR, DS
ST10	JavaTok	token types	valid words	PorterStemmer	FS, CR
ST11	JavaTok	token types	valid words	PorterStemmer	FS
ST12	JavaTok	token types	valid words	PorterStemmer	-

In addition to the single-terms, multi-term indices are maintained to improve matching of term sequences. Table 8.3 presents the set of multi-term indices used. In contrast to the single-terms, multi-terms define a controlled vocabulary. This is necessary because the large number of unique term combinations is inapplicable for run-time query computation. The four indices include the multi-terms extracted in Section 5.7 and Section 5.8.

Table 8.3: Multi-term indices maintained by the system

Index	Description
MT01	JavaTok-based composite nouns of arbitrary length ($\sim$ 340.000 terms)
MT02	JavaTok-based named entities of arbitrary length ($\sim$ 65.000 terms)
MT03	JavaTok-based formulaic speech of arbitrary length ($\sim$ 245.000 terms)
MT04	JavaTok-based full forms of acronyms of arbitrary length ($\sim$ 17.000 terms)

Since indexing only stores term frequency vectors in the database, weight computation is totally executed on-the-fly during the retrieval process.

8.4.2 Retrieval

This section explains the retrieval process. In particular, it describes how the query is parsed, analyzed, and expanded. Term weighing takes place during retrieval only. The matching process is guided by mechanisms improving performance in both regards, computational complexity and quality of retrieval results. These mechanisms include, besides clustering, a set of pre- and post-filtering steps.

Query Parser

Queries introduced to the server by the client are formulated in an extended XOR language syntax. Queries further come in chains of subqueries. Each subquery are parsed by a JavaCC parser implementation, which creates an Abstract Syntax Tree (AST). Listing 8.2, which is replicated from Listing 8.1 for better readability, provides a XOR query example.

Listing 8.2: XOR query example

```
/DOC[meta(.,documentMeta.title like '\%Computer\%')]
//SEC[(about(.,retrieval) AND about(./FRA,information))]
/FRA[about(.,processing data storage instructions logic)]
```

In a first step the search paths and retrieval paths are resolved. The paths of subsequent subqueries are obtained by concatenation. For instance, the retrieval path of subquery 2 is `/DOC//SEC`, and the two search paths are `/DOC//SEC` and `/DOC//SEC/FRA`.

NLP Analysis

According to the index that is used for matching, query terms occurring in the `about()`-predicates are extracted by the DataIndexer (see Section 8.4.1) with the same settings. This results in a term frequency vector of query terms. Terms not occurring in the document term space are neglected. Terms marked with `+/-` are separately kept for fast pre-filtering of single-term representations during matching. Quoted terms are remembered for post-filtering of matched results. During retrieval, the similarity of a query representation and the document representations stored serves to compute the relevance of an XML component.

Metadata information such as titles or author names are not indexed using the vector space model. In order to avoid missing query terms that are not included in the term space (neglected terms), all query terms are compared unstemmed to metadata fields. For instance, the subquery `//*[about(.,to be or not to be)]` matches all elements with metadata fields that contain at least one of the terms `to`, `be`, `or`, or `not` to a certain degree.

Query Expansion

Queries often contain quite specific search terms. In case of general answers expected, related terms can be added to query terms automatically, increasing the number of results returned. Generally, this step covers synonyms of wanted terms and antonyms of unwanted terms. If the query terms added are well chosen, additional results may also be relevant to the same query with high confidence.

However, adding terms to a query automatically may lead to confusion of results retrieved not containing any of the initial query terms (i.e., only terms expanded match the query). Since many terms can be used synonymously, the set of query terms expanded increases rapidly. Further, longer and thus more specific queries are expanded by a larger set of terms than shorter queries. This may be unsatisfying, if the user tried to restrict the search to specific contents excluding somehow similar elements using a specific vocabulary.

Especially domain-specific acronyms pose the possibility of meaningful query term extension. The system proposed uses the acronyms and their full forms (extracted in Section 5.8) to add context-specific acronyms and/or their full forms.

Term Weighing

Weighing of both, query terms and document terms, takes place during retrieval only. The system supports two kinds of term spaces, a static term space and dynamic term spaces (see Section 3.5). Both methods apply the same weighing formula (see Equations 3.3 and 3.7). The weight $w_{i,j}$ of a term i in a component j is given by the term frequency $tf_{i,j}$ multiplied by the inverse element frequency $ief_{i,c}$ of the current search path c (i.e., term space).

During weighing, the granularity aspect is considered in a first place. This means that, according to the searched XML components, the term space and the corresponding inverse element frequency vector are determined first. In order to minimize the number of (dynamic) ief calculations and term space generations, queried elements are grouped according to their expanded search paths. For instance, the search path `/DOC//SEC/FRA` is resolved to `/DOC/SEC/FRA`, `/DOC/SEC/SEC/FRA`, `/DOC/SEC/SEC/SEC/FRA` etc. The corresponding $ief_{i,c}$ values are calculated once per path and kept in main memory. Both, query terms and document terms are mapped onto this common term space and weighed using the vector of inverse element frequencies. Note that using a normalized term weighing function such as the vector space model, the length of a node's content is already taken into account.

Result Computation

The process of computing the result set of a subquery is illustrated in Figure 8.20.

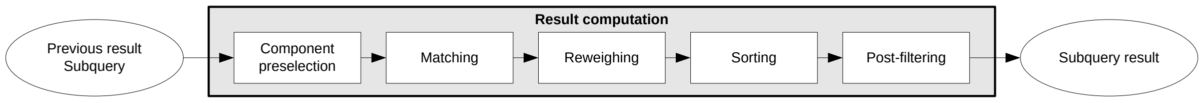


Figure 8.20: Result computation

Each of the subqueries is processed sequentially, where the result of one subquery serves as a filter for the result of subsequent subqueries. A preselection mechanisms observes only components that are feasible for further investigation. It is based on three steps:

1. Components must match the structural constraint (retrieval path). This is accomplished by a single database query on the `XMLstructure` table specifying the `pathID` attribute.
2. If a previous result is available, the component *comp* itself or any of the component's ancestors *anc* must be included. This is equivalent to a retrieval status value above zero ($rsv > 0$). The containment is computed by checking for components $preorder_{anc} \leq preorder_{comp}$ and $postorder_{anc} \geq postorder_{comp}$. In cases where no previous result is set, all components are investigated.
3. Finally, `+`/`-` conditions on query terms exclude components. This filtering step is applied on single-term indices only, because these are based on uncontrolled vocabulary. By checking a components' term vector for the existence (`+`) or absence (`-`) of query terms, the performance of retrieval is boosted.

This concept of preselection ensures that only suitable components are processed, which reduces the number of comparisons computed tremendously.

Matching includes two types of information, metadata and content. While metadata is queried by using the `meta(path,condition)` predicate, content is searched by using `about(path,keywords)`. Metadata similarity, *simMeta*, is computed by addressing metadata fields directly in an SQL-like manner. Consequently, the `meta()` predicate retrieves components that strictly match the condition. If components are retrieved, *simMeta* is defined as the maximum value $1,0$. Otherwise, *simMeta* is set to zero. This allows to compute metadata queries efficiently, because the database retrieves the results directly without further processing. Alternatively, the similarity of content, *simContent*, is evaluated by using the `about()` predicate. The matching procedure is straightforward, applying the cosine similarity formula of the vector space model (see Section 3.5) on the weighed term vectors of the query and the component. In case of multiple indices being used, *simCont* is defined as the average of all computed values.

Hierarchical clustering is applied to improve matching performance of the `about()` predicates. Therefore, the document level (e.g., XML path `/DOC`) is clustered according to the approach described in Chapter 7. Each document initially creates its own cluster. Recursively, the two most similar clusters are merged. The process ends if a single cluster remains. Matching starts at the top cluster. If the content of the cluster is somehow similar to the query (i.e., similarity above zero), the two child clusters are investigated. Leaf clusters are not further investigated but added to the list of investigated components. These remaining components are then compared to the query in the usual manner. For the classification, parameters are set to $\alpha_{struct} = 0,0$ and $\beta_{parent} = 0,2$, which achieved the best evaluation results (see Sections 7.6.2 and 7.6.5). Besides comparing the contents only, the keywords of the `about()` predicate are also compared to the metadata information available. The overall metadata similarity *simMeta* is defined as the number of matching single-terms divided by the smaller number of terms (either keywords or metadata terms).

Subqueries may contain constraints combined with boolean operators. For instance, the second subquery in Listing 8.1, `//SEC[about(.,retrieval) AND about(./FRA,information)]`, combines sections about `retrieval` with sections that include fragments containing the term `information`. During processing, both of these result sets are computed independently. The `AND` operator then computes the intersection of both sets, where elements occurring in both sets are assigned the minimum similarities. The `OR` operator, instead, defines the union of both sets, where elements occurring in both sets are assigned the maximum similarities. The same procedure is applied to `OR`-connected retrieval paths (e.g., `//(SEC|FRA)[about(.,information retrieval)]`.

Having computed the similarity values sim_{new} of the components, reweighing combines these similarities with the similarities of the previous result sim_{prev}. This is done by using the generality factor, the gf parameter, specified in the query. The overall similarities are given by $sim = gf \cdot sim_{prev} + (1,0 - gf) \cdot sim_{new}$. Reweighing is applied on the metadata similarity *simMeta*, the content similarity *simCont*, and the retrieval status value *rsv*.

The computed result consists of tuples of the form (*ID*, *preorder*, *postorder*, *simMeta*, *simCont*, *rsv*). Document *ID*, *preorder*, and *postorder* come directly from the database. *simMeta* and *simCont* are the calculated meta similarity and content similarity. The *rsv* value combines *simMeta*, *simCont*, and the parent-child relationship according to the query parameters *ci* (content importance) and *gf* (generality factor). The list of preliminary results is sorted according to the *rsv* in descending order. Thus, components ranked at the top of the list are most relevant to the query.

Post-filtering of results is done in three steps: First, if the result type is specified as focused, only the component with the highest *rsv* along the path to the root node of it's document is added to the result as best entry point. Retrieval results of the type unfocused are not filtered. Second, result elements not meeting the minimum similarity criteria *minSim* are further discarded. Third, if the number of results still exceeds the number of maximum results *maxRes*, the list is truncated. The final list of results is transmitted to the client for result presentation.

8.4.3 Classification

The system supports classification of arbitrary document components according to user-defined classes. While the client enables users to define those classes and assign components to them, the server is able to retrieve components that are similar to the components of a class. The result of such a 'classification task' is displayed similarly to the retrieval results responding to a user query. This kind of retrieval needs no additional parameters being set by the user.

As mentioned, the key issue of classifying documents is the similarity function that compares two structured document trees. Therefore, the system relies on the *TED_CM* approach of combining tree edit distance *TED_SO* (structure only) with content matrix matching *CM_any* (ignores labels), described in Chapter 6. According to the preliminary evaluation, the parameter combining both approaches, α, is set to $0,5$. *TED_SO* specific parameters are set to $\alpha = 0,5$, $\beta = 2,0$, $C_{del} = 1,0$, and $C_{ins} = 1,0$.

During classification, each component in the database is compared to the set of components assigned to that class. The distances to all components of the class are summed up and averaged over the size of the class. In order to get comparable 'relevance' values, the result list of component distances is normalized on a [0,1] scale and subtracted from 1,0 (similarity instead of distance). Elements not relevant are discarded. The final list is sorted according to the relevance in descending order and sent to the client.

8.4.4 Direct Data Requests

In a client-server architecture, the client often needs to access and/or alter the database directly. These direct data requests help reducing data loads and access times. Further, some of the application logic can be transferred to the client instead of being computed on the server.

The data abstraction layer of the system developed comprises methods for accessing whole documents (structure, metadata, and content), the set of available element paths, document processing information, links to the original source of a document, and general statistics on documents and the document collection.

8.5 Future Extensions

Changing environments, new application domains, and tuning are inherent in software development and software engineering. Consequently, system adaptations are inevitable over time. Besides these changes, a number of future extensions would improve the value of X-DOSE considerably:

- **Metadata:** The treatment of metadata has to be refined. At the moment multiple occurrences of the same metadata, e.g. two authors of a paper, are ignored. Also mechanisms for structured metadata are to be developed.

- **Representation:** The XSLT transformation applied on the INEX corpus is based on a stylesheet consisting of over 2180 lines of code. Thus, the transformation surely faced mapping difficulties with some documents. Those errors were corrected manually. More consistent mapping mechanisms may improve the retrieval results.

- **Index enhancements:** The natural language core processing depends on different internal (JavaTok tokenizer) and external tools (tagger, stemmer). For the experiments, only a small set of possible configurations was tested. Advanced corpus-based analysis techniques may be used to generate resources that further improve language analysis and representation.
- **Content types:** A valuable extension of X-DOSE may be additional indices on other content types than plain text. Database index structures are already provided by X-DOSE to store that data (e.g., binary data). However, models for representing and matching formulae, figures, pictures, and multimedia data still have to be integrated.
- **Retrieval:** Performance improvements, especially of the database, are necessary to reduce query response time.
- **Classification and clustering:** The approaches proposed illustrate the benefits of grouping similar components automatically. However, other approaches with lower computational complexity should be tested for information retrieval.
- **Distributed information retrieval:** In order to cope with large amounts of data, a distributed version of the system seems to be necessary. The systems' architecture is designed to operate independently. However, a central core engine managing the distribution and merging of multiple result sets needs to be designed and implemented.
- **Incorporation of external resources:** Integrating external resources or services like multilingual thesauri or wordnet ontologies may improve the quality of representations. Including the output of other search engines as intermediate results (e.g., Google) may further extend the search space, decrease complexity, and improve the overall retrieval performance.

8.6 Summary

This chapter presented X-DOSE, the XML-Document Retrieval Engine developed. X-DOSE is based on a client-server architecture and relies on a relational MySQL database storage. The systems' main tasks, indexing, retrieval, and classification are described and details about the interaction of the client and the server are highlighted. An easy-to-use graphical interface of the client enables the user to access the server's functionality. Index, retrieval, and classification requests are sent to the server and processed in parallel. A multi-level preselection mechanism and hierarchical clustering are applied to improve computational performance during query processing.

Chapter 9

The outcome of any serious research can only be to make two questions grow where only one grew before.

Thorstein Veblen

Evaluation of X-DOSE

This chapter is concerned with the evaluation of the system described in the previous chapter. Based on an official data set provided by INEX, several experiments were conducted that show the effects and benefits of different search engine settings. Final results are compared to a former X-DOSE version as well as to similar systems of other INEX 2005 participants.

9.1 Experimental Settings

This section presents the preliminaries and experimental settings used for rating the performance of the system implemented. The evaluation is based on the official data set provided by INEX in 2005[1] (see Section 2.9). This data set includes a large document repository, a set of queries or topics, and the corresponding query results handpicked by human experts. A set of retrieval tasks and evaluation metrics used at INEX 2005 enable a comparison of different systems operating on the same data.

9.1.1 Document Repository

The latest version of the INEX document collection v1.9 (used in the Ad Hoc Retrieval track and the Natural Language track) consists of 16.819 articles of the IEEE Computer Society's publications from the field of computer science. The 764 MB collection is organized in 24 folders, each of them representing a particular journal (magazine or transaction). Within these folders, the documents are further subclassified according to the year they were published, starting from 1995 to 2004. All documents are structured according to a DTD of about 700 lines of code, defining 178 different tags. In total, the XML documents contain over 11 million document components. On average, each document contains 687,46 elements an reaches a nested depth of 8,09.

[1] `http://inex.is.informatik.uni-duisburg.de/2005`, (10.10.2008)

In order to feed the collection to the system, the documents are transformed into the generic XML format described in Section 3.3. The XSLT 2.0 stylesheet used for that transformation consists of about 2.200 lines of code, reflecting the structural richness of the documents. The generic schema defines only 33 different tags, which is less than 18,54% of the original DTD. The average depth of the transformed documents is about 9,07, where each document contains 737,53 elements on the average. Since the basic XML elements of the generic format (`DOC`, `SEC`, `FRA`) contains a synthetic level (`METADATA`, `CONTENT`) that is not indexed, the average nesting is reduced to one half ($\sim 4,54$) and the average number of elements per document decreases to one third ($\sim 245,78$). Table 9.1 summarizes the statistics about the documents.

Table 9.1: Document repository statistics

	INEX 2005	Mapped INEX
Number of documents	16.819	
Storage requirement	764 MB	689 MB
Number of different tags	178	33
Number of components	$\sim$ 11.500.000	$\sim$ 4.100.000
Average components per document	687,46	245,78
Average nesting of a document	8,09	4,54

Using Java, SAX [12], and Saxon [13], the transformation process for the whole collection took about 45 minutes. Special characters were retained and escaped by standardized escape sequences defined by W3C in the ISO8879 character map[2], resulting in UTF-8 conformant XML documents. Only 71 out of the 16.819 documents (0,42%) needed manual correction of the structure. In most cases the corrections handled flattening of unnecessary nested structures such as lists within lists, tables within paragraphs, and paragraphs within paragraphs. Only four corrections had to be carried out on the metadata, removing paragraphs from title and keywords fields. All of these corrections had to be conducted carefully because disturbing a documents' (sub)structure consequently leads to a drop of retrieval performance. For instance, removing a table from a paragraph (e.g., `/article/par/tab` becomes `/article/tab`) changes the XPath (the position) of subsequent paragraphs (e.g., `/art[1]/par[3]` becomes `/art[1]/par[2]`).

Due to the mapping of INEX documents onto the generic document schema, some of the INEX queries cannot be evaluated exhaustively by the system. This is because the generic format gets rid of layout-related information (e.g., `<b>`, `<emph>`, etc.) and synthetic elements (e.g., `<bdy>`, `<bm>`, etc.). Other queries cannot be answered exactly: INEX topics addressing `/article/bdy` or `/article` elements are regarded as equal topics addressing the transformed `/DOC` element.

[2]http://www.w3.org/2003/entities/iso8879doc/overview.html (13.01.2009)

9.1.2 Topics

INEX topics are of two types: Content-Only (CO) and Content-And-Structure (CAS) [170]. The type of a topic reflects the knowledge about the document structure in the collection. CO topics refer to queries of users that do not have insight into structure, or simply do not make use of it. Most users are of this type. The second type, CAS topics, include structural knowledge of the documents searched. This information is used as a device for enhancing the precision of the results retrieved. Based on the types, several subtypes focusing on extension and interpretation aspects are defined in INEX 2005:

- **CO subtypes:** Investigating the usefulness of structural hints in queries, CO topics are extended to Content Only + Structure (COS) topics. While pure CO topics consist of content conditions only, COS topics formulate the same query with structural constraints. This enables an evaluation of the same topic with and without structural information across different retrieval systems.

- **CAS subtypes:** CAS topics contain conditions on the content and the structure. Structural constraints include elements that are searched (search path, support elements) and elements that are retrieved (retrieval path, target elements). Both of those elements can be considered as either strict (S) condition (path must be matched exactly) or vague (V) condition (path is simply a hint). According to these interpretations, VVCAS, SVCAS, VSCAS, and SSCAS topics are distinguished. The first letter defines the elements retrieved, and the second letter the elements searched.

All of the topics were created by the INEX 2005 participants, where each party was asked to submit up to 6 candidate topics (3 CO/COS and 3 CAS). The final set of 87 topics consisted of 40 CO (28 COS), and 47 CAS topics. The complete list of INEX 2005 topics can be found in the Appendix in Section A.6.

Result assessments of the topics were also carried out by the participants. Each party was assigned two to three topics to evaluate, where topics were assigned multiple times to cross-check the assessments. Supported by the XML Retrieval Assessment Interface (X-RAI), relevant topics results were highlighted manually. After the assessment phase, the participant was allowed to access the complete set of topic assessments and granted access to the results submitted by other INEX participants.

As the system operates on transformed INEX documents, the topics were adapted to fit the generic document model. This step included element path renaming (e.g., `article` elements became `DOC` elements) and metadata inclusion (e.g., `//fm/au[about(.,Einstein)]` became `/DOC[meta(AUTHOR,Einstein)]`, `.//yr<=2000` became `/DOC[meta(YEAR,<=2000)]`). COS topic 212 was the only topic that included a structural count constraint (`.//en > 3`). This condition on the structure had to be removed because this type of constraint is not yet implemented in X-DOSE.

9.1.3 Retrieval Tasks

In INEX 2005, the Ad Hoc retrieval track was concerned with the evaluation of the topic results achieved by different retrieval systems. A set of assumptions regarding the output of those systems led to the definition of special retrieval tasks. According to the query types, several retrieval strategies were distinguished in INEX:

- **CO.Thorough:** This task is considered as the 'basic' retrieval task that returns all relevant components within the collection. Overlap within the results is not a concern of this tasks, which may lead to a large number of overlapping elements. Main focus is put on the ranking mechanisms of the results.
- **CO.Focused:** This strategy is supposed to return the most exhaustive and most specific element along a single XPath within a document. No overlapping elements are allowed in the result, targeting the appropriate level of granularity. If parent and child elements are equally relevant, the parent element is to be returned.
- **CO.FetchBrowse:** The fetch and browse task combines document retrieval and element retrieval strategies. It consists of two phases: A first fetching phase ranks the documents according to their relevance. In a second browsing phase, elements within a document are compared to other elements within the same document. According to this intra-document relevances, elements are ranked and returned by the system.
- **COS.Thorough:** Same task as the CO.Thorough strategy, but considering constraints on the structure.
- **COS.Focused:** Same task as the CO.Focused strategy, but considering constraints on the structure.
- **COS.FetchBrowse:** Same task as the CO.FetchBrowse strategy, but considering constraints on the structure.
- **VVCAS:** Following the thorough strategy, vague matching of elements retrieved and vague matching of elements searched is applied.
- **SVCAS:** Following the thorough strategy, strict matching of elements retrieved and vague matching of elements searched is applied.
- **VSCAS:** Following the thorough strategy, vague matching of elements retrieved and strict matching of elements searched is applied.
- **SSCAS:** Following the thorough strategy, strict matching of elements retrieved and strict matching of elements searched is applied.

Out of these tasks, X-DOSE is evaluated on the basis of CO.Thorough, CO.Focused, COS.Thorough, and COS.Focused. FetchBrowse tasks were not considered because X-DOSE has not implemented that strategy. Since X-DOSE processes structural conditions as strict filters to improve computational retrieval performance, SSCAS is the appropriate strategy that is evaluated. Vague matching of support and target elements was left open for further experiments.

9.1.4 Evaluation Metrics

In INEX 2005, two kinds of official metrics were introduced to evaluate the performance of XML retrieval systems. A recall-oriented measure at fixed ranks based on cumulated gain, and a precision-oriented effort-precision/gained-recall measure. Both measures are computed by EvalJ [3], an open source evaluation software implemented for INEX. Since performance values of other INEX 2005 systems are available for comparison, X-DOSE is evaluated according to these measures for comparison.

eXtended Cumulated Gain (xCG) Measures

Cumulated gain [143] measures reflect the number of results expected among the number of results retrieved at a fixed cutoff point. An extension of those metrics at INEX led to a new set of eXtended Cumulated Gain (xCG) measures [149], making structured document retrieval performance judgements more accurate. The xCG measure is defined as a vector of accumulated gain. The cumulated gain at a given rank i is computed as the sum of all relevance scores $xG[j]$ up to that rank (Equation 9.1).

$$xCG[i] = \sum_{j=1}^{i} xG[j] \tag{9.1}$$

For each topic, the ideal gain vector xI is derived from the recall-base acquired in the topic assessment phase. The corresponding accumulated ideal gain vector of the optimal results is referred to as xCI. The final normalized extended cumulated gain $nxCG$ score is given by Equation 9.2. For any rank, $nxCG = 1,0$ represents an ideal result.

$$nxCG[i] = \frac{xCG[i]}{xCI[i]} \tag{9.2}$$

The relevance of a component is computed based on its exhaustivity ($e \in \{0, 1, 2\}$)[4] and specificity ($s \in [0, 1]$) values assigned during the assessment. According to these (e, s) pairs, three quantization functions are defined:

[3]`http://evalj.sourceforge.net` (12.10.2008)

[4]Officially, INEX defined exhaustivity values as $e \in \{?, 0, 1, 2\}$, where $e =?$ denotes elements judged as 'too small'. Applying the $nxCG$ measure, these elements were processed as $e = 0$. [149, pp. 17]

$$quant_{strict}(e,s) = \begin{cases} 1 & \text{if } e = 2 \text{ and } s = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9.3}$$

$$quant_{gen}(e,s) = e \cdot s \tag{9.4}$$

$$quant_{genLifted}(e,s) = (e+1) \cdot s \tag{9.5}$$

The last function, $quant_{genLifted}$, enables 'too small' elements to be considered as near-misses.

These quantization functions are applied to compute the $xG[j]$ values used in Equation 9.2. According to the different retrieval tasks, a set of relevance value functions, referred to as rv, were defined. For the thorough retrieval tasks, the cumulated gain is given by

$$xG[j] = rv(c_i) = quant((e,s)_i) \tag{9.6}$$

where c_i is the component at rank j, $quant$ is one of the three quantization functions mentioned, and $(e,s)_i$ is the assessed exhaustivity-specificity pair of c_i. For the focused retrieval tasks, two aspects of structured document retrieval results are considered: Near-misses (e.g., neighboring paragraphs, container sections) and overlap (e.g., a paragraph and its container section are both retrieved). Near-misses are introduced as rewards of non-ideal components retrieved that are structurally related to ideal components. The set of structurally related components consists of relevant components (as per quantization function) that are not included in the ideal result set. Overlap is explicitly included in the relevance value function:

$$xG[j] = rv(c_i) = \begin{cases} quant((e,s)_i) & \text{if } c_i \text{ has not yet been seen} \\ (1-\alpha) \cdot quant((e,s)_i) & \text{if } c_i \text{ has been fully seen} \\ \alpha \cdot \frac{\sum_{j=1}^{m}(rv(c_j) \cdot |c_j|)}{|c_i|} + (1-\alpha) \cdot quant((e,s)_i) & \text{if } c_i \text{ has been partially seen} \end{cases} \tag{9.7}$$

m is the number of child components of c_i, $|\cdot|$ is the length of an element in characters or words, and $\alpha \in [0,1]$ is a user's intolerance factor of redundant components in the result. The higher the value α, the less interested the user is in any overlapping result.

A normalization function rv_{norm} safeguards against higher relevance values of components (by summing the relevances of its child nodes) than that of the ideal node.

$$xG[j] = rv_{norm}(c_i) = \min(rv(c_i), rv(c_{ideal}) - \sum^{S} rv(c_j)) \tag{9.8}$$

In the formula, c_{ideal} is the ideal node that lies on the same relevant path as c_i, and S is the set of child nodes c_j of the ideal node that has already been seen. Figure 9.1 illustrates the behavior of the normalization function. In the example, yellow nodes c_i and c_j are retrieved. Their relevance values are computed by the retrieval system and given by rv. The ideal node to be retrieved is c_{ideal}, which is not included in the result set. For c_{ideal}, the human judgement of its retrieval value is $rv = 0,9$. In the given scenario, the normalization function limits the retrieval value of node c_i to $rv_{norm} = min(0,7,\ 0,9 - (0,2 + 0,2 + 0,2)) = 0,3$, although the retrieval value computed for c_i is $rv = 0,7$.

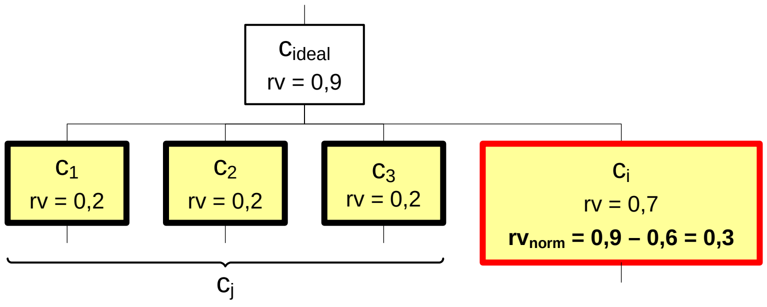


Figure 9.1: Normalization function rv_{norm} of $nxCG$

Effort-Precision and Gain-Recall ($ep - gr$) Measures

The *ep*-*gr* measure [149] reflects the effort of users required to reach a given level of cumulated gain. Therefore, the given result ranking is compared to the ideal ranking. Formally, effort-precision ep is given by

$$ep[r] = \frac{i_{ideal}}{i_{run}} \tag{9.9}$$

where i_{ideal} is the rank position at which the cumulated gain of r is reached by the ideal gain vector xCI, and i_{run} is the rank position at which the same cumulated gain is reached by the system xCG. A value of $1,0$ refers to an optimal performance.

Gain-recall, gr, is computed as the cumulated gain value divided by the total cumulated gain achievable:

$$gr[i] = \frac{xCG[i]}{xCI[n]} = \frac{\sum_{j=1}^{i} xG[j]}{\sum_{j=1}^{n} xI[j]} \tag{9.10}$$

where n is the total number of documents in the recall base and $xI[j]$ is the ideal gain vector. In *ep*-*gr* graphs, effort-precision is plotted against gain-recall (similar to traditional recall/precision graphs), providing a global summary of a system's overall performance.

9.2 Results

This section describes the experiments conducted and results achieved. Nine different sets of experiments were performed to evaluate several indexing strategies, X-DOSE parameter settings, time measurements, performance improvements due to clustering, and an overall comparison to similar systems. The main focus of the experimental evaluation was put on the following INEX retrieval tasks: CO.Thorough, CO.Focused, COS.Thorough, COS.Focused, and SSCAS. For each retrieval task, the complete set of CO topics (40), COS topics (28), and CAS topics (47) was processed.

For better readability, experiments and discussions focus on the strict $nxGC$ measure. The complete set of $nxCG$ evaluation measures is included in the Appendix in Section A.7. Times for indexing and retrieval are relative times according to the maximum time needed for the current set of tasks investigated. Since the experiments were run on multiple virtual machines and in parallel, measurements are meant for relative comparison only. Relying on absolute times would be misleading and inappropriate.

9.2.1 Experiment I - Single-Term Index Performance

The goal of the initial experiment was to identify the single-term index that performs best. All twelve single-term indices (ST01 – ST12, see Table 8.2) were evaluated and compared to each other. This allowed to (1) measure the effect of each text analysis step independently, and to (2) find an optimal overall configuration of analysis steps. For the tests, the complete set of CO topics was used. Query parameters were fixed at $maxRes = 1500$ (official threshold used at INEX), $minSim = 0,0$, $ci = 0,2$, $gf = 0,2$, and $rt = unfocused$ (CO.Thorough). Further, a static term space was presupposed in the experiment. For better readability, Table 8.2 explaining the single-term index configurations is replicated in Table 9.2.

Table 9.2: Single-term indices maintained by the system

Index	Tokenizer	Tagger	Extractor	Stemmer	Stopword Filtering
ST01	SimpleTokenizer	-	all	-	-
ST02	OpenNLPTokenizer	-	all	-	-
ST03	JavaTok	-	all	-	-
ST04	OpenNLPTokenizer	QTag	nouns, verbs	PorterStemmer	Fox (the best)
ST05	JavaTok	QTag	nouns, verbs	PorterStemmer	Fox (the best)
ST06	JavaTok	QTag	nouns, verbs	PorterStemmer	FS, CR, DS
ST07	JavaTok	QTag	nouns, verbs, adjectives, adverbs	PorterStemmer	FS, CR, DS
ST08	JavaTok	QTag	nouns, verbs, adjectives, adverbs	-	FS, CR, DS
ST09	JavaTok	token types	valid words	PorterStemmer	FS, CR, DS
ST10	JavaTok	token types	valid words	PorterStemmer	FS, CR
ST11	JavaTok	token types	valid words	PorterStemmer	FS
ST12	JavaTok	token types	valid words	PorterStemmer	-

The $nxCG$ results of each of the indices at ranks 10, 25, and 50 (official ranks used at INEX), are presented in Table 9.3. Note that in the $nxCG$ figures, these official ranks lay at very small x-axis values (0,007, 0,017, and 0,033). In the sequel, each of the text analysis steps is investigated in more detail.

Table 9.3: *nxCG* of the CO.Thorough task (single-terms)

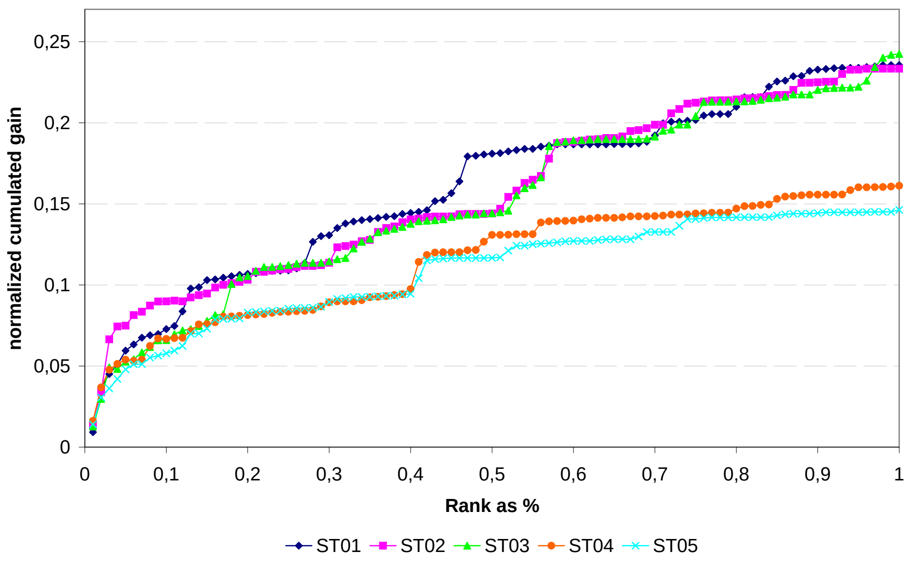
		gen *nxCG*			strict *nxCG*			genLifted *nxCG*		
Index	Size	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
ST01	1.272.103	0,1524	0,1649	0,1660	0,0115	0,0442	0,0514	0,1745	0,1841	0,1809
ST02	741.688	0,1544	0,1760	0,1850	0,0231	0,0387	0,0760	0,1795	0,1998	0,2034
ST03	711.832	0,1403	0,1660	0,1744	0,0115	0,0277	0,0474	0,1666	0,1889	0,1924
ST04	479.644	0,1559	0,1575	0,1504	0,0197	0,0399	0,0502	0,1831	0,1783	0,1672
ST05	460.992	0,1415	0,1490	0,1426	0,0158	0,0328	0,0417	0,1647	0,1693	0,1591
ST06	461.205	0,1506	0,1519	0,1406	0,0197	0,0328	0,0487	0,1763	0,1719	0,1569
ST07	516.742	0,1567	0,1570	0,1524	0,0197	0,0381	0,0454	0,1806	0,1760	0,1686
ST08	555.777	0,1478	0,1575	0,1537	0,0115	0,0308	0,0373	0,1767	0,1815	0,1719
ST09	302.204	0,1435	0,1334	0,1378	0,0235	0,0274	0,0447	0,1606	0,1509	0,1530
ST10	302.248	0,1462	0,1396	0,1378	0,0235	0,0304	0,0482	0,1673	0,1591	0,1541
ST11	302.559	0,1459	0,1418	0,1385	0,0235	0,0320	0,0431	0,1680	0,1608	0,1542
ST12	302.755	0,1430	0,1422	0,1401	0,0197	0,0313	0,0416	0,1649	0,1615	0,1540

Tokenizer Performance

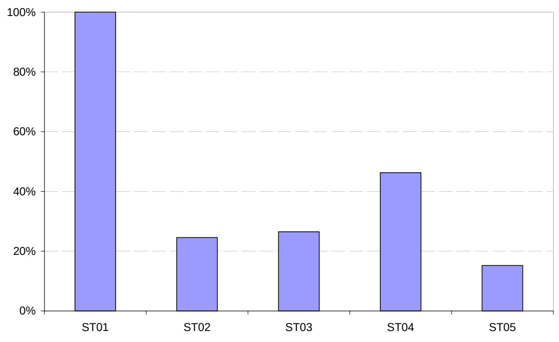
The performance of tokenization was investigated by comparing the results achieved by the indices ST01, ST02, ST03, ST04, and ST05. Figure 9.2 shows the *nxCG* curves and the corresponding processing times. The x-axis in Figure 9.2a denotes the percentage of result components achieving the current cumulated gain. For the INEX tasks, only the 1.500 top-ranked results are considered. The y-axis is the normalized cumulated gain at a given number of results.

Retrieval results achieved by the indices ST01 (no tokenizer, no tagger), ST02 (OpenNLP-Tokenizer, no tagger), and ST03 (JavaTok, no tagger) differed only to a small degree. There was no obvious trend that one of the three tokenization approaches outperformed the others. Taking tagging into account, ST04 (OpenNLPTokenizer, QTag) returned marginally better results than ST05 (JavaTok, QTag). For the indices (ST01, ST02, ST03, ST04, and ST05), the number of index terms differed considerably (1.272.102, 741.688, 711.832, 479.644, and 460.992). Compared to ST01, term space was reduced by 41,7% in ST02, 44,0% in ST03, 62,3% in ST04, and 63,8% in ST05. This reduction led to a considerable speed up of indexing and retrieval (Figures 9.2b and 9.2c). As for indexing time, 100% was equivalent to 75,8 hours and 100% of retrieval time stood for 141,8 hours. In subsequent experiments, the longest taking indexing (resp. retrieval) time is also referred to as 100% indexing (resp. retrieval) time.

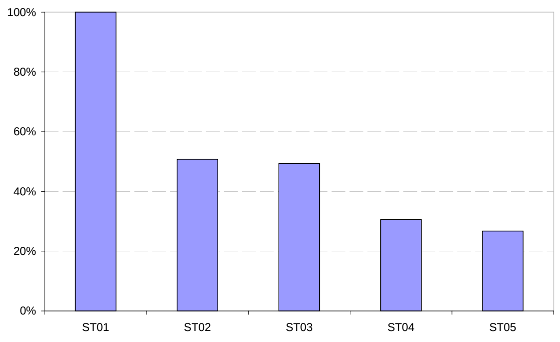
Interestingly, the OpenNLPTokenizer slightly outperformed JavaTok according to the retrieval quality in case when tagging was involved. This became evident by the ST04 and ST05 curves. The reason for this behavior is threefold: First, JavaTok does not include the complete set of punctuation marks used in the documents. This leads to tokens that include unknown markers at the beginning or at the end of token strings. Such tokens are not typed as 'proper' words and are, consequently, not included in the index. Second, OpenNLPTokenizer segments hyphenated words into multiple separate tokens (e.g., `element-based` is split into



(a) strict $nxCG$ Performance



(b) Indexing times



(c) Retrieval times

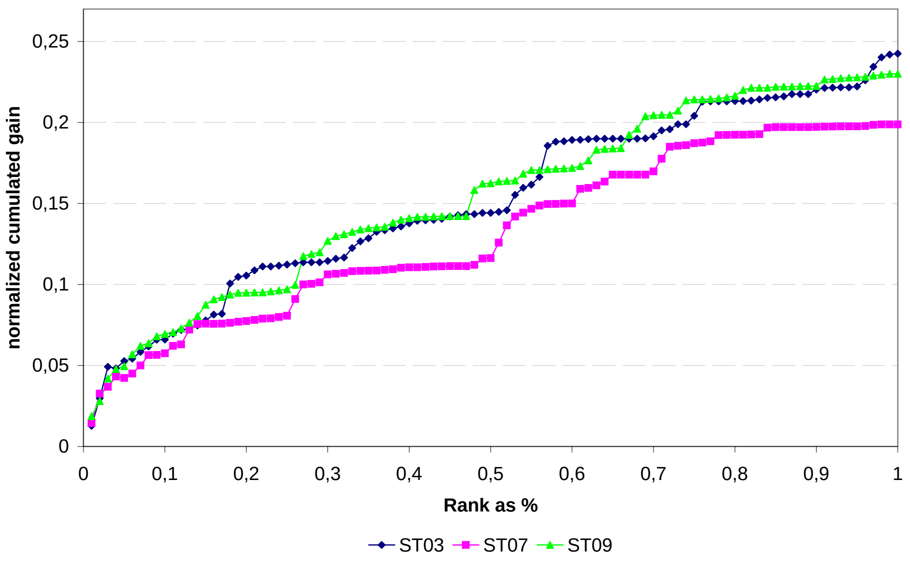
Figure 9.2: Tokenizer performance

`element` and `based`). This behavior increases the index size (compare ST02 and ST03), but allows to retrieve additional XML components due to more general index terms. However, precision may be lost because strings such as '`element based`' and '`element-based`' match completely. Finally, QTag is not optimized for preprocessed JavaTok inputs. Multi-tokens and token types are not supported by QTag. Both, the OpenNLPTokenizer and JavaTok operated at the same speed during indexing and retrieval (ST02 and ST03). Incorporating tagging in the processing (ST04 and ST05), especially during indexing, JavaTok preprocessed inputs speeded up QTag processing considerably.

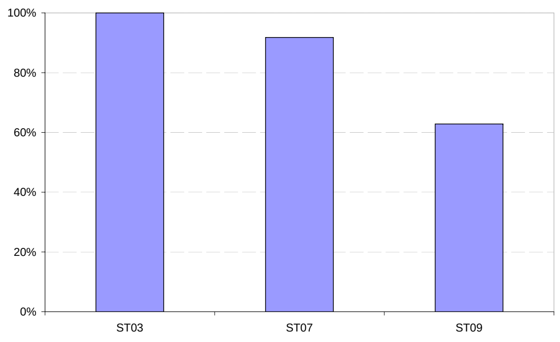
Both tokenization shortcomings, the treatment of punctuation marks and the splitting of hyphenated words, could be integrated in JavaTok by extending the charset definition and the rule base. Due to limited time, re-computation of several indices including this JavaTok update was not conducted. In the experiments, JavaTok (ST03 and ST05) achieved similar *nxCG* performance compared to the OpenNLPTokenizer (ST02 and ST04). Additionally, JavaTok speeded up indexing and retrieval. This processing improvement, the capability of extending and tailoring the functionality of JavaTok, the smaller indices, and the improvements of tagging favored JavaTok tokenization.

Tagger Performance

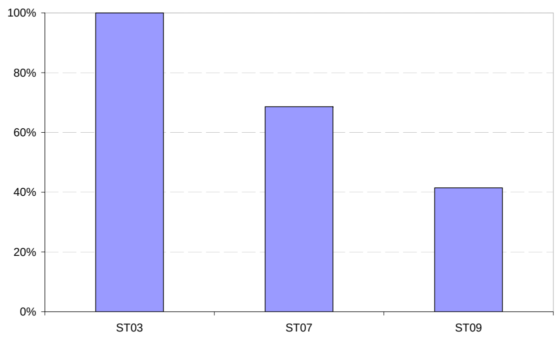
After tokenization, the influence of tagging on the retrieval performance was measured. Figure 9.3 shows the effect of applying QTag to identify syntactically relevant information that was used to represent the content.



(a) strict *nxCG* Performance



(b) Indexing times



(c) Retrieval times

Figure 9.3: Tagger performance

A comparison of ST03 (no tagging, using all terms), ST07 (tagging, using nouns, verbs, adjectives, and adverbs), and ST09 (no tagging, using token types) showed that QTag did not improve the *nxCG* retrieval performance. Both indices, ST03 and ST09, clearly outperformed ST07. The size of the term space did not influence the *nxCG* performance directly: ST03 (711.832), ST07 (516.742), and ST09 (302.204). Fastest indexing times (100% was equivalent to 20,1 hours) and retrieval times (100% was equivalent to 70,0 hours) were achieved for index ST09.

Since tagging (re-)assigns syntactic tags based on word chains, any kind of token might be tagged as a noun if it occurs in a certain context. Thus, tagging quality drops instantly if word categories are not identified correctly. A brief investigation of the ST07 index terms showed that over 37.000 terms did not even start with a letter or a number. Thus, a large subset of the terms indexed seemed to be inappropriate or irrelevant for searching. Instead, index terms extracted on the basis of JavaTok assigned token types (ST09) better fitted the notion of 'meaningful' words. As a result of this experiment, the small index size, the better index terms, and the faster processing clearly favored index ST09 without tagging.

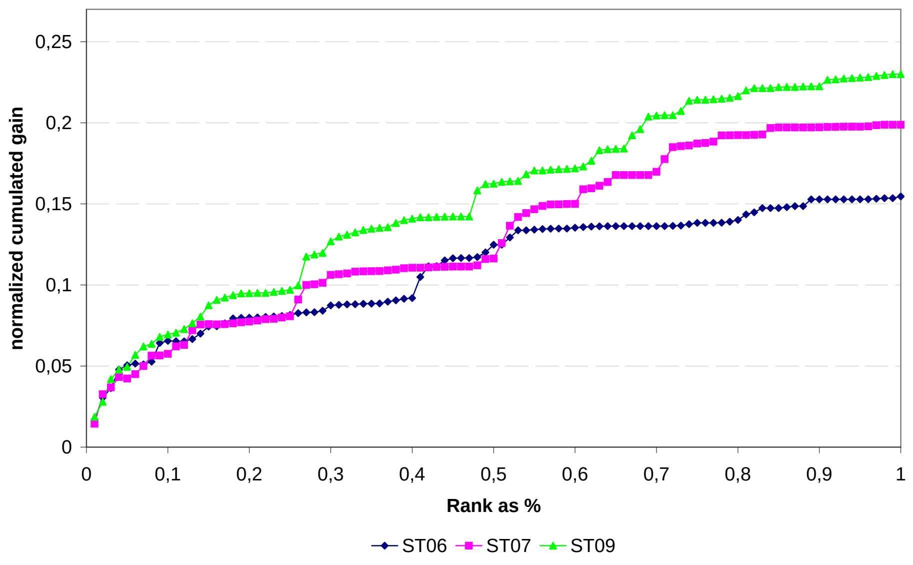
Extractor Performance

The effect of including different (syntactic) categories of words in the index is shown in Figure 9.4. ST06 selected nouns and verbs only, ST07 considered nouns, verbs, adjectives, and adverbs, and ST09 extracted all 'proper' words identified via token types.

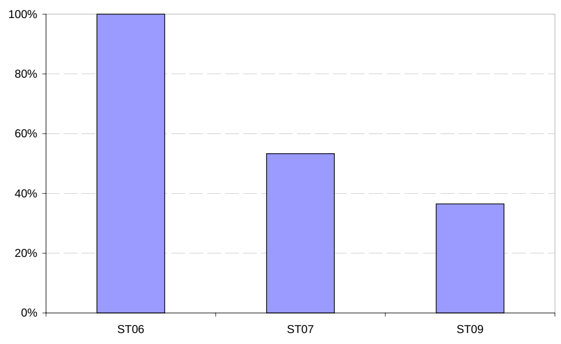
Taking into account only the first results returned ($0 - 0,12\%$), performance was nearly identical. As in previous results, performance was independent of the size of the index: ST06 (461.205 terms), ST07 (516.742 terms), and ST09 (302.204 terms). Again, ST09 achieved fastest indexing time (100% was equivalent to 34,6 hours) and retrieval time (100% was equivalent to 48,0 hours).

As expected, index ST06 resulted in the worst performance. This is because adjectives and adverbs containing important information were neglected. Clearly better results were achieved by ST07, which included all terms of ST06 plus adjectives and adverbs. Both indices were outperformed by ST09. Looking at the curves for ST06 and ST07, the number of index terms extracted linearly increased the retrieval performance. However, only correct terms relevant to user queries were helpful. Thus, the considerably smaller index ST09 achieved best results.

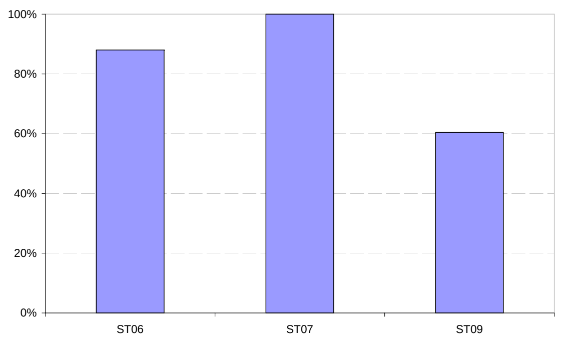
This experiment confirmed that the selection of index terms based on JavaTok token types is a promising procedure. *nxCG* performance and fast processing during indexing and retrieval support that argument.



(a) strict *nxCG* Performance



(b) Indexing times



(c) Retrieval times

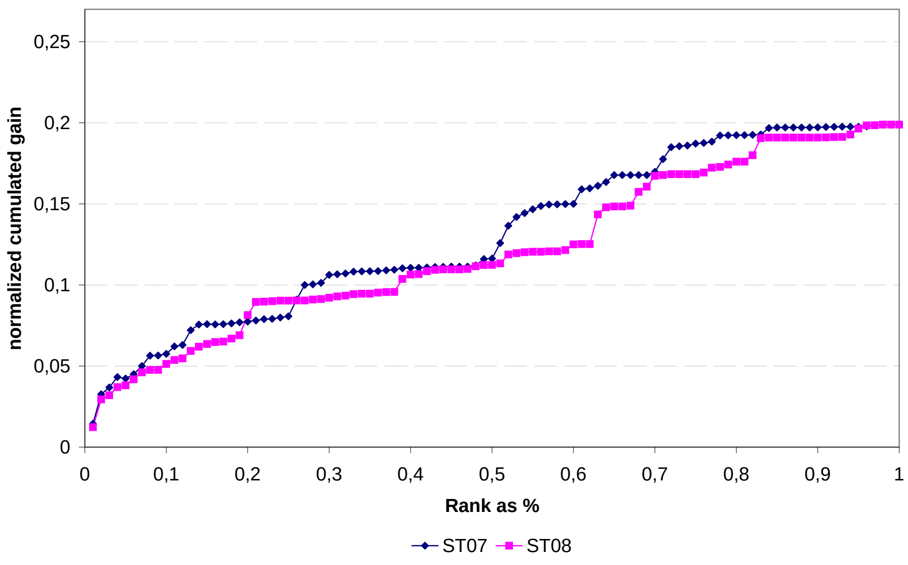
Figure 9.4: Extractor performance

Stemmer Performance

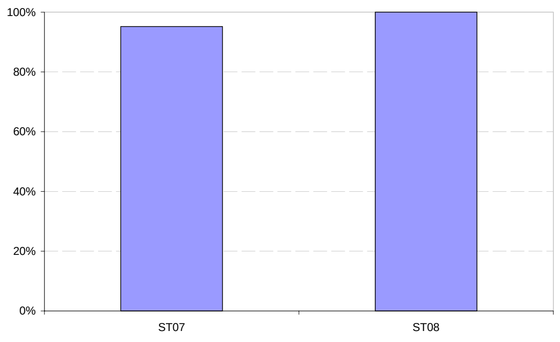
A performance comparison of index ST07 (with stemming) and index ST08 (without stemming) is given in Figure 9.5.

The performance of both indices were nearly the same. In the experiment, stemming seemed to achieve slightly better retrieval results. The term space of ST07 contained 516.742 terms, while the term space of ST08 included 555.777 terms. Since stemming is a lightweight process that runs very fast, there was nearly no difference in the indexing times (100% was equivalent to 19,4 hours) and retrieval times (100% was equivalent to 48,0 hours).

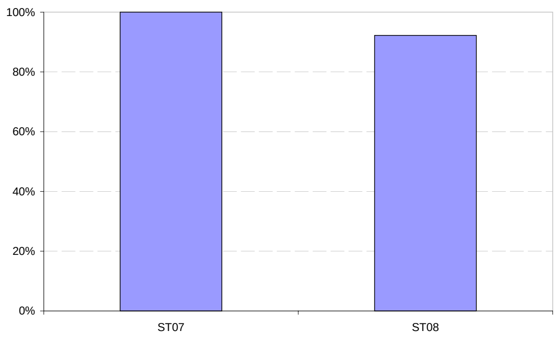
One may conclude that stemming is an appropriate procedure. It improved information retrieval by reducing the size of the vocabulary and by providing concept-like index terms.



(a) strict $nxCG$ Performance



(b) Indexing times



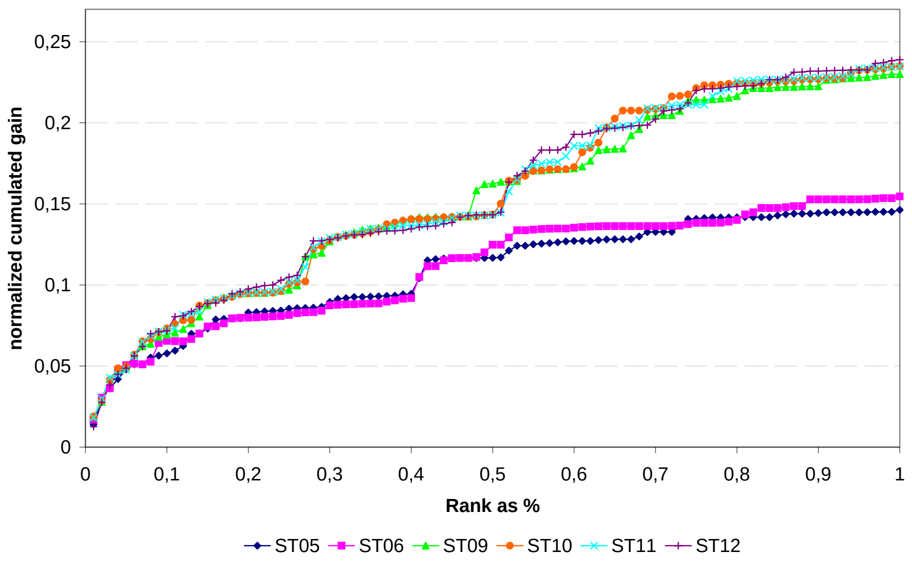
(c) Retrieval times

Figure 9.5: Stemmer performance

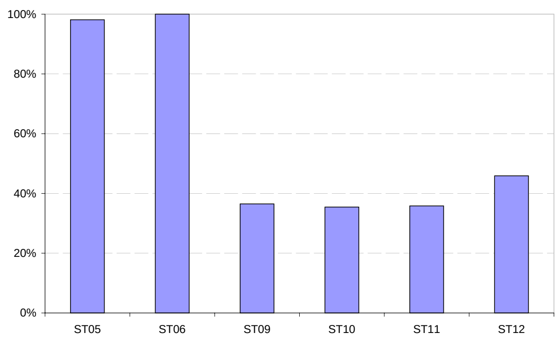
At the same time it operates very fast and nearly without additional computational costs. During retrieval, it speeded up processing and reduced query answer times.

Stopword Filtering Performance

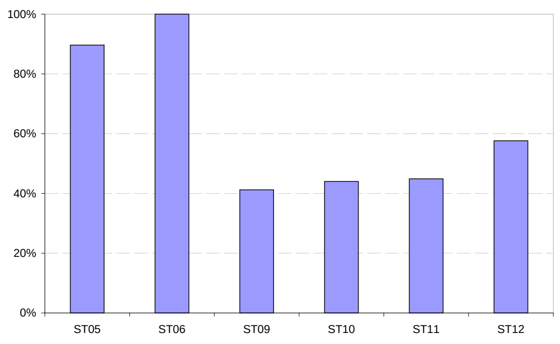
Finally, the effect of filtering stopwords was investigated. Figure 9.6 summarizes the results of the experimental runs using the indices ST05 (QTag, Fox's stopwords), ST06 (QTag, functional FS, content-related CR, and domain-specific DS stopwords), ST09 (no tagger, FS, CR, DS), ST10 (no tagger, FS, CR), ST11 (no tagger, FS), and ST12 (no tagger, no stoplist). For each comparison (ST05 and ST06, ST09 to ST12), only the single stopword filtering step is alternated.



(a) strict $nxCG$ Performance



(b) Indexing times



(c) Retrieval times

Figure 9.6: Stopword filtering performance

First, one notes that both stopword lists, the stopwords proposed by Fox (ST05) and the stopwords generated in this work (ST06) performed nearly the same. Also, the index sizes of ST05 (460.992 terms) and ST06 (461.205 terms) could be considered as equal. Both, ST05 and ST06, included tagging. As a consequence, indexing times (100% was equivalent to 34,6 hours) and retrieval times (100% was equivalent to 70,5 hours) were considerably higher than that of the other indices.

Iterative exclusion of different stopword layers led to the indices ST09, ST10, ST11, and ST12. As expected, the size of the index increased linearly with the exclusion of layers (302.204, 302.248, 302.559, and 302.755). These minimal changes had only a slight impact on index computation and retrieval. However, the large number of comparisons during retrieval favored an approach based on the ST09 index.

Summarizing the stopword filtering procedure, appropriate selection of index terms in advance showed that the effect of stopword filtering was reduced. Content-related and domain-specific stopwords did not influence the retrieval performance to a high degree.

According to these experiments, ST09 turned out to achieve best retrieval results while reducing the computational complexity and processing times. Hence, subsequent experiments were conducted using ST09 as the best performing single-term index.

9.2.2 Experiment II - Multi-Term Index Performance

Continuing the previous experiment, multi-term index performance metrics of MT01, MT02, MT03, MT04, and their combination MT were evaluated. MT refers to the performance achieved by combining all four multi-term indices using equal weights ($\frac{1}{4} = 0,25$). None of the initial query parameters was changed. For better readability, Table 8.3 explaining the multi-term indices is repeated in Table 9.4. Table 9.5 summarizes the results of this experiment.

Table 9.4: Multi-term indices maintained by the system

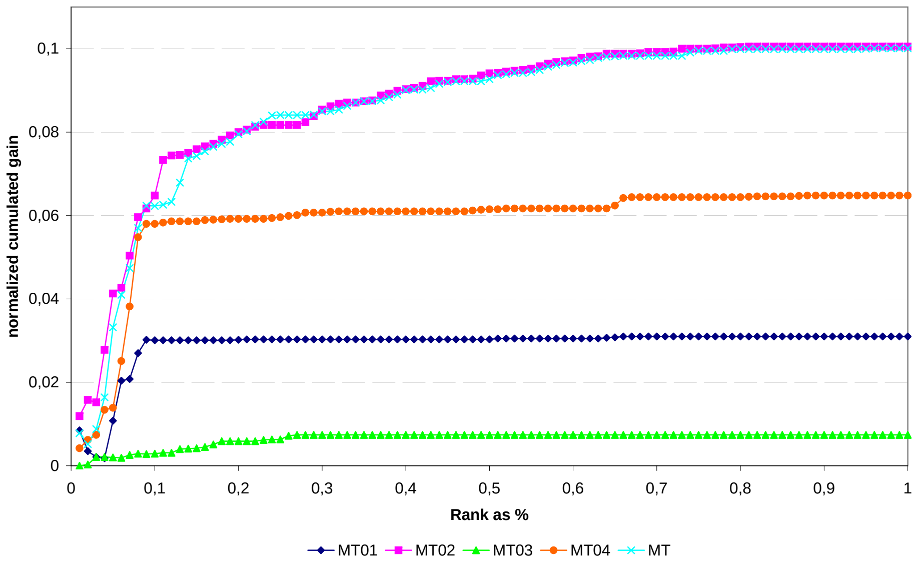
Index	Description
MT01	JavaTok-based composite nouns of arbitrary length ($\sim$ 340.000 terms)
MT02	JavaTok-based named entities of arbitrary length ($\sim$ 65.000 terms)
MT03	JavaTok-based formulaic speech of arbitrary length ($\sim$ 245.000 terms)
MT04	JavaTok-based full forms of acronyms of arbitrary length ($\sim$ 17.000 terms)

Table 9.5: $nxCG$ of the CO.Thorough task (multi-terms)

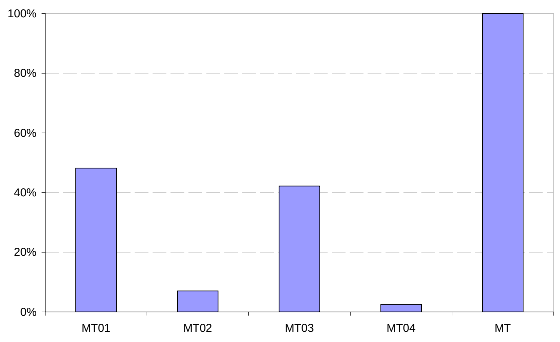
		gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Index	Size	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
MT01	318.116	0,0653	0,0492	0,0606	0,0077	0,0031	0,0015	0,0765	0,0615	0,0702
MT02	50.950	0,0816	0,0744	0,0803	0,0173	0,0158	0,0213	0,1031	0,0923	0,0961
MT03	213.183	0,0541	0,0435	0,0388	0,0000	0,0000	0,0024	0,0561	0,0462	0,0397
MT04	15.603	0,0661	0,0606	0,0536	0,0055	0,0070	0,0133	0,0716	0,0659	0,0563
MT	597.852	0,0603	0,0644	0,0751	0,0077	0,0046	0,0165	0,0822	0,0825	0,0887

Since MT is a combination of all other multi-term indices, the (theoretical) size of the term space was given by the sum of all other term space sizes. As expected, the performance of multi-term indices was much lower compared to the performance achieved by single-term indices (see Table 9.3 and the nCG scales of Figures 9.2 to 9.6). This was because multi-terms are less frequent and do not allow partial matching of query terms. Figure 9.7 presents the $nxCG$ performance achieved.

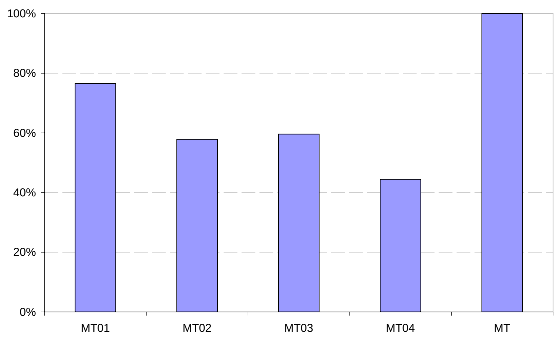
Indexing and retrieval times correlated with the sizes of the indices. For MT, the theoretical indexing time (100% was equivalent to 822,6 hours) and retrieval time (100% was equivalent to 22,9 hours) are given by the cumulated times of all multi-term indices.



(a) strict *nxCG* Performance



(b) Indexing times



(c) Retrieval times

Figure 9.7: Multi-term index performance

The evaluation showed that some categories of multi-terms were more useful to information retrieval than others. For instance, named entities (MT02) and full forms of acronyms (MT04) turned out to be better index terms than composite nouns (MT01) and formulaic speech (MT03). Interestingly, the term spaces of MT02 and MT04 were much smaller than the term spaces of MT01 and MT03. Most astonishing was the fact that MT02 performed equal or even slightly better than the combination of all multi-terms together. This means that in the INEX topics named entities are the most important type of multi-terms. Looking at the gen and genLifted graphs in the Appendix (see Section A.7.2), only the complete set of the multi-term indices summed up to MT.

The benefit of each index depends on the user queries. In the INEX topics, named entities and acronyms were frequently used, while composite nouns and formulaic speech were not.

One reason for that is that the topics were constructed by domain experts that searched for specific pieces of information. One explanation would be that named entities and acronyms are more likely to be used to express such special information needs. However, this may not be true for common users querying more general topics. Thus, this work relies on a combination of all multi-term indices.

9.2.3 Experiment III - Combined Single-Term and Multi-Term Index Performance

In a third experiment, a combination of the best performing single-term index ST09 and the multi-term index MT (consisting of MT01, MT02, MT03, and MT04) was studied. This approach was called TOP. Again, none of the initial query parameters was changed. As before, the overall relevance of a document component was averaged over the relevances computed for each index separately using equal weights ($\frac{1}{5} = 0,2$). Table 9.6 summarizes the results.

Table 9.6: $nxCG$ of the CO.Thorough task (single-terms and multi-terms)

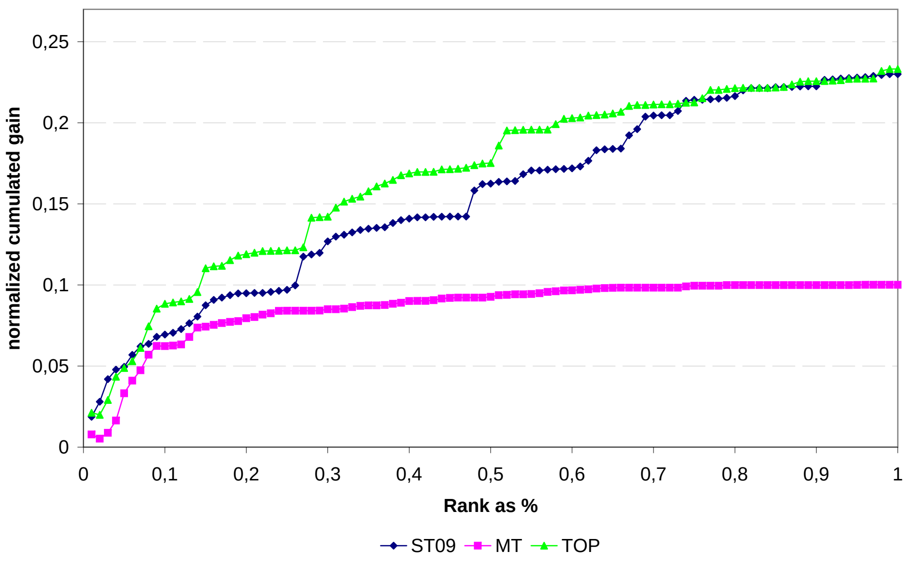
	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Index	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
ST09	0,1435	0,1334	0,1378	0,0235	0,0274	0,0447	0,1606	0,1509	0,1530
MT	0,0603	0,0644	0,0751	0,0077	0,0046	0,0165	0,0822	0,0825	0,0887
TOP	0,1524	0,1415	0,1503	0,0231	0,0240	0,0424	0,1804	0,1660	0,1718

The results in the table show that a combination of single-terms and multi-terms achieved better results than each of the indices separately. Only among the first 8% of the results retrieved, the strict $nxCG$ performance of the ST09 index is minimally higher than the performance of the TOP index. This was not upheld for the gen and genLifted $nxCG$ curves, where in both cases TOP achieved best results. In Figure 9.8, the performance curves of the three indices are depicted.

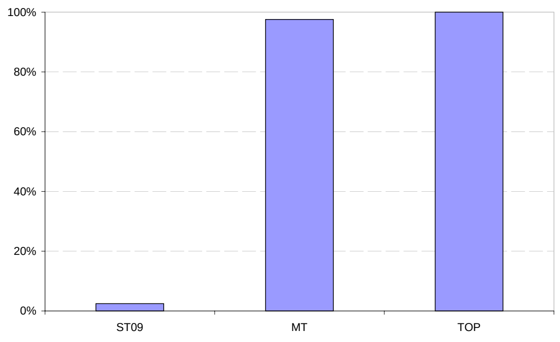
As in the previous experiment, indexing time (100% was equivalent to 843,3 hours) for the index TOP was cumulated from the indexing times of ST09 and MT. Since each index is compared sequentially, retrieval time (100% was equivalent to 54,6 hours) using the TOP index took, as expected, nearly as long as the retrieval of ST09 and MT together. The figure also shows that the contribution of the single-term index was much higher than that of the multi-term index. However, using additional multi-terms in combination with single-terms increased retrieval performance. Subsequent experiments were, therefore, conducted using the TOP index.

9.2.4 Experiment IV - Content and Structure

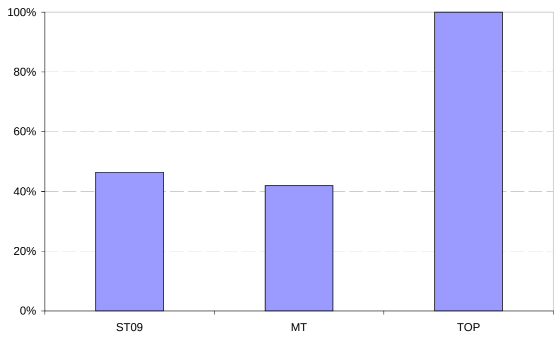
This experiment focused on the impact of structural constraints on the quality of the retrieval results. Query parameters remain unchanged at $maxRes = 1500$, $minSim = 0,0$, $ci = 0,2$, and



(a) strict $nxCG$ Performance



(b) Indexing times



(c) Retrieval times

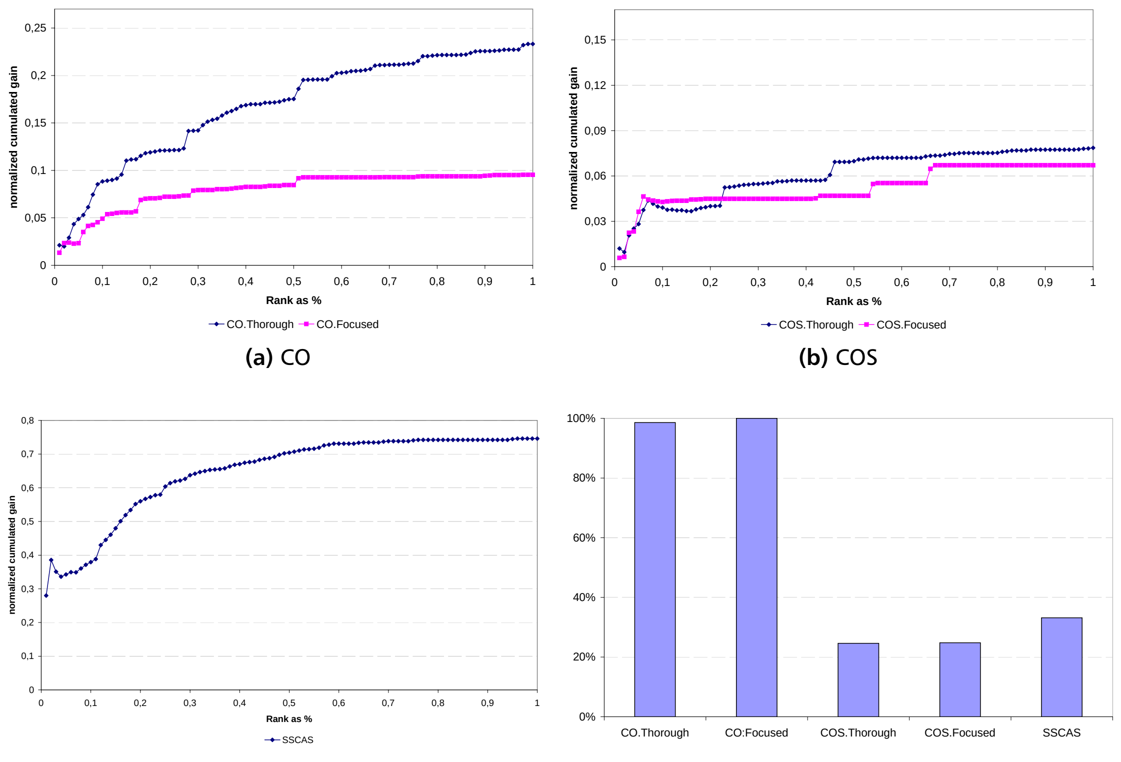
Figure 9.8: Combined single-term and multi-term index performance

$gf = 0,2$. The INEX tasks CO.Thorough, CO.Focused, COS.Thorough, COS.Focused, and SSCAS were evaluated and compared to each other. In all cases, the TOP index was used for retrieval. The results are given in Table 9.7.

Table 9.7: $nxCG$ of CO, COS, and SSCAS

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Index	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
CO.Thorough	0,1524	0,1415	0,1503	0,0231	0,0240	0,0424	0,1804	0,1660	0,1718
CO.Focused	0,1350	0,1285	0,1225	0,0192	0,0231	0,0234	0,1449	0,1380	0,1368
COS.Thorough	0,0977	0,0846	0,0852	0,0118	0,0118	0,0235	0,1046	0,0897	0,0878
COS.Focused	0,1037	0,0876	0,0957	0,0059	0,0071	0,0225	0,1084	0,0890	0,0957
SSCAS	0,1932	0,2857	0,3365	0,4000	0,3739	0,3389	0,2063	0,3011	0,3470

The performance indicators of the different INEX tasks varied remarkably. CO topic performance obtained 20% of the maximum cumulated gain possible. Worst results were achieved for the COS topics (< 15%). Best results were computed for complex SSCAS topics, reaching up to 40% of $nxCG$. For both tasks, CO and COS, the thorough strategy turned out to perform better than the focused strategy.



(a) CO **(b)** COS

(c) SSCAS **(d)** Retrieval times

Figure 9.9: Strict $nxCG$ Performance

The figures show that focused retrieval was not as successful as thorough retrieval. Especially for the CO tasks, the difference was considerable. Taking the complete result sets of 1500 results per topic into account, the CO task achieved 24% of the total cumulated gain possible. The performance of the COS tasks showed a similar behavior. Thorough retrieval outperformed focused retrieval. However, the difference was not as large as for the CO tasks. In general, the performance of the COS task was quite low. The most complex topics, the SSCAS retrieval task, achieved markedly better results. The complete SSCAS results reached about 75% of the total cumulated gain possible.

As indicated by Figure 9.9d, result computation for CO topics took longest. In the figure, the average retrieval times per topic for each of the task are given. 100% retrieval time was equivalent to 1,3 hours. These processing times are explained by the computational complexity of the on-the-fly weight computation. In a productive system these unfeasible retrieval times can be avoided by storing pre-computed term weights redundantly in the database. Since X-DOSE implements structural constraints as filter criteria, results for COS and SSCAS topics – although more complex – were computed three times faster.

9.2.5 Experiment V - Static Term Space versus Dynamic Term Spaces

The differences of applying a single static term space instead of multiple dynamic term spaces are described in Section 3.5. This experiment showed its impact on the retrieval performance. According to experiment IV, all INEX tasks were evaluated applying the same query parameters. The results are summarized in Table 9.8.

Table 9.8: Static term space versus dynamic term spaces

		gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Index	Type	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
CO.Thorough	static	0,1524	0,1415	0,1503	0,0231	0,0240	0,0424	0,1804	0,1660	0,1718
CO.Thorough	dynamic	0,1524	0,1415	0,1503	0,0231	0,0240	0,0424	0,1804	0,1660	0,1718
CO.Focused	static	0,1350	0,1285	0,1225	0,0192	0,0231	0,0234	0,1449	0,1380	0,1368
CO.Focused	dynamic	0,1350	0,1285	0,1225	0,0192	0,0231	0,0234	0,1449	0,1380	0,1368
COS.Thorough	static	0,0977	0,0846	0,0852	0,0118	0,0118	0,0235	0,1046	0,0897	0,0878
COS.Thorough	dynamic	0,0977	0,0846	0,0852	0,0118	0,0118	0,0235	0,1046	0,0897	0,0878
COS.Focused	static	0,1037	0,0876	0,0957	0,0059	0,0071	0,0225	0,1084	0,0890	0,0957
COS.Focused	dynamic	0,1037	0,0876	0,0957	0,0059	0,0071	0,0225	0,1084	0,0890	0,0957
SSCAS	static	0,1932	0,2857	0,3365	0,4000	0,3739	0,3389	0,2063	0,3011	0,3470
SSCAS	dynamic	0,1932	0,2857	0,3365	0,4000	0,3739	0,3389	0,2063	0,3011	0,3470

Unexpectedly, dynamic term spaces had no impact at all on the retrieval performance. Although term weights of the document component representations differed (because of different ief values), the ranking of components retrieved maintained the same.

The explanation for that behavior was the large number of leaf components and, consequently, the amount of text within the leave nodes (FRA elements). Due to this fact, dynamic term spaces of leave components performed nearly equal to the complete static term space. Since term spaces of components higher in the hierarchy contain the term spaces of descendant components, this effect was even enforced for dynamic term spaces at intermediate levels.

The retrieval times (100% retrieval time was equivalent to 78,5 hours), as given in Figure 9.10, indicated that dynamic term space computation consumed up to 45% more time. Learning from that experience, subsequent experiments were conducted using a single static term space.

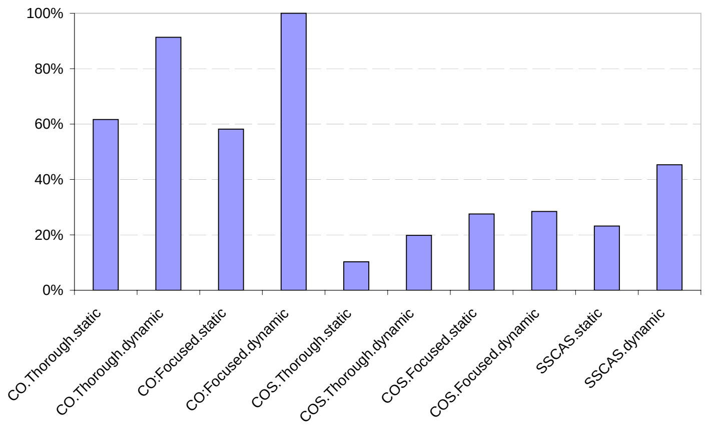


Figure 9.10: Retrieval times of static and dynamic term spaces

9.2.6 Experiment VI - The Effect of Content Importance ci

In order to find an optimal parameter setting for the importance of content relative to metadata, several ci settings were tested. Since the ci factor combines the impact of metadata and content relevance, this experiment was conducted on each of the CO, COS, and CAS tasks. Previous query parameters remained unchanged ($maxRes = 1500$, $minSim = 0,0$, and $gf = 0,2$). Experimental results are provided in Table 9.9.

The results in the table show that higher values of ci generated better results than lower ci values. This indicates that the content of a document component was more important than its' metadata information. The explanation for that is simple: The majority of the components retrieved, the FRA elements, did not contain metadata that was queried explicitly (except tables and figures).

Figure 9.11 shows the impact of the content importance factor ci on the retrieval performance. Colors are used to distinguish the different ci values.

For all tasks, low ci values led to worse results. Best performance was achieved for $ci = 0,8$ and $ci = 1,0$. Therefore, the ci parameter was fixed at $0,8$ for subsequent experiments. This value was assumed to give best results, because much weight is put on the similarity of contents while metadata similarity is not ignored completely.

9.2.7 Experiment VII - The Effect of the Generality Factor gf

The generality factor gf controls the influence of the ancestor components' relevances stated in the user query on the components' relevance itself. $gf = 0,0$ means that the relevance of a component is independent of the component ancestors' relevances. $gf = 1,0$ defines that a components' relevance is given by the component ancestors' relevance only. The query

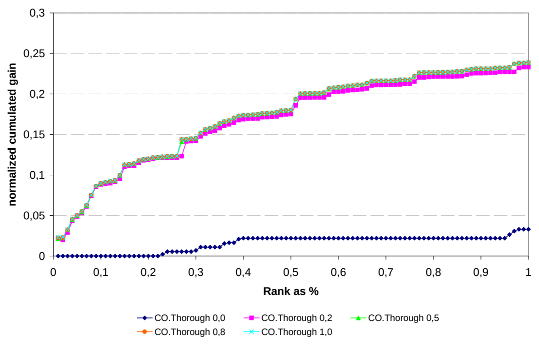
Table 9.9: Impact of content importance *ci*

		gen *nxCG*			strict *nxCG*			genLifted *nxCG*		
Index	*ci*	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
CO.Thorough	0,0	0,0509	0,0303	0,0209	0,0000	0,0000	0,0000	0,0520	0,0302	0,0199
CO.Thorough	0,2	0,1524	0,1415	0,1503	0,0231	0,0240	0,0424	0,1804	0,1660	0,1718
CO.Thorough	0,5	0,1451	0,1401	0,1498	0,0231	0,0270	0,0440	0,1767	0,1656	0,1729
CO.Thorough	0,8	0,1522	0,1400	0,1541	0,0231	0,0270	0,0440	0,1845	0,1669	0,1777
CO.Thorough	1,0	0,1519	0,1395	0,1523	0,0231	0,0286	0,0440	0,1841	0,1665	0,1757
CO.Focused	0,0	0,0573	0,0357	0,0245	0,0000	0,0000	0,0000	0,0553	0,0329	0,0215
CO.Focused	0,2	0,1350	0,1285	0,1225	0,0192	0,0231	0,0234	0,1449	0,1380	0,1368
CO.Focused	0,5	0,1312	0,1267	0,1225	0,0192	0,0277	0,0264	0,1461	0,1402	0,1393
CO.Focused	0,8	0,1368	0,1268	0,1250	0,0192	0,0277	0,0264	0,1506	0,1399	0,1418
CO.Focused	1,0	0,1339	0,1250	0,1233	0,0231	0,0277	0,0264	0,1503	0,1373	0,1400
COS.Thorough	0,0	0,0218	0,0122	0,0150	0,0000	0,0000	0,0012	0,0238	0,0129	0,0148
COS.Thorough	0,2	0,0977	0,0846	0,0852	0,0118	0,0118	0,0235	0,1046	0,0897	0,0878
COS.Thorough	0,5	0,0986	0,0851	0,0865	0,0118	0,0118	0,0247	0,1066	0,0905	0,0894
COS.Thorough	0,8	0,1007	0,0836	0,0862	0,0118	0,0118	0,0247	0,1085	0,0890	0,0893
COS.Thorough	1,0	0,1007	0,0824	0,0864	0,0118	0,0118	0,0247	0,1085	0,0875	0,0895
COS.Focused	0,0	0,0257	0,0156	0,0190	0,0000	0,0000	0,0012	0,0261	0,0150	0,0169
COS.Focused	0,2	0,1037	0,0876	0,0957	0,0059	0,0071	0,0225	0,1084	0,0890	0,0957
COS.Focused	0,5	0,1056	0,0919	0,0989	0,0059	0,0118	0,0272	0,1117	0,0933	0,0986
COS.Focused	0,8	0,1078	0,0896	0,0990	0,0059	0,0118	0,0272	0,1136	0,0915	0,0986
COS.Focused	1,0	0,1051	0,0895	0,0992	0,0059	0,0118	0,0272	0,1101	0,0914	0,0989
SSCAS	0,0	0,0000	0,0439	0,0495	0,0000	0,0300	0,0550	0,0190	0,0639	0,0606
SSCAS	0,2	0,1932	0,2857	0,3365	0,4000	0,3739	0,3389	0,2063	0,3011	0,3470
SSCAS	0,5	0,3361	0,2857	0,3365	0,4000	0,3739	0,3388	0,3492	0,3011	0,3470
SSCAS	0,8	0,3363	0,3013	0,3371	0,4000	0,3839	0,3389	0,3543	0,3178	0,3479
SSCAS	1,0	0,2525	0,3013	0,3371	0,4000	0,3839	0,3389	0,2705	0,3178	0,3479

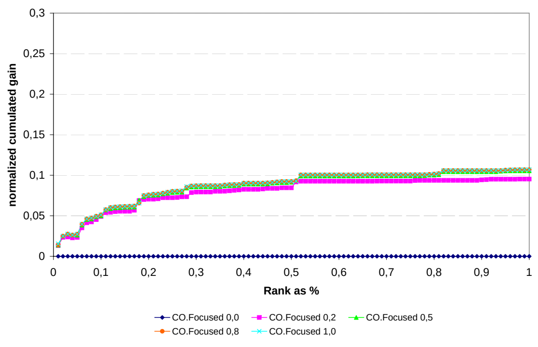
parameters for the experiment were fixed at $maxRes = 1500$, $minSim = 0,0$, and $ci = 0,8$. Since CO topics do not contain ancestor components (a single, unchained subquery), these tasks (thorough and focused) were skipped. For the COS and SSCAS tasks five different gf values in the range between 0 and 1 were evaluated. Table 9.10 presents the results of this experiment.

As Table 9.10 shows, the minimum $gf = 0,0$ and the maximum $gf = 1,0$ values were not optimal. Best results were achieved by $gf = 0,5$ for COS topics and $gf = 0,8$ for CAS topics.

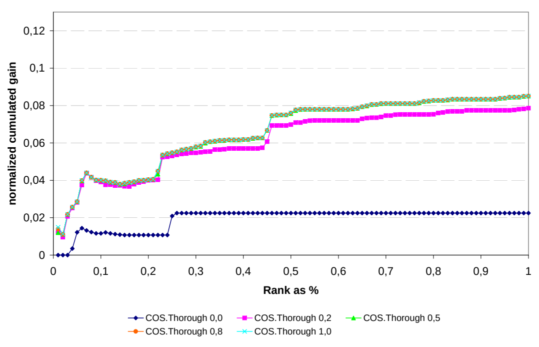
Figure 9.12 sketches the *nxCG* performance of the different gf values. In the figure, colors decode the different gf values used. The experiment showed that the relevance of ancestor components had no large impact on the components' relevance. This was because many of the topics addressed components directly without specifying ancestor components. Such topics were unaffected by the gf parameter. Out of the 28 COS and 47 CAS topics, only 12 COS and 37 CAS specified container elements explicitly. Of course, some topics implicitly included ancestor relationships such as `//SEC` addressed `/DOC/SEC` and `/DOC/SEC/SEC` components. On the other hand, container components were mostly given by their structure without any content restrictions (e.g., `/DOC/SEC`).



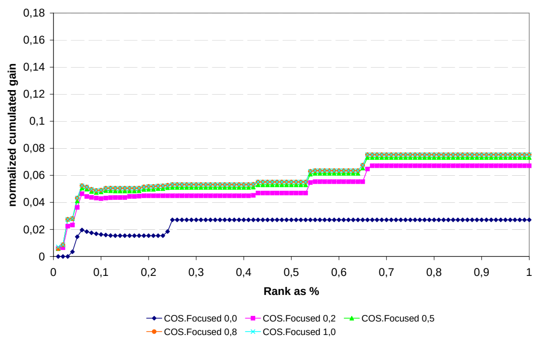
(a) CO.Thorough



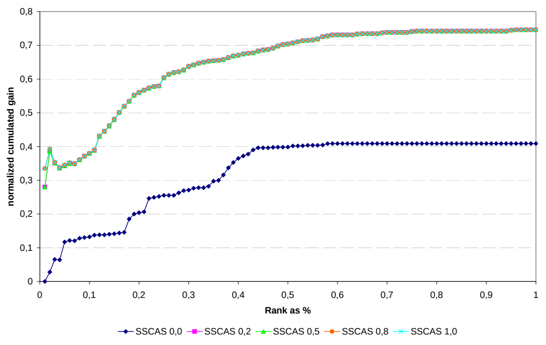
(b) CO.Focused



(c) COS.Thorough



(d) COS.Focused



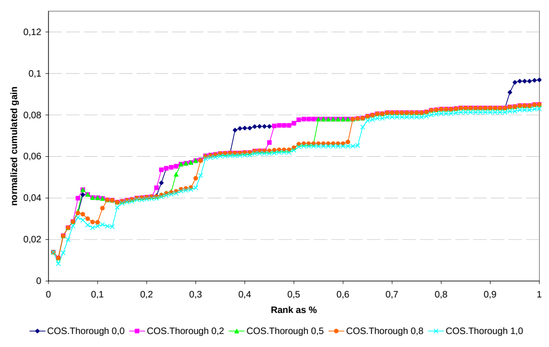
(e) SSCAS

Figure 9.11: Strict $nxCG$ Performance of ci

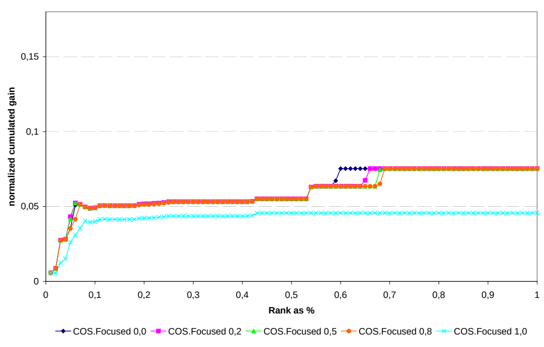
In the experiment, the maximum value of $gf = 1,0$ led to bad performance. The performance of other gf values were close to each other. Generally, lower values that put more emphasis on the components relevance, achieved better performance than higher ones. The small differences indicated that a factor smaller than $1,0$ did not influence the result too much. According to the figures, values of $gf = 0,5$ for COS topics and $gf = 0,8$ for CAS topics were chosen.

Table 9.10: Impact of the generality factor gf

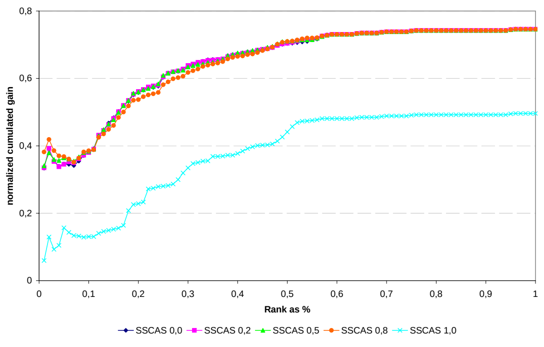
		gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Index	gf	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
COS.Thorough	0,0	0,1007	0,0836	0,0872	0,0118	0,0118	0,0247	0,1085	0,0890	0,0903
COS.Thorough	0,2	0,1007	0,0836	0,0862	0,0118	0,0118	0,0247	0,1085	0,0890	0,0893
COS.Thorough	0,5	0,1034	0,0837	0,0862	0,0118	0,0118	0,0247	0,1120	0,0893	0,0893
COS.Thorough	0,8	0,1007	0,0832	0,0856	0,0118	0,0118	0,0247	0,1069	0,0877	0,0884
COS.Thorough	1,0	0,0789	0,0551	0,0617	0,0118	0,0071	0,0176	0,0849	0,0592	0,0652
COS.Focused	0,0	0,1078	0,0896	0,0990	0,0059	0,0118	0,0272	0,1136	0,0915	0,0986
COS.Focused	0,2	0,1078	0,0896	0,0990	0,0059	0,0118	0,0272	0,1136	0,0915	0,0986
COS.Focused	0,5	0,1104	0,0898	0,0979	0,0059	0,0118	0,0272	0,1171	0,0917	0,0976
COS.Focused	0,8	0,1091	0,0895	0,0973	0,0059	0,0118	0,0272	0,1144	0,0910	0,0969
COS.Focused	1,0	0,0854	0,0594	0,0725	0,0059	0,0071	0,0129	0,0902	0,0612	0,0711
SSCAS	0,0	0,3504	0,2967	0,3419	0,4250	0,3739	0,3389	0,3634	0,3112	0,3524
SSCAS	0,2	0,3363	0,3013	0,3371	0,4000	0,3839	0,3389	0,3543	0,3178	0,3479
SSCAS	0,5	0,3504	0,2912	0,3408	0,4250	0,3739	0,3539	0,3682	0,3110	0,3540
SSCAS	0,8	0,3646	0,3115	0,3484	0,4500	0,4039	0,3689	0,3825	0,3350	0,3647
SSCAS	1,0	0,2365	0,2657	0,2756	0,0750	0,1300	0,0950	0,2737	0,2926	0,2964



(a) COS.Thorough



(b) COS.Focused



(c) SSCAS

Figure 9.12: Strict $nxCG$ Performance of gf

9.2.8 Experiment VIII - INEX 2005 Comparison

The results achieved in this work were compared to other INEX 2005 participants using the same documents, topics, and evaluation metrics. At INEX 2005, a former version of the X-DOSE system participated (see [120]). In order to point out the progress of the X-DOSE development over the last years, the former X-DOSE'05 performance is shown and compared to the current X-DOSE'09 version.

For the comparison, the best performing X-DOSE'09 parameter settings for the CO tasks ($ci = 0,8$, $gf = 0,8$ although irrelevant), the COS tasks ($ci = 0,8$, $gf = 0,5$), and the SSCAS task ($ci = 0,8$, $gf = 0,8$) were selected. Table 9.11 summarizes the results of X-DOSE'09 and X-DOSE'05.

Table 9.11: Progress of the X-DOSE development

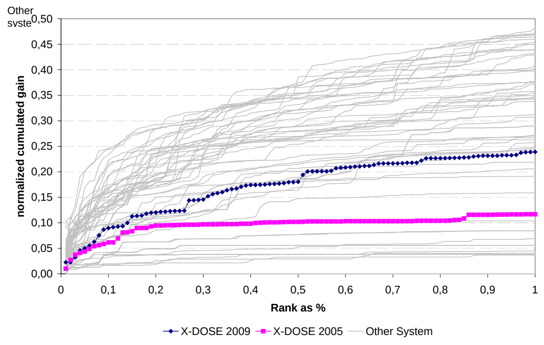
		gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Task	System	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
CO.Thorough	X-DOSE'05	0,1486	0,1229	0,1085	0,0115	0,0320	0,0407	0,1592	0,1320	0,1141
CO.Thorough	X-DOSE'09	0,1522	0,1400	0,1541	0,0231	0,0270	0,0440	0,1845	0,1669	0,1777
CO.Focused	X-DOSE'05	0,1247	0,0913	0,0819	0,0160	0,0112	0,0117	0,1283	0,0926	0,0785
CO.Focused	X-DOSE'09	0,1368	0,1268	0,1250	0,0192	0,0277	0,0264	0,1506	0,1399	0,1418
COS.Thorough	X-DOSE'05	0,1036	0,0889	0,0719	0,0000	0,0183	0,0312	0,1077	0,0907	0,0709
COS.Thorough	X-DOSE'09	0,1034	0,0837	0,0862	0,0118	0,0118	0,0247	0,1120	0,0893	0,0893
COS.Focused	X-DOSE'05	0,1216	0,0875	0,0827	0,0000	0,0212	0,0365	0,1206	0,0821	0,0732
COS.Focused	X-DOSE'09	0,1104	0,0898	0,0979	0,0059	0,0118	0,0272	0,1171	0,0917	0,0976
SSCAS	X-DOSE'05	0,1672	0,1494	0,1548	0,3500	0,3578	0,3828	0,1781	0,1641	0,1654
SSCAS	X-DOSE'09	0,3646	0,3115	0,3484	0,4500	0,4039	0,3689	0,3825	0,3350	0,3647

As the table shows, X-DOSE'09 outperforms X-DOSE'05 on the CO tasks. On the COS tasks, X-DOSE'05 seemed to perform even a bit better than the current version. For the SSCAS task, the improvements of X-DOSE'09 boosted the retrieval performance.

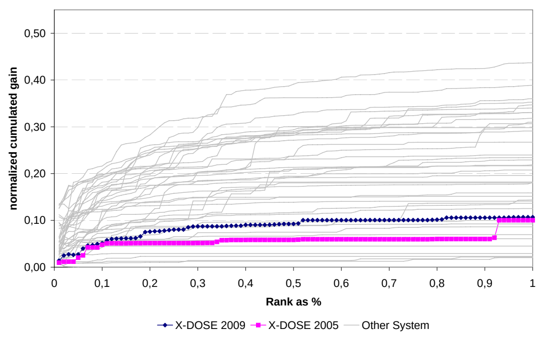
Figure 9.13 plots the corresponding $nxCG$ curves of X-DOSE'05, X-DOSE'09, and all participating INEX'05 systems. All graphs show that for all retrieval tasks X-DOSE'09 performed clearly better than X-DOSE'05. For the first 5%-10% of the retrieval results, $nxCG$ performance laid within a narrow margin. The more results included, the more the cumulated gain measure differed for both systems. Taking the complete number of retrieval results into account, X-DOSE'09 outperformed X-DOSE'05 clearly.

In Figure 9.13, gray curves denote performance profiles of other systems competing at INEX. Compared to other systems, X-DOSE was outperformed for the CO and COS tasks. Instead, for the SSCAS task X-DOSE outperformed most of the other systems. Tables 9.12–9.16 show the performance measures of the top 10 ranked INEX'05 systems. The reason for the low performance of the CO and COS retrieval tasks is threefold:

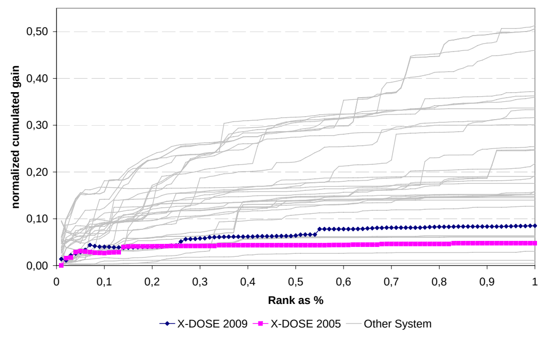
- The mapping procedure of the INEX documents onto the generic document format changed the document structure to some degree (e.g., corrections of the structure). Some



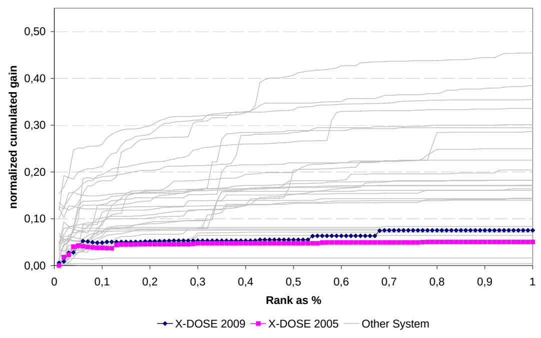
(a) CO.Thorough



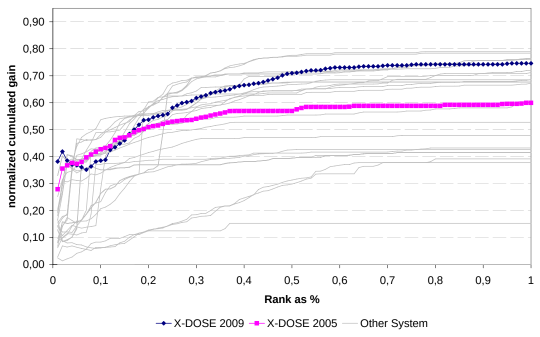
(b) CO.Focused



(c) COS.Thorough



(d) COS.Focused



(e) SSCAS

Figure 9.13: Strict *nxCG* Performance at INEX 2005

structural components of the initial INEX schema did not occur in the new format (e.g., tags for layouting, synthetic elements such as `/article/bdy`), were generalized to more abstract components (e.g., all different paragraph tags became a `FRA` elements of type `text`), or were reordered during correction (e.g., `p[2]/tbl[1]` became `tbl[1]`, further `p[i]` elements were changed to `p[i-1]`).

- INEX topics had to be adapted according to the generic document structure. This transformation included new `meta()` predicates and renaming of element paths. Thus, some of the initial topics could not be translated exactly.

- INEX assessments (optimal results) included elements that were derived automatically to the optimal recall base. For instance, if `/article/bdy/sec` was a relevant result, both containers `/article/bdy` and `/article` were also relevant to some degree and got added. Many of these elements derived were only of synthetic nature (e.g., `/article/bdy`. During evaluation, all elements retrieved had to include the correct path within the document. In X-DOSE, this path had, after retrieval, to be reconstructed from the metadata information (`sourcepath`) of the component. Obviously, this could only be done for components that were in fact mapped. Since a large number of INEX elements were not mapped one to one, such a reconstruction was often not possible. During evaluation, such results were – unfortunately – judged as missing, although these elements never existed in the mapped documents. Especially in the case of CO and COS topics, this large number of synthetic elements not retrieved by X-DOSE led to that drop of performance.

Table 9.12: Top-10 INEX 2005 systems (CO.Thorough)

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Rank	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
1	0,3037	0,2771	0,1004	0,1189	0,1931	0,2546	0,3401	0,3167	0,2717
2	0,2820	0,2654	0,0923	0,1174	0,1539	0,2529	0,3345	0,3032	0,2689
3	0,2797	0,2634	0,0846	0,1079	0,1298	0,2480	0,3274	0,2855	0,2670
4	0,2673	0,2573	0,0749	0,0974	0,1222	0,2448	0,2993	0,2814	0,2639
5	0,2665	0,2570	0,0747	0,0921	0,1157	0,2377	0,2939	0,2782	0,2628
6	0,2637	0,2552	0,0746	0,0910	0,1134	0,2350	0,2908	0,2777	0,2613
7	0,2593	0,2539	0,0739	0,0855	0,1093	0,2343	0,2905	0,277	0,2593
8	0,2574	0,2441	0,0615	0,0847	0,1092	0,2339	0,2816	0,2732	0,2468
9	0,2561	0,2399	0,0598	0,0846	0,1049	0,2330	0,2806	0,2693	0,2423
10	0,2552	0,2386	0,0538	0,0842	0,1035	0,2299	0,2756	0,2667	0,2411

For better comparison, the best results achieved by X-DOSE were ranked in Table 9.17. The number between parentheses in each cell indicates the rank of X-DOSE in relation to other participating systems. As in the figure and tables above, the results illustrated that X-DOSE was less competitive in the case of CO and COS tasks. In contrast to that, it was ranked among the top-10 systems in the case of SSCAS. However, taking the performance of the SSCAS runs as the most accurate ones into account, X-DOSE seems to be a competitive structured document retrieval system. Further evaluation tasks based on other corpora and metrics would surely provide better insights in the real performance of the system. Unfortunately, there were no other corpora available that included queries and assessments for evaluation

Table 9.13: Top-10 INEX 2005 systems (CO.Focused)

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Rank	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
1	0,2688	0,2325	0,2190	0,1401	0,1543	0,1902	0,3118	0,2505	0,2380
2	0,2561	0,2178	0,2122	0,1363	0,1513	0,1730	0,2942	0,2449	0,2371
3	0,2458	0,2139	0,2084	0,1324	0,1432	0,1627	0,2763	0,2415	0,2279
4	0,2538	0,2152	0,2085	0,1324	0,1442	0,1730	0,2859	0,2437	0,2305
5	0,2349	0,2134	0,2078	0,1266	0,1294	0,1549	0,2729	0,2347	0,2274
6	0,2316	0,2130	0,2031	0,1209	0,1095	0,1317	0,2729	0,2319	0,2264
7	0,2313	0,2110	0,1998	0,1074	0,1077	0,1261	0,2664	0,2293	0,2165
8	0,2290	0,2073	0,1985	0,0960	0,0997	0,1240	0,2612	0,2234	0,2079
9	0,2275	0,2034	0,1924	0,0959	0,0887	0,1209	0,2588	0,2230	0,2077
10	0,2244	0,2025	0,1914	0,0901	0,0886	0,1176	0,2428	0,2099	0,2012

Table 9.14: Top-10 INEX 2005 systems (COS.Thorough)

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Rank	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
1	0,3153	0,2858	0,2665	0,0824	0,1050	0,1366	0,3375	0,3045	0,2792
2	0,3111	0,2828	0,2607	0,0824	0,1050	0,1360	0,3373	0,3016	0,2775
3	0,2766	0,2754	0,2583	0,0588	0,1027	0,1354	0,3078	0,3003	0,2747
4	0,2747	0,2741	0,2542	0,0588	0,0956	0,1307	0,3044	0,2929	0,2655
5	0,2690	0,2649	0,2490	0,0529	0,0912	0,0901	0,3021	0,2916	0,2626
6	0,2666	0,2634	0,2462	0,0507	0,0771	0,0883	0,3017	0,2904	0,2601
7	0,2659	0,2621	0,2367	0,0507	0,0736	0,0755	0,2939	0,2836	0,2493
8	0,2650	0,2620	0,2361	0,0471	0,0736	0,0712	0,2893	0,2829	0,2491
9	0,2625	0,2586	0,2347	0,0471	0,0660	0,0659	0,2881	0,2810	0,2416
10	0,2607	0,2582	0,2343	0,0448	0,0576	0,0653	0,2795	0,2798	0,2364

Table 9.15: Top-10 INEX 2005 systems (COS.Focused)

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Rank	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
1	0,2908	0,2520	0,2439	0,1588	0,1996	0,2510	0,3194	0,2579	0,2598
2	0,2860	0,2372	0,2375	0,1261	0,1788	0,1809	0,3036	0,2541	0,2441
3	0,2767	0,2370	0,2315	0,1206	0,1308	0,1576	0,3014	0,2399	0,2306
4	0,2637	0,2327	0,2179	0,1125	0,1235	0,1574	0,2906	0,2397	0,2251
5	0,2534	0,2248	0,2147	0,1063	0,1213	0,1264	0,2878	0,2307	0,2182
6	0,2462	0,2137	0,2101	0,0971	0,0975	0,1178	0,2513	0,2200	0,2141
7	0,2457	0,2068	0,1936	0,0672	0,0908	0,0856	0,2499	0,2177	0,2101
8	0,2427	0,1942	0,1931	0,0613	0,0815	0,0830	0,2495	0,2089	0,2086
9	0,2368	0,1939	0,1889	0,0496	0,0684	0,0780	0,2441	0,2046	0,1963
10	0,2286	0,1891	0,1816	0,0466	0,0637	0,0697	0,2417	0,2018	0,1804

purpose. Other performance metrics than $nxCG$ and ep/gr would not have been comparable to systems of INEX 2005 participants.

9.2.9 Experiment IX - Clustering Performance

Due to limited time, the evaluation of clustering performance was restricted to the best performing single-term index ST09. In this evaluation, clustering was used to improve answer

Table 9.16: Top-10 INEX 2005 systems (SSCAS)

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Rank	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
1	0,4730	0,4816	0,5192	0,4500	0,4278	0,4356	0,4810	0,4936	0,5233
2	0,4699	0,4335	0,4211	0,4500	0,4278	0,4078	0,4780	0,4342	0,4299
3	0,4031	0,3978	0,4194	0,3250	0,3956	0,4067	0,4053	0,4007	0,4299
4	0,3643	0,3978	0,4194	0,3250	0,3956	0,3967	0,3719	0,3980	0,4288
5	0,3288	0,3971	0,4138	0,2250	0,3839	0,3894	0,3243	0,3958	0,4177
6	0,3246	0,3950	0,4094	0,2000	0,3839	0,3756	0,3215	0,3921	0,4157
7	0,3147	0,3927	0,4076	0,1750	0,3200	0,3639	0,3163	0,3907	0,4140
8	0,3071	0,3433	0,3959	0,1500	0,1817	0,3589	0,3157	0,3600	0,4111
9	0,2995	0,3373	0,3951	0,1500	0,1856	0,3489	0,3140	0,3512	0,3961
10	0,2862	0,3364	0,3845	0,1500	0,3100	0,3200	0,3140	0,3478	0,3891

Table 9.17: Best results of X-DOSE

	gen $nxCG$			strict $nxCG$			genLifted $nxCG$		
Task	[10]	[25]	[50]	[10]	[25]	[50]	[10]	[25]	[50]
CO.Thorough	(40)	(42)	(38)	(33)	(40)	(41)	(40)	(39)	(37)
CO.Focused	(33)	(30)	(27)	(34)	(34)	(36)	(31)	(30)	(25)
COS.Thorough	(31)	(32)	(31)	(20)	(21)	(21)	(30)	(32)	(31)
COS.Focused	(21)	(23)	(23)	(26)	(21)	(23)	(23)	(23)	(21)
SSCAS	(4)	(15)	(15)	(1)	(3)	(7)	(4)	(13)	(13)

times during retrieval applying a document preselection. Each document in the database was transformed into an XML tree representation that contained the ST09 single-term content vectors in the leaf nodes. Thus, document trees were always rooted in the `/DOC` elements.

The complete set of documents was clustered using hierarchical clustering. The parameter settings were chosen according to the experiments described in Section 7.6 ($\alpha_{struct} = 0,8$, β_{parent}=0,2). The two most similar documents were merged into a supertree and got stored in the database. During retrieval, the cluster hierarchy was kept in memory and served as a filter. For a set of query terms, the cluster hierarchy was traversed. The content of each cluster was used as a filter. If its similarity with the query terms was above zero, both child clusters were investigated recursively. The search within the cluster hierarchy ended at the leaf nodes (single documents) or at clusters that were completely dissimilar. As a result, a list of documents that are at least similar to some degree was returned. Based on that list, only components that were contained in one of the documents on that list were matched.

The clustering of the 16.819 documents took 50,6 hours. Since each document tree (resp. supertree) is compared to each other, 792.956.310.739 comparisons were needed. However, caching of pairwise similarities reduced this number to 282.845.123 'unique' comparisons that had to be calculated. The experiments given in Figure 9.14 show that retrieval time was reduced to a high degree by exploiting the hierarchical clusters computed.

On the average, retrieval time was reduced by more than one half for all retrieval tasks. Since clustering was applied for preselection only, neither the ranking, nor the similarities

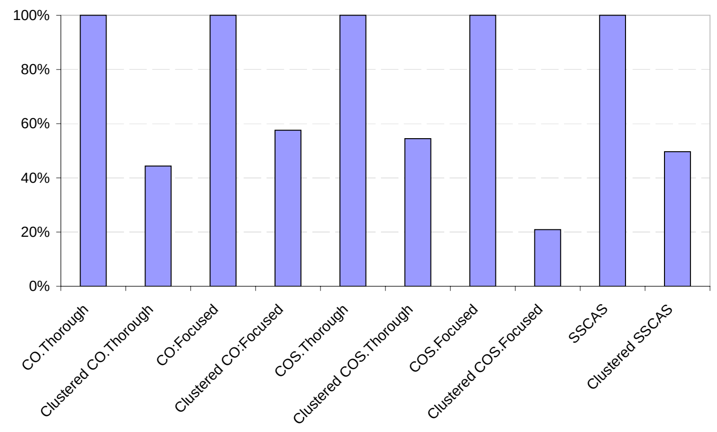


Figure 9.14: Retrieval times of clustered document components

computed did change. Summarizing these results, hierarchical clustering turned out to be an excellent improvement of X-DOSE retrieval performance.

9.2.10 Experiments not Conducted

Two experiments planned were not conducted: A first one on query expansion of acronyms and a second one on classification. While query expansion would have had no effect because queries always contained the short and the full form of an acronym, an evaluation of classification performance was lacking pre-classified data.

Query Expansion

One possibility to evaluate the effect of query expansion is to add terms included in the multi-term index to the query automatically. This kind of expansion only would effect queries that contain acronyms of both variants, short forms and full forms. However, all INEX queries that included acronyms did (1) already include the short and the long form of the acronym, or (2) were not included in the index. Thus, query expansion would not have had any effect on the results retrieved. Experiments on query expansion were left for further X-DOSE improvements and evaluation tasks.

Classification

An evaluation of XML document classification was not conducted in this work. The reason for this is that no pre-classified test data was included in the INEX 2005 document retrieval collection. Since classification of similar document components into user-defined classes without any comparison data is only a matter of subjective attitude, an objective evaluation could not be conducted.

9.3 Summary

This chapter described the evaluation of the X-DOSE system. For this purpose, the document collection and queries of the INEX 2005 evaluation workshop were used. The experiments included a comparison of twelve single-term indices, four multi-term indices, and combinations of them. Five different INEX retrieval tasks were run on X-DOSE applying numerous parameter settings.

The experiments clearly indicated that the size of the index is not correlated directly with the performance achieved. The quality of the index terms was essential. The four types of multi-term indices, which were used in addition to the best performing single-term index, turned out to improve performance considerably for all content-oriented retrieval tasks. Important to information retrieval, indexing times and retrieval times were measured and interpreted for each of the experiments. Best performance in both regards, retrieval quality and processing speed, was achieved by token-type based index term extraction and stemming. Tagging turned out to be not optimal because of wrongly identified word categories and its computational complexity. Dynamic term space generation had no impact on the retrieval results. A comparison of a former X-DOSE version and the current version revealed the progress of development and highlighted the improvements achieved.

Compared to other systems used at INEX 2005, X-DOSE was not competitive in the Content-Only (CO and COS) tasks. This was because the evaluation procedure used at INEX penalized retrieval results computed on mapped documents and topics, as it is proposed in this research. In contrast, Content-And-Structure (CAS) queries were processed very fast. Since CAS evaluation was done based on strict structural matching (SSCAS), the results much better reflected the performance of X-DOSE. For this task, X-DOSE even outperformed most other systems.

Chapter 10

The best way to get a good idea is to get a lot of ideas.

L. Pauling

Conclusion

This chapter summarizes the research done in this dissertation. In particular, it highlights the scientific contributions to the field of structured document retrieval and lessons learned. Moreover it sheds light on some perspectives and future research directions.

10.1 Concerns

This thesis introduces structured document retrieval as a further development of traditional information retrieval. Documents are no longer understood as flat content containers. Instead, they look like a hierarchical structure of elements (called document components). In other terms, the contents of a document are structured into various levels of granularity. Retrieval systems that take advantage of that additional information source have to be adjusted to fit the challenges coming along with this new paradigm.

XML is used as an appropriate means to represent structured documents. In order to generalize structural elements, documents are mapped onto a generic XML document format and are stored in a relational database. Clues about the structure, including available metadata, are used in combination with the content to satisfy a users' information need more accurately. This benefits retrieval in several ways: (1) Parts of the documents usually provide more focused answers to queries than complete documents; (2) The search engine focuses on relevant information avoiding further search and browsing activities by users; (3) Expert users are able to formulate highly specific queries.

Natural Language Processing (NLP) transforms the textual content of document components into term frequency vector representations. It includes an analysis of complex linguistic patterns that are extracted autonomously. This results in multiple representations of a text that are used conjointly and support each other. These representations of a text allow the adoption of modified versions of the traditional vector space model to match and rank components according to a user query.

During retrieval, the same NLP procedures are applied on the query terms to obtain a term frequency vector. Weighing of document and query terms is done on-the-fly. Content similarity, metadata similarity, and structural similarity are combined to compute the relevance of a document component to the user's query given a set of user-defined query parameters.

Classification of similar document components into user-defined categories is proposed as a means of enhancing and facilitating the organization and navigation in large document repositories. Because of the hierarchical structure, components are represented as ordered trees and compared using a distance measure. Providing the system with pre-assigned components and categories, classification is learned and unseen components are assigned automatically.

Computational performance is a critical issue of retrieval systems and strongly depends on the underlying data structures and matching mechanisms. In this work, hierarchical clustering is proposed to lower retrieval response time by creating clusters of closely related document components. During retrieval (result computation), these clusters are used to filter documents that do not correspond to any of the query constraints before weighing and matching are applied.

Owing to the challenges described, X-DOSE (XML-Document Oriented Search Engine) is developed. Based on a client-server architecture, the system processes various requests (indexing, retrieval, classification, and clustering) in parallel. Results are transferred back to the client, which displays them to the user. The client allows the user to browse and refine the query too.

The performance of X-DOSE is evaluated using an official collection of real-world XML documents provided by INEX. Well defined retrieval tasks and evaluation measures enable a comparison of its performance against other systems using the same data and retrieval tasks. The results demonstrate that linguistically-motivated text representations can improve retrieval quality efficiently. The evaluation also shows that, especially for complex retrieval tasks, X-DOSE is competitive to other systems and even outperforms them.

10.2 Contributions

The contributions achieved by this research are of two types, theoretical issues and tool aspects. Theoretical issues cover concepts inherent in structured document retrieval, natural language processing, text mining, and document mining (classification and clustering). Tool aspects include the implementation of a structured document retrieval system, X-DOSE, incorporating those concepts. It comprises natural language processing and text mining components. Further, linguistic resources relevant to information retrieval are generated. X-DOSE has been evaluated extensively with experiments focusing on retrieval performance.

In addition to that, experimental results of classification and clustering improving retrieval complement the evaluation.

10.2.1 Theoretical Contributions

- **Background:** An extensive overview of structured document retrieval is presented. It includes a description of XML, the standard format for representing the structure within text-based documents, historical development of structured document retrieval, retrieval models, representation, indexing, storage of documents, matching and ranking, query languages, result representation, and retrieval evaluation. Along with each of these topics, the challenges are pinpointed and existing solutions are provided.
- **Storage:** The documents are stored in a relational database. In order to ensure fast access to the structure, content, and metadata information, an existing approach is extended to fit these requirements.
- **Dynamic term spaces:** Since structured documents consist not only of a single (kind of) content, dynamic term spaces are defined. According to different structural levels and varying content types, dynamic term spaces enable structure-related indexing and retrieval of document components of arbitrary granularity.
- **XOR query language:** In cooperation with colleagues of other universities, the XML Oriented Retrieval (XOR) language is defined. XOR supports query constraints on the structure and on the content, and distinguishes components searched and components retrieved. Further, boolean operators that combine subqueries, qualifiers for the structure and the query terms, and term operators such as quotes and negation are provided.
- **Extended tokenization:** Representation issues of natural language texts have been addressed in Chapter 4. In this context, extended tokenization is introduced as initial processing step that includes token typing and multi-term concepts on different levels. Based on language-dependent dictionaries and tokenization rules, texts are preprocessed for subsequent analysis steps without any interpretation of their content.
- **Multi-layered stopword model:** In this work, stopword filtering is applied to reduce the amount of index terms. A multi-layered stopword model of functional, content-related, and domain-specific stopwords is motivated and implemented. Formulae for ranking and extracting stopwords automatically are defined and tested.
- **Text mining:** Chapter 5 is concerned with corpus-based text mining tasks based on the output of extended tokenization. A process that generates tokenization rules and dictionaries of single-tokens and multi-tokens is defined and illustrated.

- **XML classification:** Aspects relevant to classification of XML components into predefined categories have been discussed in Chapter 6. In this context, a similarity function that compares two document trees is defined. Two measures that combine structural similarity and content similarity are proposed. A first approach based on tree edit distance focuses on the structural match of two document trees. In contrast, a second approach based on content matrix distance puts more emphasis on the content match. The evaluation shows that best performance is achieved when both measures are combined.
- **XML clustering:** Clustering of structured documents is covered in Chapter 7. A novel representation of cluster representatives based on supertrees is introduced. A merge operation that inserts a document tree into a supertree is defined. Two similarity functions, one that compares a document tree to a supertree, and another one that compares two supertrees to each other, are provided. Both similarity functions support ordered and unordered tree comparisons.
- **Result ranking:** Ranking, as proposed in this work, combines the similarities computed separately for the content, structure, and metadata information. User-defined query parameters allow to control the impact of each of these factors on the overall relevance of a result component. Therefore, a set of ranking formulae is defined.

10.2.2 Tool Aspects

- **Generic XML format:** A generic XML document format is developed together with a mapping mechanism that transforms incoming documents into that format. The motivation of exploiting a single format for retrieval is to guarantee (1) valid document structures, (2) uniform access to standardized components, and (3) efficient processing.
- **XOR parser:** In order to process queries formulated in the XOR query language correctly, a parser based on JavaCC is implemented. It parses a valid XOR query and transforms it into an Abstract Syntax Tree (AST).
- **JavaTok:** Referring to the objectives of extended tokenization, JavaTok, a tokenizer that covers these issues, is implemented. JavaTok is available as stand-alone software, Java class framework, and online version. Texts are processed in a pipelined architecture of eights steps. Each step is optional and can be parameterized via a configuration file. Token type assignment and multi-token creation are done by dictionary lookup and rule matching. UTF-16 support and multiple output formats such as HTML and XML facilitate the integration of JavaTok into other systems.
- **Stopword list:** In several experiments, a set of 804 categorized stopwords (functional, content-related, and domain-specific) is extracted from the experimental corpus used.

In focused experiments, performance evaluations regarding the retrieval quality of each stopword layer are carried out.

- **Generation of natural language resources:** From the document collection, various natural language resources are generated to improve natural language text representation. 72 basic token types relevant to information retrieval are identified. Using that token types, 322 corpus-specific abbreviations, over 650.000 multi-terms (composite nouns, named entities, and formulaic speech), and about 17.000 acronyms with their full forms are extracted into separate dictionaries. In addition to that, a rule miner is implemented. For a given input pattern (a sequence of input tokens), it extracts statistically-motivated identification rules based on the left and right token context of the pattern. Using that rule miner, a set of complex multi-token rules was identified (e.g., phone numbers, citations, acronyms).
- **Experiments on XML classification:** Several experiments on five different data sets provided by INEX are conducted to evaluate the performance of XML classification. A comparison with other classification systems shows that the approach proposed is competitive compared to others.
- **Experiments on XML clustering:** Clustering performance is evaluated on six different data sets provided by INEX. In five experiments, the performance of *k*-Means and agglomerative hierarchical clustering is compared using several parameter settings.
- **X-DOSE prototype:** X-DOSE (XML-Document Oriented Search Engine) is implemented to measure the effectiveness of the retrieval approach proposed in this work. X-DOSE is based on a client-server architecture. It supports three types of queries, depending on the expertise of the user. Internally, queries are represented in an extended XOR syntax. Results are displayed by the client, providing browsing functionality of single documents. A class manager enables the classification of XML components retrieved into user-defined categories. XML clustering is integrated to speed up retrieval internally.
- **Result representation:** A novel graphical presentation of result components based on treeMaps and textBars is implemented. It enables navigation within the structure of a result document/component by supporting users with a relevance-based color schema that reflects content and metadata similarities.
- **Evaluation of X-DOSE:** An extensive evaluation of the X-DOSE retrieval performance focusing on computational complexity and retrieval quality is conducted. Relying on an official corpus of about 17.000 computer science documents, nine separate experiments focus on the quality of text representations, the interaction of structure, metadata, and content information, optimal parameter settings, and response time reduction. An

evaluation metric based on cumulated gain (*nxCG*) is used to express the performance of each retrieval task. Performance comparisons of similar systems using the same data show that X-DOSE is competitive and even outperforms other systems.

10.3 Lessons Learned

Research in the field of structured document retrieval is up-to-date and just at its beginning. Only with fundamental knowledge about the internal processing and functioning of these systems, adequate solutions to the difficulties described become feasible. Especially, computational performance is an issue that researchers and developers must be aware of. Although hardware becomes more powerful, faster, and cheaper, it is surely not the answer to manage the complexity of processing structured documents. Hence, it might be interesting for other researchers in this field to draw some conclusions from the theoretical work and the experiments.

- **Text representation:** Experimental results demonstrate that even in structured documents content information is more important than structural or metadata information. Thus, improvements of text representations improve the retrieval quality of structured documents. Especially extended tokenization supporting token typing and multi-terms, as proposed in this work, turns out to be an effective technique to pre-process textual content efficiently. Subsequent processing steps such as tagging strongly rely on the correctness of the tokenization output. Filtering of functional, content-related, and domain-specific stopwords further improves the quality of index terms while reducing the size of the vocabulary.

- **Metadata matching:** Experiments show that metadata matching and content matching has to be performed differently. For instance, skipping stopword filtering for matching metadata leads to retrieval of a huge number of unwanted components (e.g., documents, sections, tables, figures). This is because functional stopwords (e.g., 'a', 'the', 'and') are included frequently in metadata fields such as titles and captions. Retrieval based on metadata constraints has also to be reconsidered. A two-step procedure turned out to perform best: Metadata requests are transformed into SQL statements and processed by the database efficiently. For the results retrieved, similarity of metadata terms and query terms is computed using an overlap-based function on the two term sets.

- **Performance:** Indexing, retrieval, classification, and clustering of structured documents is more complex than it is in traditional information retrieval. Since performance is crucial for retrieval systems, processing must be efficient to ensure acceptable response times even for large amounts of data. This is especially true for structured document retrieval.

- **Dynamic aspects:** The possibility to keep term weights dynamic (on-the-fly computation during retrieval) becomes expensive in terms of processing performance. Although dynamic aspects are relevant and should be maintained, pre-computed and static representations stored might be necessary to speed up processing.
- **Data structures:** Related to the performance issue, appropriate data structures are the foundation of efficient processing. Especially tree data structures, as created by hierarchical clustering for document pre-selection, speeded up retrieval performance considerably. This concept might be transferred to improve the access to other data such as hierarchical text representations where common terms of two representations are raised one level higher in a representation tree.

10.4 Future Research Directions

There are a variety of exciting future directions in which the present work can be continued. Some of these directions were already presented at the end of Chapters 6 and 8. Other aspects worthy to be investigated further include:

- **Text mining:** Identification of textual patterns that go beyond multi-tokens such as named entities or citations is definitely a promising step to improve text analysis and, consequently, information retrieval. Besides the generation of static dictionaries, mechanisms that derive rules to dynamically identify these patterns in texts are an interesting issue. Applying these resources, for instance, in taggers, tagging statistics and tagging rules are reduced and simplified while the overall performance is increased regarding both, tagging quality and processing speed.
- **Shallow parsing:** The output of extended tokenization might be used to identify constituents of sentences such as noun phrases or verb phrases. For instance, the sequence $FS_i + FS_j + ALPHA\_COMMON\_LOWER$ (two functional stopwords followed by a lower case term) might be glued together as a single noun phrase (e.g., `in the house`, `on the tree`, `of the road`). Applying this mechanism recursively using different rule repositories, shallow parsing functionality based on token types is accomplished. Because of JavaTok's processing speed, this kind of parsing requires only little computational time. Hence, it is applicable to process even large amounts of data.
- **Additional experiments:** Most evaluations conducted in this research are based on the INEX document collection containing documents of a single domain. Further

experiments using other corpora and queries are necessary to gain higher external validity. This includes that dictionaries and rules generated might also be enriched by patterns of other domains.

- **Document structure:** The generic XML format proposed is an initial step towards a general model for structuring (text) documents. From the retrieval perspective, the three main elements for structuring a document, (`DOC`, `SEC`, and `FRA`), seem to be sufficient. However, metadata fields for each component were chosen according to the information available in the source documents. These data fields have to be reconsidered for the general case. The generic format further supports sub-structured content within fragments (`FRA` elements). However, there are no standards available that ensure a correct syntax. According to the type of a fragment (e.g, table, list, figure, formula), schemata that define these substructures are needed. To realize this, there exist several alternatives such as the usage of HTML-like table syntax and MathML markup for formulae.

- **Structural similarity:** In the current approach, constraints on the structure are implemented as strict filters. This speeds up retrieval, but prohibits a vague matching of structural constraints (meant as structural hints). Assuming complex nested document structures, vague matching might be beneficial to both, query formulation and retrieval quality.

In sum, the goal of this thesis has been successfully achieved. In line with this research, a fully functioning structured document retrieval system called X-DOSE has been implemented. Of course, further developments may concentrate on the extension, optimization, and tuning of X-DOSE. Beyond that, the broadness and application area of structured document retrieval leaves much open space for future researchers.

Appendix A

Creativity is the power to connect the seemingly unconnected.
William Plomer

Appendix

A.1 Extracted Stopword Lists

Tables A.1 to A.3 summarize all stopwords extracted according to their word categories and stopword layers, functional *F*, content-related *C*, and domain-specific *D*. The domain of these stopwords is computer science and information technology.

Table A.1: Final list of functional stopwords (English, INEX)

Category	Terms
F_DET (10)	`a, an, here, some, that, the, there, these, this, those`
F_AUX (33)	`am, are, be, became, become, becomes, becoming, been, being, can, cannot, could, did, do, does, doing, done, had, has, have, having, is, make, may, might, must, ought, shall, should, was, were, will, would`
F_PREP (47)	`about, above, across, after, against, along, among, amongst, around, aside, at, before, beforehand, behind, below, beside, between, beyond, by, down, for, from, in, into, near, of, off, on, onto, out, outside, per, since, through, thru, to, toward, towards, under, until, unto, up, upon, via, with, within, without`
F_PRON (70)	`another, anybody, anyhow, anyone, anything, anyway, anywhere, during, elsewhere, everybody, everyone, everything, everywhere, he, her, hers, herself, him, himself, his, how, i, it, its, itself, me, mine, my, myself, nobody, none, noone, nowhere, our, ours, ourselves, she, somebody, somehow, someone, something, sometime, somewhat, somewhere, such, that, their, theirs, them, themselves, they, us, we, what, when, whence, where, which, whither, who, whoever, whom, whomever, whose, why, you, your, yours, yourself, yourselves`
F_PART (17)	`almost, as, down, even, just, no, off, out, over, quite, rather, so, to, too, up, very, yes`
F_CONN (37)	`after, although, and, because, before, but, further, furthermore, hence, howbeit, if, insofar, instead, like, neither, nonetheless, nor, not, or, since, than, then, thence, therefore, though, thus, unless, until, whenever, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, while`
F_LOG_OP (3)	`and, not, or`
F_Q (43)	`all, billion, both, each, eight, eighty, eleven, every, few, fifteen, fifth, fifty, first, five, forty, four, many, million, much, multiple, nine, ninety, often, one, second, secondly, seven, seventy, six, sixty, some, ten, third, thirty, thousand, three, trillion, twelve, twenty, twice, two, various, zero`

Table A.2: Final list of content-related stopwords (English, INEX)

Category	Terms
CR_ADV (139)	accordingly, actually, actually, afterwards, again, along, already, also, always, any, apart, approximately, away, awfully, back, besides, certainly, clearly, clearly, closely, completely, consequently, currently, definitely, differently, directly, downwards, due, easily, either, else, entirely, especially, even, evenly, ever, exactly, explicitly, extremely, far, finally, first, formerly, frequently, fully, generally, hardly, hereafter, hereby, herein, hereupon, highly, hither, hopefully, however, immediately, inasmuch, increasingly, indeed, independently, inward, lately, latterly, less, likely, mainly, meanwhile, merely, more, moreover, most, mostly, much, namely, nearly, necessarily, never, nevertheless, non, normally, nothing, now, nowhere, obviously, often, once, only, otherwise, over, overall, particularly, perhaps, possibly, preferably, presumably, previously, primarily, probably, quickly, rather, really, reasonably, recently, relatively, respectively, seriously, significantly, similarly, simply, simultaneously, slightly, sometimes, soon, specifically, still, successfully, sure, thereafter, thereby, therein, thereof, thereto, thereupon, thorough, thoroughly, throughout, today, together, truly, typically, unfortunately, unfortunately, unlikely, usually, very, well, whatever, widely, yet
CR_ADJ (102)	additional, alone, appropriate, available, basic, best, better, big, brief, certain, clear, common, complete, current, different, difficult, due, early, easy, enough, entire, except, former, forth, full, general, good, great, greater, greatest, high, higher, highest, immediate, important, independent, initial, inner, kind, large, largely, larger, last, later, latest, latter, least, little, long, longer, longest, low, lower, main, main, major, mean, near, necessary, new, newer, newest, next, novel, old, older, oldest, original, other, own, particular, possible, potential, previous, real, recent, regardless, right, same, sensible, serious, several, significant, similar, simple, single, small, smaller, smallest, sorry, special, specific, standard, suitable, thick, thin, total, useful, whole, young, younger, youngest
CR_N (76)	ability, abstract, addition, address, advantage, amount, area, areas, basis, case, cases, change, changes, cost, curricula, degree, details, difference, effect, end, example, fact, field, focus, form, future, goal, group, hand, help, individual, interest, interests, issue, issues, key, level, means, need, needs, ones, order, others, paper, part, place, point, potential, problem, problems, project, range, regards, result, results, section, set, sets, size, solution, space, step, study, support, terms, time, times, type, types, use, view, vitae, way, ways, work, years
CR_V (189)	able, according, achieve, achieved, added, adding, address, allow, allowing, allows, applied, apply, applying, associated, assume, based, began, beginning, being, building, called, came, change, changing, come, compared, consider, considered, considering, consists, contain, containing, contains, corresponding, create, created, creating, depending, depends, derived, describe, described, detailed, determine, determined, differ, discussed, effect, end, existing, expected, find, finding, fixed, focus, following, follows, form, found, gave, get, gets, getting, give, given, gives, go, goes, going, gone, got, gotten, group, hand, help, improve, improved, include, included, includes, including, increase, increased, increasing, interested, interesting, introduced, involved, issue, issues, keep, keeps, kept, knew, know, known, knows, leading, less, let, lets, like, liked, liked, likes, limited, made, makes, making, maybe, need, needed, needing, needs, obtain, obtained, okay, order, organized, part, place, point, preferred, present, presented, presenting, presents, produce, proposed, provide, provided, provides, providing, published, put, puts, received, reduce, reduced, reducing, regarding, related, remaining, require, required, requires, result, resulting, see, seeing, seem, seemed, seeming, seems, selected, self, set, show, shown, shows, specify, specifying, starting, step, study, sub, support, take, taken, takes, taking, took, type, underlying, understanding, unlike, use, used, uses, using, want, wanted, wanting, wants, went, work, worked, working, works

Table A.3: Final list of domain-specific stopwords (English, INEX)

Category	Terms
D_ADV (3)	automatically, effectively, efficiently
D_ADJ (6)	complex, effective, efficient, local, national, technical
D_N (58)	access, algorithm, algorithms, analysis, application, applications, approach, approaches, architecture, complexity, components, computer, control, data, department, design, development, environment, features, function, functions, hardware, implementation, information, input, institute, knowledge, management, member, method, methods, model, models, network, number, operations, performance, process, professor, program, requirements, research, science, software, structure, system, systems, technique, techniques, technology, tools, trans, university, user, users, value, values, version
D_V (38)	access, computing, control, define, defined, design, designed, develop, developed, developing, distributed, engineering, extended, function, generate, generated, implemented, input, integrated, model, modeling, operating, perform, performed, performing, process, processing, programming, represent, represented, representing, represents, run, running, sets, specified, supported, testing

A.2 Extracted Patterns Suited for Composite Nouns

Table A.4: Top 24 extracted composite noun suited patterns of length two

Multi-term	tf	df
execution time	6.870	1.405
electrical engineering	6.333	3.316
response time	3.803	804
source code	3.768	1.403
fault tolerance	3.758	1.024
upper bound	3.369	1.228
worst case	3.035	1.305
experimental results	2.930	1.517
test cases	2.839	511
wide range	2.657	1.969
image processing	2.563	1.307
programming language	2.347	1.294
programming languages	2.218	1.235
neural networks	2.167	868
virtual channels	2.156	135
shared memory	2.127	686
load balancing	2.073	561
distributed computing	2.022	1.237
power consumption	1.968	611
pattern recognition	1.947	917
simulation results	1.915	893
fault coverage	1.864	315
state space	1.839	381
computation time	1.838	750

Table A.5: Top 24 extracted composite noun suited patterns of length three

Multi-term	tf	df
digital signal processing	385	301
natural language processing	303	186
partial differential equations	302	210
cache hit ratio	282	46
directed acyclic graph	251	198
average response time	251	121
test pattern generation	249	148
distributed shared memory	249	150
computational fluid dynamics	247	188
finite state machine	243	129
personal digital assistants	218	205
dynamic load balancing	209	90
middle stage switches	205	7
path delay faults	199	22
cache coherence protocol	193	68
artificial neural networks	191	144
finite state machines	186	111
average execution time	175	66
intellectual property rights	174	110
statistical pattern recognition	170	102
cache line size	170	69
state transition diagram	166	71
consistent global checkpoint	154	11
false alarm rate	150	46

Table A.6: Top 24 extracted composite noun suited patterns of length four

Multi-term	tf	df
automatic test pattern generation	79	68
extended channel dependency graph	62	8
dynamic buffer allocation scheme	55	1
linear feedback shift register	51	38
parallel task completion time	48	1
worst case response time	40	7
call channel occupancy time	35	1
worst case computation time	31	9
linear feedback shift registers	30	24
identically distributed random variables	30	23
solving partial differential equations	29	26
maximum average waiting time	28	1
state space explosion problem	27	16
optimal linear schedule vector	26	3
worst case execution time	25	20
optical flow constraint equation	25	9
hierarchical aggregate selection queries	25	1
distributed memory parallel computers	25	16
cumulative call variable usage	24	1
byte error correcting code	24	5
white box code inheritance	23	2
uniform leader election protocol	23	1
parallel discrete event simulation	23	16
timewheel atomic broadcast protocol	22	2

Table A.7: Top 24 extracted composite noun suited patterns of length five

Multi-term	*tf*	*df*
handoff call channel occupancy time	19	1
submit queries concerning historical events	10	10
average message latency versus traffic	10	4
scholarly archival journals inform readers	9	9
procedurally generated partial product reduction	9	1
minimal cost distribution tree problem	9	1
generated partial product reduction tree	9	1
vertex versus maximal clique incidence	8	1
row shift invariant wavelet packet	8	1
robust path delay fault coverage	8	2
shift invariant wavelet packet transform	7	1
scheduling precedence constrained parallel tasks	7	1
recoverable distributed shared virtual memory	7	1
maximum average waiting time requirement	7	1
fault tolerant wormhole routing strategy	7	7
extended multicast channel dependency graph	7	1
adaptive row shift invariant wavelet	7	1
time varying flow field visualization	6	1
symmetric symbol error correcting codes	6	1
conditional steady state probability vector	6	3
versus maximal clique incidence matrices	5	1
spanning tree carry lookahead adder	5	2
solicits papers giving preliminary results	5	5
random field modeled textured images	5	1

Table A.8: Top 24 extracted composite noun suited patterns of length six

Multi-term	*tf*	*df*
procedurally generated partial product reduction tree	9	1
row shift invariant wavelet packet transform	7	1
adaptive row shift invariant wavelet packet	7	1
vertex versus maximal clique incidence matrix	4	1
vertex versus maximal clique incidence matrices	4	1
produces dependency preserving nested database schemes	4	1
nonbalanced identically distributed binary random variables	4	1
beam addressed swept volume display unit	4	1
systolic redundant residue arithmetic error correction	3	2
redundant residue arithmetic error correction circuit	3	2
plot normalized deadlocks versus load rate	3	1
partial differential equations describing physical phenomena	3	3
interactive event service giving conference dates	3	3
handoff call channel occupancy time distribution	3	1
disk array controller signals service completion	3	1
database schema satisfies generalized entity integrity	3	1
coarse time scale traffic smoothing mode	3	1
cluster generative statistical dynamic time warping	3	1
wird etwas knapp bei mir sagen	2	1
unstructured sparse symmetric positive definite matrices	2	1
une courbe qui remplit toute une	2	2
ultimately simplify text composition tasks faced	2	2
trusted graphics server applet stored locally	2	2
time instant object component retrieval request	2	1

Table A.9: Top 24 extracted composite noun suited patterns of length seven

Multi-term	tf	df
adaptive row shift invariant wavelet packet transform	7	1
systolic redundant residue arithmetic error correction circuit	3	2
wird etwas knapp bei mir sagen wir	2	1
une courbe qui remplit toute une aire	2	2
temporal strata translate temporal query language statements	2	1
recommended practices define ethical standards define educational	2	2
practices define ethical standards define educational curricula	2	2
law enforcement agencies continually analyze vast amounts	2	2
knapp bei mir sagen wir lieber vierzehn	2	1
etwas knapp bei mir sagen wir lieber	2	1
courbe qui remplit toute une aire plaine	2	2
classification assumes locally constant class conditional probabilities	2	2
block sharing implies strong interprocess spatial locality	2	2
beam addressed swept volume display unit employing	2	1
authors propose task assignment effort adjustment factors	2	2
atlas anatomy removes individual anatomical shape variations	2	2
allocation strategies dramatically outperform contiguous allocation strategies	2	1
acyclic channel dependency graph guarantees deadlock freedom	2	1
zeroset copyset clrbit setbit tstbit xrealloc xmalloc	1	1
xrealloc freelist lex error enlist dfainit insert	1	1
xmalloc dfaerror addtok xrealloc freelist lex error	1	1
write aent ante acrt cart aent neat	1	1
woodword ted kaczynski competency birmingham islam blaze	1	1
wiring channels cause extra wiring path delays	1	1

A.3 Extracted Patterns Suited for Named Entities

Table A.10: Top 24 extracted named entity suited patterns of length two

Multi-term	tf	df
Pattern Recognition	5.914	1.575
Machine Intelligence	5.670	1.568
Artificial Intelligence	4.802	1.819
Distributed Computing	3.885	1.749
Parallel Processing	3.057	1.227
Image Processing	2.797	1.200
World Wide	2.734	1.537
Wide Web	2.658	1.508
Neural Networks	2.230	715
Signal Processing	2.039	1.006
Carnegie Mellon	2.024	1.147
International Conference	1.889	1.144
Electrical Engineering	1.581	1.076
Programming Languages	1.509	810
United States	1.473	826
Hong Kong	1.403	511
Reader Service	1.381	303
San Diego	1.336	741
Los Angeles	1.332	888
Parallel Computing	1.283	723
Machine Learning	1.235	475
Semantic Web	1.231	179
Monte Carlo	1.216	397
Air Force	1.139	628

Table A.11: Top 24 extracted named entity suited patterns of length three

Multi-term	tf	df
World Wide Web	2.650	1.505
Pattern Recognition Letters	444	287
Unified Modeling Language	394	274
Internet Engineering Task	328	284
Ad Hoc Networks	311	74
Object Request Broker	300	207
North Carolina State	298	209
Jet Propulsion Laboratory	282	186
Wide Web Consortium	274	232
Engineering Task Force	274	238
International Test Conference	269	183
Artificial Neural Networks	269	180
Extensible Markup Language	264	217
Upper Saddle River	244	129
Distributed Shared Memory	237	102
Stochastic Petri Nets	234	43
Java Virtual Machine	234	138
Guest Editors Introduction	223	223
Reader Interest Survey	211	205
Digital Signal Processing	211	144
Interest Survey Indicate	204	204
Wall Street Journal	203	149
Tau Beta Pi	195	178
Naval Postgraduate School	194	134

Table A.12: Top 24 extracted named entity suited patterns of length four

Multi-term	tf	df
World Wide Web Consortium	272	230
Internet Engineering Task Force	272	237
Reader Interest Survey Indicate	204	204
Goddard Space Flight Center	166	118
Virtual Reality Modeling Language	150	136
Mobile Ad Hoc Networks	114	47
San Diego Supercomputer Center	90	63
Ad Hoc Wireless Networks	83	45
Web Services Description Language	77	61
Field Programmable Gate Arrays	73	50
Wireless Ad Hoc Networks	69	34
Markov Chain Monte Carlo	68	43
Generalized Stochastic Petri Nets	66	28
Accelerated Strategic Computing Initiative	62	55
Synchronized Multimedia Integration Language	61	45
North Atlantic Test Workshop	60	54
San Jose Mercury News	56	47
Ordered Binary Decision Diagrams	46	28
International Parallel Processing Symposium	46	42
Mobile Ad Hoc Networking	45	30
Enterprise Distributed Object Computing	43	42
Digital Millennium Copyright Act	40	36
Imagery Pattern Recognition Workshop	35	31
World Intellectual Property Organization	34	33

Table A.13: Top 24 extracted named entity suited patterns of length five

Multi-term	tf	df
Sara Reese Hedberg Sara Reese	23	23
Reese Hedberg Sara Reese Hedberg	23	23
Linda Dailey Paulson Linda Dailey	23	23
Dailey Paulson Linda Dailey Paulson	23	23
Markov Regenerative Stochastic Petri Nets	22	8
Iowa State College Statistical Laboratory	22	1
Upsilon Pi Epsilon Student Award	19	11
Unified Modeling Language Reference Manual	19	19
Neural Networks Outstanding Paper Award	19	19
Fault Tolerant Wormhole Routing Strategy	19	19
Lance Stafford Larson Student Scholarship	18	13
International World Wide Web Conference	15	12
Virtual Reality Annual International Symposium	14	13
Inverse Visual Problems Involving Discontinuities	13	13
Electronic Delay Storage Automatic Calculator	13	8
Air Force Flight Dynamics Laboratory	12	12
Mary Jean Harrold Mary Jean	11	11
Jean Harrold Mary Jean Harrold	11	11
Enterprise Distributed Object Computing Workshop	11	11
British Association Mathematical Tables Committee	11	3
Air Force Scientific Advisory Board	11	10
Ad Hoc Mobile Wireless Networks	11	11
Wright Patterson Air Force Base	10	8
Robot World Cup Soccer Games	10	9

Table A.14: Top 24 extracted named entity suited patterns of length six

Multi-term	tf	df
Sara Reese Hedberg Sara Reese Hedberg	23	23
Linda Dailey Paulson Linda Dailey Paulson	23	23
Mary Jean Harrold Mary Jean Harrold	11	11
Shari Lawrence Pfleeger Shari Lawrence Pfleeger	9	9
Khaled El Emam Khaled El Emam	9	9
Alberto Del Bimbo Alberto Del Bimbo	8	8
Mo Kim Cheng Albert Mo Kim	7	7
Kim Cheng Albert Mo Kim Cheng	7	7
Hee Yong Youn Hee Yong Youn	7	7
Andy Hunt Dave Thomas Andy Hunt	7	7
Albert Mo Kim Cheng Albert Mo	7	7
Timothy Mark Pinkston Timothy Mark Pinkston	6	6
Sung Yong Shin Sung Yong Shin	6	6
Norris Parker Smith Norris Parker Smith	6	6
Lizy Kurian John Lizy Kurian John	6	6
Laxmi Narayan Bhuyan Laxmi Narayan Bhuyan	6	6
Giovanni De Micheli Giovanni De Micheli	6	6
Dik Lun Lee Dik Lun Lee	6	6
David Alan Grier David Alan Grier	6	6
Beng Chin Ooi Beng Chin Ooi	6	6
Song Chun Zhu Song Chun Zhu	5	5
Sang Lyul Min Sang Lyul Min	5	5
Reversible Jump Markov Chain Monte Carlo	5	4
Pam Frost Gorder Pam Frost Gorder	5	5

Table A.15: Top 24 extracted named entity suited patterns of length seven

Multi-term	tf	df
Mo Kim Cheng Albert Mo Kim Cheng	7	7
Albert Mo Kim Cheng Albert Mo Kim	7	7
Optimal Infinite Impulse Response Edge Detection Filters	5	5
Stochastic Petri Nets Representing Generalized Service Networks	4	4
Reversible Jump Markov Chain Monte Carlo Computation	4	4
Oak Ridge Association Junior Faculty Enhancement Award	3	3
International Test Conference Tutorials Washington Sheraton Hotel	3	3
Whitaker Jane Wilhelms Yves Willems Peter Williams	2	2
Victor De La Luz Victor De La	2	2
Time Warp Synchronized Parallel Discrete Event Simulation	2	2
Stereoscopic Image Pairs Assuming Piecewise Continuous Surfaces	2	2
Sillion Bruno Silva Claudio Silva Deborah Silver	2	2
San Diego Supercomputer Center Networked Volume Renderer	2	2
San Diego Supercomputer Center Creative Computing Award	2	2
Robertson Phil Robertson Alyn Rockwood Jon Rokne	2	2
Petri Nets Representing Generalized Service Networks Abstract	2	2
Nets Representing Generalized Service Networks Abstract Abstract	2	2
Natural Microbial Populations Reveals Tertiary Structural Elements	2	2
Marcello Pelillo Josiane Zerubia Guest Editors Curricula	2	2
Larry Aupperle Rick Avila Ron Azuma Norman	2	2
James Arvo Larry Aupperle Rick Avila Ron	2	2
Hee Beng Kuan Tan Hee Beng Kuan	2	2
Hans Hagen Bernd Hamann Pat Hanrahan Chuck	2	2
Hancock Marcello Pelillo Josiane Zerubia Guest Editors	2	2

A.4 Extracted Patterns Suited for Formulaic Speech

Table A.16: Top 24 extracted formulaic speech suited patterns of length two

Multi-term	tf	df
computer science	29.743	8.233
research interests	14.241	6.737
et al	12.946	3.645
software engineering	11.901	2.927
interests include	11.382	5.923
vitae curricula	11.048	11.048
computer vision	10.632	1.851
other hand	10.266	5.282
data set	8.604	1.525
computer society	8.379	3.859
software development	8.023	2.546
distributed systems	7.985	2.749
computer graphics	7.927	1.880
data sets	6.943	1.528
information systems	6.554	2.577
operating system	6.451	2.425
other words	6.054	3.301
see figure	6.043	2.201
total number	5.978	2.601
pattern analysis	5.691	1.546
same time	5.126	3.376
operating systems	5.106	2.434
computer engineering	5.020	2.477
user interface	4.892	1.963

Table A.17: Top 24 extracted formulaic speech suited patterns of length three

Multi-term	tf	df
vitae curricula vitae	11.048	11.048
curricula vitae curricula	11.048	11.048
research interests include	10.260	5.479
parallel and distributed	6.937	2.158
analysis and machine	5.425	1.487
degree in computer	4.603	2.397
institute of technology	4.563	2.775
shown in figure	4.430	2.078
number of processors	3.559	793
described in section	3.545	1.819
department of computer	3.513	2.221
national science foundation	3.390	2.464
university of california	3.315	1.951
science and engineering	2.985	1.829
shown in table	2.901	1.570
professor of computer	2.663	2.010
number of nodes	2.652	932
electrical and computer	2.486	1.442
hardware and software	2.392	1.506
discussed in section	2.301	1.372
point of view	2.300	1.534
organized as follows	2.152	2.115
degree in electrical	2.112	1.339
university of illinois	2.058	1.151

Table A.18: Top 24 extracted formulaic speech suited patterns of length four

Multi-term	tf	df
curricula vitae curricula vitae	11.048	11.048
pattern analysis and machine	5.404	1.480
analysis and machine intelligence	5.384	1.478
science from the university	3.796	2.575
degree in computer science	3.765	2.125
parallel and distributed systems	3.052	1.344
department of computer science	2.938	1.934
professor in the department	2.342	1.796
engineering from the university	2.337	1.742
electrical and computer engineering	2.334	1.383
professor of computer science	2.287	1.739
science at the university	2.173	1.593
parallel and distributed computing	2.133	1.146
vision and pattern recognition	1.947	745
computer vision and pattern	1.937	737
computer science and engineering	1.917	1.225
degree in electrical engineering	1.814	1.220
engineering at the university	1.389	1.060
degrees in computer science	1.347	1.091
presented in this paper	1.280	917
engineering and computer science	1.259	898
due to the fact	1.176	851
electrical engineering and computer	1.149	808
current research interests include	1.060	878

Table A.19: Top 24 extracted formulaic speech suited patterns of length five

Multi-term	*tf*	*df*
pattern analysis and machine intelligence	5.378	1.476
computer science from the university	3.602	2.455
paper is organized as follows	2.005	2.005
computer science at the university	1.998	1.465
computer vision and pattern recognition	1.918	733
electrical engineering and computer science	1.055	756
curricula vitae of curricula vitae	1.024	564
electrical engineering from the university	961	798
authors would like to thank	877	865
department of electrical and computer	822	599
associate professor in the department	771	691
assistant professor in the department	740	654
member of the technical staff	627	505
university of texas at austin	624	425
computer engineering from the university	599	522
vitae curricula vitae of curricula	550	550
work was supported in part	545	537
state university of new york	540	382
computer engineering at the university	537	429
associate professor of computer science	527	480
lecture notes in computer science	521	357
knowledge discovery and data mining	500	256
engineering from the indian institute	477	390
programming languages and operating systems	470	269

Table A.20: Top 24 extracted formulaic speech suited patterns of length six

Multi-term	*tf*	*df*
professor in the department of computer	1.224	986
department of electrical and computer engineering	818	597
vitae curricula vitae of curricula vitae	550	550
curricula vitae curricula vitae of curricula	550	550
science from the university of california	542	459
professor in the department of electrical	525	457
annals of the history of computing	469	218
department of computer science and engineering	466	357
vitae of curricula vitae of curricula	457	241
support for programming languages and operating	453	259
rest of the paper is organized	452	452
transactions on knowledge and data engineering	358	257
computer science department at the university	356	305
professor of electrical and computer engineering	354	317
rest of this paper is organized	350	350
professor in the computer science department	347	309
science from the university of illinois	330	286
research interests are in the areas	328	294
transactions on parallel and distributed systems	312	263
science at the university of california	303	242
science and engineering at the university	296	233
national institute of standards and technology	294	253
transactions on pattern analysis and machine	290	197
remainder of this paper is organized	290	290

Table A.21: Top 24 extracted formulaic speech suited patterns of length seven

Multi-term	tf	df
degree in computer science from the university	1.139	857
professor in the department of computer science	1.001	833
professor of computer science at the university	624	548
department of computer science at the university	580	476
curricula vitae curricula vitae of curricula vitae	550	550
computer science from the university of california	515	434
vitae of curricula vitae of curricula vitae	457	241
curricula vitae of curricula vitae of curricula	457	241
degrees in computer science from the university	452	412
support for programming languages and operating systems	448	255
architectural support for programming languages and operating	441	258
electrical and computer engineering at the university	426	346
associate professor in the department of computer	417	391
engineering from the indian institute of technology	413	353
assistant professor in the department of computer	404	363
degree in electrical engineering from the university	351	311
computer science from the university of illinois	328	284
computer science at the university of california	292	232
transactions on pattern analysis and machine intelligence	288	196
computer science and engineering at the university	280	218
engineering and computer science at the university	246	197
electrical and computer engineering from the university	242	218
department of electrical engineering and computer science	242	166
circling the appropriate number on the reader	208	208

A.5 Extracted Acronyms

Table A.22: Top 24 extracted acronyms of length two

Acronym	Full form	tf	df
IP	internet protocol	90	2
IP	intellectual property	63	2
VR	virtual reality	58	2
AI	artificial intelligence	52	4
ML	maximum likelihood	45	1
PE	processing element	43	3
CA	cellular automata	43	2
CT	computed tomography	42	3
RF	radio frequency	41	2
SA	simulated annealing	39	1
GA	genetic algorithm	33	1
IR	information retrieval	32	3
EM	expectation maximization	31	2
OS	operating system	30	2
IT	information technology	26	2
DP	dynamic programming	23	2
NN	nearest neighbor	23	7
PC	program counter	22	2
MR	magnetic resonance	22	4
VE	virtual environment	20	1
MD	molecular dynamics	19	1
SC	sequential consistency	18	1
EU	european union	18	1
NI	network interface	17	2

Table A.23: Top 24 extracted acronyms of length three

Acronym	Full form	tf	df
NSF	national science foundation	207	16
ATM	asynchronous transfer mode	154	2
UML	unified modeling language	105	3
RDF	resource description framework	98	4
GPS	global positioning system	95	4
PCA	principal component analysis	90	1
DAG	directed acyclic graph	90	3
API	application programming interface	86	3
MAP	maximum a posteriori	86	3
FFT	fast fourier transform	83	3
LRU	least recently used	80	4
SVD	singular value decomposition	79	2
MIT	massachusetts institute of technology	74	2
DCT	discrete cosine transform	72	3
MPI	message passing interface	72	4
IDL	interface definition language	71	3
GUI	graphical user interface	70	2
RMI	remote method invocation	68	2
CGI	common gateway interface	67	2
JVM	java virtual machine	65	3
SSL	secure sockets layer	62	2
MRI	magnetic resonance imaging	59	7
UDP	user datagram protocol	59	2
MDL	minimum description length	58	3

Table A.24: Top 24 extracted acronyms of length four

Acronym	Full form	tf	df
IETF	internet engineering task force	97	4
VRML	virtual reality modeling language	92	3
VLIW	very long instruction word	69	2
SOAP	simple object access protocol	62	10
ISCA	int'l symp. computer architecture	56	1
VLSI	very large scale integration	55	3
NIST	national institute of standards and technology	54	3
SGML	standard generalized markup language	47	3
WSDL	web services description language	43	7
PSTN	public switched telephone network	42	3
SNMP	simple network management protocol	39	2
VLDB	very large data bases	39	2
PARC	palo alto research center	39	3
LFSR	linear feedback shift register	38	2
SMIL	synchronized multimedia integration language	38	2
ARPA	advanced research projects agency	38	3
PODS	principles of database systems	38	3
ATPG	automatic test pattern generation	36	2
ICDE	int'l conf. data eng.	36	2
CDMA	code division multiple access	35	3
CVPR	computer vision and pattern recognition	35	9
MTTF	mean time to failure	35	2
ANSI	american national standards institute	34	3
LDAP	lightweight directory access protocol	34	2

Table A.25: Top 24 extracted acronyms of length five

Acronym	Full form	tf	df
DARPA	defense advanced research projects agency	88	6
CORBA	common object request broker architecture	81	3
KAIST	korea advanced institute of science and technology	52	9
CIPIC	center for image processing and integrated computing	20	6
ICDCS	int'l conf. distributed computing systems	20	2
ISSTA	int'l symp. software testing and analysis	16	3
EPSRC	engineering and physical sciences research council	16	3
TAPOS	theory and practice of object systems	16	2
ENIAC	electronic numerical integrator and computer	16	2
NSERC	natural sciences and engineering research council	15	1
RIACS	research institute for advanced computer science	12	3
ICASE	institute for computer applications in science and engineering	11	5
ICANN	internet corporation for assigned names and numbers	11	2
EDSAC	electronic delay storage automatic calculator	10	2
DARPA	defense advanced research project agency	10	2
IJCAI	int'l joint conf. artificial intelligence	10	1
ISCAS	international symposium on circuits and systems	10	1
ICDCS	international conference on distributed computing systems	10	3
TOSEM	transactions on software engineering and methodology	10	3
NPACI	national partnership for advanced computational infrastructure	9	3
AFIPS	american federation of information processing societies	9	1
PPOPP	principles and practice of parallel programming	9	4
HKUST	hong kong university of science and technology	9	3
NSERC	natural science and engineering research council	8	1

Table A.26: Top 24 extracted acronyms of length six

Acronym	Full form	tf	df
ASPLOS	architectural support for programming languages and operating systems	25	3
TAPADS	theoretical aspects of parallel and distributed systems	11	5
SIGMOD	special interest group on management of data	10	1
PECASE	presidential early career award for scientists and engineers	9	3
ICLASS	illinois computer laboratory for aerospace systems and software	9	8
DOCSIS	data over cable service interface specification	7	2
LSSDSV	large scientific and software data set visualization	6	4
UMIACS	university of maryland institute for advanced computer studies	6	2
ASPLOS	architecture support for programming languages and operating systems	5	1
CESDIS	center for excellence in space data and information sciences	4	1
DCGMRP	delay constrained group multicast routing problem	4	1
NCCUSL	national conference of commissioners on uniform state laws	4	1
TOMACS	transactions on modeling and computer systems	3	3
YUPPIE	yorktown ultra parallel polymorphic image engine	3	1
ESPRIT	european strategic programme of research in information technology	3	3
DOCSIS	data over cable system interface specification	3	2
EBCDIC	extended binary coded decimal interchange code	3	1
BARWAN	bay area research wireless access network	3	1
FMOODS	formal methods for open object-based distributed systems	3	1
PCMCIA	personal computer memory card international association	3	1
AHPCRC	army high performance computing research center	3	1
MICCAI	medical image computing and computer assisted intervention	2	1
TIPHON	telecommunications and internet protocol harmonization over networks	2	1
PECASE	presidential early career awards for scientists and engineers	2	2

Table A.27: Top 13 extracted acronyms of length seven

Acronym	Full form	tf	df
EMMCVPR	energy minimization methods in computer vision and pattern recognition	7	3
WYSIWYG	what you see is what you get	5	1
SHOSLIF	self-organizing hierarchical optimal subspace learning and inference framework	2	1
YCAGWYS	you can always get what you see	1	1
ESORICS	european symposium on research in computer security	1	1
SIGCAPH	special interest group for computers and the physically handicapped	1	1
IKIWISI	i'll know it when i see it	1	1
EMERALD	event monitoring enabling responses to anomalous live disturbances	1	1
INSPASS	immigration and naturalization service passenger accelerated service system	1	1
ICIMADE	international conference on intelligent multimedia and distance education	1	1
ICANNGA	int'l conf. artificial neural networks and genetic algorithms	1	1
WYSIWYR	what you see is what you record	1	1
WYSIWYC	what you see is what you compute	1	1

A.6 INEX Topics

A.6.1 CO Topics

Table A.28: CO Topics used at INEX 2005

ID	Query
202	ontologies case study
203	code signing verification
204	moldovan semantic networks
205	marshall mcluhan
206	problems physical limits miniaturization microprocessor
207	DOM and SAX
208	"Artificial Intelligence" history
209	mining frequent pattern itemset sequence graph association
210	+multimedia "document models" "content authoring"
211	applications for mobile devices gps "global positioning system"
212	HMM "hidden Markov model" equation
213	Gibbs sampler
214	"adaptive learning" and "interactive learning" in education
215	Conference on Information and Knowledge Management CIKM
216	multimedia retrieval system architecture
217	user-centered design of web sites
218	computer assisted composing music notes MIDI
219	learning object granularity
220	image annotation ontology
221	capabilities limitations commercial speech recognition software
222	eletronic commerce business strategies
223	wireless ATM multimedia
224	incomplete information database
225	xml security
226	corba java
227	Adaboost Bagging "ensemble learning"
228	"IPv6 deployment" "IPv6 support"
229	"latent semantic anlysis" "latent semantic indexing"
230	+brain research +"differential geometry"
231	markov chains in graph related algorithms
232	Dempster Shafer theory Database experiment
233	Synthesizers for music creation
234	"call for papers" conference workshop +multimedia
235	"Central Intelligence Agency" "Federal Bureau of Investigation" personal privacy surveillance concerns +Carnivore
236	machine translation approaches -programming
237	"Natural Language Processing" techniques "Artificial Intelligence" "Intelligent Information Retrieval" +"Medical Informatics"
238	neural network algorithm for chess
239	quantum computation
240	Software quality control and measurement
241	Single sign on + LDAP

A.6.2 COS Topics

Table A.29: COS Topics used at INEX 2005

ID	Query
202	//article[about(., ontologies)]//sec[about(., ontologies case study)]
203	//sec[about(., code signing verification)]
204	//*[about(.//au, moldovan) and about(., "semantic networks")]
205	//bdy//*[about(., "Marshall McLuhan")]
207	//*[about(., "DOM and SAX")]
208	//article[about(., "Artificial Intelligence" history)]
210	//article//(abs|sec)[about(.,+multimedia "document models" "content authoring")]
211	//article//sec[about(.//p, applications mobile devices gps "global positioning system")]
212	//*[(about(., HMM equation) OR about(., "hidden Markov model" equation)) AND .//en > 0]
216	//sec[about(., multimedia retrieval system architecture) or about(.//fig, multimedia retrieval architecture)]
219	//sec[about(., learning object granularity)]
220	//article[about(., image retrieval)]//sec[about(., annotation ontology)]
222	//article[about(. , bussiness strategies)]//sec[about(. , eletronic commerce e-commerce)]
223	//article[about(.//sec, wireless ATM multimedia)]
224	//article[about(.//bb, Lipski)]//*[about(., incomplete information database)]
225	//*[about(.//p, xml security)]
226	//*[about(.//sec, corba java)]
228	//article[about(.//abs, IPv6)]//sec[about(., "IPv6 deployment") or about(., "IPv6 support")]
229	//article[about(.//bdy,"latent semantic analysis" "latent semantic indexing")]
230	//article//sec[about(.,brain research "differential geometry")]
231	//article//sec[about(.,+"markov chains" +algorithm +graphs)]
232	//article[about(.//abs, Dempster-Shafer theory)]//sec[about(., Dempster Shafer database experiment)]
233	//article[about (.//bdy, synthesizers) and about (.//bdy, music)]
234	//article[about(.//atl,"upcoming events") OR about(.//atl,"call for papers")]//sec[about(., +multimedia conference workshop)]
236	//article[about(., machine translation approaches -programming)]
238	//article[about(.//bdy, "artificial intelligence") and .//yr<=2000]//bdy[about(., chess) and about(., algorithm)]
239	//article[about(.//bdy//sec, quantum computation) and (.//yr=2000 or .//yr=2001) and about(.//(atl|abs|kwd), - mechanics)]
240	//article[about(.//(abs|kwd),quality control measure)]//sec[about(.//p,software quality)]

A.6.3 CAS Topics

Table A.30: CAS Topics used at INEX 2005

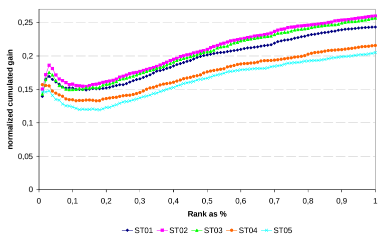
ID	Query
242	//article//sec[about(., web personalization approaches)]
243	//article//bb[about(., Schafer Anand Mulvenna Riecken)]
244	//article[about (.//fm, "query optimization")]//sec[about (., "join query optimization")]
245	//article//fm[about(., "query optimization")]
246	//article//sec[about (., "join query optimization")]
247	//article[about(.//abs,clustering) or about(.//tig,clustering)]//sec[about(.,evaluation measure)]
248	//article//abs[about(.,clustering)]
249	//article//tig[about(.,clustering)]
250	//article//sec[about(.//p, web retrieval) and about(.//p, link analysis)]
251	//article//sec//p[about(., web retrieval)]
252	//article//sec//p[about(., link analysis)]
253	//article[about(.//abs,evaluation "usability experiment" "digital libraries")]//sec[about(., evaluation methodology measures "usability testing")]
254	//article//abs[about(.,evaluation "usability experiment" "digital libraries")]
255	//article//sec[about(., evaluation methodology measures "usability testing")]
256	//article[about(.//p,"data embedding")]//p[about(.,watermarking)]
257	//sec[about(.,free public licenses gnu Linux "open source")]
258	//article[about(.,intellectual property)]//sec[about(., copyright law)]
259	//article[about(.,intellectual property)]
260	//bdy//*[about(., model checking state space explosion)]
261	//article[about(., gesture recognition)]//sec[about(., application HMM "hidden Markov model")]
262	//article[about(., gesture recognition)]
263	//article//sec[about(., application HMM "hidden Markov model")]
264	//article[about(., "machine learning") AND about(.//sec, "mutual information criterion")]
265	//article[about(.//fm//atl, "digital libraries")]//sec[about(.,"information retrieval")]
266	//article//bdy[about (., thread implementation)]
267	//article//fm//atl[about(., "digital libraries")]
268	//article//sec[about(., "information retrieval")]
269	//article[about(.,interconnected networks)]//p[about(., Crossbar networks)]
270	//article//sec[about(., introduction information retrieval)]
271	//article//p[about(.,watermarking)]
272	//article//p[about(.,embedding data)]
273	//article//sec[about(., "frequent itemsets")]
274	//article//abs[about(., "data mining")]
275	//article[about(.//abs, "data mining")]//sec[about(., "frequent itemsets")]
276	//article//sec[about(.,evaluation measure)]
277	//article//bb[about(., Baeza-Yates)]
278	//sec[about(. , string matching)]
279	//sec[about(.,approximate algorithm)]
280	//article[about(.//bb, Baeza-Yates) and about(.//sec , string matching)]//sec[about(., approximate algorithm)]
281	//article//sec[about(., copyright law)]
282	//article[about(., "machine learning")]
283	//article//sec[about(., "mutual information criterion")]
284	//article[about (.//bdy, thread implementation) and about (.//bdy, operating system)]
285	//article//bdy[about(., operating system)]
286	//article[about(.,interconnected networks)]
287	//article//p[about(., Crossbar networks)]
288	//article[about(.//bb, Schafer Anand Mulvenna Riecken)]//sec[about(., web personalization approaches)]

A.7 Evaluation Results - $nxCG$ Performance of INEX Topics

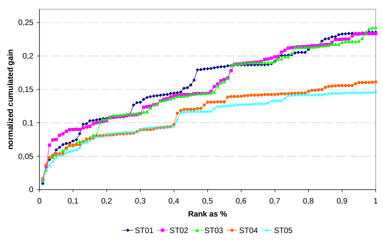
This chapter provides the complete set of performance figures of X-DOSE. Due to readability and limited space, discussions in the evaluation chapter concentrated on the strict $nxCG$ metric. Here, the $nxCG$ results of all three quantization functions, gen, strict, and genLifted, are given.

A.7.1 Experiment I - Single-Term Index Performance

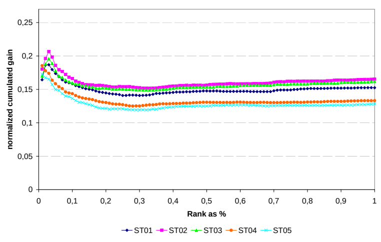
The colors in Figure A.1 refer to ST01 (blue), ST02 (purple), ST03 (green), ST04 (orange), and ST05 (cyan).



(a) gen $nxCG$



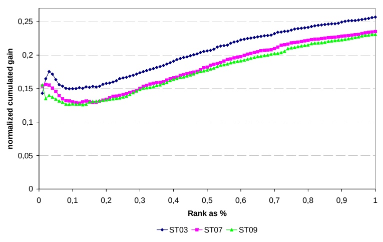
(b) strict $nxCG$



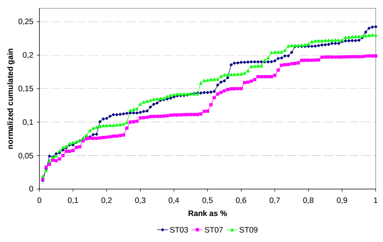
(c) genLifted $nxCG$

Figure A.1: Tokenizer performance

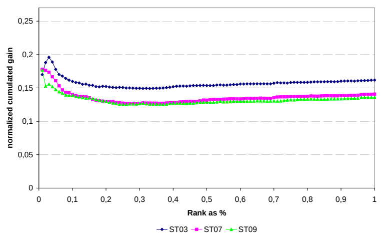
The colors in Figure A.2 refer to ST03 (blue), ST07 (purple), and ST09 (green).



(a) gen $nxCG$



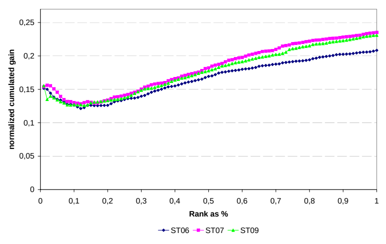
(b) strict $nxCG$



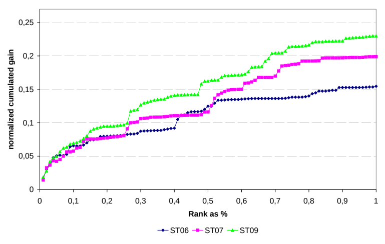
(c) genLifted $nxCG$

Figure A.2: Tagger performance

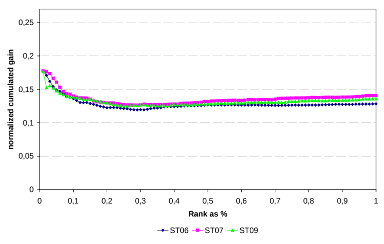
The colors in Figure A.3 refer to ST06 (blue), ST07 (purple), and ST09 (green).



(a) gen *nxCG*



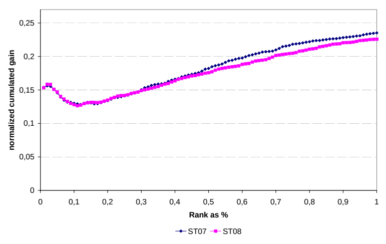
(b) strict *nxCG*



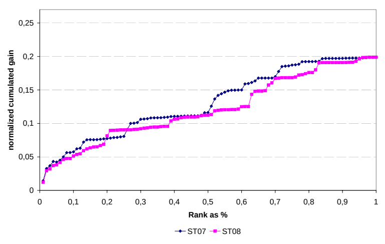
(c) genLifted *nxCG*

Figure A.3: Extractor performance

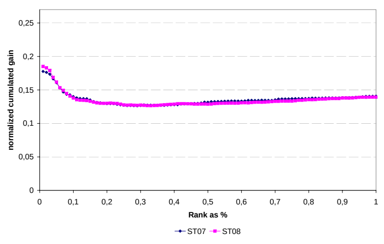
The colors in Figure A.4 refer to ST07 (blue) and ST08 (purple).



(a) gen *nxCG*



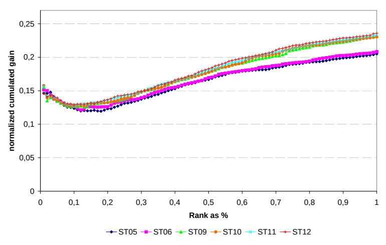
(b) strict *nxCG*



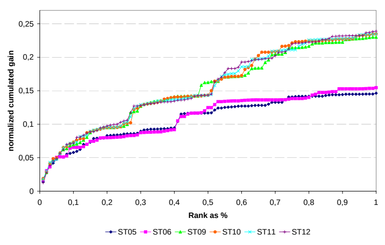
(c) genLifted *nxCG*

Figure A.4: Stemmer performance

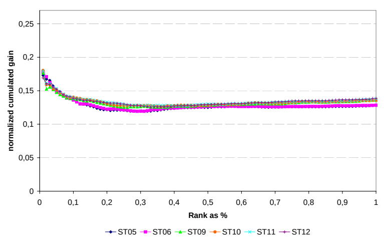
The colors in Figure A.5 refer to ST05 (blue), ST06 (purple), ST09 (green), ST10 (orange), ST11 (cyan), and ST12 (red).



(a) gen *nxCG*



(b) strict *nxCG*

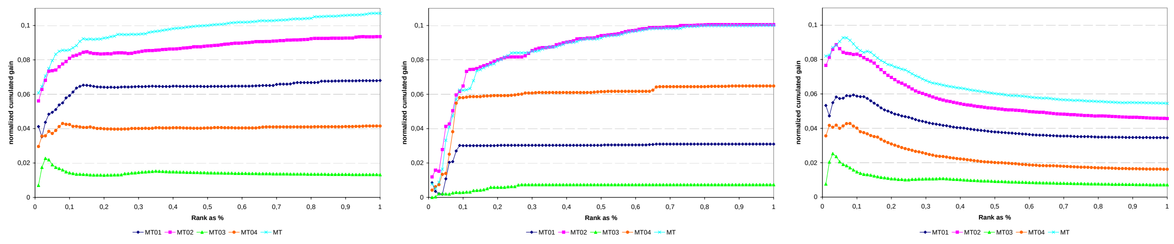


(c) genLifted *nxCG*

Figure A.5: Stopword filtering performance

A.7.2 Experiment II - Multi-Term Index Performance

The colors in Figure A.6 refer to MT01 (blue), MT02 (purple), MT03 (green), MT04 (orange), and MT (cyan).

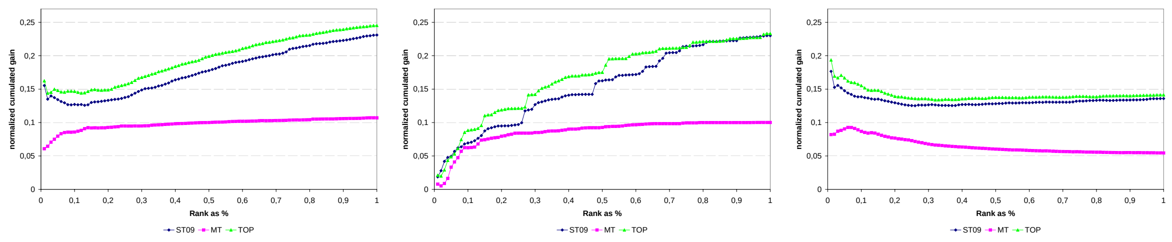


(a) gen $nxCG$ **(b)** strict $nxCG$ **(c)** genLifted $nxCG$

Figure A.6: Multi-term index performance

A.7.3 Experiment III - Combined Single-Term and Multi-Term Index Performance

The colors in Figure A.7 refer to ST09 (blue), MT (purple), and TOP (green).

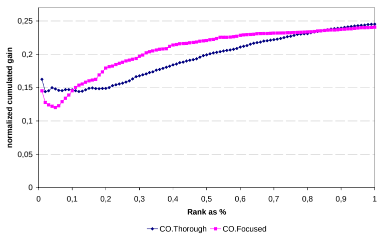


(a) gen $nxCG$ **(b)** strict $nxCG$ **(c)** genLifted $nxCG$

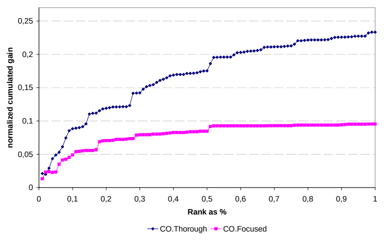
Figure A.7: Combined single-term and multi-term index performance

A.7.4 Experiment IV - Content and Structure

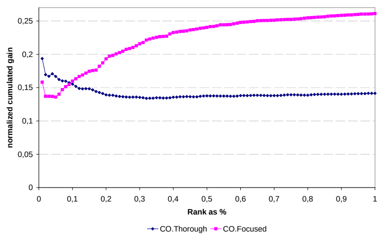
The colors in Figure A.8 refer to CO.Thorough (blue) and CO.Focused (purple).



(a) gen $nxCG$



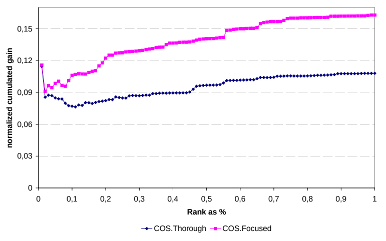
(b) strict $nxCG$



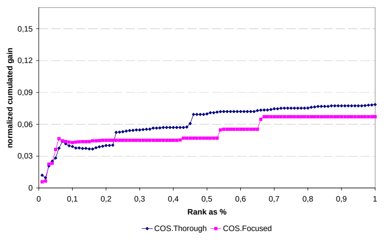
(c) genLifted $nxCG$

Figure A.8: Performance of CO topics

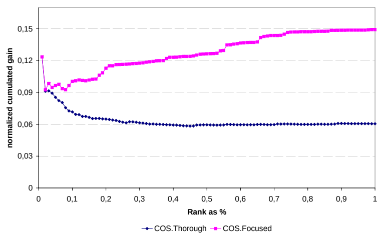
The colors in Figure A.9 refer to COS.Thorough (blue) and COS.Focused (purple).



(a) gen $nxCG$



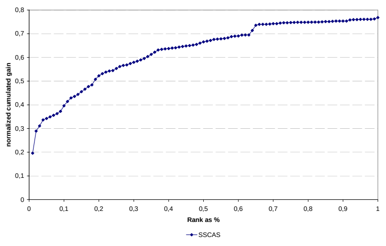
(b) strict $nxCG$



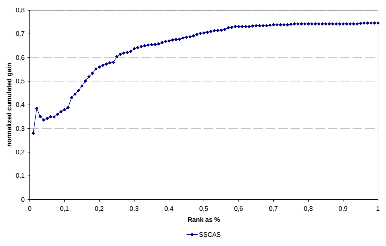
(c) genLifted $nxCG$

Figure A.9: Performance of COS topics

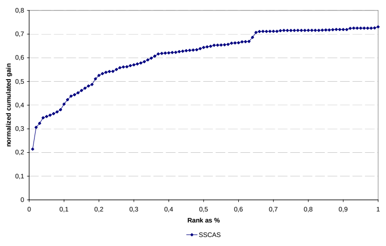
The color in Figure A.10 refers to SSCAS (blue).



(a) gen $nxCG$



(b) strict $nxCG$

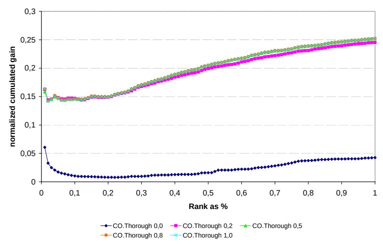


(c) genLifted $nxCG$

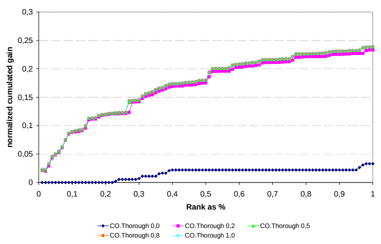
Figure A.10: Performance of SSCAS topics

A.7.5 Experiment VI - The Effect of Content Importance ci

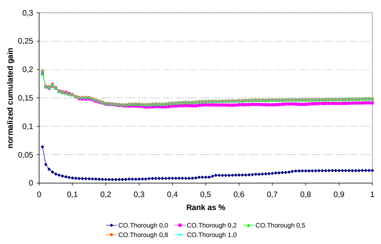
In the Figures A.11 to A.15, colors are used to decode different ci values of 0,0 (blue), 0,2 (purple), 0,5 (green), 0,8 (orange), and 1,0 (cyan).



(a) gen $nxCG$

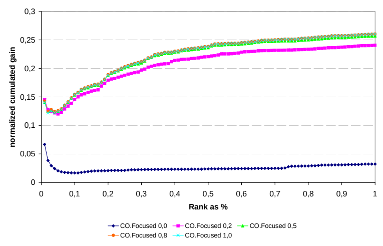


(b) strict $nxCG$

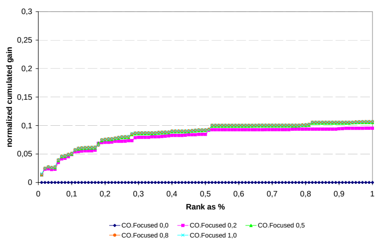


(c) genLifted $nxCG$

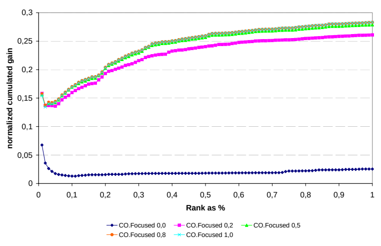
Figure A.11: CO.Thorough performance of ci



(a) gen $nxCG$

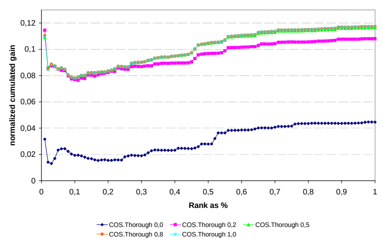


(b) strict $nxCG$

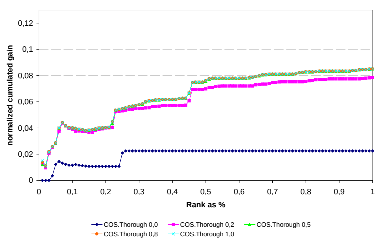


(c) genLifted $nxCG$

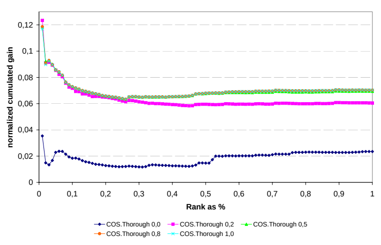
Figure A.12: CO.Focused performance of ci



(a) gen $nxCG$

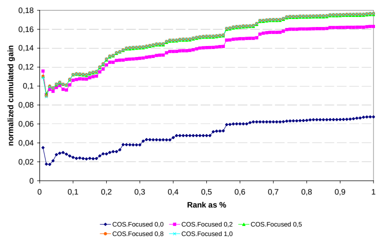


(b) strict $nxCG$

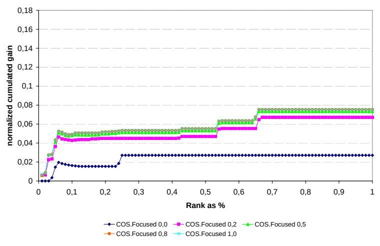


(c) genLifted $nxCG$

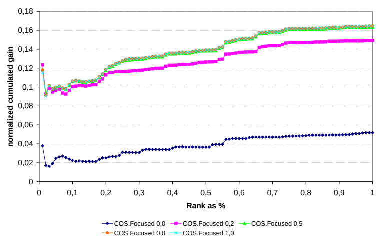
Figure A.13: COS.Thorough performance of ci



(a) gen $nxCG$

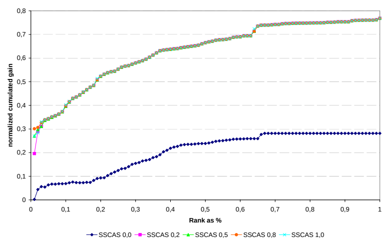


(b) strict $nxCG$

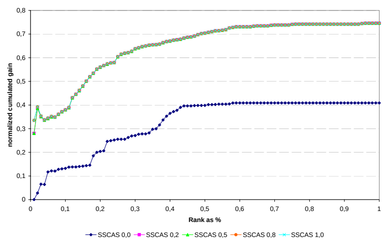


(c) genLifted $nxCG$

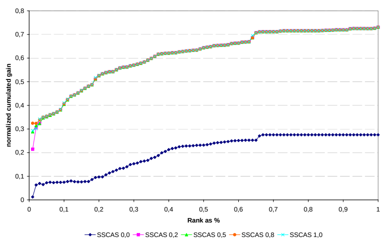
Figure A.14: COS.Focused performance of ci



(a) gen $nxCG$



(b) strict $nxCG$

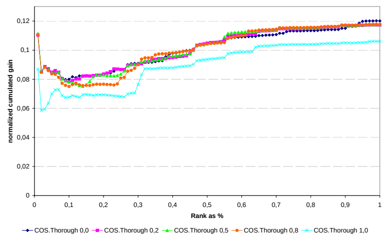


(c) genLifted $nxCG$

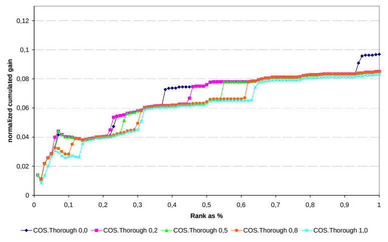
Figure A.15: SSCAS performance of ci

A.7.6 Experiment VII - The Effect of the Generality Factor gf

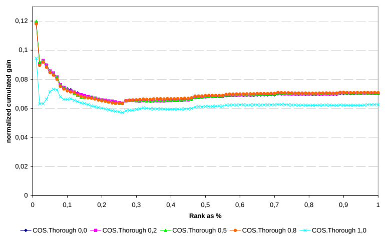
In the Figures A.16 to A.18, colors are used to decode different gf values of 0,0 (blue), 0,2 (purple), 0,5 (green), 0,8 (orange), and 1,0 (cyan).



(a) gen $nxCG$

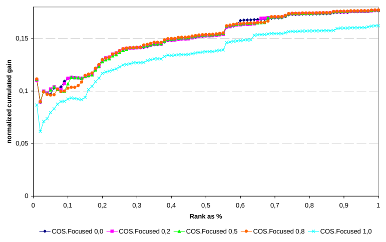


(b) strict $nxCG$

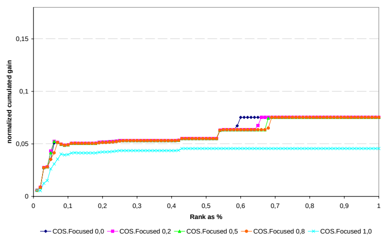


(c) genLifted $nxCG$

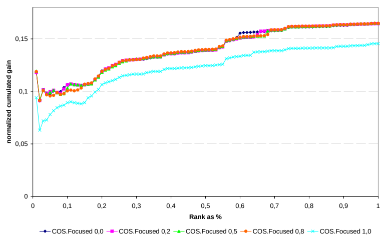
Figure A.16: COS.Thorough performance of gf



(a) gen $nxCG$

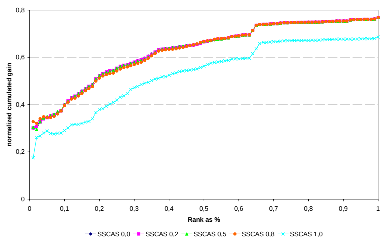


(b) strict $nxCG$

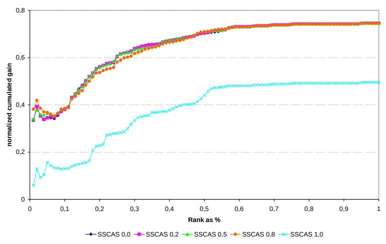


(c) genLifted $nxCG$

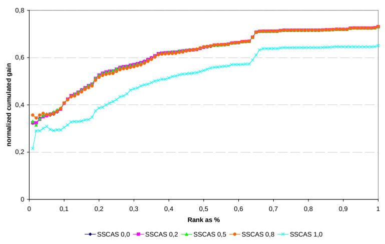
Figure A.17: COS.Focused performance of gf



(a) gen $nxCG$



(b) strict $nxCG$

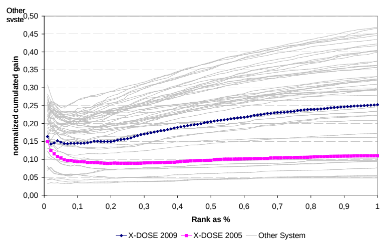


(c) genLifted $nxCG$

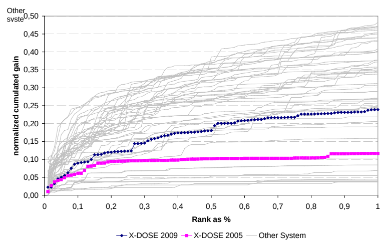
Figure A.18: SSCAS performance of gf

A.7.7 Experiment VIII - INEX 2005 Comparison

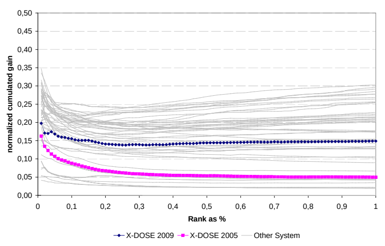
In the Figures A.19 to A.23, colors are used to decode the performance of X-DOSE'09 (blue), X-DOSE'05 (purple), and other INEX'05 systems.



(a) gen $nxCG$

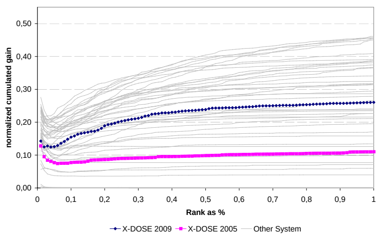


(b) strict $nxCG$

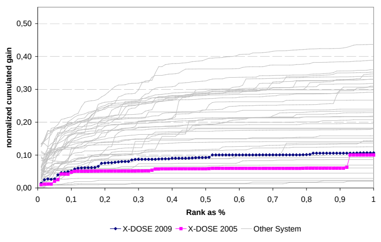


(c) genLifted $nxCG$

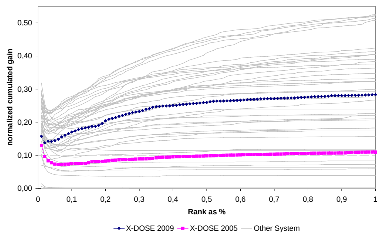
Figure A.19: CO.Thorough performance at INEX 2005



(a) gen $nxCG$

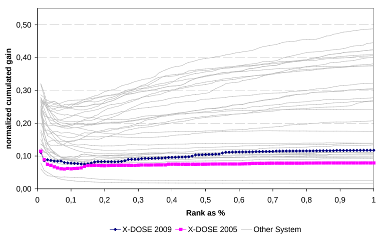


(b) strict $nxCG$

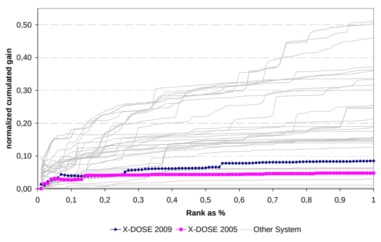


(c) genLifted $nxCG$

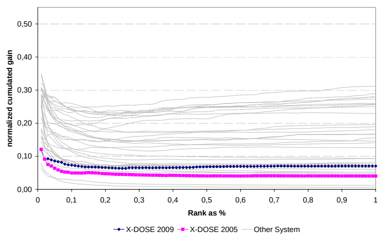
Figure A.20: CO.Focused performance at INEX 2005



(a) gen $nxCG$

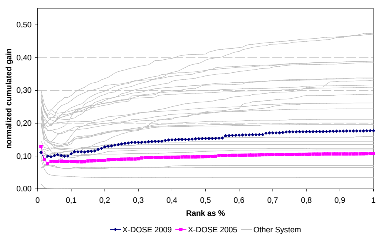


(b) strict $nxCG$

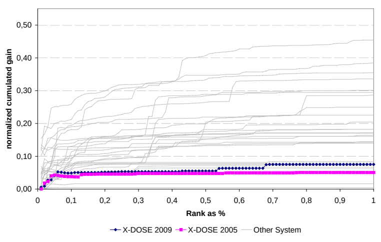


(c) genLifted $nxCG$

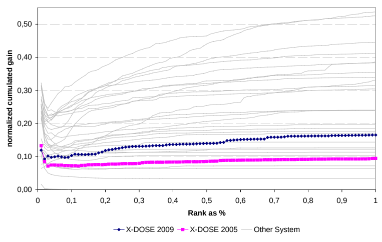
Figure A.21: COS.Thorough performance at INEX 2005



(a) gen $nxCG$

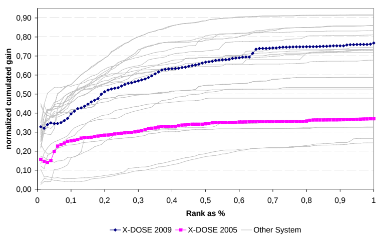


(b) strict $nxCG$

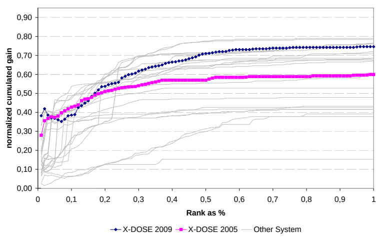


(c) genLifted $nxCG$

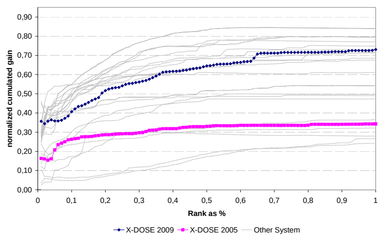
Figure A.22: COS.Focused performance at INEX 2005



(a) gen $nxCG$



(b) strict $nxCG$



(c) genLifted $nxCG$

Figure A.23: SSCAS performance at INEX 2005

Bibliography

[1] XML representation of a relational database. http://www.w3.org/XML/RDB.html, July 1997. (17)

[2] XML Linking Language (XLink) Version 1.0. http://www.w3.org/TR/xlink/, May 27 2001. (15)

[3] XML Pointer Language (XPointer). http://www.w3.org/TR/xptr/, August 16 2002. (15)

[4] Extensible Markup Language (XML). http://www.w3.org/XML, May 2006. (3, 14)

[5] Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/REC-xml, May 2006. (3, 15)

[6] HyperText Markup Language (HTML) Home Page. http://www.w3.org/MarkUp, May 2006. (3)

[7] LaTeX – A document preparation system. http://www.latex-project.org, May 2006. (3)

[8] PDF Reference. http://partners.adobe.com/public/developer/pdf/index_reference.html, May 2006. (3)

[9] The Extensible Stylesheet Language Family (XSL). http://www.w3.org/Style/XSL, May 2006. (15)

[10] Word Reference Documentation. http://msdn.microsoft.com/office/understanding/word/documentation/default.aspx, May 2006. (3)

[11] XML Schema. http://www.w3.org/XML/Schema, May 2006. (3, 15)

[12] SAX (Simple API for XML). http://sax.sourceforge.net, February 2008. (236)

[13] SAXON – The XSLT and XQuery Processor. http://saxon.sourceforge.net, October 2008. (236)

[14] Kjersti Aas and Line Eikvil. Text Categorisation: A Survey. Technical report, Norwegian Computing Center, June 1999. (167)

[15] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997. (52)

[16] Steven Abney. *Corpus-Based Methods in Language and Speech*, chapter Part-of-Speech Tagging and Partial Parsing. Kluwer Academic Publishers, Dordrecht, 1996. (91, 92)

[17] Mohammad Abolhassani and Norbert Fuhr. Applying the Divergence From Randomness Approach for Content-Only Search in XML Documents. In Sharon McDonald and John Tait, editors, *Proceedings of the 26th European Conference on Information Retrieval Research (ECIR)*, volume 2997 of *Lecture Notes in Computer Science*, Sunderland, UK, April 4–7 2004. University of Sunderland, Springer Verlag. (27, 206)

[18] Mohammad Abolhassani, Norbert Fuhr, Norbert Gövert, and Kai Großjohann. HyREX: Hypermedia Retrieval Engine for XML. Technical report, University of Dortmund, Department of Computer Science, Dortmund, Germany, 2002. (205)

[19] James F. Allen. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, USA, second edition, 1994. (92)

[20] Ofer Arazy and Carson Woo. Enhancing Information Retrieval through Statistical Natural Language Processing: A Study of Collocation Indexing. *Management Information Systems Quarterly (MIS)*, 31(Issue 3):525–546, September 3 2007. (127)

[21] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, ACM Press, New York, Essex, England, 1999. (1, 8, 12, 13, 21, 27, 38, 58, 73, 76, 79, 80, 81, 94, 95, 98, 112, 171, 183)

[22] Peter Bailey and David Hawking. A Parallel Architecture For Query Processing over a Terabyte of Text. Technical Report TR-CS-96-04, Department of Computer Science, Australian National University, Canberra 0200 ACT, Australia, 1996. (210)

[23] Francois Bancilhon, Gilles Barbedette, Véronique Benzaken, Claude Delobel, Sophie Gamerman, Christophe Lécluse, Patrick Pfeffer, Philippe Richard, and Fernando Velez. The Design and Implementation of O2, an Object-Oriented Database System. pages 1–22, New York, NY, USA, 1988. Springer-Verlag New York, Inc. (53)

[24] Francisco-Mario Barcala, Jesús Vilares Ferro, Miguel A. Alonso, Jorge Graña, and Manuel Vilares. Tokenization and Proper Noun Recognition for Information Retrieval. In A. Min Tjoa and Roland R. Wagner, editors, *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 246–250, Washington, DC, USA, September 2-6 2002. IEEE, IEEE Computer Society Press. (80)

[25] David T. Barnard, Gwen Clarke, and Nicholas Duncan. Tree-to-Tree Correction for Document Trees. Technical Report 95-372, Department of Computing and Information Science, Queen's University, 1995. (151, 159, 173)

[26] Philip Bille. A Survey on Tree Edit Distance and Related Problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005. (153)

[27] Patrick Billingsley. *Probability and Measure*. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons, Inc., 1979. (27, 206)

[28] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery`, January 23 2007. (31)

[29] Angela Bonifati and Stefano Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1):68–79, 2000. (31)

[30] Abdelhamid Bouchachia and Marcus Hassler. Classification of XML Documents. In *IEEE Symposium Series on Computational Intelligence (SSCI), Computational Intelligence and Data Mining (CIDM)*, pages 390–396. Honolulu, Hawaii, United States, IEEE Computational Intelligence Society, April 1-5 2007. (147)

[31] Thorsten Brants. TnT – A Statistical Part-Of-Speech Tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP)*, April 29 – May 3 2000. (69, 92)

[32] Thorsten Brants. Natural Language Processing in Information Retrieval. In *Proceedings of the 14th Meeting of Computational Linguistics in the Netherlands (CLIN)*, pages 1–13, Antwerp, The Netherlands, December 19 2003. (63)

[33] Andrej Bratko and Bogdan Filipič. Exploiting Structural Information in Semi-Structured Document Classification. In *Proceedings of the 13th International Electrotechnical and Computer Science Conference (ERK)*, 2004. (150)

[34] Eric Brill. A Simple Rule-based Part of Speech Tagger. In *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP)*, pages 152–155, Morristown, NJ, USA, 1992. Laboratory for Computer Science, Massachusetts Institute of Technology, Association for Computational Linguistics. (67, 92)

[35] Eric Brill. *A Corpus-Based Approach to Language Learning*. PhD thesis, University of Pennsylvania, Philadelpha, PA, USA, December 1 1993. (67)

[36] Eric Brill. Some Advances in Transformation-Based Part of Speech Tagging. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, volume 1, pages

722–727, Menlo Park, CA, USA, 1994. Laboratory for Computer Science, Massachusetts Institute of Technology, American Association for Artificial Intelligence. (67, 92)

[37] Eric Brill. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, 21(4):543–565, 1995. (67)

[38] Eric Brill. Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In David Yarovsky and Kenneth Church, editors, *Proceedings of the 3rd Workshop on Very Large Corpora (WVLC)*, pages 1–13, Somerset, New Jersey, 1995. Association for Computational Linguistics. (67)

[39] Eric Brill. *Part-Of-Speech Tagging*, pages 403–414. Volume 1 of Dale et al. [61], 2000. (91)

[40] Horst Bunke and Abraham Kandel. Mean and Maximum Common Subgraph of Two Graphs. *Pattern Recognigion Letters*, 21(2):163–168, 2000. (151)

[41] Forbes J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Text. In Nicholas J. Belkin, Peter Ingwersen, and Annelise Mark Pejtersen, editors, *Proceedings of the 15th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 112–125, New York, NY, USA, June 1992. ACM. (13, 18)

[42] James P. Callan. Passage-Level Evidence in Document Retrieval. In W. Bruce Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 302–310, Dublin, Ireland, July 1994. Springer-Verlag New York, Inc. (18)

[43] Laurent Candillier, Isabelle Tellier, and Fabien Torre. Transforming XML Trees for Efficient Classification and Clustering. In Fuhr et al. [90], pages 469–480. (151, 152, 172, 177)

[44] David Carmel, Einat Amitay, Miki Herscovici, Yoëlle Maarek, Yael Petruschka, and Aya Soffer. Juru at TREC 10 – Experiments with Index Pruning. In *Proceedings of the 10th NIST Text Retrieval Conference (TREC)*, pages 228–237, Haifa 31905, Israel, November 2001. Haifa IBM Labs. (26, 208)

[45] David Carmel, Nadav Efrati, Gad M. Landau, Yoelle S. Maarek, and Yosi Mass. An Extension of the Vector Space Model for Querying XML Documents via XML Fragments. In Ricardo Baeza-Yates, Norbert Fuhr, and Yoelle S. Maarek, editors, *Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval*, 15. August 2002. (208)

[46] Kai-Uwe Carstensen, Christian Ebert, Cornelia Endriss, Susanne Jekat, Ralf Klabunde, and Hagen Langer, editors. *Computerlinguistik und Sprachtechnologie – Eine Einführung*. Elsevier GmbH, Spektrum Akademischer Verlag, second edition, 2004. (91, 93)

[47] Sudarshan S. Chawathe. Comparing Hierarchical Data in External Memory. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 90–101, 1999. (151, 152, 177)

[48] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 26–37, 1997. (151, 152)

[49] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change Detection in Hierarchically Structured Information. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 493–504, 1996. (151, 152, 153, 177)

[50] Yves Chiaramella. Information Retrieval and Structured Documents. *Lecture Notes on Computer Science (LNCS)*, 1980:286–309, 2001. (18, 19)

[51] Yves Chiaramella, Philippe Mulhem, and Franck Fourel. A Model for Multimedia Information Retrieval. Technical Report FERMI ESPRIT BRA 8134, University of Glasgow, July 4 1996. (33, 205)

[52] Heting Chu. *Information Representation and Retrieval in the Digital Age*. Thomas H. Hogan, Sr. for the American Society for Information Science and Technology, second edition, 2005. (94)

[53] James Clark and Steve DeRose. XML Path Language (XPath). `http://www.w3.org/TR/xpath`, November 1999. (15, 31)

[54] Grégory Cobéna, Serge Abiteboul, and Amélie Marian. Detecting Changes in XML Documents. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, San Jose, CA, 2002. (151, 152)

[55] Ronald Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Battista Varile, Annie Zaenen, and Antonio Zampolli, editors. *Survey of the State of the Art in Human Language Technology*. Cambridge University Press and Giardini, Center for Spoken Language Understanding CSLU, Carnegie Mellon University, Pittsburgh, PA., web edition edition, 1997. (30)

[56] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. Perfomance Evaluation of the VF Graph Matching Algorithm. In *Proceedings of the 10th Interna-*

tionall Conference on Image Analysis and Processing (ICIAP), volume 2, pages 1172–1177, Washington, DC, USA, 1999. IEEE Computer Society. (151)

[57] Gianni Costa, Giuseppe Manco, Riccardo Ortale, and Andrea Tagarelli. A Tree-Based Approach to Clustering XML Documents by Structure. In Jean-Francois Boulicaut, Dino Pedreschi, Floriana Esposito, and Fosca Giannottl, editors, *Knowledge Discovery in Databases: Proceeding of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, volume 3202, pages 137–148, Pisa, Italy, September 20-24 2004. Springer Lecture Notes in Computer Science (LNCS). (176)

[58] W. Bruce Croft and Jinxi Xu. Corpus-Specific Stemming using Word Form Co-occurence. In *Proceedings of the 4th Symposium on Document Analysis and Information Retrieval*, pages 147–159, Las Vegas, Nevada, April 1995. (72, 110)

[59] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A Practical Part-of-Speech Tagger. In *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP)*, pages 133–140, Morristown, NJ, USA, 1992. Association for Computational Linguistics. (68)

[60] Theodore Dalamagas, Tao Cheng, Klaas-Jan Winkel, and Timos Sellis. A Methodology for Clustering XML Documents by Structure. *Information Systems*, 31(3):187–228, 2006. (152, 177)

[61] Robert Dale, Hermann Moisl, and Harold Somers, editors. *Handbook of Natural Language Processing*, volume 1. Marcel Dekker, Inc., 2000. (308, 322)

[62] John L. Dawson. Suffix Removal and Word Conflation. *ALLC Bulletin*, 2(3):33–46, 1974. (70)

[63] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via de EM Algorithm. *The Journal of Royal Statistical Society*, 39:1–38, 1977. (20)

[64] Ludovic Denoyer and Patrick Gallinari. Bayesian Network Model for Semi-Structured Document Classification. *Information Processing and Management, Pergamon Press, Inc., Tarrytown, NY, USA*, 40(5):807–827, 2004. (149, 150)

[65] Ludovic Denoyer, Patrick Gallinari, and Anna-Marie Vercoustre. XML Mining Challenge at INEX 2005. Technical report, University of Paris VI, INRIA, 2006. (167, 194)

[66] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML. `http://www.w3.org/TR/NOTE-xml-ql/`, August 19 1998. (31)

[67] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, David Maier, and Dan Suciu. Querying XML Data. *IEEE Data Engineering Bulletin*, 22(3):10–18, 1999. (31)

[68] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., second edition, 2001. (164)

[69] Christos Faloutsos and Douglas W. Oard. A Survey of Information Retrieval and Filtering Methods. Technical Report CS-TR-3514, University of Maryland at College Park, College Park, MD, USA, 1995. (175)

[70] Mary Fernández, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the Boat with Strudel: Experiences with a Web-site Management System. In *Proceedings of the 1998 ACM International Conference on Management of Data (SIGMOD)*, pages 414–425, New York, NY, USA, 1998. ACM. (52)

[71] Sergio Flesca, Giuseppe Manco, Elio Masciari, and Luigi Pontieri. Fast Detection of XML Structural Similarity. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):160–175, 2005. (151, 177)

[72] Günther Fliedl. *Natürlichkeitstheoretische Morphosyntax – Aspekte der Theorie und Implementierung*. Gunter Narr Verlag (GNV), Dischingerweg 5, D-72070 Tübingen, 1999. (104)

[73] Daniela Florescu and Donald Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Technical report, May 1999. (43, 45, 55)

[74] Richard Foster. *Document Clustering in Large German Corpora Using Natural Language Processing*. PhD thesis, University of Zurich, 2006. (195)

[75] Christopher Fox. *Information Retrieval: Data Structures and Algorithms*, chapter Lexical Analysis and Stoplists, pages 102–130. Prentice-Hall, Inc., 1992. (64, 65, 80, 93)

[76] Christopher Fox. A Stop List for General Text. *SIGIR Forum*, 24(1-2):19–21, r 90. (69, 70, 93, 98, 108)

[77] William Bill Frakes and Ricardo A. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ, USA, 1992. (80)

[78] Francesco De Francesca, Gianluca Gordano, Riccardo Ortale, and Andrea Tagarelli. Distance-Based Clustering of XML Documents. In Luc De Raedt and Takashi Washio, editors, *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences (MGTS)*, pages 75–78. ECML/PKDD'03 Workshop Proceedings, September 2003. (178)

[79] Norbert Fuhr and Norbert Gövert. Index Compression vs. Retrieval Time of Inverted Files for XML Documents. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM)*, pages 662–664, New York, NY, USA, 2002. ACM Press. (24, 207)

[80] Norbert Fuhr and Norbert Gövert. Index Compression vs. Retrieval Time of Inverted Files for XML Documents. Technical report, University of Dortmund, 2002. (24, 207)

[81] Norbert Fuhr, Norbert Gövert, and Kai Großjohann. HyREX: Hyper-Media Retrieval Engine for XML. In Kalervo Järvelin, Micheline Beaulieu, Ricardo Baeza-Yates, and Sung Hyon Myaeng, editors, *Proceedings of the 25th ACM SIGIR International Conference on Research and Development in Information Retrieval*, page 449, New York, NY, USA, 2002. ACM. (33, 205)

[82] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors. *Advances in XML Information Retrieval and Evaluation, Proceedings of the 1st International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, ERCIM Workshop Proceedings, Sophia Antipolis, France, December 9–11 2002. ERCIM. (8, 35, 314, 316, 317, 319, 320, 321, 322, 323, 326, 327)

[83] Norbert Fuhr, Norbert Gövert, and Thomas Rölleke. DOLORES: A System for Logic-Based Retrieval of Multimedia Objects. In *Proceedings of the 21st ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 257–265, New York, NY, USA, 1998. ACM Press. (19, 27)

[84] Norbert Fuhr and Kai Großjohann. XIRQL – An Extension of XQL for Information Retrieval. In Ricardo Baeza-Yates, Norbert Fuhr, Ron Sacks-Davis, and Ross Wilkinson, editors, *Proceedings of the SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 28, 2000 2000. ACM. (23, 31)

[85] Norbert Fuhr and Kai Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In Kraft et al. [157], pages 172–180. (xix, 15, 23, 24, 26, 42, 206)

[86] Norbert Fuhr and Kai Großjohann. XIRQL: An XML Query Language Based on Information Retrieval Concepts. *ACM Transactions on Information Systems*, 22(2):313–356, 2004. (13, 23, 28, 31, 33, 205)

[87] Norbert Fuhr, Kai Großjohann, and Sasha Kriewel. *A Query Language and User Interface for XML Information Retrieval*, volume 2818 of *Lecture Notes in Computer Science*, pages 59–75. Springer, Heidelberg, 2003. (23, 31, 33, 34, 205, 206)

[88] Norbert Fuhr and Mounia Lalmas, editors. *Advances in XML Information Retrieval and Evaluation, Proceedings of the 5th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, ERCIM Workshop Proceedings, Sophia Antipolis, France, December 17–19 2006. ERCIM Springer LNCS. (35)

[89] Norbert Fuhr, Mounia Lalmas, and Saadia Malik, editors. *Advances in XML Information Retrieval and Evaluation, Proceedings of the 2nd International Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)*, ERCIM Workshop Proceedings, Sophia Antipolis, France, December 15–17 2003. ERCIM Springer LNCS. (35, 313, 321, 322)

[90] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Gabriella Kazai, editors. *Advances in XML Information Retrieval and Evaluation, Proceedings of the 4th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, volume 3977 of *ERCIM Workshop Proceedings*, Sophia Antipolis, France, November 28–30 2005. Dagstuhl Castle, Germany, ERCIM Springer Lecture Notes in Computer Science (LNCS), Springer-Verlag GmbH. (10, 35, 167, 194, 308, 316, 319, 320, 322, 327)

[91] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Gabriella Zoltan Szlavik, editors. *Advances in XML Information Retrieval and Evaluation, Proceedings of the 3rd International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, volume 3493 of *ERCIM Workshop Proceedings*, Sophia Antipolis, France, December 06–08 2004. ERCIM Springer LNCS. (35, 321, 326)

[92] Norbert Fuhr, Mounia Lalmas, and Andrew Trotman, editors. *Advances in XML Information Retrieval and Evaluation, Proceedings of the 6th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, ERCIM Workshop Proceedings, Sophia Antipolis, France, December 17–19 2007. ERCIM Springer LNCS. (35)

[93] Norbert Fuhr, Saadia Malik, and Mounia Lalmas. Overview of the INitiative for the Evaluation of XML Retrieval (INEX) 2003. In Fuhr et al. [89], pages 1–18. (36, 38)

[94] Norbert Fuhr and Thomas Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems*, 15(1):32–66, 1997. (206)

[95] Norbert Fuhr and Thomas Rölleke. HySpirit – A Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, pages 24–38, Heidelberg et al., 1998. Springer. (207)

[96] Shlomo Geva, Marcus Hassler, and Xavier Tannier. XOR – XML Oriented Retrieval Language. In Andrew Trotman and Shlomo Geva, editors, *Proceedings of the ACM SIGIR*

Workshop on XML Element Retrieval Methodology, pages 5–12, Seattle, WA, USA, August 10 2006. ACM Press, New York City, NY, USA. (7, 23)

[97] Joydeep Ghosh. *The Handbook of Data Mining*, chapter Scalable Clustering, pages 247–277. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 2003. (176, 192, 193)

[98] Emmanuel Giguet. The Stakes of Multilinguality: Multilingual Text Tokenization in Natural Language Diagnosis. In *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence Workshop Future issues for Multilingual Text Processing*, Cairns, Australia, August 27 1996. (66, 80)

[99] Kevin Glass and Shaun Bangay. Evaluating Parts-Of-Speech Taggers for Use in a Text-to-Scene Conversion System. In *Proceedings of the 2005 Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries (SAICSIT)*, pages 20–28, Republic of South Africa, September 2005. South African Institute for Computer Scientists and Information Technologists. (92)

[100] Marco Gori, Marco Maggini, and Lorenzo Sarti. Exact and Approximate Graph Matching Using Random Walks. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 27(7):1100–1111, 2005. (151)

[101] Norbert Gövert. Bilingual Information Retrieval with HyREX and Internet Translation Services. In Carol Peters, editor, *Workshop on Cross-Language Information Retrieval and Evaluation*, volume 2069 of *LNCS*, pages 237–244, Heidelberg, 2001. Springer-Verlag, Lecture Notes in Computer Science. (43, 51, 206)

[102] Norbert Gövert, Norbert Fuhr, Mohammad Abolhassani, and Kai Großjohann. Content-Oriented XML Retrieval with HyREX. In Fuhr et al. [82], pages 26–32. (205)

[103] Norbert Gövert and Gabriella Kazai. Overview of the INitiative for the Evaluation of XML Retrieval (INEX) 2002. In Fuhr et al. [82], pages 1–17. (36, 38, 39)

[104] Torsten Grabs, Klemens Böhm, and Hans-Jörg Schek. XMLTM: Efficient Transaction Management for XML Documents. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM)*, pages 142–152, New York, NY, USA, 2002. ACM. (52)

[105] Torsten Grabs and Hans-Jörg Schek. ETH Zürich at INEX: Flexible Information Retrieval from XML with PowerDB-XML. In Fuhr et al. [82], pages 141–148. (21)

[106] Torsten Grabs and Hans-Jörg Schek. Generating Vector Spaces On-the-Fly for Flexible XML Retrieval. In *Proceedings of the XML and Information Retrieval Workshop - 25th Annual*

International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 4–13, Tampere, Finland, August 2002. ACM Press. (21, 27, 57)

[107] Gregory Grefenstette and Pasi Tapanainen. What is a word, What is a Sentence? Problems of Tokenization. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX)*, pages 79–87. Xerox Research Centre Europe, MLTT, 1994. (65, 78, 80, 81, 126)

[108] Kai Grosjohann, Norbert Fuhr, Daniel Effing, and Sasha Kriewel. Query Formulation and Result Visualization for XML Retrieval. In *Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval*. ACM, 2002. (33, 206)

[109] Kai Großjohann, Norbert Fuhr, Daniel Effing, and Sasha Kriewel. A User Interface for XML Document Retrieval. In *Informatik bewegt: Informatik 2002 - 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI)*, Informatik 2002, pages 166–170. Springer GI, Heidelberg, 2002. (33, 34, 206)

[110] David A. Grossman and Ophir Frieder. *Information Retrieval - Algorithms and Heuristics*. Springer, second edition, 2004. (19, 94, 99, 110)

[111] David A. Grossman, Ophir Frieder, David O. Holmes, and David C. Roberts. Integrating Structured Data and Text: A Relational Approach. *J. Am. Soc. Inf. Sci.*, 48(2):122–132, 1997. (53)

[112] Torsten Grust. Accelerating XPath Location Steps. In *Proceedings of the 2002 ACM International Conference on Management of Data (SIGMOD)*, pages 109–120. ACM Press, 2002. (44, 45, 53, 55)

[113] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate XML Joins. In *Proceedings of the 2002 ACM International Conference on Management of Data (SIGMOD)*, pages 287–298, New York, NY, USA, 2002. ACM Press. (152)

[114] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. In *Proceedings of the 15th International Conference on Data Engineering (ICDE)*, pages 512–521, Washington, DC, USA, 1999. IEEE Computer Society. (177)

[115] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *Information Systems*, 25(5):345–366, 2000. (177)

[116] Jin Guo. Critical Tokenization and its Properties. *Computational Linguistics*, 23(4):569–596, 1997. (66, 74, 80, 83)

[117] Jin Guo. One Tokenization Per Source. In Christian Boitet and Pete Whitelock, editors, *Proceedings of the 36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 457–463, Morristown, NJ, USA, 1998. Association for Computational Linguistics. (66, 74, 80, 83)

[118] Markus Hagenbuchner, Alessandro Sperduti, Ah Chung Tsoi, Francesca Trentini, Franco Scarselli, and Marco Gori. Clustering XML Documents Using Self-Organizing Maps for Structures. In Fuhr et al. [90], pages 481–496. (172)

[119] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of Data Mining*. MIT Press, Cambridge, MA, USA, 2001. (1)

[120] Marcus Hassler and Abdelhamid Bouchachia. Searching XML Documents – Preliminary Work. In Fuhr et al. [90], pages 119–133. (41, 260)

[121] Marcus Hassler, Abdelhamid Bouchachia, and Roland Mittermeir. Classification of XML Documents. *International Journal of Information Technology and Intelligent Computing (Int. J. IT&IC)*, 2(4):26, November 2007. (147)

[122] Marcus Hassler and Günther Fliedl. Text Preparation through Extended Tokenization. In Alessandro Zanasi, Carlos A. Brebbia, and Nelson F.F. Ebecken, editors, *Data Mining VII: Data, Text and Web Mining and their Business Applications*, volume 37, pages 13–21. Prague, Czech Republic, WIT Press, Wessex Institute of Technology, July 11-13 2006. (63, 81)

[123] Marcus Hassler, Christian Hofbauer, and Günther Fliedl. The Klagenfurt Computer Linguistic Resource Portal. `http://clr.uni-klu.ac.at`, September 2006. (89)

[124] Marcus Hassler and Franz Kollmann. Secure Management of Structured Documents. In Veljko Milutinovic, editor, *Proceedings of the International Conference on Advances in the Internet, Processing, Systems, and Interdisciplinary Research (IPSI)*, page 18. IPSI Belgrade, Academic Mind, November 10–13 2005. (41)

[125] Kenji Hatano, Hiroko Kinutani, Masahiro Watanabe, Masatoshi Yoshikawa, and Shunsuke Uemura. Determining the Unit of Retrieval Results for XML Documents. In Fuhr et al. [82], pages 57–64. (22, 25)

[126] David Hawking. PADRE – A Parallel Document Retrieval Engine. In *Proceedings of the 3rd Fujitsu Parallel Computing Workshop*, Kawasaki, Japan, November 1994. (210)

[127] David Hawking. The Design and Implementation of a Parallel Document Retrieval Engine. Technical Report TR-CS-95-08, Department of Computer Science, Australian National University, 1995. (210)

[128] David Hawking. Document Retrieval in OCR-Scanned Text. In *Proceedings of the 6th Parallel Computing Workshop*, pages P2–F, Kawasaki, Japan, November 1996. paper P2-F. (210)

[129] David Hawking. PADRE for COWs. In Paul Mackerras, editor, *Proceedings of the 7th Parallel Computing Workshop*, Canberra, Australia, September 1997. Department of Computer Science, ANU. (210)

[130] David Hawking. Scalable Text Retrieval for Large Digital Libraries. In *Proceedings of the 1st European Conference on Research and Advanced Technology for Digital Libraries*, pages 127–145. Springer-Verlag, 1997. (210)

[131] David Hawking and Peter Bailey. *PADRE v. 2.4 User Manual*. Department of Computer Science, Australian National University, August 28 1996. (210)

[132] David Hawking, Peter Bailey, and David Campbell. A Parallel Document Retrieval Server for the World Wide Web. In *Proceedings of the Australian Document Computing Symposium*, pages 73–78, Melbourne, Australia, March 1996. (210)

[133] David Hawking, Peter Bailey, David Campbell, Paul B. Thistlewaite, and Andrew Tridgell. A PADRE in MUFTI (A Multi User Free Text retrieval Intermediary). In *Proceedings of the 4th Parallel Computing Workshop*, pages 75–84, London, England, September 1995. (210)

[134] David Hawking, Peter Bailey, and Nick Craswell. Efficient and Flexible Search Using Text and Metadata. Technical report, CSIRO Mathematical and Information Sciences, May 2000. (210)

[135] David Hawking, Paul Thistlewaite, and Peter Bailey. ANU/ACSys TREC-5 Experiments. In D. K. Harman, editor, *Proceedings of the 5th International Text Retrieval Conference (TREC)*, pages 275–290, Gaithersburg, MD, February 1 1997. U.S. National Institute of Standards and Technology. (210)

[136] David Hawking and Paul B. Thistlewaite. Searching for Meaning with the Help of a PADRE. In D. K. Harmann, editor, *Proceedings of the 3rd International Text Retrieval Conference (TREC)*, pages 257–267, Gaithersburg, MD, November 1994. (210)

[137] Djoerd Hiemstra. A Database Approach to Content-Based XML Retrieval. In Fuhr et al. [82], pages 111–118. (45, 53)

[138] Adel Hlaoui and Shengrui Wang. A New Algorithm for Inexact Graph Matching. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR)*, volume 4, 2002. (151)

[139] Adel Hlaoui and Shengrui Wang. A New Median Graph Algorithm. In Edwin R. Hancock and Mario Vento, editors, *Proceedings of the 4th IAPR International Workshop on Graph Based Representations in Pattern Recognition (GbRPR)*, volume 2726 of *Lecture Notes in Computer Science*, pages 225–234, York, UK, June 30 - July 2 2003. Springer. (151)

[140] Xiuzhen Huang and Jing Lai. Maximum Common Subgraph: Upper Bound and Lower Bound Results. *First International Multi-Symposium of Computer and Computational Sciences (IMSCCS)*, 1:40–47, 2006. (151)

[141] Peter Jackson and Isabelle Moulinier. *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorisation*. John Benjamins Publishing Company, Amsterdam, Netherlands, Wolverhampton, United Kingdom, 2002. (73, 80, 81)

[142] Hosagrahar V. Jagadish, Shurug A. Al-Khalifa, Adriane Chapman, Laks V. S. Lakshmanan, Andrew Nierman, Stylianos Paparizos, Jaqdish Himatlal Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu, and Cong Yu. TIMBER: A Native XML Database. *The VLDB Journal*, 11(4):274–291, April 2002. (53)

[143] Kalervo Järvelin and Jaana Kekäläinen. Cumulated Gain-Based Evaluation of IR Techniques. *ACM Transactions on Information Systems*, 20(4):2002, 2002. (239)

[144] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of Trees – An Alternative to Tree Edit. In *Journal on Theoretical Computer Science*, volume 143, pages 137–148, 1995. (153)

[145] Jaap Kamps, Maarten de Rijke, and Börkur Sigurbjörnsson. Length Normalization in XML Retrieval. In *Proceedings of the 27th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 80–87, New York, NY, USA, 2004. ACM Press. (19)

[146] Jaap Kamps, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. XML Retrieval: What to Retrieve? In *Proceedings of the 26th ACM SIGIR International Conference on Research and Development in Informaion Retrieval*, pages 409–410, New York, NY, USA, 2003. ACM. (29)

[147] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A Local Search Approximation Algorithm for k-Means Clustering. In *Proceedings of the 18th ACM Symposium on Computational Geometry (SCG)*, pages 10–18, New York, NY, USA, 2002. ACM Press. (176, 192)

[148] Marcin Kaszkiel and Justin Zobel. Passage Retrieval Revisited. In *Proceedings of the 20th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 178–185, Philadelphia, July 1997. ACM Press. (13, 18)

[149] Gabriella Kazai and Mounia Lalmas. INEX 2005 Evaluation Measures. In Fuhr et al. [90], pages 16–29. (239, 241)

[150] Gabriella Kazai and Mounia Lalmas. Notes on What to Measure in INEX. In Andrew Trotman, Mounia Lalmas, and Norbert Fuhr, editors, *Proceedings of the INEX 2005 Workshop on Element Retrieval Methodology*, Second Edition, pages 22–38. University of Glasgow, Glasgow, Scotland, July 30 2005. (39)

[151] Gabriella Kazai, Mounia Lalmas, and Thomas Rölleke. Focussed Structured Document Retrieval. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 2476, pages 241–247, London, UK, 2002. Springer-Verlag. (4, 11, 12, 26)

[152] Gabriella Kazai and Thomas Roelleke. A Scalable Architecture for XML Retrieval. In Fuhr et al. [82], pages 49–56. (22, 207)

[153] Latifur Khan and Yan Rao. A Performance Evaluation of Storing XML Data in Relational Database Management Systems. In *Proceedings of the 3rd International Workshop on Web Information and Data Management (WIDM)*, pages 31–38, New York, NY, USA, 2001. ACM. (4, 52)

[154] Pekka Kilpeläinen and Heikki Mannila. Ordered and Unordered Tree Inclusion. *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing*, 24(2):340–356, 1995. (153)

[155] Donald Knuth. *The Art of Programming*, volume 1-3. Addison-Wesley, 1968. (153)

[156] Evangelos Kotsakis. Structured Information Retrieval in XML Documents. In *Proceedings of the 17th ACM International Symposium on Applied Computing (SAC)*, pages 663–667, New York, NY, USA, 2002. ACM Press. (12, 26)

[157] Donald H. Kraft, Bruce W. Croft, David J. Harper, and Justin Zobel, editors. *Proceedings of the 24th ACM SIGIR International Conference on Research and Development in Information Retrieval*, New Orleans, Louisiana, United States, September 9-13 2001. ACM Press. (312, 324)

[158] Robert Krovetz. Viewing Morphology as an Inference Process. In *Proceedings of the 16th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 191–202, New York, NY, USA, 1993. ACM. (71, 111)

[159] Robert Jeffrey Krovetz and W. Bruce Croft. Word Sense Disambiguation Using Machine-Readable Dictionaries. *SIGIR Forum*, 23(SI):127–136, 1989. (111)

[160] Daniel T. Larose. *Discovering Knowledge in Data – An Introduction to Data Mining*. John Wiley and Sons Inc, November 2004. (1)

[161] Ray R. Larson. Cheshire II at INEX: Using a Hybrid Logistic Regression and Boolean Model for XML Retrieval. In Fuhr et al. [82], pages 18–25. (209)

[162] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966. (151)

[163] Wang Lian, David Wai lok Cheung, Nikos Mamoulis, and Siu-Ming Yiu. An Efficient and Scalable Algorithm for Clustering XML Documents by Structure. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):82–96, 2004. (177)

[164] Shaorong Liu, Qinghua Zou, and Wesley W. Chu. Configurable Indexing and Ranking for XML Information Retrieval. In *Proceedings of the 27th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 88–95. ACM Press, 2004. (22)

[165] Julie Beth Lovins. Development of a Stemming Algorithm. *Mechanical Translation and CDomputational Linguistics*, 11(1–2):22–31, 1968. (70, 110)

[166] Anna Lubiw. Some NP-Complete Problems Similar to Graph Isomorphism. In *SIAM Journal of Computing*, volume 10, pages 11–21, 1981. (151)

[167] Robert W. P. Luk, Alvin T. S. Chan, Tharam S. Dillon, and Hong Va Leong. A Survey of Search Engines for XML Documents. In *Proceedings of the ACM SIGIR Workshop on XML and Information Retrieval*, Athens, Greece, July 2000. (17, 18)

[168] Robert W. P. Luk, Hong Va Leong, Tharam S. Dillon, Alvin T. S. Chan, W. Bruce Croft, and James Allan. A Survey in Indexing and Searching XML Documents. *Journal of the American Society for Information Science and Technology (JASIST)*, 53(6):415–437, 2002. (17, 26, 31)

[169] J. B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In L. M. LeCam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Berkeley University of California Press, 1967. (176, 192)

[170] Saadia Malik, Gabriella Kazai, Mounia Lalmas, and Norbert Fuhr. Overview of INEX 2005. In Fuhr et al. [90], pages 1–15. (237)

[171] Murali Mani, Dongwon Lee, and Makoto Murata. Normal Forms for Regular Tree Grammars. Technical report, UCLA Computer Science Department, 2001. (151)

[172] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. (42)

[173] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, London, England, fifth printing edition, 2002. (73, 76, 77, 78, 79, 80, 81, 84, 91, 92, 93, 94, 108, 111)

[174] Yosi Mass and Matan Mandelbrod. Retrieving the Most Relevant XML Component. In Fuhr et al. [89], pages 53–58. (22)

[175] Yosi Mass and Matan Mandelbrod. Component Ranking and Automatic Query Refinement for XML Retrieval. In Fuhr et al. [91], pages 134–140. (22)

[176] Yosi Mass, Matan Mandelbrod, Einat Amitay, David Carmel, Yoelle Maarek, and Aya Soffer. JuruXML – An XML Retrieval System at INEX'02. In Fuhr et al. [82], pages 73–80. (208)

[177] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallas Quass, and Jennifer Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997. (52)

[178] Wolfgang Meier. eXist: An Open Source Native XML Database. In Erhard Rahm, B. Chaudri, Mario Jeckle, and Rainer Unland, editors, *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, volume 2593, pages 169–183, London, UK, 2003. Springer-Verlag LNCS. (53)

[179] Bruno T. Messmer and Horst Bunke. A New Algorithm for Error-Tolerant Subgraph Isomorphism Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–505, 1998. (151)

[180] Andrei Mikheev. Periods, Capitalized Words, etc. *Computational Linguistics*, 28(3):289–318, 2002. (67, 74, 83, 84)

[181] Elke Mittendorf and Peter Schäuble. Document and Passage Retrieval Based on Hidden Markov Models. In *Proceedings of the 17th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 318–327, Dublin, Ireland, July 1994. Springer-Verlag New York, Inc. (18)

[182] Muc. Message Understanding Conference (MUC-6). `http://www.cs.nyu.edu/cs/faculty/grishman/muc6.html`, September 2006. (82)

[183] Muc. Message Understanding Conference (MUC-7). `http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/muc_7_toc.html`, September 2006. (82)

[184] Richard Myers, Richard C. Wilson, and Edwin R. Hancock. Bayesian Graph Edit Distance. In *Proceedings of the 10th International Conference on Image Analysis and Processing (ICIAP)*, page 1166, Washington, DC, USA, 1999. IEEE Computer Society. (151)

[185] Gonzalo Navarro and Ricardo A. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Transactions on Information Systems*, 15(4):400–435, October 1997. (13)

[186] Richi Nayak and Sumei Xu. XML Documents Clustering by Structures. In Fuhr et al. [90], pages 432–442. (202, 203)

[187] Andrew Nierman and Hosagrahar Jagadish. Evaluating Structural Similarity in XML Documents. In *Proceedings of the 5th International Workshop on the Web and Databases (WebDB)*, June 2002. (152, 155)

[188] John O'Connor. Retrieval of Answer-Sentences and Answer-figures from Papers by Text Search. In *Information Processing and Management*, volume 11, pages 155–164, 1975. (18)

[189] John O'Connor. Answer-Passage Retrieval by Text Searching. *Journal of the American Society for Information Science*, 31(4):227–239, July 1980. (18)

[190] Paul Ogilvie and Jamie Callan. Language Models and Structured Document Retrieval. In Fuhr et al. [82], pages 33–40. (19)

[191] Richard A. O'Keefe and Andrew Trotman. The Simplest Query Language That Could Possibly Work. In Fuhr et al. [89], pages 167–174. (23)

[192] openNLP Tagger. openNLP Tagger. `http://opennlp.sourceforge.net`, September 2006. (90)

[193] Chris D. Paice. Another Stemmer. *SIGIR Forum*, 24(3):56–61, 1990. (71)

[194] Chris D. Paice. An Evaluation Method for Stemming Algorithms. In *Proceedings of the 17th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 42–50, New York, NY, USA, 1994. Springer-Verlag New York, Inc. (71)

[195] Sukomal Pal. XML Retrieval – A Survey. Technical report, Indian Statistical Institute, Kolkata, Computer Vision and Pattern Recognition Unit (CVPR), June 30 2006. (12, 17, 18, 23, 26, 39)

[196] David D. Palmer. *Tokenisation and Sentence Segmentation*, pages 11–35. Volume 1 of Dale et al. [61], 2000. (81)

[197] David D. Palmer and Marti A. Hearst. Adaptive Multilingual Sentence Boundary Disambiguation. *Computational Linguistics*, 23(2):241–267, June 1997. (66)

[198] Uchang Park. An Implementation of XML Documents Search System Based on Similarity in Structure and Semantics. In *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration (WIRI)*, pages 97–103. Yeojin Seo Duksung Women's University, April 08–09 2005. (151)

[199] Dan Pelleg and Andrew Moore. Accelerating Exact k-Means Algorithms with Geometric Reasoning. In *Proceedings of the 5th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 277–281, New York, NY, USA, 1999. ACM Press. (176, 192)

[200] Luuk Peters. Change Detection in XML Trees: A Survey. In *Proceedings of the 4th Twente Student Conference on IT*, 2005. (xxii, 152, 153)

[201] Karen Pinel-Sauvagnat and Mohand Boughanem. A Survey on XML Focussed Component Retrieval. In *Large-Scale Semantic Access to Content (Text, Image, Video and Sound) (RIAO)*, page Electronic Medium, Pittsburgh, USA, June 2007. Centre de Hautes Etudes Internationales D'Informatique Documentaire (C.I.D.). (26)

[202] Benjamin Piwowarski, Georges-Etienne Faure, and Patrick Gallinari. Bayesian Networks and INEX. In Fuhr et al. [82], pages 149–154. (20)

[203] Martin F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, July 1980. (70, 111)

[204] QTag. QTag. `http://www.english.bham.ac.uk/staff/omason/software/qtag.html`, September 2006. (90)

[205] Adwait Ratnaparkhi. A Maximum Entropy Model for Part-of-Speech Tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 133–142, Somerset, New Jersey, 1996. University of Pennsylvania, Association for Computational Linguistics. (92)

[206] Jonathan Robie, Joe Lapp, and David Schach. XML Query Language (XQL). `http://www.w3.org/TandS/QL/QL98/pp/xql.html`, September 1998. (23, 31)

[207] Thomas Rölleke and Norbert Fuhr. HySpirit – A Flexible System for Investigating Probabilistic Logical Information Retrieval. Technical report, University of Dortmund, Dortmund, Germany, 1997. (207)

[208] Thomas Rölleke and Norbert Fuhr. Retrieving Complex Objects with HySpirit. In J. Furner and D. J. Harper, editors, *Proceedings of the 19th BCS-IRSG Colloquium on IR Research*, pages 32–43, Aberdeen, 1997. Robert Gordon University. (207)

[209] Thomas Rölleke, Ralf Lübeck, and Gabriella Kazai. The HySpirit Retrieval Platform. In Kraft et al. [157], page 454. (207)

[210] Carl Sable and Ken Church. Using Bins to Empirically Estimate Term Weights for Text Categorization. In Lillian Lee and Donna Harman, editors, *Proceedings of the 6th International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 58–66, Pittsburgh, US, 2001. Association for Computational Linguistics, Morristown, US. (164)

[211] Gerard Salton. *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971. (8, 21, 27)

[212] Gerard Salton, J. Allan, and Chris Buckley. Approaches to Passage Retrieval in Full Text Information Systems. In *Proceedings of the 16th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 49–58. ACM Press, 1993. (18)

[213] Gerard Salton and Micheal E. Lesk. Computer Evaluation of Indexing and Text Processing. *Journal of the ACM*, 15(1):8–36, 1968. (8, 21, 27)

[214] Gerard Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620, 1975. (8, 21)

[215] Christer Samuelsson. Morphological Tagging Based Entirely on Bayesian Inference. In Robert Eklund, editor, *Proceedings of the 9th Nordic Conference on Computational Linguistics (NODALIDA)*, pages 225–238. Stockholm University, 1993. (92)

[216] Bilge Say. *An Information-Based Approach to Punctuation*. PhD thesis, Institute of Engineering and Science af Bilkent University, 1998. (80)

[217] Bilge Say and Varol Akman. An Information-Based Treatment of Punctuation. In *Proceedings of the 2nd International Conference on Mathematical Linguistics (ICML)*, pages 93–94, Tarragona, Spain, 1996. (80)

[218] Anne Schiller, Simone Teufel, Christine Stöckert, and Christine Thielen. Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Institut für maschinelle Sprachverarbeitung, Stuttgart, 1999. (91)

[219] Torsten Schlieder and Holger Meuss. Result Ranking for Structured Queries against XML Documents. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, Zurich, Switzerland, December 2000. (13)

[220] Torsten Schlieder and Holger Meuss. Querying and Ranking XML Documents. *JASIST*, 53(6):489–503, 2002. (13)

[221] Helmut Schmid. Probabilistic Part-Of-Speech Tagging Using Decision Trees. In *International Conference on New Methods in Language Processing*, Manchester, UK, September 1994. (68, 92)

[222] Helmut Schmid. Improvements in Part-Of-Speech Tagging with an Application to German. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, pages 172–176, 1995. (68, 92)

[223] Albrecht Schmidt, Martin L. Kersten, Menzo Windhouwer, and Florian Waas. Efficient Relational Storage and Retrieval of XML Documents. In *Selected papers from the 3rd International Workshop WebDB 2000 on the World Wide Web and Databases (WebDB)*, volume 1997 of *Lecture Notes in Computer Science*, pages 137–150, London, UK, 2001. Springer-Verlag. (43)

[224] Fabrizio Sebastiani. Machine Learning in Automated Text Categorisation. *ACM Computing Surveys*, 34(1):1–47, 2002. (167, 168, 172)

[225] Stanley M. Selkow. The Tree-to-Tree Editing Problem. In *Information Processing Letters*, volume 6, pages 184–186, 1977. (151, 177)

[226] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. (53)

[227] Dennis Shasha and Kaizhong Zhang. Approximate Tree Pattern Matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997. (151, 153)

[228] Kurt A. Shoens, Allen Luniewski, Peter M. Schwarz, James W. Stamos, and Joachim Thomas. The Rufus System: Information Organization for Semi-Structured Data. In *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB)*, pages 97–107, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. (52)

[229] Peter Shoubridge, Miro Kraetzl, and David Ray. Detection of Abnormal Change in Dynamic Networks. In *Proceedings of Information Decision and Control*, pages 557–562. IEEE Inc., 1999. (151)

[230] Alexander Strehl, Joydeep Ghosh, and Raymond Mooney. Impact of Similarity Measures on Web-Page Clustering. In *Proceedings of the 17th National Conference on Artificial*

Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI), pages 58–64, Austin, Texas, USA, 30–31 July 2000. AAAI. (195, 196)

[231] Stanford Tagger. Stanford Tagger. `http://nlp.stanford.edu/software/tagger.shtml`, September 2006. (90)

[232] Kuo-Chung Tai. The Tree-to-Tree Correction Problem. *Journal of the Association for Computing Machinery (ACM)*, 26(3):422–433, 1979. (151)

[233] Xavier Tannier, Alan Woodley, Shlomo Geva, and Marcus Hassler. Approaches to Translating Natural Language Queries for Use in XML Information Retrieval Systems. Technical Report 2006-400-008, Ecole Nationale Supérieure des Mines de Saint-Etienne, Saint-Étienne, France, July 2006. (5, 32, 33)

[234] Tei. The Text Encoding Initiative (TEI). `http://www.tei-c.org`, September 2006. (82)

[235] Tei. The Text Encoding Initiative (TEI) Guidelines. `http://www.tei-c.org/Guidelines2/index.html`, September 2006. (82)

[236] Anja Theobald and Gerhard Weikum. Adding Relevance to XML. In *Selected papers from the 3rd International Workshop WebDB 2000 on the World Wide Web and Databases (WebDB)*, pages 105–124, London, UK, 2001. Springer-Verlag. (208)

[237] Anja Theobald and Gerhard Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT)*, pages 477–495, London, UK, 2002. Springer-Verlag. (23, 208)

[238] Richard M. Tong. Tarragon Consulting at INEX 2002: Experiments using the K2 Search Engine from Verity. In Fuhr et al. [82], pages 88–94. (209)

[239] Kristina Toutanova and Christopher D. Manning. Enriching the Knowledge Sources used in a Maximum Entropy Part-Of-Speech Tagger. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 63–70, Morristown, NJ, USA, 2000. Association for Computational Linguistics. (69)

[240] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In Fuhr et al. [91], pages 16–40. (23, 38)

[241] Andrew Trotman and Börkus Sigurbjörnsson. NEXI, Now and Next. In Fuhr et al. [91], pages 41–53. (23)

[242] Dan Tufis and Oliver Mason. Tagging Romanian Texts: A Case Study for QTAG, a Language Independent Probabilistic Tagger. In *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC)*, pages 589–596, May 28–30 1998. (69, 92)

[243] Unicode. The Unicode Consortium. `http://www.unicode.org`, September 2006. (65, 88)

[244] UTF. The Unicode Standard, Unicode Transformation Format (UTF). `http://www.unicode.org/standard/standard.html`, September 2006. (82, 88)

[245] Hans van Halteren, Jakub Zavrel, and Walter Daelemans. Improving Data Driven Wordclass Tagging by System Combination. In *Proceedings of the 36th Meeting on Association for Computational Linguistics (COLING)*, pages 491–497, Morristown, NJ, USA, 1998. Association for Computational Linguistics. (92)

[246] Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Dept. of Computer Science, University of Glasgow, London, England, 2 edition, 1979. (1, 175)

[247] Anne-Marie Vercoustre, Mounir Fegas, Saba Gul, and Yves Lechevallier. A Flexible Structure-Based Representation for XML Document Mining. In Fuhr et al. [90], pages 443–457. (202, 203)

[248] Anne-Marie Vercoustre, James A. Thom, Alexander Krumpholz, Ian Mathieson, Peter Wilkins, Mingfang Wu, Nick Craswell, and David Hawking. CSIRO INEX Experiments: XML Search using PADRE. In Fuhr et al. [82], pages 65–72. (210)

[249] Ellen M. Voorhees. The Cluster Hypothesis Revisited. In *Proceedings of the 8th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 188–196, New York, NY, USA, 1985. ACM Press. (175)

[250] Ellen M. Voorhees. Natural Language Processing and Information Retrieval. In M. T. Pazienza, editor, *Information Extraction: Towards Scalable, Adaptable Systems*, volume 1714, pages 32–48, London, UK, 1999. LNCS Springer-Verlag. (63)

[251] Jason T. L. Wang, Kaizhong Zhang, and Gung-Wei Chirn. Algorithms for Approximate Graph Matching. *Information Sciences Information Computer Science*, 82(1-2):45–74, 1995. (151)

[252] Jason Tsong-Li Wang, Kaizhong Zhang, Karpjoo Jeong, and Dennis Shasha. A System for Approximate Tree Matching. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 6(4):559–571, 1994. (151)

[253] Yuan Wang. X-Diff: A Fast Change Detection Algorithm for XML Documents. Master's thesis, University of Wisconsin, WI, USA, 2003. (151, 152)

[254] Yuan Wang, David J. DeWitt, and Jin-Yi Cai. X-Diff: A Fast Change Detection Algorithm for XML Documents. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, Bangalore, India, March 5-8 2003. IEEE Computer Society. (151)

[255] Jonathan J. Webster and Chunyu Kit. Tokenization as the Initial Phase in NLP. In International Center of Computational Logic (ICCL), editor, *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, volume 4, pages 1106–1110, Morristown, NJ, USA, August 23-28 1992. University of Trier. (64, 65, 80)

[256] Ralph Weischedel, Richard Schwartz, Jeff Palmucci, Marie Meteer, and Lance Ramshaw. Coping with Ambiguity and Unknown Words through Probabilistic Models. *Comput. Linguist.*, 19(2):361–382, 1993. (92)

[257] Ross Wilkinson. Effective Retrieval of Structured Documents. In *Proceedings of the 17th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 311–317, Dublin, Ireland, 1994. Springer-Verlag New York, Inc. (12, 13, 18)

[258] Ross Wilkinson and Justin Zobel. Comparison of Fragmentation Schemes for Document Retrieval. In *Proceedings of the 3rd International Text Retrieval Conference (TREC)*, pages 81–84, 1994. (18)

[259] Alan Woodley, Marcus Hassler, Xavier Tannier, and Shlomo Geva. Natural Language Processing and XML Retrieval. In Lawrence Cavedon and Ingrid Zukerman, editors, *Proceedings of the Australasian Language Technology Workshop (ALTW)*, pages 165–166, Sidney, Australia, November 30 – December 1 2006. Australian Language Technology Association. (31, 32)

[260] Tatsuo Yamashita and Yuji Matsumoto. Language Independent Morphological Analysis. In *Proceedings of the 6th International Conference on Applied Natural Language Processing*, pages 232–238, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. (66, 80)

[261] Yiming Yang. An Evaluation of Statistical Approaches to Text Categorization. *Information Retrieval*, 1(1/2):69–90, 1999. (167)

[262] Yiming Yang and Xin Liu. A Re-Examination of Text Categorization Methods. In Marti A. Hearst, Fredric Gey, and Richard Tong, editors, *Proceedings of the 22nd ACM*

International Conference on Research and Development in Information Retrieval (SIGIR), pages 42–49, Berkley, US, August 1999. ACM Press, New York, US. (164)

[263] Øystein Grøvlen. Natural Language Processing in Information Retrieval. Phd term paper, Norwegian Institute of Technology, January 1995. (63)

[264] Mohammed J. Zaki and Charu C. Aggarwal. XRules: An Effective Structural Classifier for XML Data. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003. (150, 152)

[265] Kaizhon Zhang, Jason Wang, and Dennis Sasha. On the Editing Distance between Undirected Acyclic Graphs. *International Journal of Foundations of Computer Science*, 7(13), 1995. (151)

[266] Kaizhong Zhang and Dennis Elliott Shasha. Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *Society for Industrial and Applied Mathematics (SIAM) Journal on Computing*, 18(6):1245–1262, 1989. (151, 152, 153, 177)

[267] Zhongping Zhang, Rong Li, Shunliang Cao, and Yangyong Zhu. Similarity Metric for XML Documents. In *Workshop on Knowledge and Experience Management (FGWM)*, 2003. (151, 152)

[268] Ying Zhao and George Karypis. Criterion Functions for Document Custering: Experiments and Analysis. Technical Report TR 01–40, Department of Computer Science, University of Minnesota, Minneapolis, MN, 2001. (195)