University of Kaiserslautern
Department of Computer Sciences
Research Group Databases and Information Systems
Prof. Dr.-Ing. Dr. h.c. Theo Härder

Improving XML Retrieval by Exploiting Content and Structure

– Diploma Thesis –

Christoph R. Hartel

November 2007

Advisor: Prof. Dr.-Ing. Dr. h.c. Theo Härder
Co-Advisor: Dipl.-Inf. Philipp Dopichaj

Contents

List of Tables

List of Figures

Part I.

Foundations

1. Preliminaries

1.1. Goals

The overall goal of this thesis is to explore how information on the structure of XML documents can be used to improve the quality of XML Retrieval. We define *retrieval quality* as how well the results generated by an IR system satisfy the user's information need. Please note that this definition excludes improvements on user interaction as well as improvements on retrieval efficiency; these fields – albeit being equally important – are thus not in the scope of this thesis. We operationalise our overall goal by breaking it down into the following three subordinate goals:

1. The **design of a conceptional framework** for Content-and-Structure (CAS) Retrieval. This framework should integrate the various aspects discussed in present day CAS literature, confine and structure them in a systematic manner, and provide a foundation for developing improvements of these aspects.
2. The **proposal of improvements**. We should analyse existing solutions for the implementation of individual CAS aspects, identify quality improvement needs, and then devise possible solutions.
3. The **evaluation of improvements**. Due to the limited scope of this thesis we will likely not be able to conduct practical experiments to test our improvement proposals. However, we should clearly define how this evaluation is to be performed to enable future work in this direction.

1.2. Notation and Terms

Throughout this thesis we endeavour to establish a clear and consistent wording and notation. Whenever a new term is introduced, it is set in italics to highlight its definition (e.g. *definition*). All important term definitions are also listed in the glossary at the end of this thesis (cf. appendix D). In appendix B we have included several tables which list general notations we use: Table B.1 provides an overview of formal entities; table B.2 lists the predefined mathematical sets we use. The various symbols and general notations we employ (like null value indicators, for example) are listed in table B.3.

1.3. Organisation

This thesis consists of four parts. In part I, after some preliminary notes, we introduce the key concepts underlying our work and devise the foundations of our XML Retrieval framework. Based on these foundations, part II describes the details of our framework and discusses related work as well as our improvement proposals; it is the main part of this thesis. In part III we propose means to evaluate the effect of our improvement proposals on retrieval quality. Finally, in part IV we sum up the contents of this thesis, draw conclusions, and outline perspectives for future work.

2. Foundations of XML Retrieval

2.1. Introduction

This thesis is about XML Retrieval or, more precisely, one specific kind of XML Retrieval called Content-and-Structure Retrieval. Before we actually turn to this, however, we will briefly introduce its two constituents: Information Retrieval and the eXtensible Markup Language, commonly known as XML.

2.1.1. Information Retrieval

Information Retrieval has been an active area of research for many years [Sin01], with researchers mainly coming from the fields of computer science, library science, and linguistics[1]. Despite of this, the term "Information Retrieval" is still anything but unambiguous, but can bear many different meanings [MRS08, chpt. 1]. We define it based on proposals by Lancaster (as cited in [vR79, chpt. 1]) and Baeza-Yates/Ribeiro-Neto [BYRN99, chpt. 1], but with emphasis on its process-oriented nature as follows[2]: *Information Retrieval* (IR for short) is the process of informing a user "on the existence (or non-existence) and whereabouts" of documents and/or document parts which to some degree satisfy an information need the user has expressed. Traditional Information Retrieval only considers whole documents and we regard XML Retrieval as a special case of Information Retrieval. However, one of the key features of XML Retrieval (as defined on page 17) is the handling of partial documents, so we include them in our definition of Information Retrieval. To ease understanding, we still use the term "document" for both whole documents and partial documents in the remainder of this section. As opposed to question answering, Information Retrieval "does *not* inform (i.e. change the knowledge of) the user on the subject of his inquiry", but only points him to appropriate sources of information.

Roughly, the general process of Information Retrieval looks like follows: Initially, the user has an arbitrary information need. For example, he likes cats and is interested in finding information on appropriate cat food. He then formulates this information need in a query and enters it in the Information Retrieval system of his choice for processing. The IR system then evaluates the query on a set of documents. During evaluation it determines for each document in the set how well it thinks the document will satisfy

[1]Although the latter two perspectives are also important, we take the computer science perspective throughout this thesis. In particular, we focus on the concepts underlying Information Retrieval *Systems* which we regard as a special kind of Information Systems.

[2]All quotations in this paragraph are from Lancaster's definition as cited in [vR79, chpt. 1].

the user's information need. After that, the system presents the user with a list of results. This list includes all documents which the system considers to (partially) satisfy the information need with the best documents at the top. Finally, the user decides by looking at these documents whether they satisfy his information need. In case they do not (or if by looking at the documents his information need has changed), the user adapts his query and iteratively runs through the process again until his information need is eventually satisfied.

We will later on refine this process and adapt it to the peculiarities of XML Retrieval. For now, we confine ourselves to discussing two important facets in more detail: queries and relevance. A *query* is system of conditions on the documents (or document parts) to be retrieved; the conditions are related by operators (such as `AND`, `OR`, and `NOT`, for example). In our example, the query might look like this:

```
cats AND NOT dogs AND food
```

An IR system may actually provide arbitrary means for the user to formulate his query. A textual representation following a defined notation (like our sample query) is very common for web search engines, for example. Alternatively the user may also be allowed to enter a natural language query, construct a query by using some sort of graphical interface, or use any other means of interaction. To abstract from the way the query is actually formulated, we regard a query as a tree whose nodes are conditions and operators. We assume that any query can be transformed into this representation. Figure 2.1 illustrates such a tree corresponding to our example.

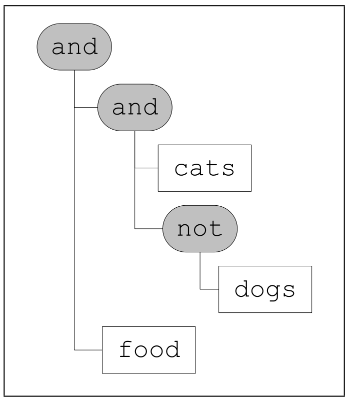

Figure 2.1.: Sample of a query tree

We can interpret a query in different ways. Most importantly, we distinguish two semantics of query conditions: constraints and hints. A *constraint* imposes a restriction which *must* be met by every item in the result set. A *hint*, on the other hand, only provides an indication of what results might be desired. It may not cover all result items, contain errors such as misspellings, or may be wrong altogether. Constraints are the typical semantic used by data-oriented systems such as databases. Their aim is to retrieve all records which exactly match a query. More precisely, no record not matching the query must be retrieved and every record matching it has to be retrieved. For IR applications this paradigm is too restrictive as IR queries are per se imprecise: Information needs are innately vague and thus no perfect transformation into a query is possible. Also IR queries are often formulated by non-expert users who are unsure of how to formulate a query and who invest little time and effort on doing so. Therefore we generally regard query conditions as hints in the context of Information Retrieval.

Relevance is a key concept in Information Retrieval [vR79, chpt. 1]. We define it as the degree $r \in [0, 1]$ to which a document satisfies an information need. If not explicitly stated otherwise, we always refer to the relevance regarding the information need expressed in a query processed by an IR system. For ease of expression we define a function $\mathrm{rv} : \mathcal{D} \longrightarrow [0, 1]$ which returns the relevance value of an arbitrary document $d \in \mathcal{D}$. By $\mathcal{D}$ we denote the *document space*, that is, the set of all documents on which a given IR query is evaluated. When *d is relevant*, this implies $\mathrm{rv}(d) \in (0, 1]$. If d is *not relevant*, $\mathrm{rv}(d)$ is zero. This view on relevance is commonly referred to as *binary relevance* [Peh06, sec. 1]. It primarily aims at determining which fragments ought to appear in the query result. (Of course additional conditions can be applied to further diminish the set of returned documents, e.g. when a maximum number of result items is enforced.) *Non-binary relevance*, on the other hand, differentiates more than two relevance levels. We can, for example, express non-binary relevance as ordinal values such as “not relevant”, “partially relevant”, “mostly relevant”, “highly relevant” or as a real number as shown above.

A concept closely related to – but not identical with – relevance is that of scoring: The *score* of a document is an IR system’s estimation of its relevance. We determine the score by using a *scoring function* $\mathrm{s} : \mathcal{D} \longrightarrow [0, 1]$. To illustrate this, consider the document shown in listing 2.1: Assuming that the user has been looking for an article about dogs, he would judge the document to have little relevance, say $r = 0.2$. As both the terms “pets” and “dogs” appear in prominent positions in the document, however, an Information Retrieval engine may assign it a rather high score (e.g. 0.8), thus providing a bad approximation of its relevance. In an ideal system, the score of a document is always equal to its relevance. By comparing the relevance of result items to the scores they are assigned by a particular IR system, we can judge this system’s retrieval quality; we will look at this in more detail in part III.

```
Popular Pets

Abstract
This article provides an overview of popular pets. We focus on
   cats, whereas dogs are only of marginal interest.

1. Introduction
...
```

Listing 2.1: Excerpt from a sample plain-text document

2.1.2. The eXtensible Markup Language

Over the last decade, the *eXtensible Markup Language* (XML) [BPSM+06] has become an increasingly popular document format which is widely used in particular for electronic data exchange and integration, today. It has been developed by the World Web Web Consortium[3] (W3C) in the late 1990's based on SGML[4]. We define an *XML document* as an ordered, labelled tree [MRS08, sec. 10.1]. This idea originates in the Document Object Model (DOM) [HHW+04] also defined by the W3C. The nodes of this tree are *XML elements*[5], each of which has exactly one parent element (except for the document root which has none) and arbitrarily many child elements. The order of the nodes in the tree is the *document order* which is defined as the order in which the corresponding elements appear in an XML document. In addition, every element has a tag name, an unordered, possibly empty set of attributes, and optionally textual content. The tag name is a string consisting of certain non-whitespace characters such as letters, digits, and hyphens. It does *not* have to be unique, that is, there can be arbitrarily many elements with the same tag name within a document. Typically, the tag name provides an indication of an element's semantics, such as in `<section>` or `<author>`, for example. For our purposes, we assume that this holds for every tag name[6]. An *attribute* consists of a name and an optional value; the name has to be unique within the containing element, the value, on the other hand, is arbitrary. The textual content contains arbitrary character strings (again, with the exception of certain special characters). It does not have to be a monolithic block, but can actually be split in several parts per containing element; the parts are treated like nodes and interleave with that element's children according to the document order. Listing 2.2 shows the customary textual representation of an XML document, figure 2.2 illustrates the corresponding logical tree structure.

```
<article>
  <title>
    Popular Pets
  </title>
  <abstract>
    This article provides an <font color="red">overview</font>
       of popular pets. We focus on cats, whereas dogs are only
       of marginal interest.
  </abstract>
  ...
</article>
```

[3]`http://www.w3.org`

[4]See `http://www.w3.org/MarkUp/SGML` for an overview.

[5]Please note that the XML specification [BPSM+06] allows for documents to include other types of nodes such as comments and parsed character data. To avoid unnecessary complexity, we use a simplified definition, however.

[6]In theory, tag names can, of course, be just random character strings. As they are an integral part of XML's aim to create a meaningful, self-describing document structure, however, we believe this assumption does not impose an actual restriction.

Listing 2.2: Sample XML fragment

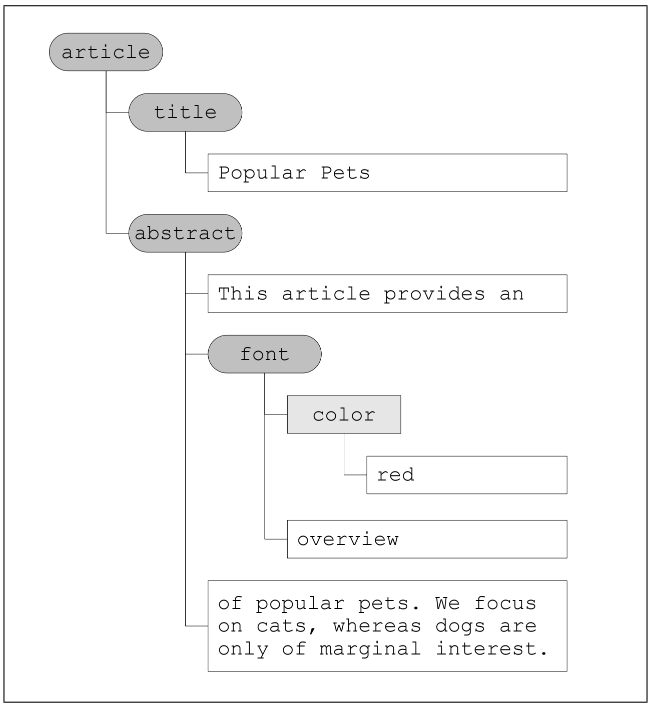
article
title
Popular Pets
abstract
This article provides an
font
color
red
overview
of popular pets. We focus on cats, whereas dogs are only of marginal interest.

Figure 2.2.: Sample XML document tree

Besides these basic aspects of XML, many extensions have been defined. Four such extensions which are of particular interest in our context are schemas, namespaces, addressing, and references. A *schema* defines constraints regarding XML documents. For example, it may define rules for elements' nesting structures and for the use of attributes; also, aspects like the typing of values may be addressed by a schema. We say that an XML document whose nesting structure observes the rules of the XML specification (cf. [BPSM$^+$06]) is *well-formed*; a document conforming to a schema is *valid*. For XML there exist two major frameworks for schema definition: document type definitions (DTDs) [BPSM$^+$06] and XML schema [TBMM04], [BPM04]. The former is more concise, the latter more powerful. For our purposes, however, it suffices to consider schemas on a more abstract level without actually looking into which of these (or other) frameworks is used. On the implementation level these details would, of course, be of interest. Namespaces are related to schemas: A *namespace* provides a vocabulary of element and attribute names [BHLT06] to enable reuse and ease the creation of uniform XML documents; a schema can then be employed to define rules on how these elements and attributes may be used. For addressing parts of an XML document, several languages have evolved. The most prominent such language is XPath [CD99]. As we will see later on, XPath is also the basis for most other XML-related languages including query languages for XML Information Retrieval. It defines 13 *axes* which represent possible relationships

Table 2.1.: Overview of important XPath axes

Axis	**Example (from listing 2.2)**
Self	`<abstract>` $\longrightarrow$ `<abstract>`
Parent	`<font>` $\longrightarrow$ `<abstract>`
Child	`<abstract>` $\longrightarrow$ `<font>`
Ancestor	`<font>` $\longrightarrow$ `<article>`
Descendant	`<article>` $\longrightarrow$ `<font>`
Preceding Sibling	`<abstract>` $\longrightarrow$ `<title>`
Following Sibling	`<title>` $\longrightarrow$ `<abstract>`
Attribute	`<font>` $\longrightarrow$ `color`

between elements, attributes, and namespaces. Table 2.1 lists the axes which are of interest to us; the second column shows a sample from listing 2.2 which illustrate the respective axis. *References* serve to create logical links from one XML element to a different element. To address specific elements within an XML document when creating a reference, we can employ the addressing mechanisms described above. There are both frameworks to reference elements within the same XML document as the referencing element (the IDREF mechanism, for example [BPSM+06]) and frameworks to reference elements in other documents (such as the XPointer framework [DJG+07]). To account for references on the conceptional level, we define an additional "reference" axis which is not included in the XPath specification; it represents the referencing element/referenced element relationship.

Before we go on to discuss XML Retrieval, we first introduce some auxiliary constructs to ease expression; these constructs will also provide the foundation for the conceptional framework of XML Retrieval which we devise in the following chapters. First of all, we define a *(document) fragment* f_d as an arbitrary subtree of an XML document d. Please note that – unlike elements – fragments are highly redundant due to the nesting structure of XML. Figure 2.3 illustrates this (the black rectangles represent fragments). The largest possible fragment is the document itself. In the context of XML Retrieval, we require f to be always well-formed and non-empty. Analogously to the document space $\mathcal{D}$ we define the *fragment space* $\mathcal{F}$ as the set of all fragments belonging to documents in $\mathcal{D}$ and the *element space* $\mathcal{E}$ as the set of all elements in these fragments. The *term space* $\mathcal{T}$ is the set of all terms (e.g. words, numbers) which occur in any document in $\mathcal{D}$. Based on these constructs, we define some auxiliary functions listed in table 2.2. In particular, these functions include some important refinements of XML content. We will use these functions in the remainder of this thesis.

Another key concept we need to introduce are (document) collections. In Information Retrieval, documents are typically grouped according to their genre, provider, or other characteristics. Thus we define a *collections* as an arbitrary set of documents. Without loss of generality, we assume every document to belong to exactly one collection and every collection to contain at least on document. For simplicity reasons we further assume that collections used for XML Retrieval only contain XML documents (that is,

Table 2.2.: XML-related auxiliary functions

Declaration	**Specification**
$\mathrm{root} : \mathcal{F} \longrightarrow \mathcal{E}$	Returns the root element of a fragment (or document); as we assume fragments to be non-empty, the root function never returns a null value ($\perp$).
$\mathrm{parent} : \mathcal{E} \longrightarrow \mathcal{E}$	Returns the parent element of an element e or $\perp$, if e is the root element.
$\mathrm{chld} : \mathcal{E} \longrightarrow \mathcal{E}^n$	Returns the set of child elements of a given element e; if e does not have any children, the function returns an empty set.
$\mathrm{desc} : \mathcal{E} \longrightarrow \mathcal{E}^n$	Returns the set of descendant elements of a given element e; if e does not have any descendants, the function returns an empty set.
$\mathrm{anc} : \mathcal{E} \longrightarrow \mathcal{E}^n$	Returns the set of ancestor elements of a given element e; if e does not have any ancestors (i.e. the root element), the function returns an empty set.
$\mathrm{sibl} : \mathcal{E} \longrightarrow \mathcal{E}^n$	Returns the set of (preceding and following) sibling elements of a given element e; if e does not have any siblings, the function returns an empty set.
$\mathrm{linkto} : \mathcal{E} \longrightarrow \mathcal{E}^n$	Returns the set of elements which reference a given element e; if e is not referenced by any element, the function returns an empty set.
$\mathrm{linkfrom} : \mathcal{E} \longrightarrow \mathcal{E}^n$	Returns the set of elements which are referenced by a given element e; if e does not reference any element, the function returns an empty set.
$\mathrm{cont}_{\mathrm{d}} : \mathcal{E} \longrightarrow \mathcal{T}^n$	Returns the *direct content* of $e \in \mathcal{E}$, that is, the ordered set of all term occurrences in e without regarding its descendant elements, attributes, attribute values, or e's tag name.
$\mathrm{cont}_{\mathrm{r}} : \mathcal{E} \longrightarrow \mathcal{T}^n$	Returns the *recursive content* of $e \in \mathcal{E}$, that is, e's direct content collated with the direct contents of all its descendants; the resulting set is ordered in document order.
$\mathrm{cont}_{\mathrm{df}} : \mathcal{E} \longrightarrow \mathcal{T}^n$	Returns the *direct full content* of $e \in \mathcal{E}$ which consists of its tag name, all its attribute names and their respective values, and its direct content, in this order.
$\mathrm{cont}_{\mathrm{rf}} : \mathcal{E} \longrightarrow \mathcal{T}^n$	Returns the *recursive full content* of $e \in \mathcal{E}$, that is, e's direct full content collated with the direct full contents of all its descendants; the resulting set is ordered in document order.
$\lvert \mathrm{cont}_{\mathrm{df}}(e \in \mathcal{E})\rvert$	The *direct content length* (number of terms) of $e \in \mathcal{E}$.
$\lvert \mathrm{cont}_{\mathrm{rf}}(e \in \mathcal{E})\rvert$	The *recursive content length* of $e \in \mathcal{E}$.
$\lvert d \in \mathcal{D}\rvert$	The *document length* of a document $d \in D$. We define it as the recursive length of d's root element, formally $\lvert d\rvert = \lvert \mathrm{cont}_{\mathrm{r}}(\mathrm{root}(d))\rvert$.

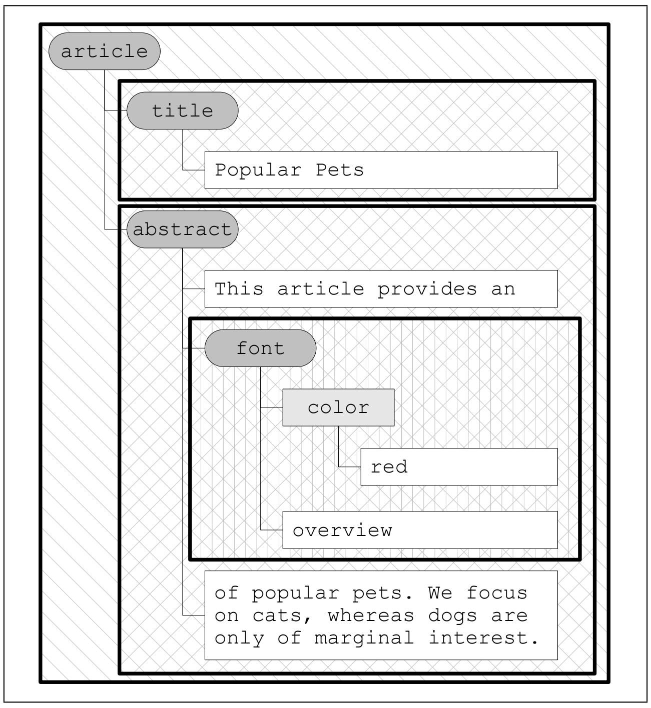
article
title
Popular Pets
abstract
This article provides an
font
color
red
overview
of popular pets. We focus on cats, whereas dogs are only of marginal interest.

Figure 2.3.: Sample XML document tree with fragments highlighted

no documents of any other format). Please note that both the document space and the fragment space are collection-independent, that is they contain *all* documents (or fragments of documents, respectively) on which a given IR query is evaluated.

2.1.3. XML Retrieval

We define *XML Retrieval* simply as Information Retrieval over XML documents [MRS08, p. 181] as opposed to the traditional "flat" retrieval over unstructured documents. Albeit sounding trivial at first, XML Retrieval introduces some interesting potentials which, at the same time, pose new challenges to Information Retrieval systems; we sum up these potentials in the following two aspects:

1. Explicitness and genericness of the logical document structure
2. Fragment-oriented retrieval

In IR literature, the first aspect is generally referred to as retrieval on "structured" documents documents (as opposed to "unstructured" ones). We argue, though, that virtually *any* document does have a structure in some sense: Even a plain text file describes content which is logically segmented in different parts such as an introduction, body, and a conclusion. Unlike XML, however, other document formats such as plain text typically do not (or only to some limited extend) allow to express this logical structure explicitly: In the case of a plain text file, for example, the logical structure can only be reflected in the physical document structure such as the spacing (paragraphs, indentation, and so on). More enhanced formats such as HTML [RHJ99] can explicitly express the logical structure, but lack the genericness of XML: In HTML we can only express predefined concepts like paragraphs and headings, whereas XML is able to express arbitrary ones; for example, we can easily represent custom entities such as `<author>` or `<play>` in an XML structure. Therefore we restrain from using the terms "structured" and "unstructured" in this context and instead refer to XML documents as being able to express the logical document structure explicitly and generically. Also, when we refer to "the" structure of a document, we always mean its logical structure, not the physical one.

For Information Retrieval this means, that when operating on XML documents, we can not only exploit the textual content of documents, but also their structure. There are two ways how we can use the structure of a document: by explicit and by implicit conditions. An *explicit condition* is provided by the user as part of his query string, whereas *implicit conditions* are generated by the IR system. Implicit conditions may either be additional, system-induced query conditions or scoring rules hard-coded in the query evaluation mechanism. Orthogonally to the explicitness of conditions, we distinguish three kinds of query conditions [TL05]:

1. *Keywords* are character strings regarding the content of documents. For example, the keyword "cat" expresses that the user wants to retrieve all documents which somehow concern cats. In traditional Information Retrieval they are the only conditions available; thus an IR system matches them against all parts of a document, be it textual or somehow related to the logical structure. In XML Retrieval we restrict keywords to the textual contents of XML elements and attribute values. Most IR systems allow for complex keywords such as strings consisting of multiple words, grouping mechanisms like phrases, etc. (cf. [TPL06, p. 278], for example). For simplicity reasons, we restrict keywords to single words consisting of alphanumeric characters, only. As keywords are not our focus, we believe this to be only a minor restriction.

2. *Support conditions* express that certain patterns in the document structure will render the corresponding part of the document helpful regarding the uses's information need. For example, when looking for documents on cats, the user may think that documents containing a chapter with the title "Popular Pets" are of particular interest to him.

3. *Target conditions* define which parts of a document to return to the user. This is closely related to fragment-oriented retrieval which we will discuss below.

Like for Information Retrieval in general, we regard all of these conditions as hints. In XML Retrieval we consider this to be mandatory[7], as formulating a precise and correct

[7]In XML Retrieval literature these interpretations of query conditions are commonly known as strict and vague evaluation, respectively [TL06]. There has been an intense debate on which interpretation is preferable, but more recently most researchers have agreed that vague evaluation is necessary; cf. e.g. [AYLP04], [KLP04].

query becomes even more difficult than in traditional IR. For ease of expression we define $\mathcal{Q}$ to be the set of all conditions contained in a given query; $\mathcal{Q}_{\text{kwd}}, \mathcal{Q}_{\text{sup}}, \mathcal{Q}_{\text{tgt}} \subseteq \mathcal{Q}$ denote the subsets of keyword, support, and target conditions, respectively.

Fragment-oriented retrieval means that the result set not only includes entires documents (as in traditional Information Retrieval), but parts of documents as well. To illustrate this, image searching for information in a library. Traditional Information Retrieval would only return entire books, whereas fragment-oriented retrieval can return the exact chapters, sections, or even paragraphs of a book which best address the information need. This has the obvious advantage of relieving the user from having "to look for the right paragraphs" manually. On the other hand, this feature also introduces some new challenges: In XML Retrieval, the parts of a document are fragments. As we have discussed in the preceding section, fragments of a document are not independent of each other, but strongly related. Namely, there is a containment relationship between fragments which in XML Retrieval literature is commonly referred to as *overlap*. Due to this items in the result set are to a large extend redundant and often have to be regarded in a particular context. Redundancy means that when looking at result items in a linear fashion, the user may come across the same content several times. This happens, for example, if a result item the user looks at is contained in another result item located higher up in the ranking. This redundancy may be undesired, so XML Retrieval has to include means for generating overlap-free result sets. The second aspect (need of context) is related to this: When the user is presented one paragraph of a book, he may need to know which book it belongs to. For example, because a paragraph from a fiction novel might be a less reliable source of information than a paragraph from a well-reputed scientific lexicon. As a consequence of XML Retrieval being fragment-oriented, we generalise the relevance and scoring functions from section 2.1.1 to operate on fragments instead of just documents as follows: $\text{rv} : \mathcal{F} \longrightarrow [0,1]$ and $\text{s} : \mathcal{F} \longrightarrow [0,1]$.

The two aspects we have just discussed (explicitness and genericness of structure and fragment-oriented retrieval) are the core ideas of XML Retrieval. There are, however, two major views on how they should be reflected in an IR system: *Content-only Retrieval* (CO) and *Content-and-Structure Retrieval* (CAS) [FML03]. CO Retrieval only partially exploits the explicitness and genericness of structure: The IR system uses the document structure for fragment-oriented retrieval, but does not allow the user to provide target or support conditions. CAS Retrieval, on the other hand, takes into account all three condition types. In this thesis we focus on CAS retrieval and in particular on target and support conditions. They impose the greatest challenges, but we also expect them to also yield the highest potential regarding high-quality XML Retrieval.

Before we go on to devising the XML Retrieval process in the next section, we will now provide a brief introduction to some organisational aspects of XML IR research: Information Retrieval in general has been an active area of research for a long time (in computer science terms, that is). Major conferences focussing on Information Retrieval are held regularly such as the Text REtrieval Conference[8] (TREC) and the SIGIR

[8]`http://trec.nist.gov`

conference which is held by the ACM Special Interest Group on Information Retrieval[9]. XML Retrieval, on the other hand, has only in recent years started to gain attention. As a main forum for XML IR research the INitiative for the Evaluation of XML Retrieval[10] (INEX) has been established in 2002 [GK02]. Besides networking the XML IR research community, one of its main aims is to provide an infrastructure for the comparative evaluation of XML Retrieval approaches. Throughout this thesis, we will at various points reference parts of the infrastructure (such as the INEX query language(s), test document collections and evaluation metrics). Also we generally follow the terminology used by INEX, except where it impedes our work.

2.2. The XML Retrieval Process

In this section we define the process of XML Retrieval. This process is *not* intended as an implementation basis (i.e. a system design), but serves as a conceptual framework for the issues discussed in this thesis. It refines the general Information Retrieval process we have outlined in section 2.1.1. Each phase is marked with a letter to indicate by whom it is performed: “U” stands for the end-user, “F” for the front-end of the IR system (i.e. its user interface), and “P” for the actual query processor. Please note that only the latter is in the scope of this thesis, whereas the other aspects – albeit being equally important – are only listed for the sake of completeness. Figure 2.4 illustrates the process.

Information Need (U) The user has an arbitrary information need. This analogous to traditional Information Retrieval and an initiates the XML Retrieval process, conceptionally.

Query Formulation (U) The user formulates his information need using the means the IR system front-end provides. This may be a natural language query, a query using some formal query language, filling out of a HTML form, using a graphical editor, or any other form of interaction the front-end supports. Query formulation marks the practical start of the process.

Query Translation (F) The IR system front-end translates the query formulated in some end-user language to the language used by the query processor. We discuss the various kinds of query languages and their translation in more detail in section 3.2.

Query Rewriting (P) The query processor optionally rewrites the query by exploiting additional knowledge which the user does not possess. This might include the expansion of query conditions, for example, to handle vagueness or generation of additional structural hints. In the rewriting phase we typically do not access the data to be queried directly, but use indexes or repositories of statistical information.

[9] `http://www.sigir.org`
[10] `http://inex.is.informatik.uni-duisburg.de`

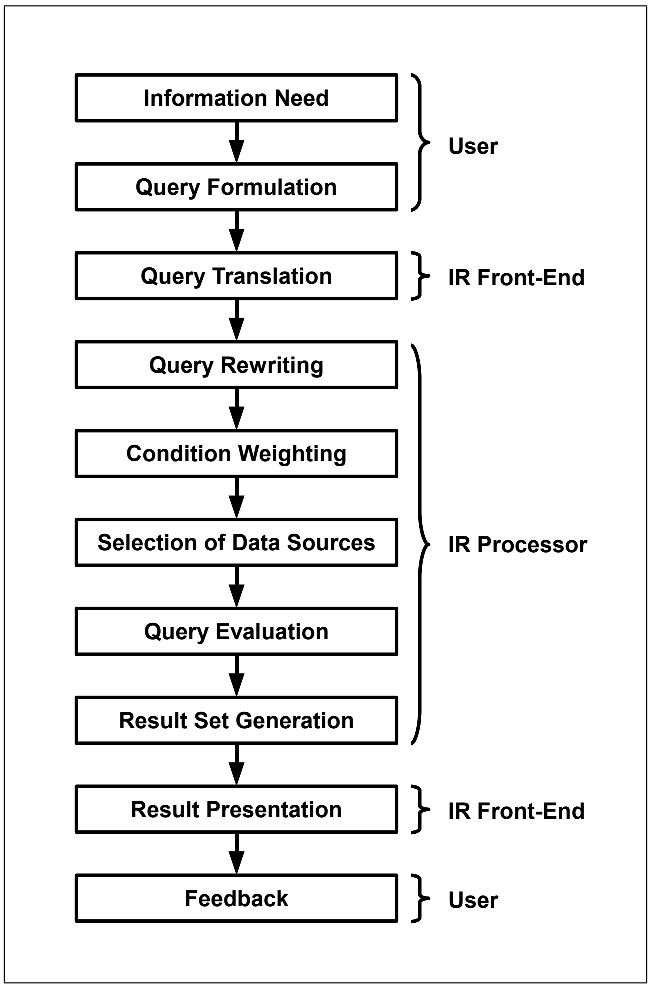

Figure 2.4.: The XML Retrieval process

Condition Weighting (P) After having rewritten the query, the processor assigns each query condition a weight. This weight controls, how "important" the condition is in regard to other conditions, that is, how strongly it influences the result. As in some contexts assigning uniform weights to all conditions suffices, this activity is also optional; in most scenarios weighting has to performed, though. We discuss weighting in section 3.5.

Selection of Data Sources (P) The IR processor decides on which data the query is to be evaluated. To support this decision, the user may have provided hints during query formulation; for example, he may have been presented with a list of available sources and selected one. In addition, data source selection may include issues like looking up remote sources, judging their credibility, and so on. None of this is in the focus of our work, so we restrict the scope to the selection of one or more collections of documents which we assume to be known in advance, perfectly credible, and locally accessible by the IR processor.

Query Evaluation (P) The previous steps have provided the IR processor with a set of weighted query conditions (potentially related by explicit operators) and a set of document collections to operate on. Based on this it evaluates each keyword and support condition producing a set of scored XML fragments as output. Target conditions are not evaluated yet in this activity. Apart from the actual query, the IR processor may also exploit implicit hints (i.e. hints not contained in query conditions) for this task.

Result Set Generation (P) Based on the scores assigned during query evaluation, the IR processor performs score propagation. This means, that it considers the context of the scored fragments (e.g. their ancestor and descendant elements in the XML document tree) and adapts the fragments' relevance scores. Independent of score propagation, it evaluates target conditions. Based on all resulting scores, it then generates the final score for each fragment. Finally, the processor selects appropriate fragments of the documents as result fragments. Doing so, it takes into account overlap and other scenario-dependant factors, if needed.

Result Presentation (F) The IR front-end prepares the result fragments for (typically) visual display. As part of this activity it decides how the results are structured (e.g. linear display, clustered per document, etc.) and ordered. It also determines all other presentation-related aspects, for example, if summaries or entire pieces of content are displayed, if result entries contain links to the original content, and so forth. It optionally provides the user with means to enter feedback for further refining the results.

Feedback (U) If supported by the IR system, the user enters feedback to further refine the retrieval results. In this case, he triggers another iteration of the XML Retrieval process which from an IR system point of view is independent of the first.

2.3. Use Cases

XML Retrieval may be useful in various scenarios. A *scenario*[11] is one specific configuration of a set of environmental properties of an IR system. It includes, for example, the kind of users operating the system, the collections used, preferences regarding the visual presentation of results, and many other factors. Depending on the respective scenario, we can make different assumptions regarding the kind of information provided in the query, properties of the document collection used, the aims of the user when querying this collection, etc. In this section we thus define use cases for XML Retrieval involving the use of structural information. By doing so we strive to determine which aspects of CAS Retrieval might be relevant in practice and which combinations of aspect configurations are likely to appear. We first classify and discuss those aspects; some of these aspects we assume to be use case invariant, whereas most aspects vary on a per use case basis. We thus derive a use case template which contains the use case-dependent aspects. Based on this template, we then define several concrete use cases.

2.3.1. Use Case Classification

We distinguish three kinds of aspects: document-related, query-related, and presentation-related ones. The first kind are aspects of the documents which the IR system operates on; the second kind characterise the queries which a user formulates; the third kind define how the user prefers the retrieval results to be composed. We mark aspects which vary per use case with a "V" (for varying) and general assumptions with an "F" (for fixed).

Document-related Aspects

- *Unique identifier (F)* We assume that every document in the document space has a *document ID* which uniquely identifies it. This may be a file name or a URL, for example.

- *Document-centric vs. data-centric (F)*: We can distinguish two kinds of XML fragments: *document-centric* and *data-centric* ones [KMdRS06][12]. A document-centric fragment typically includes very long elements like paragraphs and elements which contain both text and element nodes (so-called *mixed content*). Listing 2.3 shows an excerpt from a famous Oscar Wilde play transformed into XML to illustrate a document-centric fragment. Data-centric XML focusses on describing entities in a key/value fashion. It usually contains short elements and no mixed content, i.e. elements either contain only textual content or further elements. An example of a data-centric XML fragment is shown in listing 2.4; the fragment is an

[11]We use the term "scenario" to avoid ambiguities to the concept of "contexts" which we introduce in part III to describe parts of an XML document.

[12]To be precise, Kamps et al. [KMdRS06] (like most authors) do not categorise fragments, but whole documents as document- or data-centric. This does not affect the general idea of this categorisation, however.

excerpt from the bibliography sample in [Goo02, chpt. 8]. Information Retrieval usually focusses on document-centric content, whereas fact retrieval (i.e. common database system functionality) concentrates on data-centric content. However, data-centric and document-centric XML fragments are often mixed in a single document. A document representing a book, for example, is likely to have a document-centric part representing the book's body and various data-centric parts like the front-matter and the bibliography. Throughout this thesis we will assume purely document-oriented content for simplicity reasons. It is an interesting aspect to be addressed by future work, however, if we can exploit data-centric parts of an otherwise document-centric document as a source of additional metadata regarding the content we are interested in.

```
<play>
  <body>
    <act>
      ...
      <paragraph>
        ...
        <speaker>Cecily</speaker> I can't understand how you
            are here at all. Uncle Jack wont be back till
           Monday afternoon.
        <speaker>Algernon</speaker> That is a great
           disappointment. I am obliged to go up by the
           first train on Monday morning. I have a business
           appointment that I am anxious... to miss?
        <speaker>Cecily</speaker> Couldn't you miss it
           anywhere but in London?
        <speaker>Algernon</speaker> No: the appointment is
           in London.
        <speaker>Cecily</speaker> Well, I know, of course,
           how important it is not to keep a business
           engagement, if one wants to retain any sense of
           the beauty of life, but still I think you had
           better wait till Uncle Jack arrives. I know he
           wants to speak to you about your emigrating.
        ...
      </paragraph>
    </act>
  </body>
</play>
```

Listing 2.3: Example of document-centric XML

```
<biblioentry id="bib.TDG99">
  <abbrev id="bib.TDG99.abbrev">TDG1999</abbrev>
  <authorgroup>
    <author>
```

```
      <firstname>Norman</firstname>
      <surname>Walsh</surname>
    </author>
    <author>
      <firstname>Leonard</firstname>
      <surname>Muellner</surname>
    </author>
  </authorgroup>
  <title>DocBook</title>
  <subtitle>The Definitive Guide</subtitle>
  <pubdate>1999</pubdate>
  <edition>1</edition>
  <isbn>ISBN: 156592-580-7</isbn>
  <pagenums>648</pagenums>
</biblioentry>
```

Listing 2.4: Example of data-centric XML

- *Duplicates (F)*: Document collections often contain *duplicates*, that is, several equivalent instances of the same logical document. For example, a single scientific paper is usually available via multiple URLs (i.e. document IDs) from different web sites and therefore a duplicate from the point of view of an (XML-based) web search engine. However, duplicate handling belongs to the area of *data cleaning*[13] which is not in our scope. We therefore assume that the collections used have been perfectly cleaned beforehand and thus no duplicates exist.

- *Schema availability (V)*: A schema (e.g. a DTD [BPSM+06] or XML Schema [TBMM04]) defines the set of elements and attributes which one or more documents use and rules for nesting these elements. We can distinguish two cases regarding the existence of schemas: explicit and implicit schemas. If a schema is explicit, then for every document in the collection there exists a schema and this schema is defined as metadata and thus available to the IR system. There may either be a single schema covering all documents, or multiple schemas covering a subset of documents each. It is non-trivial to generate schemas automatically based on a given set of documents. (See [Chi02] for a general approach of schema generation and [XX07, sec. 3] for a proposal in the context of XML Retrieval.) Hence the existence of an explicit schema typically requires manual effort in creating it. An implicit schema, on the other hand, has not been documented as metadata. Thus the IR system has to operate solely on the instance level of documents without access to schematic information (apart from statistically gathered data, see below).

- *Homogeneity (V)*: Documents in the collections used may either be all of the same kind (*homogeneous*) or a mixture of various kinds of documents (*heterogeneous*).

[13]See [Har06] for an overview of data cleaning.

Fox example, if all documents in a collection represent scientific articles the collection is homogeneous, whereas a collection containing articles mixed with medical records and records on legal bills is heterogeneous. Homogeneity does *not* imply the existence of a common (explicit or implicit) schema: For example, a collection of scientific articles is homogeneous, but may contain articles from a different publishers (e.g. ACM, IEEE, Springer); as different publishers presumably use a different albeit similar schemas, the collection does not feature a single common schema although it is homogeneous.

- *Stability (F)*: Data used in IR scenarios typically changes over time: new documents are added to the document space and existing documents are modified or removed. We refer to this as *instability* of the document collection, and to the opposite case analogously as its *stability*. If data changes, an XML IR system has to reflect this in two ways: Firstly, it has to update its index structures, and secondly, if the system uses statistical metadata, it has to update that, too. Index updates a time-critical as otherwise the IR system will not be able to locate locate XML fragments correctly. Also, in most scenarios we assume that there exists little or no maintenance windows for the IR system. Hence the system has to conduct index update efficiently at runtime. We will formulate this as a requirement when discussing indexing in section 4.2.3; as index implementations are not in our scope, however, we do not need to consider this issue beyond that. Unlike index updates, the updates of statistical metadata are not time-critical: The IR system uses this data for vague decisions such as relevance assignments which allow for a certain slack. The restrictions regarding maintenance windows also apply here, though. Therefore the statistics repository also needs to be modified at runtime and without having a major impact on system performance; the modification can be performed over longer periods of time, however. Analogous to index updates, the implementation details of this update process are also not of concern to us, so we only formulate an according requirement. All other concepts we discuss in this thesis are independent of data modifications. For simplicity reasons we thus assume the document collections used to be stable without reducing the generality of our proposals.

- *Atomic values (V)*: Atomic values (e.g. dates, numbers) are expressed in XML by attribute values, element content, or may even be embedded in free text. Consider the examples of date values in XML shown in listing 2.5 to illustrate this. To evaluate conditions on such values, the IR system has to interpret them, that is, detect atomic values and then transform the values into some internal representation suitable for further processing. This task becomes considerably easier when metadata on the value formats used by documents in the collection is available, e.g. as part of the schema. Even easier to handle are uniformly normalised values. Thus the following questions regarding our use cases are of interest: Is the evaluation of conditions on atomic values required? If so, is metadata on relevant value formats available to the IR system? Lastly, are atomic values normalised throughout all documents?

```
<mo>JANUARY-MARCH</mo><yr>2007</yr>
<validity>01/01/2007 -- 31/03/2007</validity>
<account valid-from="2007-01-01" valid-until="2007-03-31"/>
<subject>Meeting on March 31 regarding ...</subject>
```

Listing 2.5: Example of atomic values

Query-related Aspects

- *Availability of support conditions (V)*: Support conditions address the document structure and are thus more difficult to formulate than mere keywords. Hence, we have to distinguish per use case, if an end-user is likely to provide such conditions. Orthogonal to this, we must also define, if the IR system can reasonable provide support conditions on its own.

- *Vague matching (V)*: Traditional Information Retrieval is closely related to the vague evaluation of queries formulated by the user: We consider query conditions as mere hints on what information is relevant to the user as opposed to strict constraints. Yet in our context the question arises to what *extent* vague matching of conditions is actually required. More specifically: Is it necessary to only apply vague matching to content conditions (like in traditional IR) or do we also need match target and/or support conditions vaguely? If vague matching of support conditions is required, does this only cover minor deviations like typographic errors in element names or do entirely different XML modellings of a structure defined in the query have to be matched, too? We believe, that the answers to these questions vary depending on the use case.

Presentation-related Aspects

- *Result granularity (V)*: The *granularity* of retrieval results may differ per query or be invariant for all queries in a particular scenario. Only in the former case, it makes sense for the user to provide granularity-focussed target conditions. In either case we have to ask, whether the user is likely to manually specify target elements, at all.

- *Result context (V)*: Fragment-oriented retrieval enables us to include arbitrary parts of XML documents in the result set. Depending on the scenario, the user may need to know the context from which a result item originates or not. For example, he might want to know the collection, document, author, modification date, and similar metadata per result item.

- *Result size (V)*: The user may wish to see *all* relevant fragments (optionally with the exception of filtering mechanisms, see below) or only a fixed maximum number of best matches. The latter is commonly known as *Top k approach*.

- *Clustering (V)*: We can list result fragments linearly in order of their relevance or cluster them per document. In the latter case, we can either rank results within a document or display them in document order. Optionally, we can also display result abstractions like a relevance distribution within a document (a so-called *heat map*), for example. We expect all of these choices to vary between different scenarios.
- *Filtering (V)*: Retrieval results may contain fragments which overlap, that is, which contain one another. This may not be a problem (or even desired) in some scenarios, whereas we need to avoid it in others. Also the user may wish results to be compacted in certain cases, for example by returning a parent element in lieu of many individual, relevant children.

2.3.2. Use Cases

In the following we define several use cases for XML Retrieval. Each use case corresponds to one particular configuration of the variable aspects which we have discussed in the preceding section. Table 2.3 shows the template we use to summarise this configuration per use case.

Table 2.3.: Use case configuration template

Schemas	Count Explicit/Implicit Homogeneous/Heterogeneous	e.g. 1 e.g. explicit e.g. heterogeneous
Atomic values	Evaluation required? Metadata on formats? Normalised?	...
Support elements	By end-user? By IR system?	
Vague matching	Support elements? Target elements? Semantic relativism?	
Result granularity	Fixed/Variable User/System/Invariant Granularity only?	
Context needed	Yes/No	
Result size	All/Top k	
Clustering	Linearly/Per document? Ranked within document? Heat map?	
Filtering	Overlap Removal Compaction	

Use Case: Book Search

A common use case in XML Retrieval (as well as traditional IR) is the search for books in either an online store or library: A user is looking for books on a particular topic. As he is considering to purchase a book, but only has a limited budget available, he is interested in identifying a single book that is very exhaustive regarding his topic of interest and at the same time as cheap as possible. This book does not necessarily have to be restricted to his topic of interest, but in order to maximise his personal efficiency in learning about the topic he would prefer a book covering little other fields (i.e. a highly specific book). He does not prefer any particular author or publisher. In order to not get tempted to buy a very expensive book, he wishes to constrain the search to books below a certain price limit. If, however, there is a book at a higher price which is far better suited than all other books available, he would like to see this book as well.

Our user is searching the website of a single online book store. Thus we can assume the underlying collection to conform to a single schema[14] describing the store's internal format which is explicitly available. As the store is only selling books, the collection is homogeneous. The user is primarily using keywords to formulate his query. Due to his restrictions on book prices, however, he also formulates support conditions which include atomic values. Prices are coded as an XML element in a normalised and clearly defined manner, as shown in listing 2.6. Thus evaluation of atomic values is only needed for normalised values not embedded in any textual content.

```
<book>
  <price currency="eur">47.11</price>
  ...
</book>
```

Listing 2.6: Sample XML fragment representing a book entity

As he is entirely unaware of the store's XML schema, the user enters his condition on the price in a GUI form field; therefore the resulting query condition is actually formulated by the IR system taking into account the document structure precisely and correctly. Hence structural conditions on the support elements are provided, but do not require taking into account semantic relativism. According to the weak nature of this condition (which does not apply for very good books), vague matching still has to be performed. The primary result elements desired by the user are books. Yet he is interested in judging their exhaustivity and specificity and also in seeing those parts of the books most relevant to him. Thus there are no target elements provided by the user, but the system is configured to locate chapters, sections, and subsections and display them clustered by book, ranked according to relevance within books. In this view, overlapping book parts are rather not desired by the user. The result context (i.e. the book including author, price, etc.) is obviously needed. Vague matching of target elements is not really needed, however, as the system has explicitly listed elements at

[14]In this scenario having a small number of schemas (i.e. more than one) is actually more likely. As we cover this case in subsequent uses cases, however, we assume a single schema to increase use case diversity.

a suitable granularity (i.e. paragraphs would be to fine-grained, entire books obviously not useful).

The result display can be expected to support paging (i.e. distributing the result list across several pages) and the user is unlikely to look beyond the first couple of pages. To constrain the book prices according to his budget he may adjust the price condition. As within this price range he is looking for the best-suited book, we do not expect him to re-order the result list. Thus a Top k processing approach would be sufficient. The resulting query (generated by the IR system based on the user's input) could look like this:

```
cas retrieval
  support://books/book@price[<50]
  ( target:chpt || target:sec || target:sec2 )
```

The configuration of the use case aspects we have identified is summarised in table 2.4.

Table 2.4.: Configuration of the Book Search use case

Schemas	Count	1
	Explicit/Implicit	Explicit
	Homogeneous/Heterogeneous	Homogeneous
Atomic values	Evaluation required?	Elements only
	Metadata on formats?	Yes
	Normalised?	Yes
Support elements	By end-user?	Yes (price)
	By IR system?	No
Vague matching	Support elements?	Yes
	Target elements?	No
	Semantic relativism?	No
Result granularity	Fixed/Variable	Fixed
	User/System/Invariant	Invariant
	Granularity only?	Yes
Context needed	Yes/No	Yes
Result size	All/Top k	Top k
Clustering	Linearly/Per document?	Per book
	Ranked within document?	Yes
	Heat map?	No
Filtering	Overlap Removal	Yes
	Compaction	Yes

Use Case: Re-finding

Another interesting use case for XML Retrieval is the endeavour of a user to locate a particular piece of information *again* which he has come across before ("re-finding").

This is similar to a use case Trotman et al. have proposed in [TPL06]. In our case, however, the emphasis lies more on user-provided support conditions rather than on content conditions. Let us assume that the user has recently come across a paper on how to implement vague structural matching by using path edit distances which he would now like to read again in detail. Unfortunately though, he does not remember the author, title, or any other strongly discriminating information. He does, however, recall that there was an interesting section containing the word "path" in its title whose content included at least one word printed in bold font.

As the user does not recall the publisher, he uses an imaginary XML-based search engine for scientific papers from various sources which allows to specify structural conditions. The underlying collection is homogeneous (i.e. only contains scientific papers), but missing a common schema. Processing atomic values is not required, as the user cannot recall any such information. The user is able to specify conditions on both content and structure, but is unsure of the actual structure (i.e. element names etc.) used in the collection; thus all conditions have to be processed very vaguely and with regard to semantic relativism. His query might look like this:

```
path edit distance
  support://sec//title[path]
  support://sec//b
  target:sec
```

The user expresses both granularity and type of desired result elements in his query by including the target condition. We need to be interpreted the target condition vaguely, too, as the passage the user remembers might also have been a subsection or chapter, although the user assumes it not to be. The result display should list relevant sections ranked by relevance. Because we expect many partially relevant matches and as the user will have to go through the matches manually to identify the one he was looking for, he would only want to see a reasonable amount of results (i.e. Top k approach). The results should be ranked decreasingly according to relevance and for each match the context (document, title, author, publisher) must be displayed: The user looks through the result fragment to identify the fragment which looks like the one whose structure he remembered; once he finds the correct one (i.e. the one he was actually looking for), he needs the context information to cite the source. We have to allow for overlap in results and disable compaction as the user has to identify the right match based on visual characteristics and thus needs to come across the exact same fragment which he is looking for in the result set. Table 2.5 summarises this use case's configuration.

Use Case: Corporate Intranet Search

This use case is concerned with locating relevant information in a confined yet heterogeneous environment: a corporate intranet. Similar scenarios are the the BioMedNet example given in [TPL06, p. 281], where jobs, store items, articles, and other entities can be searched, and even the field of personal document management (cf. [FHM05]), provided each of the individual data sources can be mapped to an XML structure. In

Table 2.5.: Configuration of the Re-finding use case

Schemas	Count	Many
	Explicit/Implicit	Explicit
	Homogeneous/Heterogeneous	Homogeneous
Atomic values	Evaluation required?	No
	Metadata on formats?	n/a
	Normalised?	n/a
Support elements	By end-user?	Yes
	By IR system?	No
Vague matching	Support elements?	Yes
	Target elements?	Yes
	Semantic relativism?	Yes
Result granularity	Fixed/Variable	Fixed
	User/System/Invariant	User
	Granularity only?	No
Context needed	Yes/No	Yes
Result size	All/Top k	Top k
Clustering	Linearly/Per document?	Linearly
	Ranked within document?	n/a
	Heat map?	n/a
Filtering	Overlap Removal	Yes
	Compaction	No

our setting, the user is an employee of a larger company who has the task of establishing a maintenance project. This project's goal is to make some adaptions to an information system developed in a previous project several years back. The user first wants to contact the members of the original project for assistance. He is unaware, however, of how the original project was organised and thus unsure whom he should contact for information. Therefore the user searches the corporate intranet for information on roles and role assignments in that project; he formulates the following query:

```
project XYZ role assignment tgt:role
```

The result list may include parts of organisational and technical documentation, contracts, company news bulletins, and other items. Within these pieces of information it should contain all existing roles and their respective assignments. Overlap is clearly undesired; ideally, each role should even only be listed once. Listing 2.7 illustrates a sample result fragment; the exact structure of the results and the naming of relevant elements may vary widely, however. We assume that all concerned data sources have been mapped to (different) XML schemas and can be queried by a central IR system. Table 2.6 summarises the configuration.

```
<role name="Requirements Engineer">
  <assignments>
    Peter Anderson, John Smith
  </assignments>
</role>
```

Listing 2.7: Sample result fragment for the Corporate Intranet Search use case

Use Case: XML Web Search

Another scenario where XML Retrieval is of interest is XML web search. Today, searching the world wide web for information is one of the most common Information Retrieval applications. However, today's web search is document-centred with little regard to document structure. In particular, a user is unable to define conditions involving the document structure and there is hardly any use of implicit structural conditions, either. Given the increasing popularity of XML – especially in web-based contexts –, XML documents may replace current document formats in the long run. In this case, Information Retrieval focussing on both the content and structure of XML documents will enter the focus of web search applications, thus forming the field of XML web search.

As an actual example, consider a user who uses his favourite (XML-enabled) web search engine to look up information on cat food. He is particularly interested in finding scientific articles on how to feed cats, as opposed to news items and advertisements on the topic. To ensure that the results have a certain scientific credibility, he only wishes to retrieve results whose author holds at least a PhD. Hence he formulates the following query:

```
cat food target:article support://article//author/title=PhD
```

Table 2.6.: Configuration of the Corporate Intranet Search use case

Schemas	Count	≈ 10
	Explicit/Implicit	Explicit
	Homogeneous/Heterogeneous	Heterogeneous
Atomic values	Evaluation required?	No
	Metadata on formats?	n/a
	Normalised?	n/a
Support elements	By end-user?	No
	By IR system?	Yes
Vague matching	Support elements?	Yes
	Target elements?	Yes
	Semantic relativism?	Yes
Result granularity	Fixed/Variable	Fixed
	User/System/Invariant	User
	Granularity only?	No
Context needed	Yes/No	No
Result size	All/Top k	All
Clustering	Linearly/Per document?	Linearly
	Ranked within document?	n/a
	Heat map?	n/a
Filtering	Overlap Removal	No
	Compaction	Yes

Of course authors holding equivalent degrees such as "Dr.", "MD", and so on as well as abbreviated and non-abbreviated notations should match the query. Also, web search involves many different sources so that the structure of matching XML fragments may vary widely. Results should not overlap and mainly contain elements with the granularity of an article. We list the details of this use case in table 2.7.

Table 2.7.: Configuration of the XML Web Search use case

Schemas	Count	Many
	Explicit/Implicit	Implicit
	Homogeneous/Heterogeneous	Heterogeneous
Atomic values	Evaluation required?	Only string values
	Metadata on formats?	No
	Normalised?	No
Support elements	By end-user?	Yes
	By IR system?	No
Vague matching	Support elements?	Yes
	Target elements?	Yes
	Semantic relativism?	Yes
Result granularity	Fixed/Variable	Fixed
	User/System/Invariant	User
	Granularity only?	No
Context needed	Yes/No	No
Result size	All/Top k	Top k
Clustering	Linearly/Per document?	Linearly
	Ranked within document?	n/a
	Heat map?	n/a
Filtering	Overlap Removal	No
	Compaction	Yes

2.4. Summary

In this chapter we have laid the foundations for discussing XML Retrieval. We have defined *Information Retrieval* as the process of informing a user on the existence (or non-existence) and whereabouts of documents and/or document parts which to some degree satisfy an information need the user has expressed. In the course of this process, the user formulates a query which we regard as a system of explicit (i.e. user-provided) and implicit (i.e. system-provided) conditions and interpret vaguely. The IR system evaluates this query and tries to approximate the *relevance* (the degree to which a document satisfies an information need) by assigning *scores*.

XML is a popular document format and widely used today. We have defined an *XML document* as an ordered, labelled tree whose nodes are XML *elements*. Elements have a non-unique tag name and optionally contain textual content. We can use key/value pairs called *attributes* to further describe elements and to *reference* other elements. *Namespaces* serve to provide a common vocabulary to XML documents and *schemas* enable us to define rules for their composition. A document adhering to the nesting rules of XML is *well-formed*, whereas a document conforming to a schema is *valid*. The various directed relations between elements, attributes, and namespaces we refer to as *axes*; important axes include the parent, child, ancestor, and descendant relation.

Information Retrieval over XML documents is called *XML Retrieval*. It provides additional potential by allowing us to generically and explicitly express the logical document structure and by enabling fragment-oriented retrieval. A *fragment* is an arbitrary subtree of an XML document tree; thus when using fragments to compose the result set we have to handle redundancies due the nesting structure of XML which is commonly known as the *overlap* problem. Of particular interest to us is *Content-and-Structure Retrieval* (CAS) which distinguishes tree types of conditions (keyword, support, and target conditions). To provide a conceptional framework for this kind of retrieval we have devised the XML Retrieval process. It consists of several phases, each of which we can refine into distinct activities.

Because XML Retrieval is applicable to a broad range of applications, we have finally introduced the concept of scenarios. A *scenario* is one specific configuration of a set of environmental properties of an IR system. We have explored the space of these properties by discussing document-related, query-related, and presentation-related aspects of XML Retrieval. Based on this we have then defined several use cases to estimate which configurations are likely to occur in practical applications. These use cases consequently provide a basis for deciding which assumptions we can validly make regarding XML Retrieval and where configurability is required.

Part II.

The XML Retrieval Process

3. Query Specification and Weighting

3.1. Criteria for Query Languages

Numerous query languages have been proposed to search XML documents from a data-oriented as well as a document-oriented point of view. In order to identify (or define, if necessary) a language suitable for our purposes, we will first state our general query language requirements in the following. We then introduce two classifications of XML query languages to narrow down the set of candidates: First we classify languages by the kind of user who is formulating the queries. This aims at determining the relevant scenarios in which we need the query language and the requirements each scenario imposes on such a language. We then provide a second classification proposed in [AYL06] which aims at grouping concrete query languages according to their respective expressiveness. Based on our requirements and classifications we then evaluate potentially suitable appropriate query languages in the next section.

3.1.1. Requirements

We now define requirements which a query language has to meet in order to suit our needs. Please note that we consider the processing semantics to be independent of the query language; hence these requirements solely define the expressive power of a language regarding an information need, but not how a query is processed. To state our requirements, we use the following simple wording convention: The term "should" expresses that a functionality is desirable, but not mandatory; the term "must", on the other hand, denotes mandatory functionalities.

- The query language should enable us to easily formulate document-oriented, imprecise information needs.
- It should enable us to express "traditional" IR queries, that is, queries only consisting of keywords.
- It must enable us to optionally provide an arbitrary number of target conditions (including zero).
- It must enable us provide an arbitrary number of support elements per target element.
- It must enable us to provide arbitrary support elements *below* a target element; for example, the target element may be a section with a support condition that this section contains a `<b>` element indicating bold font.

- It must enable us to express paths.
- It must enable us to express keyword conditions in the context of one particular path component of a support condition.
- It should enable us to optionally assign an individual weight to each condition. The weight indicates how important the condition is to the user[1].

Throughout this thesis we concentrate on the conceptional view of XML Retrieval, not a practical implementation. This enables us to make certain assumptions which do not impose restrictions on the generality of our concepts, although they would impede a user in a real-world setting. The aim of these assumptions is to avoid unnecessary complexity. Namely, we assume that query conditions are always atomic and that they are related by the following implicit operators, only: Keyword and support conditions are treated vaguely conjunctive (that is, a result item *should* match all conditions, but does not *have to* match any), target conditions are treated disjunctive. We consider target conditions separately, as unlike other conditions a result item can typically only match at most one target condition. (We will discuss this in more detail in section 5.1.) To formulate disjunctive keyword or support conditions (e.g. “cats OR dogs”), two independent queries are thus needed. A third assumption is due to the limited scope of this thesis and therefore does impose a conceptional restriction: We will not take into account atomic values like numbers, dates, etc. For example, in the XML fragment `<book price="42.00" currency="EUR"/>` we ignore the semantics of the attribute value (i.e. that the book has a price of 42.00 Euros). Thus we cannot handle information needs like that for books in a certain price range (as in the Book Search use case, for example).

Based on these assumptions, we can now explicitly admit certain relaxations of our query language requirements. In this context, the expression “does not have to” denotes that a functionality may be provided by a query language, but is not needed. Namely, we admit the following relaxations:

- The query language does not have to enable us to formulate non-atomic conditions (i.e. conditions which consist of a system of conditions themselves).
- It does not have to enable us to use explicit operators (like `and`, `or`, `not`, $<$, $>$, and so on).
- It does not have to enable us to explicitly specify the kind of axis separating two path components (i.e. child, descendant, attribute, reference).

These relaxations do not impede the concepts we propose for our XML Retrieval framework. For the general discussion of XML Retrieval issues, having a query language including these features would be helpful, however. We therefore strive to identify *two* query languages: one featuring the minimal set of requirements needed for our XML Retrieval framework, and another to enable a precise discussion of XML Retrieval in general. For the latter the above relaxations consequently do not apply.

[1]User-defined weights are orthogonal to system-defined ones. If both user- and system-defined weights are assigned, we evaluate the user-provided weights relative to the system-defined ones.

3.1.2. Classification by User

We differentiate three classes of query languages which differ in terms of their requirements:

1. End-user query languages
2. Expert query languages
3. System query languages

An *end-user query language* is the means by which the end-user of an IR system expresses his information need. The end-user is typically not an IR expert and has little (if any) knowledge of both the IR system and the document collection. He is unlikely to know, for example, the XML schema of the collection or details of the query processing. Thus an end-user query language should be centred around expressing information needs and not around giving query processing instructions. All information provided to the IR system in this query language should be treated as possibly incomplete and erroneous and thus processed vaguely (i.e. the system should not rely on any information provided by the end-user, but rather regard it as hint on what information he desires). Ideally, an end-user language is not a formal query language in a narrow sense, but consists in a use case-specific GUI or natural language processor. We expect queries formulated by the end-user to be translated into a more system-oriented language before actually evaluating them. This translation is far from being trivial. Nonetheless, we believe that it can be performed independently of the actual IR processor (i.e. by the IR front-end) and that the end-user language is thus transparent to us. Therefore neither the the design of a suitable end-user query language nor its translation to the language used by the IR processor are primary aims of this thesis and are thus not considered further.

An *expert query language* is used to express an information need and additionally provide processing instructions like structural metadata, condition weights, etc. It should cover all conceptional features deemed necessary for the specific IR system, yet it should be simple enough to allow IR experts to efficiently formulate queries. (By efficient we mean that the expert should be able to easily formulate queries which have the intended semantics and are free of syntactical errors; cf. [OT03]). Ideally, the IR system is able to operate on queries formulated in this language without further translation (to reduce complexity and potentially increase efficiency); this is not mandatory, however. Generally we should treat expert language queries vaguely as well. If the IR system supports this, however, the query may contain detailed information on the reliability of each condition and the level of strictness that is desired.

Finally, the *system query language* is the language used internally by the IR processor when handling user queries. It typically features a non-textual representation like an operator tree. This is needed, for example, for query rewriting and relevance feedback mechanisms. Also the system potentially enriches the query with additional information aimed at increasing processing efficiency (e.g. "Ignore elements of type X as they never contain relevant content"). Compared to expressions formulated in the expert query language, we expect system query conditions to have a higher degree of strictness, as the system "knowns" the exact document structure and thus is likely to provide reliable information. As stated above, the system query language should ideally be identical to the expert user language, but does not have to be. At least it must feature a textual representation which can be read by an expert user to enable debugging.

3.1.3. Classification by Expressiveness

Amer-Yahia et al. [AYL06] propose a classification scheme for XML query languages which is based on expressiveness. They distinguish the following four classes based on which we narrow down the set of query languages feasible for our work:

- *Keyword-Only Queries Languages* originate in non-XML contexts (i.e. flat Information Retrieval) and only allow to list a number of keywords. Optionally these keywords can be related by operators and structured by means of nesting. Operators can include both common logical operators like `and`, `or`, `not`, etc. and complex operators like `has-sibling`. Nesting is typically achieved by using brackets. A very simple query expressed in a keyword-only language is:

  ```
  animals cats dogs
  ```

  A more enhanced one could look like this:

  ```
  animals AND (cats has-sibling dogs)
  ```

  Representatives of such languages include XRank [GSBS03] and XKSearch [XP05], both featuring pure keyword search, and the unnamed query language proposed in [AVF06], featuring the use of complex operators to define structural conditions. In a keyword-only language, means of formulating structural hints are naturally either very limited (e.g. keywords can be used to match tag names as well as element content) or very complex (e.g. deep nesting of complex operators). Hence they are not feasible when focussing on CAS-based retrieval.

- *Tag and Keyword Query Languages* enhance the former class by allowing to add *tags* to the keywords contained in the query. A tag provides additional metadata on the meaning of a keyword. They can be used, for example, to specify a certain confidence value or a hint that the keyword has to appear in a tag name like in the following example:

  ```
  animals AND tagname:pet AND (attrvalue:cat OR attrvalue:dog)
  ```

  The XSEearch[2] language [CMKS03] is one example of such a language: It is a very simple, end-user-centric query language that allows for the use of so-called labels to express structural constraints. Queries in such languages are comparatively easy to formulate and process. Also we can easily extend a tag and keyword language to meet additional requirements by adding new tags. On the downside, however, they lack the ability to (easily) specify path information just like keyword-only languages. Therefore this class of languages is feasible, if we want to formulate

[2]Not to be confused with the XKSearch language [XP05] mentioned above.

structural query conditions on a very high abstraction level. For the kind of structural conditions we want to express, this would make query formulation rather cumbersome and hard to understand, though; we will therefore refrain from using such languages.

- *Path and Keyword Query Languages* address this issue by allowing for path expressions in addition to keywords. This definitely makes formulating queries more complex for simple queries (i.e. queries with mostly keywords and little structure), but provides powerful means for expressing structural constraints. For example, the following query belongs to this class of languages:

  ```
  animals AND /book//chapter/title[cats]
  ```

  Many query languages pursuing the idea of CAS-based retrieval proposed in recent years belong to this class of languages. Early examples include ApproXQL [Sch01], EquiX [CKK+02], and XIRQL [FG01]; more recent approaches are often extensions of the NEXI language [TS04a], such as XOR [GHT06]. As discussed in the previous section, we do not aim at defining an end-user language; thus the level of complexity of formulating path and keyword language queries seems adequate in our context. Also these languages feature a sufficient expressive power. Hence we consider path and keyword languages to be a feasible choice; we thus compare existing languages of this category in detail in section 3.2.

- *XQuery and Keyword Query Languages* enable XQuery-like querying capabilities. In particular these languages allow for constructs like defining variables, ordering, joins etc. as well as very specific structural conditions as shown in the following sample query:

  ```
  let $col := doc('books.xml')
      for $chapter in $col//book//chapter,
      $title in $col//book//chapter/title
      where contains($title, 'Cats')
      order by $title
      return $chapter
  ```

  Apart from XQuery [BCF+07] itself, its predecessors (e.g. Quilt [CRF00]) and recently proposed extensions for textual search such as XQuery-FT [AYBB+07], belong to this category. As these languages have been developed with data-centric applications in mind, most of the additional features listed above have arguably little or no use in Information Retrieval scenarios. On the contrary: Due their focus on fine-grained, complex query conditions the additional expressive power makes formulating a query a as set of vague hints (as opposed to an exact specification of where to look and what to retrieve) rather difficult compared to languages of the former classes. Thus these languages provide little additional benefit at high additional costs and are thus not feasible for our purposes.

- *Query-by-Example Languages*: Another kind of XML query languages which do not fit into the classes discussed above are those following the query-by-example paradigm: The user provides input that resembles the result items he is interested in. In the XML IR context this means that the user actually provides complete XML fragments as the following sample query demonstrates:

```
<chapter><title>animals</title>cats dogs</chapter>
```

Although this enables the user to formulate queries in a rather intuitive manner similar to keyword-only languages, the user is able to provide a fair amount of structural information which resembles that of path and keyword languages in expressiveness. Optionally the language may also permit embedding operators to express, for example, boolean conditions or confidence values. An example of such a language are so-called XML fragments[3] proposed by Carmel et al. [CMM$^+$03]. We consider this as an interesting option for an end-user language, but deem it impractical for expert or even system usages due to its lengthy and imprecise nature. We therefore do not consider this class of languages as a candidate to suit our needs.

3.2. Existing Query Languages for CAS Retrieval

The two classifications of query languages we have introduced enable us to greatly narrow down the set of existing query languages: We foremost need an expert language to express queries featuring the various aspects discussed throughout this thesis (like support conditions, weighting, and so forth). To support implicit query conditions, the language should be a feasible system language as well. End-user aspects, on the other hand, are clearly secondary in our context. Regarding the expressiveness, path and keyword languages are most appropriate for us (based on our discussion in the previous section). We will now evaluate existing query languages which belong both classes based on the general requirements we have defined. Please note that these languages are only a sample of what XML Retrieval publications have proposed so far; we consider them representative, however, in terms of their properties.

ApproXQL [Sch01] is a hybrid of tag- and path-based approaches. It allows to specify path conditions (albeit in a somewhat cumbersome fashion), but is explicitly aimed at data-centric documents and thus not well-suited for keyword matching. It also misses various features like support for descendant relations, weighting, and other features, so it is not suitable for us. Cohen et al. propose EquiX [CKK$^+$02] which uses tree-structured conditions generated by a GUI interface. Unfortunately this language seems to be completely missing a textual representation of queries (GUI input is directly translated into an internal tree model), so it is also infeasible. XIRQL [FG01] on the other hand, in spite of also being a very early proposal of a CAS-based retrieval language, fulfils most of our basic requirements: Besides allowing for the specification of support and target elements in a straight-forward manner, it supports explicit weighting of conditions and both the child and descendent axis. Not supported, however, are confidences (which could easily be added to the model, though) and references. Also,

[3]Not to be confused with our notion of a “fragment”.

specifying several target elements which share the same support conditions requires lengthy queries containing many redundancies. For an example of the latter case, see the query shown in the Book Search use case in section 2.3.2. Therefore we consider this language generally feasible (with minor extensions), but not ideal.

A more recent query language is NEXI (for Narrowed Extended XPath I) [TS04a] which has been the official query language adopted by INEX since 2004 [TS04b]. As its name suggests, NEXI is an adaption of XPath 1.0 to the requirements of INEX; the adaption mostly consists in restricting the expressiveness of the language for the sake of simplicity and only few extensions (foremost, an `about()` function used for vague content matching). Unfortunately though, several features present in either XPath or XIRQL which are important to our settings have not been integrated (or even explicitly removed); these include the differentiation of the child and descendant axis and weighting of conditions. Apart from these aspects, generally the same statements apply as for the XIRQL language discussed above. Therefore we deem NEXI less well-suited for our purposes than XIRQL, albeit being an interesting CAS language for more restricted contexts.

Geva et al. [GHT06] propose XOR as one extension of the NEXI language. It supports a tagging mechanism in the form `{key:value}` which can be used to specify various additional metadata; for example, for each condition it can be configured, if matching is to take place strictly or vaguely. Also several tags concerning natural language processing are supported, e.g. for expressing that a particular keyword is a noun or that a word is written in all upper-case letters. XOR also extends the set of operators provided by NEXI: Besides `AND` and `OR` it introduces `NOT` as well as comparison operators like `lt()` and `gt()` to process atomic values. For path matching, XOR allows for wild cards (e.g. `//book*`) and features a syntax to explicitly express references, both outgoing and incoming (`LinkTo()`, `LinkFrom()`). An appealing advantage of XOR is the easy extensibility provided by its tagging mechanism. Also, being able to handle conditions on atomic values and references are strong points, although the syntax used to express them is somewhat lengthy and cumbersome. The other features, like the wild card matching and natural language features, are unnecessary for our purposes; we can, however, simply disregard them, so they do not constrain us in using the language. Unfortunately, the criticism of XIRQL is applicable to XOR, too, to a great extent. Thus we will not it use directly, but consider it as a good basis for defining an individual language.

3.3. Query Language Definition

None of the query language proposals discussed above fully satisfy our minimal requirements. To avoid overhead, we do not extend any of these languages, but instead define a custom query language. Following the differentiation of requirements in section 3.1 (minimal vs. extended requirements), we actually define two query languages: CAS-QL (for Content-and-Structure Query Language) and CAS-QLX (for CAS-QL eXtended). The former one reflects the conceptional power of our retrieval framework, the latter one enables the precise discussion of XML Retrieval issues in general. Please note, though, that neither of these two languages is by any means intended to be “yet another query language” for wider use or generally suitable for IR system implementations. In the following, we describe the semantics of CAS-QL and CAS-QLX textually. The according grammars we have included in appendix C. CAS-QL we define as follows:

- A *term* is a character string consisting of one or more digits, lower case letters, and hyphens (-). We generally ignore case, so a lower case letter in a condition equally matches the corresponding lower and upper case letter in a document.
- A keyword condition consist of exactly one term. For example, the query `cas retrieval` contains two keyword conditions.
- A support condition consists of one or more terms separated by a slash (/) as generic axis delimiter; hence we do not distinguish specific axes. See section 4.2.3 for a detailed discussion on this. Support conditions are prefixed with “support:” or “sub:”. For example: `sup:/book/chapter/section`.
- A target condition consists of exactly one term and is prefixed with “target:” or “tgt:”. This is based on the assumption that a particular element always has the same semantics[4] (and thus contains the same kind of content) regardless of where it is used. For choosing target elements it is therefore sufficient to judge based on properties of an element (such as its tag name). We can still model all restrictions regarding the context of elements (e.g. that a `<paragraph>` inside an `<article>` is considered more relevant than one inside a `<novel>`) by using support conditions, i.e. `tgt:paragraph sup:/article/paragraph`. Therefore this definition of target conditions does not restrict the generality of our approach. It greatly simplifies target condition handling, however.
- Alternatively, a target condition may consist of a single decimal number in the set $\{0.0, 0.1, \ldots, 1.0\}$; for example: `tgt:0.2`. In this case, it merely declares a particular result granularity as the desired granularity. Will we explain this in detail in chapter 5.
- We ignore namespaces in all conditions referring to elements or attributes. Like our decision for pathless target conditions, this is due to our assumptions regarding the semantics of names (cf. section 4.2.3).
- Conditions are related by the following *implicit* operators, only, as discussed in the according requirement in section 3.1.1: Keyword and support conditions are interpreted vaguely conjunctive, target conditions vaguely disjunctive. For example, in the query

  ```
  cas retrieval tgt:section tgt:paragraph sup:/article/mainmatter
  ```

[4]We discuss this in depth in the context of name matching in section 4.2.3.

the user prefers result items to be `<section>` or `<paragraph>` elements, and wishes for them to appear in the main matter of an article which contains the keywords “cas” and “retrieval”.

- For each condition the user can optionally define a weight from the set $\{0.1, 0.2, \ldots, 1.0\}$. This is appended to the condition as shown in the following example:

  ```
  cas,weight=0.8 retrieval sup:/article/mainmatter,weight=0.3
  ```

  The default weight is 0.5. In the example, the keyword “CAS” has a weight of 0.8, “retrieval” the default weight of 0.5, and the support condition a weight of 0.3. The weights do not have to sum up to 1.0; instead the system evaluates each weight relative to sum of all weights. In our example, this sum is $\Sigma = 0.8+0.5+0.3 = 1.6$, thus the actual weights are 0.5, 0.31, and 0.19, respectively.

In addition to this we define the following for CAS-QLX:

- Conditions can be related by the explicit operators `and` (denoted by `&&`) and `or` (denoted by `||`). The implicit default operator is always the `and` operator, unlike in CAS-QL. Both operators are treated vaguely. Also, conditions can be grouped by using round brackets (`(`, `)`). Grouped conditions are treated as a single atomic condition regarding the surrounding operators. Consider the following sample to illustrate this:

  ```
  (cas (retrieval || search)) && (tgt:section || tgt:paragraph)
  ```

  This means that the user is looking for either a section or a paragraph on “cas retrieval” or “cas search”.

- Path conditions may use the following specific axis indicators: child (`/`), descendant (`//`), attribute (`@`), and reference (`->`). For example:

  ```
  sup:/books//book->author@address
  ```

  This means that the user is looking for `<book>` elements below a `<books>` element which references an `<author>` element that has an `address` attribute.

- Each path component may be suffixed with a predicate, i.e. a set of implicitly conjunctive, vague conditions regarding this path component. The conditions may be prefixed with an operator to test atomic values (namely `=`, `<`, `>`, `<=`, `>=`); an unprefixed condition is treated as keyword. Consider the following query examples:

  1. `sup:/book[information retrieval]/chapter[cas retrieval]`
  2. `sup:/book@price[>20 <50] sup:/book@author[smith] cas retrieval`

  The first example asks for books about Information Retrieval which contain a chapter on CAS Retrieval. The second example asks for books on CAS Retrieval which belong to a particular price range and were written by a particular author.

3.4. Element Type Classification

Before we discuss weighting of query conditions in the next section, we now introduce the concept of element types which we will use throughout the remainder of this thesis. It is mainly aimed at target condition handling (cf. section 5.1.1), but is also of importance in other contexts such as weighting strategies. The concept has been inspired by a publication of Lehtonen et al. [LPT06, sec. 2A]. We define an *element type* as a group of elements with common properties. Namely, we distinguish three such groups of elements based on the semantics of tag names:

- *Formatting Elements* (e.g. `<i>`, `<b>`, `<emph>`, `<large>`, `<font size="1">`): They are used to signal that the content they contain is to be formatted in a particular way when processing the document for display. Formatting elements' tag names are entirely unrelated to their content but frequently they can be interpreted as an indicator of the importance of contained terms. For example, we can reasonably assume word intended to be printed in bold letters and a large font size to be an important term, whereas for a word set in very small letters the opposite assumption usually holds.

- *Structuring Elements* (e.g. `<chapter>`, `<section>`, `<paragraph>`): These elements are used to describe a hierarchical structure of a document's contents. Tag names of structuring elements have a weak relation to their contents such that they either contain further structuring elements or a certain quantity of textual content (e.g. very little content for a heading element like `h1` or comparatively much for non-headings). Yet in the latter case the tag names do not hint towards the meaning of this text (for example, a paragraph may contain text on *any* topic). They are useful for estimating the specificity of their contents, though, as well as for selecting a suitable level of granularity for retrieval results.

- *Entity Elements* (e.g. `<author>`, `<city>`, `<dob>`, `<abstract>`): Elements of this kind usually correspond to a real-world entity and there often exists a relation between the tag name and the semantics of their contents. For example, an `<author>` element can be expected to describe a person and `<dob>` a date of birth.

Element type assignments are not necessarily exclusive. The tag `<mainmatter>`, for example, constitutes a part of a document hierarchy, yet at the same time is also an entity with implications about its content: If it is used as a target element, it both defines a desired granularity (i.e. a large amount of text) and the kind of content desired (i.e. no abstracts or summaries, but full content). Therefore we propose, that every element must be assigned to *at least* one of the above categories, but may be assigned to arbitrarily many ones. For simplicity, we assume that the assignment of element types is done manually. This suffices for our purposes, as we regard element types as an auxiliary construct to aid reasoning about the feasibility of improvements we propose. To actually use element type distinctions in an IR system implementation, automatic assignments (e.g. by using heuristics) are mandatory.

3.5. Weighting of Conditions

In a strict query evaluation environment (e.g. a database system) handling queries consisting of multiple conditions is easy: Each condition is either satisfied by a result item or not; if conditions are combined conjunctively, all conditions have to be satisfied, if they are combined disjunctively, at least one condition has to be satisfied. Performing vague query evaluation, on the other hand, is more challenging in this regard for two reasons: Firstly, a condition may be satisfied *to some extent* only, and secondly, even if a condition is not fulfilled by some result item, this item may well still be relevant to the user. In Information Retrieval we perform scoring (as opposed to binary classification); hence we must define how strongly satisfying a particular condition contributes to the overall score, i.e. how "important" that condition is. Defining this importance of a query condition relative to all other conditions is commonly referred to as *weighting.* More formally, for an arbitrary set of conditions $\mathcal{Q}$ there exists a weighting function $\mathrm{w} : \mathcal{Q} \longrightarrow (0, 1]$ for which it holds that

$$\sum_{q \in \mathcal{Q}} \mathrm{w}(q) = 1 \tag{3.1}$$

The specification of this weighting function has a great impact on the score of result items and thus the effectiveness of an IR system. We therefore discuss different weighting strategies for XML IR in the following sections.

3.5.1. Classification of Weighting Strategies

We distinguish different classes of strategies for generating condition weights. These classes are not exclusive, however, thus a weighting strategy can belong to more than one class. We use the following conventions for naming the classes: The term "static" implies that the IR system assigns weights based on configuration time rules, whereas "dynamic" weights are computed at runtime by using information on the documents. Orthogonal to this, we say that a weight is "fixed", if the system computes it once per query, and "adaptive", if the system re-weights conditions during query evaluation. Using these terms, we differentiate the following four classes of weighting strategies:

1. *User-defined Weighting* The user provides weights as part of his query. He may either formulate weights directly as part of a textual query string or in a more abstract manner, e.g. by GUI forms or natural language expressions. User-defined weights are a common feature of IR systems in general; for XML Retrieval it has been proposed by Fuhr et al. [FG00], for example. A disadvantage of this way of obtaining weighting information is that – in most IR applications – end-users will either not be able to assign useful weights due to a lack of skills, or simply not bother to do so. Hence, while being an appealing solution in theory, its practical feasibility is likely restricted to special scenarios.

2. *Fixed Static Weighting* Alternatively, we can configure an IR system to use statically predefined weights based on some classification of query conditions. For

example, all structural hints may be assigned a particular score or all keywords which are nouns. This only requires the system to analyse the query string and is thus very cost-effective. Many XML Retrieval publications apply this kind of weighting; examples are [AYLP04] and [PMM07].

3. *Fixed Dynamic Weighting* A more advanced mechanism is the assignment of weights based on an analysis of the document collections used. For example, we can adapt term frequency models like the common TF-IDF-based weighting schemes to XML to derive weights. This is proposed, for example, in [LRM05]. Unlike the former classes of weighting strategies, this not only requires query analysis, however, but also accessing documents (or at least document statistics) for weight-generation and is thus more costly.

4. *Adaptive Dynamic Weighting* Finally, weights do not have to remain fixed during query evaluation (like in the classes discussed above): Based on the query results (or a subset of it), the IR system may decide to change the weights it used initially when evaluating a query to achieve better results. This is tightly coupled to manual or automated relevance feedback which is not in our focus. For an overview of such concepts confer [Pan04].

3.5.2. Weighting Strategies

Using the above classification as a guideline, we now define several weighting strategies. User-defined weights we have already integrated as part of our query language in section 3.3. There we have also defined how to combine them with system-generated weights, so we do not need to consider them further. Adaptive dynamic weighting is outside our scope as stated above. In the following we will therefore first devise a fixed static weighting strategy and then propose several fixed dynamic weighting strategies to complement it.

Condition Type-based Weighting

The most trivial approach to weight query conditions is to assign weights uniformly, i.e. to assign the same weight to all conditions. This is equivalent to not performing weighting at all. In keyword-only environments (e.g. in traditional Information Retrieval and in Content-only XML Retrieval), this may be a reasonable approach. In CAS contexts, however, there exist several types of conditions, namely keywords, target conditions, and support conditions. Often, the latter two are merely imprecise pointers, whereas the keywords to a large extent characterise the actual information need and are thus more important. Also, depending on the scenario, target conditions may have to be treated differently with regard to their importance than support conditions. Hence, we consider it necessary to differentiate condition weights at least for the various types of conditions. In the following, let $s_{\text{kwd}}, s_{\text{sup}}, s_{\text{tgt}} \in [0, 1]$ be the scores resulting from keyword, support, and target conditions, respectively. As some proposals do not distinguish the latter two, we also define a *structural score* $s_{\text{struct}} \in [0,1]$ as the normalised sum of support and target scores:

$$s_{\text{struct}} = \frac{s_{\text{sup}} + s_{\text{tgt}}}{2} \in [0,1] \tag{3.2}$$

Our aim is to define how these scores are combined into an overall score $s \in [0,1]$. Amer-Yahia et al. [AYLP04] propose three simple strategies to weight structural conditions relative to content conditions: *Structure First*, *Keyword First*, and *Combined*. The former two are not actual weighting schemes, but define the order in which scores are used to rank results: Structure First ranks results by structural scores first, and then by content scores, Keyword First results in the opposite ordering. We consider this insufficient, as an XML Retrieval system ought to approximate relevance (i.e. a score) by taking into account both content *and* structure. The Combined strategy does combine both content and structure scores, but [AYLP04] do not define how to calculate the overall score. As an example they use the sum of both scores which is equivalent to uniform weighting and is thus infeasible.

A trivial alternative weighting scheme is the multiplication of keyword and structural scores as shown in equation 3.3. This corresponds to a conjunction of both types of query conditions: If either the keyword or the structural score is zero, then the resulting score will also be zero, meaning the element is regarded as not relevant. While this seems reasonable for strict query interpretations, it is not well suited for vague matching.

$$s = s_{\text{kwd}} \cdot s_{\text{struct}} \tag{3.3}$$

Hence we propose to perform *additive weighting* of keyword and structure conditions as is proposed in [PMM07], for example. Let $\beta \in [0,1]$ be an arbitrary factor, then equation 3.4 illustrates the idea.

$$s = \beta \cdot s_{\text{kwd}} + (1-\beta) \cdot s_{\text{struct}} \tag{3.4}$$

Additive weighting allows for an element which does not match either keyword or structural constraints to still have a non-zero score. Although this approach is generally feasible for vague matching, we believe that a fine-grained control is desirable. Namely, we propose to differentiate all three types of conditions as follows:

$$s = \alpha \cdot s_{\text{kwd}} + \beta \cdot s_{\text{sup}} + \gamma \cdot s_{\text{tgt}} \tag{3.5}$$

$$\text{with } \alpha, \beta, \gamma \in [0,1] \text{ and } \alpha + \beta + \gamma = 1 \tag{3.6}$$

One drawback of this solution is that it also assigns non-zero scores to fragments which have a keyword score of zero (i.e. which do not match a single keyword). In most scenarios this is undesirable. We can easily work around this problem, though, by slightly adapting the above weighting scheme as follows:

$$s = \begin{cases} 0 & \text{if } \alpha \cdot s_{\text{kwd}} = 0 \\ \alpha \cdot s_{\text{kwd}} + \beta \cdot s_{\text{sup}} + \gamma \cdot s_{\text{tgt}} & \text{otherwise} \end{cases} \tag{3.7}$$

This causes a zero score to be assigned, whenever the keyword conditions' total score is zero.

Tag Name ITF Strategy

An indicator of the quality of both target and support conditions might be the frequency of tag names which are used in the conditions in the document collection. For example, the tag `<section>` supposably occurs very frequently, whereas `<abstract>` is a lot less frequent; thus when the user searches for abstracts, this gives us a much more specific hint towards what he is looking for than a query requesting sections. To model this, we propose a solution based on the well-known TF-IDF weighting scheme [MRS08, chpt. 6]. For ease of expression, we first define the following auxiliary constructs: Let $d \in \mathcal{D}$ be an arbitrary document and $t \in \mathcal{T}$ an arbitrary term. Then we define a function $\text{occ} : \mathcal{T} \times X \longrightarrow \{t_1, t_2, \ldots, t_n\}$ which returns all occurrences of t in a set of arbitrary term occurrences X in the order in which they occur in X. If t does not occur in X at all, $\text{occ}(t, X)$ returns an empty set. We also define a function $\text{tn} : \mathcal{D} \longrightarrow \{t_1, t_2, \ldots, t_n\}$ which returns a list of all tag name occurrences in a document d in document order. (For simplicity, we assume that a tag name consists of exactly one term. We will later on address the problem of multi-term tag names in section 4.2.3 in the context of name matching.) To illustrate these functions, let d be the example document shown in listing 3.1. Then $\text{tn}(d)$ returns $\{\texttt{article}, \texttt{title}, \texttt{body}, \texttt{sec}, \texttt{b}, \texttt{sec}, \texttt{b}\}$ and $\text{occ}(\texttt{sec}, \text{tn}(d))$ returns $\{\texttt{sec}, \texttt{sec}\}$.

```
<article>
  <title>Examples of Tag Names in an XML Document</title>
  <body>
    <sec>Content of the <b>first</b> section.</sec>
    <sec>Content of the <b>second</b> section.</sec>
  </body>
</article>
```

Listing 3.1: Tag names in a sample XML fragment

Equipped with this, we calculate the *normalised term frequency* (TF) of t in an arbitrary document d as

$$\text{tf}(t, d) = \frac{|\,\text{occ}(t, \text{tn}(d))|}{|\,\text{tn}(d)|} \tag{3.8}$$

In other words, this tells us how often t appears as tag name in d in relation to all occurrences of any tag name in d. Applied to our example, $\text{tf}(\texttt{sec}, d)$ returns $\frac{2}{7} \approx 0.29$. For weighting purposes we analogously calculate the term frequency regarding the entire document collection $C = \{d_1, \ldots, d_n\}$ as

$$\text{tf}(t, C) = \frac{\sum_{d \in C} |\,\text{occ}(t, \text{tn}(d))|}{\sum_{d \in C} |\,\text{tn}(d)|} \tag{3.9}$$

To use this information as stated in the above example (i.e. weight `<abstract>` stronger than `<section>`), we can then use the *inverse term frequency* (ITF) which is calculated by the function shown in equation 3.10. For (tag name-based) target conditions, we can simply use the ITF value as weight, as they always contain exactly one term. For a support condition we assign the average ITF of all terms it contains as its weight. Granularity-based target conditions (e.g. `tgt:0.2`) are incompatible with this solution.

$$\mathrm{itf}(t, C) = 1 - \mathrm{tf}(t, C) \tag{3.10}$$

For collections based on a single schema, we believe this weighting strategy to be very effective, yet simple to implement. The ITF values per term should be stored in a collection-specific statistics repository. We expect the term frequencies to vary only slightly over time, so updating the statistics in large time intervals only is sufficient and without greater impact on retrieval effectiveness. For collections which are not based on a single schema, but still homogeneous (e.g. a collection of scientific articles from various publishers), we expect this strategy to be still applicable, but with a lower effectiveness. If, for example, the tags `<section>`, `<sec>`, and `<ss1>` each occur in some documents of the collection, three ITF values are tracked instead of just one, thus distorting the weights generated. A feasible countermeasure is to apply fuzzy name matching when generating the statistics: In the given example, all three tags should be recognised as equivalent and assigned a single ITF value. (We will discuss techniques for vague matching of names in section 4.2.1.) If the collection used is heterogeneous, however, generating weights based on inverse term frequencies will likely lead to very poor results: The approach assumes a common usage of tags throughout all documents of the collection, yet heterogeneous documents may vary widely in the set of tags they use and their respective frequency; in the worst case, a single tag may even be used in different contexts with different semantics. In this case, the strategy we discuss in the subsequent section is more promising.

Tag Name IDF Strategy

A measure of the discriminative power of a term commonly used in Information Retrieval is the *inverse document frequency* (IDF) [MRS08, chpt. 6]. It approximates 1.0, if very few documents contain a particular term (i.e. the term is highly discriminative), and 0.0, if many documents do. As demonstrated for term frequencies in the former section, we can also adapt the IDF measure to tag names in XML. We first obtain the *document frequency* (DF) of an arbitrary tag name t in a collection C as shown in equation 3.11. The document frequency is the number of documents in C which contain t. Based on the document frequency, equation 3.12 calculates the IDF of t in C. The IDF is greater, the less documents contain t, except when no document contains t in which case the IDF is set to zero[5].

$$\mathrm{df}(t, C) = |\{d \in C : |\mathrm{occ}(t, \mathrm{tn}(d))| > 0\}| \tag{3.11}$$

[5]A query condition which contains non-existent tag names is thus assigned a zero weight, i.e. ignored. This increases the processing efficiency, but may distort the semantics of the initial query: We assume that such a condition is erroneous, but one could also argue that if no document fulfils the condition, all result fragments should be penalised.

$$\mathrm{idf}(t,c) = \begin{cases} \log \frac{|C|}{\mathrm{df}(t,C)} & \text{if } \mathrm{df}(t,C) > 0 \land |C| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

For structuring and formatting elements in homogeneous collections (in particular such with a common schema) we expect this strategy to be perform very poorly: For example, every document in a collection of books very likely contains `<section>` elements and query conditions concerning sections are consequently assigned a zero weight; if solely using this strategy for weighting, the query

```
information retrieval tgt:section sup://book//section//title[xml]
```

would thus be mutilated to `information retrieval` – clearly not a very helpful way of processing the structural hints. In heterogeneous collections, however, this strategy may be very useful: If only a small fraction of the documents represent publications, conditions on `<section>` elements have a strong discriminating power and should thus be weighted strongly. Also, in both heterogeneous and homogeneous contexts, this strategy should work well for certain entity elements; for example, strongly weighting a condition on `<actor>` elements should be effective for selecting plays from a collection containing diverse publications like articles, books, plays, and so on.

Relationship Rareness Strategy

So far we have only considered tag name statistics. Unlike terms in flat Information Retrieval, however, XML natively introduces another powerful source of metadata in the form of containment hierarchies. There exist numerous relationships between elements in an XML document which we aim to exploit for weighting conditions in this section. We consider the XML descendant axis to be the most promising relationship for this purpose. One option we can exploit is to extend the ITF formula shown in equation 3.10 as follows: Let $t_1, t_2 \in \mathcal{T}$ be arbitrary tag names and $\mathrm{tn} : \mathcal{E} \longrightarrow \mathcal{T}$ a function which returns the tag name of an arbitrary element (again assuming that a tag name only consists of exactly one term). We then define the function $\mathrm{rc} : \mathcal{T} \times \mathcal{T} \times \mathcal{D} \longrightarrow \mathbb{N}^{\geq 0}$ in equation 3.13 to return the count of elements named t_2 which have an ancestor named t_1; in other words, this function counts the occurrences of ancestor/descendent relationships where the ancestor element has the tag name t_1 and the descendant element has the tag name t_2. Based on this we calculate the *relationship rareness* of t_1 and t_2 in C in equation 3.14. For example, if the tag `<paragraph>` occurs 100 times in the collection, and 90 of these occurrences have an ancestor element named `<section>`, then the relationship rareness of `section//paragraph` is 0.1. The relationship `section//equation`, on the other hand, might result in a much higher relationship rareness value, as it discriminates more strongly.

$$\mathrm{rc}(t_1, t_2, d) = |\,\{e_2 \in d : \mathrm{tn}(e_2) = t_2 \land \exists e_1 \in d \text{ with } \mathrm{tn}(e_1) = t_1 \land e_2 \in \mathrm{desc}(e_1)\}\,| \tag{3.13}$$

$$\mathrm{rr}(t_1, t_2, C) = 1 - \frac{\sum_{d \in C} \mathrm{rc}(t_1, t_2, d)}{\sum_{d \in C} |\operatorname{occ}(t_2, \mathrm{tn}(d))|} \tag{3.14}$$

Like the ITF strategy, we expect this approach to perform well for homogeneous, single-schema collections, but not for heterogeneous ones. The same possible extension for homogeneous collections with several schemas applies, too. Combining both strategies in a weighting function as shown in the following formula should further improve the results:

$$\mathrm{w}(t_1, t_2, C) = \mathrm{itf}(t_1, C) \cdot \mathrm{itf}(t_2, C) \cdot \mathrm{rr}(t_1, t_2, C) \tag{3.15}$$

This strongly boosts support conditions containing uncommon tags and uncommon relationships while down-weighting conditions like `sup://section//paragraph` which are of little use (in homogeneous environments).

Relationship IDF Strategy

Analogous to the Tag Name IDF strategy, we can define a relationship-based IDF variant which is feasible for heterogeneous collections. To do so, we only have to redefine the document frequency given in equation (3.11) as follows:

$$\mathrm{df}_{\mathrm{rel}}(t_1, t_2, C) = \{d \in C : \mathrm{rc}(t_1, t_2, d) > 0\} \tag{3.16}$$

Informally, this calculates the number of documents in the collection which contain an ancestor/descendant relationship of elements named t_1 and t_2, respectively. We omit the adapted variant of equation 3.12 here, as it is straightforward. The resulting IDF value then increases, if and only if very few documents contain the relationship in question. We can use it for relationship-based weighting of support conditions in heterogeneous collections.

3.6. Summary

In this chapter we have discussed query-related aspects of XML Retrieval, that is, our choice of a query language and the weighting of query conditions. For XML Retrieval there exist numerous query language proposals. In order to determine whether any of these are feasible for us, we have first stated our requirements regarding a query language. Also, we have defined certain relaxations (i.e. features which XML query languages commonly provide, but which we do not need for our purposes). To ease the comparison of existing query languages we have then introduced two classifications: by user and by expressiveness. The former classification describes whom a query language is targeted at (end-users, experts, or the IR system); the latter classification describes what kinds of information needs a query language can express.

For this thesis we need an expert and system language with the expressive power of a so-called path and keyword language. Hence we have selected existing query languages which belong to these classes and checked them against our requirements. The NEXI language [TS04a] which is common in XML Retrieval research falls short of our requirements. The most promising candidates are XIRQL [FG01] and the NEXI extension XOR [GHT06]; yet they also do not fully satisfy our needs. Hence we have resorted to defining a custom query language named CAS-QL. To keep the language as concise as possible, we have only included features which we actually use for the concepts we propose in this thesis. Features which only serve to aid our general discussion of XML Retrieval concepts we have separated to an extended query language called CAS-QLX.

After selecting the query language, we have introduced the concept of element types and discussed the weighting of query conditions in XML Retrieval. An element type is a group of XML elements with common properties such as identical tag name semantics. We use element types throughout this thesis to reason under which conditions certain solutions are applicable. Weighting defines how "important" a query condition is in relation to other conditions. We have introduced four classes of weighting strategies based on whether term weights are computed at runtime or up-front, whether the user or the system provides these conditions, and whether the weights are modified during query execution or not. Finally, we have proposed a generic weighting strategy based on condition types and several scenario-specific strategies to complement it.

4. Query Evaluation

In this chapter we will address the evaluation of XML IR queries with the exception of target conditions. We focus on the potentials and challenges related to *structural* conditions; vague content matching in XML documents, on the other hand, we only discuss as far as it directly relates to structural conditions (e.g. in the context of term proximities). We will first introduce our approach to scoring XML fragments in the subsequent section. Based on this we will then discuss the evaluation of explicit structural hints in section 4.2 and implicit structural hints in section 4.3. The generation of result sets including the handling of target conditions is the focus of chapter 5.

4.1. Scoring of XML Fragments

Due to the hierarchical nature of XML documents, the relevance of an individual element not only depends on its direct content, but also on its context (e.g. ancestor and descendant elements [PL04, sec. 1]). In listing 4.1, for example, it seems plausible that the term "cats" should contribute to the relevance of the surrounding section. In the same manner a paragraph in a section titled "pets" is likely more relevant for an according query than a paragraph in a section titled "cars".

```
<section>
  <section-title>Pets</section-title>
  <paragraph>
    Cats are often kept as domestic animals.
  </paragraph>
  <paragraph>
    Another favoured animal around households are dogs.
  </paragraph>
</section>
```

Listing 4.1: Example of relevance calculation in XML

A simple approximation is to score elements based on their recursive full content. (Please note that when we refer to the scoring of elements in the following, we regard elements as root elements and thus representatives of fragments.) In our example snippet, the `<section>` element would thus be scored using the terms "pets", "cats", "are", and so on. This is a very limited solution, however, which addresses the first example stated above ("cat" contributes to the section's relevance), but not the second one. We thus propose to calculate an element's score based on its *direct* full content, only, and then adjust it based on the scores of its context elements (which we will define shortly). We consequently refer to the score resulting from the direct content as *direct score* $\mathrm{s}_{\mathrm{direct}}$. The score resulting from an element's context is called *context score* $\mathrm{s}_{\mathrm{ctx}}$; please note that the context score does *not* modify the direct score, but is kept track of separately: We call the tuple consisting of direct score and context score *simplified score tuple*, denoted as $S_{\mathrm{SST}} = \langle \mathrm{s}_{\mathrm{direct}}, \mathrm{s}_{\mathrm{ctx}} \rangle$. As *context elements* of an arbitrary element e we define the set including

- e's ancestor elements,
- e's descendant elements,
- e's sibling elements,
- elements which e references (e.g. by `IDREF` or similar mechanisms), and
- elements which reference e.

For ease of expression we define a function $\mathrm{ctx}(e)$ to return all of e's context elements. Later on (in the result selection phase which we will discuss in chapter 5), the two scores – among other factors – are combined; this combination is a part of *score propagation.* Please note the importance of only using an element's direct content when also performing score propagation: If we would intermingle the two approaches (i.e. use the recursive content in addition to performing score propagation), this would entail the problem of *score duplication*: For example, a term occurrence in a child element of e would contribute to e's score multiple times as it is first counted as part of e's recursive content and then again by adjusting e's score based on its context elements (which include e's children).

By restricting the calculation of original scores to the direct content, we are able to ignore score propagation (almost) completely in the query evaluation phase. This is beneficial, as the way score propagation is performed has a major impact on the selection of result elements and should thus be handled in the result selection phase. The only obstacle to this separation are those query evaluation techniques which are innately context-aware. We say that context-aware evaluation techniques perform *pre-propagation*: Term proximity techniques, for example, calculate the proximity of a term A which occurs in an element x and another term B which occurs in an element y; thus they have to consider both x and y at the same time, so that the scores which are assigned to x and y already reflect their context. If we would propagate these scores, score duplication would be inevitable.

To work around score duplication, we introduce the concept of non-propagatable scores: A *non-propagatable score* (NPS for short) is a score assigned to an element $e \in \mathcal{E}$ which must not be propagated to other elements. We classify all scores resulting from context-aware evaluation techniques as NPS's and consequently ignore them in score propagation. If one element is scored by multiple context-aware techniques, this is reflected in a *single* NPS value, only, analogously to the direct score. Unlike the direct score, however, the NPS is not mandatory: An element receives an NPS, if and only if it is scored by at least one context-aware technique. If no context-aware scoring was performed for an element, this is expressed by $s_{\text{np}} = \perp$. Thus after the query evaluation phase every element in the element space features a non-propagatable score $s_{\text{np}} \in [0,1] \cup \{\perp\}$ in addition to its direct score $\text{s}_{\text{direct}} \in [0,1]$.

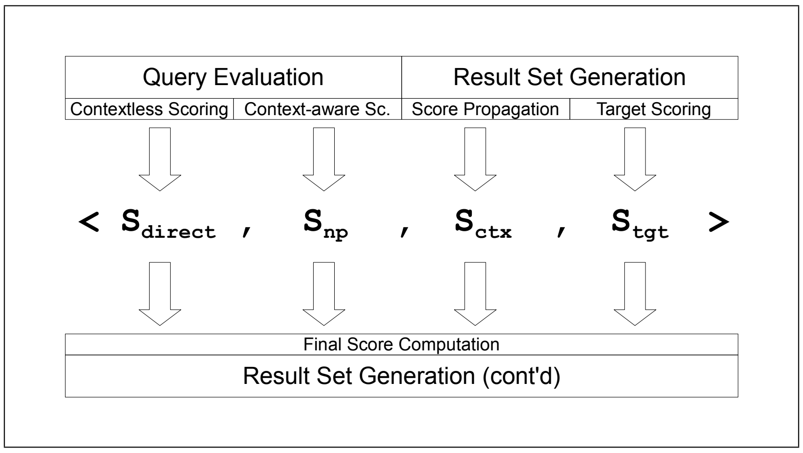

Figure 4.1.: Conceptional scoring framework

The context score is calculated in the result set generation phase based on the direct scores of an element's context elements. We refer to this calculation as *score propagation*. During the result set generation phase we also calculate another score called *target score* $s_{\text{tgt}} \in [0,1]$; as the name suggest it reflects the evaluation of target conditions. Our scoring framework hence consists of a quadruple $S_{\text{EST}} = \langle \text{s}_{\text{direct}}, s_{\text{np}}, s_{\text{tgt}}, \text{s}_{\text{ctx}} \rangle$ which we refer to as the *extended score tuple*. At the end of the query evaluation phase, only the first two of its values have been defined, that is, $S_{\text{EST}} = \langle \downarrow, \downarrow, \uparrow, \uparrow \rangle$. In the result set generation phase, after we have calculated the latter two values of S_{EST} as well, it serves as a basis to calculate a single final score ($s_{\text{final}} \in [0,1]$) which we use for ranking results. Figure 4.1 illustrates our conceptional[1] scoring framework. We will discuss its result selection-related parts in depth in the subsequent chapter. In the remainder of this chapter we concentrate on the calculation of s_{direct} and s_{np}.

4.2. Explicit Structural Hints

In this section we discuss the handling of explicit (i.e. user-provided) support conditions. We first look at the various problems related to this. Then we give an overview of related work and finally discuss our own proposals.

[1]Our scoring framework serves as a conceptional basis for our work, but is *not* intended to be a part of a system design. On the contrary, an implementation of our concepts would likely try to aggregate S_{EST} as early as possible to increase efficiency.

4.2.1. Problems

An arbitrary real world phenomenon (e.g. a scientific publication) can be modelled in quite different ways using a flexible data model such as XML. This "possibility of various ways of modelling the same [real world] object system" is generally referred to as *semantic relativism* [Pok93, sec. 1]. In XML Retrieval this is of concern to us, as conditions in XML IR queries do not necessarily reflect the modelling of the underlying document collection. This is due to the fact that the user who formulates these queries is often unaware of the schema of the XML documents queried.

Modelling Variants

In the following, we provide an overview of common *modelling variants*. We group the variants, so that each variant in a group reflects the same real-world phenomenon, but models it in a different way. There are, of course, many other modellings variants, but we consider the following list as a representative sample.

Element-based Content Modellings Modellings based on element content can have various levels of refinement [TI06, sec. 1] as shown in listings 4.2 to 4.5. Listings 4.6 and 4.7 illustrate two alternative modellings of multi-valued constructs.

```
<book>
  <author>Gerard Salton</author>
</book>
```

Listing 4.2: Single-valued, unstructured element

```
<book>
  <author>
    <name>Gerard Salton</name>
  </author>
</book>
```

Listing 4.3: Slightly structured element

```
<book>
  <author>
    <first-name>Gerard</first-name>
    <surname>Salton</surname>
  </author>
</book>
```

Listing 4.4: More structured element

```
<book>
  <author>
    <name>
      <first-name>Gerard</first-name>
      <middle-initial/>
      <surname>Salton</surname>
    </name>
  </author>
</book>
```

Listing 4.5: Highly structured element with additional data

```
<book>
  <author>Gerard Salton, Michael J. McGill</author>
</book>
```

Listing 4.6: Multi-valued, unstructured elements

```
<book>
  <authors>
    <author>Gerard Salton</author>
    <author>Michael J. McGill</author>
  </authors>
</book>
```

Listing 4.7: List of elements

Attribute-based Content Modellings Instead of elements, we can also use attributes to model content [FG00, sec. 6], [TI06, sec. 1] as listings 4.8 to 4.10 illustrate.

```
<book author="Gerard Salton"/>
```

Listing 4.8: Single-valued attribute

```
<book authors="Gerard Salton, Michael J. McGill"/>
```

Listing 4.9: Multi-valued attribute

```
<book>
  <author first-name="Gerard" surname="Salton"/>
</book>
```

Listing 4.10: Multiple single-valued attributes

Reference-based Content Modellings Entity relations we can model using references. According mechanisms are ID/IDREF [TW02] and XPointer [DJG$^+$07]. Listings 4.11 to 4.13 show examples of this.

```
<author author-id="a42"/>
<book author="a42"/>
```

Listing 4.11: IDREF instead of element

```
<author author-id="a42"/>
<author author-id="a43"/>
<book authors="a42 a43"/>
```

Listing 4.12: IDREFS instead of element

```
<author author-id="a42"/>
<author author-id="a43"/>
<book book-id="b1"/>
<participation>
  <item book-id="b1" author-id="a42" role="
     corresponding_author"/>
</participation>
```

Listing 4.13: Explicit relation element

Path Inversion In XML, many-to-many relationships can either be modelled by an explicit relation element (as illustrated in listing 4.13) or in an asymmetric way. In the latter case, there are two options to do so which are shown in the following listing [SMGL06]:

```
<book><authors><author/>...</authors></book>
<author><books><book/>...</books></author>
```

Listing 4.14: Path Inversion

Element vs. Attribute Value An element itself (i.e. the tag name, not the element content) may be modelled as the value of an attribute:

```
<book>...</book>
<publication type="book">...</publication>
```

Listing 4.15: Attribute value instead of element name

Cases of Semantic Relativism

Given these modellings, we are faced with one fundamental question: Are different modellings of the same real-world scenario are *equivalent* (i.e. matched without penalties) or not? To illustrate this, consider the following example: There is a query stating that all books are to be retrieved which were written by an author named Smith. We can formulate this query as:

```
tgt:book sup://book//author="smith"
```

The document collection we use to evaluate this query may be organised on a per-book basis or on a per-author basis as shown in listing 4.14. Also, the author of a book may be denoted as a subelement (cf. listing 4.2) or as an attribute (cf. listing 4.8). Regarding the information need stated above, all of these modellings should match the given query, that is, they should be treated as equivalents. In special scenarios like the Re-finding use case (cf. section 2.3.2), however, only the modelling specified in the query should match without penalty.

Hence the answer to this question obviously depends on the assumed scenario. As a result, we distinguish four different cases of semantic relativism. For ease of reference, we assign each case a number in the format "SR-n":

1. *SR-0: No semantic relativism:* The document collection used is homogeneous, shares a common schema, and the query already reflects the correct document structure.

2. *SR-1: Schema-restricted semantic relativism:* The document collection used is homogeneous and adheres to a single schema, but the query does not reflect the correct document structure.

3. *SR-2: Heterogeneous semantic relativism without penalties:* The document collection is heterogeneous and does not conform to a common schema, so the query is guaranteed to not correctly reflect the structure of all documents. Matches of equivalent modellings must not be penalised, even if they deviate from the modelling used in the query condition.

4. *SR-3: Heterogeneous semantic relativism with penalties:* Analogous to the previous case, except matches of equivalent modellings *are* penalised.

SR-0 (which, in fact, is a special case of SR-1) is trivial, as we do not need to consider semantic relativism at all. Unfortunately, this case is unlikely in real applications and – for reasons discussed in section 2.1.1 – we cannot rely on the query always being correct in IR scenarios. In SR-1 all documents share a common schema and thus are likely (or even guaranteed) to use the same modellings. If the schema is implicit, the user must formulate his query without a clear idea on the document's structure, and even if it is explicit it might be too complex or time-consuming for the user to take it into account. This case is special, however, from a system perspective: As there exists an (explicit or implicit) schema, it suffices – in theory[2] – for the IR system to "correct" the query on the schema-level before evaluating it. By correction we mean that the system transforms every query condition into a form which is semantically equivalent, but conforms to the schema. If the schema is explicit, this transformation could exploit it, whereas for an implicit schema the transformation would have to be based on schematic information gathered from instances. (For example, in the latter case, we might look at a small subset of documents in the collection, and then adapt the query conditions to the structure of these samples.)

For SR-2 and SR-3, correctional transformations are infeasible, as each queried document might use a different modelling. Instead of reformulating the structural conditions

[2]Schema-based corrections as mentioned here are far from trivial, particularly for the case of the schema being implicit.

we are thus forced to evaluate them in such a manner that a condition matches (ideally) all possible modellings. In SR-2 we must then treat all matches as equivalent, whereas SR-3 requires us to penalise non-perfect matches. The latter implies that we assign different relevance values to various modellings of the same real world phenomenon. We assume, however, that there is no specific modelling variant in a set of equivalent modellings that is *per se* more relevant than other variants. In other words we assume that there is no scenario where the user would generally deem `//book//author` items relevant but `//author//book` items irrelevant. The relevance of different modellings may differ, however, based on knowledge the user has about instances of these modellings. In the Re-finding use case, for example, the user looks for one specific fragment and remembers the modelling it uses; hence he considers this modelling more relevant. We expect this only to be the case in very few scenarios. We also expect that in these scenarios treating all modellings as equivalent will not cause complete failure of the retrieval logic, but merely impact its accuracy. Therefore we ignore SR-3 in the following to avoid unnecessary complexity.

In the following sections we will now discuss the problems semantic relativism poses for the matching of explicit structural conditions as well as general problems we have to tackle with regard to matching. For clarity, we differentiate problems related to the matching of names, the matching of paths, and other problems.

Name Matching

Both element and attribute names must be matched vaguely due to semantic relativism: The user might not know the correct schema and use a different modelling in his query (SR-1) or the document collections might be heterogeneous (SR-2,3). Also the user's query might contain typographic errors (e.g. misspellings) which we have to cope with. Where the name appears (e.g. that an attribute name in the query should also match an element name in the document collection) is an independent issue, however, which we discuss in the next section. Therefore we can concentrate on the following two issues only:

1. *Semantic equivalences:* A name is semantically equivalent to another one, so we have to match them.
2. *Typographic errors:* A name used in the user's query is syntactically incorrect.

These issues may, of course, also occur in combination. Please note that we have to be careful about how to define semantic equivalence: We can either define it based on general concepts such as words or in the context of a specific XML schema. To illustrate this, consider the following example: We can reasonably assume the real-world concepts "car" and "automobile" to be equivalent; nonetheless, in a particular XML schema, both concepts may be used with disjunct semantics. We believe it to be very hard, if not impossible, to decide the equivalence of two names regarding a particular XML schema without basing this decision on either the equivalences of real-world concepts or other string matching techniques. Therefore we *define* semantic equivalence of names without regard to real-world concepts as the degree of substitutability of a pair $\langle a, b\rangle$ of names. Yet we assume, that either manually predefined equivalence relations are available or equivalence can be approximated based on strings. The function equiv : $\mathcal{N} \times \mathcal{N} \longrightarrow [0, 1]$ calculates equivalence of two arbitrary names a, b in the space of all names $\mathcal{N}$, with 0 meaning not substitutable at all and 1 meaning perfectly substitutable. We assume that substitutability is symmetric[3] and reflexive, that is, $\text{equiv}(a, b) = \text{equiv}(b, a)$ and $\text{equiv}(a, a) = 1$.

So far, we have discussed name matching only in the context of support conditions. Name matching is also an issue for the evaluation of (name-based) target conditions, though. However, as both the problem and the solutions (which we discuss later on in this section) are exactly identical, we will not separately discuss it again in the context of target conditions.

Path Matching

Due to our interpretation of target conditions as not containing path information, we only need to match paths when evaluating support conditions. This, however, is by no means trivial as numerous issues related to semantic relativism arise:

- *Wrong Path Components*: This includes missing elements, excessive elements and substituted elements. A *missing element* exists in the structure to be matched, but is not included in the condition to match it. For example, the query condition `//book/section` misses the `chapter` element when matched against a `/book/chapter/section` structure. If `//book/chapter/section/p` is the condition and `/book/chapter/p` the actual structure, `section` is an *excessive element*. And finally, in the structure `/publication/chapter` the element `book` has been substituted and is thus a *substituted element*.

- *Delimiter Substitution*: We treat element-based, attribute-based, and reference-based content modellings as equivalent. Thus we have to match path expressions to each of these modelling variants. In CAS-QLX this means to allow the path delimiters (namely `/`, `//`, `@`, and `->`) to be substituted by one another.

- *Path Inversion*: A path expression must both match the path specified and its inversions (e.g. `/book/authors/author` and `/authors/author/book`).

- *Element Substitution*: In the context of path matching, we can express the element vs. attribute value problem as a combination of delimiter substitution and wrong elements. If the query aims at an element (say `<book>`) which has actually been modelled as an attribute value (i.e. `<publication type="book">`), we need to

[3]We believe asymmetric substitutability of names to be rarely needed in practical applications. Nonetheless, our restriction to the symmetric case is mainly efficiency-induced: It reduces the set of equivalence relations to half the size, as they can be specified pairwise. Apart from that, dropping the symmetry assumption entails consequences for the concept of substitution groups which we define in section 4.2.3, e.g. regarding their transitivity. Therefore our approach can be extended to the asymmetric case, but this extension is non-trivial.

substitute the element by the correct one, and append an “attribute equals value” condition.

Other Problems

Apart from name and patch matching, there are many more issues to be addressed by an XML Retrieval system. Due to the limited scope of this thesis we are unable to derive a solution which comprehensively includes *all* aspects we deem important. In this section we will therefore provide a brief overview of aspects we do not cover in detail in the remainder of this thesis, but consider important nonetheless. Please note that even this is merely a *selection* of problems remaining be solved.

One such issue is the integration of data-oriented matching with text-centric XML Retrieval. Consider the Book Search use case, for example (cf. section 2.3.2): Besides using typical XML IR conditions on the content of books and their document structure, the user specifies that he prefers books below a given price limit. As opposed to common data-oriented query languages he still wants this condition to be evaluated *vaguely* (i.e. books above the price limit may also be OK). To fulfil this information need, an IR system first of all has to detect “key = value” properties like the price and correctly identify metadata such as the data type (e.g. integer, float, date, timestamp, enumeration), unit (e.g. meter, Euro, degrees Celsius), formats, upper and lower bounds, and so on. This metadata has to be stored in an index to be accessible during query evaluation. The extraction process is commonly known as typed value extraction; confer [CLM$^+$01] for one example of how this may be done. Ideas on integrating it into an IR system are outlined in [EL00], for example.

The evaluation of conditions involving such properties also brings about various obstacles such as conversion, disambiguation, and vague matching. Conversion means that data types, units, and formats used in a query must be converted when matching against instances of the document structure; for example, the condition `price < 10 euro` should match a property `price = 14.00 dollar` due to the currencies’ exchange rates. Disambiguation refers to matching the right properties based on their meaning; in our price example, the user might be interested in matching a property `retail-price`, but not `purchase-price`. Vague matching is the behaviour described in our Book Search use case where the user is chiefly interested in books below 50 Euros, but is willing to pay more if a book suits his needs very well.

Another issue besides the integration of data-oriented matching is the conversion of XML content to string values for matching (and display). Consider the fragment shown in listing 4.16. The string representation of the `<st>` element to match against should obviously be `Introduction` as opposed to `I␣ntroduction`[4], whereas the second half of the listing (which follows the same structure and formatting) should be interpreted differently. XML is very liberal in terms of spacing and indentation so we cannot simply rely on the formatting used in the document to match to determine a string value. For a detailed discussion of this issue and possible solutions cf. [OT03].

[4]The tag name `<scp>` is commonly used to convey the formatting instruction “set in small capitals”. In our example fragment, the following formatting of the section title is thus intended: INTRODUCTION.

```
<st>
  I
  <scp>
    ntroduction
  </scp>
</st>
<p>
  I
  <i>
    wonder
  </i>
  what the right string value is.
</p>
```

Listing 4.16: String value of XML content

4.2.2. Related Work

In this section we provide an overview of other publications in the context of CAS Retrieval which address the issues which we focus on: name matching and patch matching. For the sake of clarity, we discuss related work separately for each issue, although CAS publications typically cover arbitrary combinations of these (and other) aspects.

Name Matching

There are two general approaches to deal with name matching in XML Retrieval: One is to assume that the problem can be reduced to – or at least approximated by – generic string matching techniques known as approximate string matching: This area is well-explored (see e.g. [Nav01] for an overview) and provides us with a large base of ready-to-use algorithms. The other approach is to assume that we have rule sets available which define name equivalences for the environment we are in (e.g. the document collection). The two approaches are not exclusive, but may also be used in combination. Sometimes a third approach is suggested in literature which is to ignore naming altogether. Arvola et al. [AKJ05, p. 136], for example, reduce structural conditions to element relationships with constraints on the respective content of the elements. The CAS-QLX query `/book[information retrieval]/section[cas]` would thus be reduced to `/*[information retrieval]/*[cas]`. This means that we "throw away" structural hints which are readily available and likely useful. We do not consider this kind of behaviour desirable and thus only discuss the first two approaches in the following.

Typical approximate string matching approaches include means like morphological normalisation, that is, words are reduced to a more general from like a word stem. In [KMdRS02] this technique is applied for content matching (i.e. not names) with some success. Another technique employed in [KMdRS02] for content matching are n-grams. (See [Nav01] for a general introduction to n-grams and similar concepts.) For

Name	s_{AF}
`<sec>`	1.0
`<section>`	1.0
`<subsection>`	0.8
`<chapter>`	0.7
`<subsubsection>`	0.5
`<paragraph>`	0.3

Table 4.1.: Example of a substitution group

error correction, n-grams and other character transposition means like edit distances are commonly used [LW75], [Nav01] and should also work for names in XML Retrieval, in principal. To our knowledge, this has not been tested so far, however. Many XML Retrieval publications (e.g. [MHB06], [CCPC⁺06], [PMM07], [TW02]) follow the second approach (rule sets for name equivalences) and propose the use of mapping tables for semantic equivalences of names. We will refer to these tables as *substitution groups*[5]. A substitution group contains all names which may be substituted for a given name together with a *score adjustment factor* $s_{\mathrm{AF}} \in [0, 1]$. This factor expresses the semantic equivalence of the name as defined in section 4.2.1; that is, an element whose name is listed in a substitution group for a particular name contained in a query condition will match, but its score is multiplied by the adjustment factor. Table 4.1 illustrates a sample substitution group. There are many different variations of the substitution group approach. The major variation aspects are the way substitution groups are obtained and the way they are used for query evaluation. For the first aspect we can differentiate between three (non-exclusive) means:

- *Manual generation*, e.g. by the system administrator; [FG04] describe a similar idea (although on a more generic level) where the system administrator has to define rule sets for name substitutions.
- *Automatic generation based on document statistics or machine learning*: For example, Mihajlović et al. [MHB06] propose to execute queries including a target condition for a certain name (say x); the best results with other tag names than x are then added to x's substitution group with the s_{AF} estimated based on their scores. A similar approach is proposed in [CCPC⁺06].
- *Automatic generation based on semantic relationships between names*: Such relationships includes, for example, synonym relationships (e.g. car/automobile) and hypernym-hyponym relationships (e.g. cat/animal); they can be obtained from broad, publicly available sources such as WordNet[MBF⁺90]. This technique links generic approximate string matching techniques and equivalence-based approaches. Mihajlović et al. [MHB06] use an approach like this to complement their solution.

[5]Substitution groups (and similar concepts) are known by many different names in XML IR literature. Other common ones include *expansion lists* and *equivalence classes.*

As to how substitution groups are used, there are mainly the following two approaches:

- *Ad hoc matching*: When comparing a name from a query condition (say x) to the name of an element (say y), we check whether y is contained in x's substitution group.
- *Query expansion*: Before query evaluation every name occurring the query is replaced by a disjunction of all names in its substitution group [TW02]. For example, the query `tgt:sec` would be replaced with the query `tgt:sec || tgt:section || tgt:subsection || ...` based on the substitution group defined in table 4.1.

Query expansion approach has two obvious drawbacks: Firstly, the number of query conditions grows very fast, in particular, when path conditions are used; secondly, the score adjustment factors have to be tracked (for example, by mapping them to confidence values of the generated query conditions).

Other differences between substitution group implementations include what kind of names they are applied to (In [MHB06] the authors only use them for element names, for example, whereas in [PMM07] they also include attribute names), how they integrate with existing indexes, and how they affect the scoring of matches. For details on these aspects please refer to the sources stated above.

Path Matching

Path matching compares the paths listed in support conditions with the paths of XML elements. A fundamental term in this context is the *path component*: It corresponds to one node in the XML document tree. For example, the path `//author/books/book//section` consists of the set of path components {`author`, `books`, `book`, `section`}. Thus path matching is typically considered as matching a system of such path components related by some delimiters. There exist numerous proposals in XML IR literature on how to perform path matching. The following overview thus only accounts for the most prominent approaches, but is by no means intended to be comprehensive. For further ideas on path matching refer to [Sch01], [SK05], and [XX07], for example.

Factor-based Approaches Van Zwol [vZ05] suggests two simple extensions to Content-only matching to support structural hints, the Path Factor and the Request Penalty Factor. He first scores an element $e \in \mathcal{E}$ based on keyword conditions, only. Then he multiplies the keyword score by the *Path Factor*, a number $p \in (1, \infty)$; this factor is higher the more path components of e match structural hints in the query. The *Request Penalty Factor*, on the other hand, penalises excessive path components in e's path. Carmel et al. [CMM$^+$03] use a similar, but more generic strategy. They also modify the Content-only score based on the evaluation of structural conditions, but in a more flexible manner. Their approach features a so-called *cr function* (for context resemblance); it compares the path of an element with the support path given in the query (assuming that there is only one support condition) and returns a value in the $[0, 1]$ interval, with 1.0 indicating a perfect match. Besides some trivial implementations of this function (e.g. for strict matching of structural conditions) they propose an implementation called *partial match* which returns the percentage of path components that are contained in the support condition. This addresses substituted and missing elements, but none of the remaining issues.

Path Edit Distances Thus Carmel et al. propose a more sophisticated implementation in [CEL+02] which – although not explicitly stated – essentially adapts the idea of string edit distances to XML paths. Their refined implementation additionally takes into account the following aspects:

- The order of path components. Deviations from the order given in the support condition are penalised.
- The proximity of path components. Missing elements are penalised.
- The position of path components within the path. Deviations from the requested path are penalised more strongly, the further to the left of the support path they are located; this motivated by the idea that the first elements in a path discriminate more strongly than other elements.

This extension additionally addresses excessive elements and path inversion. (Although the latter is penalised despite of potentially reflecting an equivalent modelling.) Modelling variants such as attribute-based or reference-based modellings and the remaining issues listed above are entirely ignored, however. Popovici et al. [PMM07] explicitly suggest to adapt string edit distances to the problem of path matching. Thus they define the path edit operations substitution, deletion, and insertion of elements analogous to the well-known Levenshtein distance (cf. [Bla07]) with a fixed cost per application each. As NEXI only allows the use of the descendant axis [TS04a], they assign a cost of zero to delete operations. This also addresses missing elements.

Tree-based Approaches In an early approach, Schlieder and Meuss [SM00] consider XML documents as unordered, labelled trees with only a single node type. A node can thus correspond to elements, attributes, attribute values, or textual content, without distinction. Edges in the tree represent parent-child or element-attribute relationships (without distinction), references are ignored. Queries are also formulated as tree structures following the same model; the root element is treated as a target element. Each node in the query tree is called a *structural term* and matched against nodes in the document trees contained the collection. This matching is performed based on two conditions: Firstly, the structural term's label has to match exactly that of a node in a document tree; secondly, if the structural term has ancestors, a matching node in a document tree must have corresponding ancestor nodes. This allows for a parent-child relation in the query to match an ancestor-descendant relation in a document and thus addresses the problem of missing elements. The substitution of element-based modellings by attribute-based ones (and vice versa) is also innately handled with this approach. All other issues cannot be handled with this proposal, however. (Also, both name matching and target element handling are based on strict matching which we consider infeasible. This is not a path matching issue, though.)

Ciaccia and Penzo [CP02] extend the tree matching approach by using a more detailed data model and rule-based matching. They distinguish between element and attribute nodes in their data model, for example, but allow both node types to match a path component. Also, they allow a parent-child relationship to be matched by both ancestor-descendant relationships and sibling relationships, and a sibling relationship to be matched by an ancestor-descendant relationship. All these "mismatchings" (i.e. deviations from the user's idea of the document structure) are penalised, however, thus not allowing for the equivalence of modelling variants. The remaining issues we have discussed are also ignored.

Query Relaxation Another group of proposals tries to handle path matching issues by means of query relaxation. The general idea behind this is to evaluate a query on the document collection, check if the result has a sufficiently large size, and – if not – iteratively weaken the conditions used in the query and re-evaluate it, until the result reaches the desired size (or no more relaxations are possible). A query can be relaxed by replacing conjunctions with disjunctions, by replacing conditions in a query with weaker ones, by dropping query conditions, or by any combination thereof; for a detailed discussion of these options see [AYLP04]. For ease of notation we write $q_1 \rightsquigarrow q_2$ to express that a query q_1 is relaxed to a query q_2. To still enable a meaningful result ranking, additional results produced by relaxations are penalised.

One such approach is the FleXPath system devised by Amer-Yahia et al. [AYLP04]. They define separate relaxation rules for support conditions and content conditions. For support conditions these rules essentially allow to relax parent-child relationships to ancestor-descendant relationships (e.g. `sup://a/b` $\rightsquigarrow$ `sup://a//b`) and to remove excessive elements (e.g. `sup://a//b//c` $\rightsquigarrow$ `sup://a//c`). For content conditions they define relaxations of the `contains` operator common in XML query languages such as XQuery and NEXI. The operator is used to express that a particular element in a path should contain a particular set of terms in its recursive content. The relaxation proposed by Amer-Yahia et al. is straightforward: The condition that an element a should contain a term x can be relaxed to the condition that a's parent should contain the term (e.g. `sup://a//b[x]` $\rightsquigarrow$ `sup://a[x]//b`). In the same manner, other relaxation rules can be defined, e.g. for atomic values (`sup://a//price<98` $\rightsquigarrow$ `sup://a//price<100`) or tag and attribute names (`sup://book//chapter` $\rightsquigarrow$ `sup://publication//chapter`) [AYLP04].

Scoring in FleXPath takes query relaxations into account by assigning each query condition (both original and relaxed conditions) a penalty value. Whenever a condition is dropped from a query (e.g. to be replaced with a weaker one), this penalty is subtracted from scores of elements matching (only) the resulting relaxed query. This approach not only guarantees that elements coming into the result by relaxing the query receive lower scores, but it is also invariant of the order in which conditions are dropped.

The two major drawbacks of query relaxation approaches are performance issues and maintenance complexity. We believe that a rather large number of relaxation rules would be required to address all of the path matching (and other) issues we have discussed above. These rule sets are hard to design and maintain. The existence of comprehensive rule sets presumably leads to huge relaxed queries which have to be evaluated and whose results have to be scored and joined with the initial result set. Therefore Amer-Yahia et al. aim at Top k query processing [MAYKS05] where pruning can be performed (that is, the early elimination of result candidates based on statistical estimations) and other heuristics such as in [AYKM⁺05]. As another countermeasure, they restrict the approach to very simple relaxation rules. Nonetheless, we fear query relaxation is infeasible for broader application areas.

4.2.3. Solution Approach

In the following, we discuss our approach to tackle explicit structural hints. It is composed of a proposal for (tag and attribute) name matching, path edit distances, and several index-based aspects.

Name Matching

Our name matching solution has two objectives: Firstly, it shall allow us to integrate both approximate string matching techniques *and* substitution groups, and secondly, account for name length dependencies. The first objective is based on our discussion of related work in the previous section. We believe that both approximate string matching techniques and substitution groups have certain advantages: The former allow us to cope with syntactical errors in the query string (e.g. match `tgt:publictaion` with `<publication>`) and do not depend on the existence of predefined rule set; the latter enable high-confidence matching. Therefore instead of just choosing either one technique, our solution shall be able to integrate both. The second objective is rooted in the observation that the result quality of string matching techniques strongly depends on the length of names to be matched. For example, it makes little sense to apply common correction means like edit distances or n-grams to one- or two-letter tag names like `<b>`, `<i>`, or `<st>`: This would essentially lead to arbitrary tag names being matched. For long names (like `<publication>`), on the other hand, we believe that these means yield good results.

We therefore propose a modular a name matching function which is based on name length:

$$\mathrm{namesim}(s_\mathrm{q}, s_\mathrm{d}) = \begin{cases} \mathrm{namesim}_\mathrm{short}(s_\mathrm{q}, s_\mathrm{d}) & \text{if } |s_\mathrm{q}| \leq L_\mathrm{short} \\ (1 - \frac{1}{|s_\mathrm{q}|}) \cdot \mathrm{namesim}_\mathrm{long}(s_\mathrm{q}, s_\mathrm{d}) & \text{otherwise} \end{cases} \tag{4.1}$$

It calculates the similarity of two names (i.e. character strings) s_q, s_d as a value in the $[0,1]$ interval, with 1.0 indicating identity. s_q is a name used in the query string, s_d an arbitrary name in a document which is matched against it. By $|s|$ we denote the length of a string s. Based on an arbitrary length threshold[6] $L_\mathrm{short} \in \mathbb{N}^{\geq 1}$, we delegate the actual similarity calculation either to $\mathrm{namesim}_\mathrm{short}$ or $\mathrm{namesim}_\mathrm{long}$. The former encapsulates only name matching techniques applicable to short names, whereas the latter performs the matching of long names. Both functions have the same domain and range as namesim. Which values are feasible for L_short likely depends on the scenario and collections used; as a starting point, we guess that $L_\mathrm{short} = 4$ is a reasonable assignment. Additionally we deem that from a certain minimum length onwards, matching confidence increases with increasing name length; we therefore use the length as a confidence indicator, that is, the longer a name is, the more strongly the results of string matching techniques are weighted. This is reflected in the factor $(1 - \frac{1}{|s_\mathrm{q}|})$ which we multiply by the result from $\mathrm{namesim}_\mathrm{long}$.

For the implementation of $\mathrm{namesim}_\mathrm{short}$ we propose a combination of substitution groups and abbreviation handling techniques. Substitution groups work regardless of name length, as they are based on element-specific rules instead of heuristics; we will discuss them in detail, shortly, but first turn to abbreviation handling. Abbreviation handling techniques are especially suitable for short names (as abbreviations tend to be short). One such approach which we expect to work effectively for our problems is the *recursive field-matching algorithm* (RFM) by Monge and Elkan [ME96]. Let s_1, s_2 be two arbitrary character strings; then the algorithm can handle the following common cases of abbreviations based on prefix/suffix combinations [Har06]:

1. s_1 is a prefix of s_2 (e.g. “sec” vs. “section”)
2. s_1 is a prefix of s_2 concatenated with a suffix of s_2 (e.g. “bd” vs. “bold”)
3. s_1 is a concatenation of prefixes of partial strings of s_2; partial strings are demarcated by spaces (e.g. “st” vs. “section title”).

This algorithm has been found to produce good results in other name matching scenarios, albeit being less efficient than other metrics[7]. For name matching in XML Retrieval, we propose to use it with a minor modification: As tag and attribute names in XML do not contain spaces, partial strings are demarcated by hyphens (-), underscores (_) and full stops (.). (We assume other special characters permitted for “name” tokens by the XML specification [BPSM$^+$06, sec. 2.3] are rarely used and thus ignore them for efficiency reasons.) This metric yields good results for abbreviation-related problems, but can nonetheless only determine similarities with a low confidence compared to metrics operating on longer names (and to the one discussed below); therefore we multiply a constant penalty factor $P_\mathrm{RFM} \in [0,1]$ by the results calculated by recursive field

[6]We solely use the length of the name appearing in the query (i.e. $|s_\mathrm{q}|$) as a parameter as opposed to the length of the names occurring in the documents (i.e. $|s_\mathrm{d}|$). If the latter is very different, the decision on how closely the two names match has to be made by the similarity metrics employed.

[7]For a comparative overview of name matching metrics and possible optimisations see [CRF03].

matching. We guess that setting P_{RFM} to a value around 0.5 might be appropriate; this will have to be experimentally verified, however.

As we have mentioned above, the substitution group metric introduced in section 4.2.2 is independent of name lengths. It can provide similarity values with a high confidence as it does not rely on character similarities, but uses semantic relationships instead which – ideally – are even manually defined. There are two aspects of substitution groups, however, which we need to refine in order to use them effectively: compatibility with namespaces and the transitivity of substitutions. We have introduced namespaces in section 2.1.2. They serve to reference predefined vocabularies and are of particular interest when dealing with heterogeneous collections. To avoid name clashes, we propose to transform all names used in a document to their *expanded name* before indexing. The expanded name is defined in [BHLT06, sec. 2.1] as the name including a reference to its namespace. Matching unqualified names must still be possible, however, as we discuss below. Hence the index must be able to store namespace information in such a way that matching both qualified and unqualified names is possible. If for a name no explicit namespace is stated, the default namespace of the document containing it is used; if that is undefined, too (e.g. because no default namespace has been specified), the namespace reference is empty by definition [BHLT06]. Therefore using the expanded names still does not guarantee uniqueness of names. A workaround for this case is the use of annotations (as described e.g. in [CCPC+06]) as a source of additional metadata. To limit the complexity of our approach, however, we assume that in this case the semantics of the name in question are the same across all documents (e.g. that a `<paragraph>` element without a namespace has the same meaning in all documents).

Using namespaces in substitution groups does also introduce additional challenges, however: The automatic generation of substitution groups, for example, is based on names (e.g. using WordNet as described in section 4.2.2). Using the expanded names for this purpose will likely have a negative impact on the matching quality, unless we introduce complex metrics to match both the namespace and the actual name separately. We believe that such a metric would greatly increase the complexity while the benefits are unclear. Thus the automatic generation of substitution groups should still operate on *local names*, that is, the part of the name that remains after stripping the namespace reference [BHLT06]. In broadly heterogeneous scenarios such as the XML Web Search use case we also expect it to be difficult to differentiate between namespaces, as there is likely a huge number of different albeit similar namespaces. Doing so might even limit the benefit substitution groups provide, as semantically similar names go unrecognised due to being in different namespaces – unless we would go as far as detecting groups of namespaces with semantically similar tags. Last but not least using the extended names for manual substitution group definition may require a lot more effort depending on the number of namespaces used.

Therefore we do not enforce the use of namespace information, but refine substitution groups as follows: Names used in substitution groups are local names per default to support the above cases. *Optionally* individual names may be qualified with a reference to their namespace (i.e. forming expanded names). If an expanded name is used, we observe the namespace when matching it, whereas a local name in a substitution group

Name	s_{AF}
`<ns1:sec>`	1.0
`<ns2:section>`	1.0
`<ns3:ss1>`	1.0
`<sec>`	0.8
`<section>`	0.8
`<ns1:subsec>`	0.8
`<ns2:subsection>`	0.8
`<ns3:ss2>`	0.8
`<chapter>`	0.7
`<ns1:paragaraph>`	0.5
`<ns2:subsubsection>`	0.5
`<ns3:ss3>`	0.5
`<paragraph>`	0.3

Table 4.2.: Example of a namespace-aware substitution group

matches local names in all namespaces. Names occurring in a substitution group must be unique regarding their expanded name, however. This means, that if the same local name occurs multiple times in a substitution group, each occurrence must be qualified with a different namespace and at most one occurrence may be unqualified; in the latter case the unqualified occurrence must have a strictly smaller score adjustment factor than any of its qualified variants. To illustrate this, table 4.2 adapts our substitution group example from the previous section to our enhanced proposal; “ns1”, “ns2”, and “ns3” represent arbitrary namespaces in the example. Please note that unlike substitution groups, approximate string matching techniques should still solely operate on local names. Applying edit distances, for example, to an expanded name is unlikely to yield reasonable results. Here the same possible extensions with the same drawbacks apply as we have outlined for the semantic matching of names, above. Thus we accept the inaccuracy of employing approximate string matching metrics which are ignorant to namespaces.

Another question regarding substitution groups is their transitivity. Assume, for example, that the name “section” can be substituted by “paragraph” with a s_{AF} of 0.3 and “paragraph” can be substituted by “verse” at 0.8; it then seems natural that “section” can also be substituted by “verse” at a cost of $0.3 \cdot 0.8 = 0.24$. Remember that we have defined the semantic equivalence of names to be reflexive and symmetric; thus, if we allow transitivity, we must ensure that these properties are guaranteed. One way to achieve this is to model substitution groups as a single undirected acyclic graph. This, however, would make both name matching and the (manual and automatic) generation of substitution groups a lot more complex. Therefore we define substitution groups to be non-transitive at the cost of having to list *all* desired substitutions for a name in its substitution group. This means that for all names y which are not listed in the substitution group for a name x it implicitly holds that $\text{equiv}(x, y) = 0.0$; the only exception is the special case $x = y$ where it holds that $\text{equiv}(x, y) = 1.0$ due to reflexivity.

So far we have only specified our similarity function for short names, $\text{namesim}_{\text{short}}$. For long names, we believe that a much broader range of matching techniques may be beneficial. For example, we expect traditional string similarity metrics like edit distances [LW75], n-grams [CT94], and phonetic metrics [ZD96] to perform well. Which combination of these techniques yields good results (while still being reasonably efficient) only experimental evaluation can show, however. (Confer section 7.1.3 for details on this.) One technique we definitely expect to be of use for long names as well is the substitution group metric with the adaptions discussed above.

Path Edit Distances

Regarding path matching our aim is to devise a solution which is powerful enough to address the problems listed in section 4.2.1, yet manageable in terms of complexity. We regard factor-based approaches as too inflexible to meet the first criterion; both tree-based and query expansion-based approaches, on the other hand, are growing fast in conceptional and computational complexity, when trying to solve these problems. Therefore we propose an approach based on the idea of path edit distances to keep a balance between power and complexity.

Let $\text{path} : \mathcal{E} \longrightarrow \langle p_1, \ldots, p_n \rangle$ be a function which maps an arbitrary element to an list of path components $p_1, \ldots, p_n \in \mathcal{P}$. By $\mathcal{P}$ we denote the set of all possible path components. The set of edit operations Ω we use consists of the following operations:

- Path Component Insertion (ω_{ins})
- Path Component Deletion (ω_{del})
- Path Component Substitution (ω_{sub})
- Path Component Swap[8] (ω_{swp})

The first three ($\omega_{\text{ins}}, \omega_{\text{del}}, \omega_{\text{sub}}$) we define analogous to [PMM07]: They enable the handling of missing, excessive, and substituted path components, respectively. The fourth operation (ω_{swp}) addresses path inversion. In fact, the first two operations would suffice to cover all path modifications we need. To have finer control over the cost of edit operations, however, we introduce the other two operations ω_{sub} and ω_{swp}.

Now, let $\text{editcost} : \Omega \longrightarrow [0, C_{\text{max}}]$ be a cost function which determines how strongly the application of a given operation is penalised: The greater the calculated value is, the more expensive the operation is. $C_{\text{max}} \in \mathbb{N}^{\geq 1}$ is an arbitrary, but fixed value (say 42) which defines the maximum edit cost; this serves two purposes: Firstly, whenever we reach this maximum while calculating a path edit distance, we can abort the calculation

[8]Analogous to proposals for string edit distances (cf. e.g. [LW75]) our swap operation only applies to neighbouring elements. To transform `//author/books/book` into `//books/book/author`, for example, two swaps are thus needed as opposed to only one.

Table 4.3.: Base cost assignments for edit operations

Operation	Base Cost	Repetition Factor
ω_{ins}	1.0	1.3
ω_{del}	1.0	1.3
ω_{sub}	1.8	1.1
ω_{swp}	1.5	1.0

(for efficiency reasons) and return $C_{\max}$ instead; secondly, this fixed upper bound enables us to normalise the cost value to the $[0, 1]$ interval later on, when using it for scoring.

In section 4.2.1 we discussed that we generally (i.e. with the exception of SR-3) assume different XML modellings of the same real-world scenario to be semantically equivalent. If we apply this assumption for defining our cost function, the naive solution is to simply define a cost of zero for all of the edit operations. The drawback of doing so is that with set of the operations we have defined essentially *any* modelling would match a support condition, not only semantically equivalent ones; this clearly is not a useful system behaviour. Assigning non-zero cost values, on the other hand, falls short of the goal of matching equivalent modellings without penalty. Therefore we propose a *dynamic cost function* to approximate the ideal solution. This function not only considers a single edit operation, but additionally takes into account the path components which are affected by it and all previous operations which have been performed on the path already. Thus we re-define the cost function as follows:

$$\text{editcost} : \Omega \times \mathcal{P} \times \mathcal{P} \times N \longrightarrow [0, C_{\max}] \tag{4.2}$$

The first parameter of the editcost function describes the operation used like in our initial cost function. The second and third parameter are the path components affected by this operation ($p, q \in \mathcal{P}$); the third parameter is null ($\perp$), if the operation only concerns one path component, e.g. ω_{del}. The fourth parameter $N = \langle n_{\text{ins}}, n_{\text{del}}, n_{\text{sub}}, n_{\text{swp}} \rangle$ reflects the number of executions of each edit operation for the current path (with $n_{\text{ins}}, n_{\text{del}}, n_{\text{sub}}, n_{\text{swp}} \in \mathbb{N}^{\geq 0}$). For example, $N = \langle 1, 2, 0, 0 \rangle$ means, that one the path in question, we have performed one insertion, two deletions, and no substitutions or swaps.

Now we define a constant *base cost* and a *repetition factor* for each operation as shown in table 4.3. For the ease of notation we define the following two functions to return the base cost and the repetition factor for an edit operation, respectively:

$$\text{editcost}_{\text{base}} \;:\; \Omega \longrightarrow [0, C_{\max}] \tag{4.3}$$

$$\text{editcost}_{\text{rep}} \;:\; \Omega \longrightarrow \mathbb{R}^{\geq 1} \tag{4.4}$$

The base cost function is identical with our initial cost function; the cost values it calculates serve as the basis for our dynamic cost function. For ω_{ins} and ω_{del} the base cost is arbitrary, but should be identical, as insertions and deletions complement each other. Substitutions replace one path component with a different one; because there may exist a semantic relationship between the path component to be replaced (p) and the path component replacing it (q), we assign a slightly lower base cost for a substitution than for a deletion followed by an insertion. Swaps only change the nesting of path components, but do not add or remove any component; we assume that this often only has a minor impact on the structure and thus assign it a lower base cost than that of an insertion/deletion pair or a substitution.

The repetition factor serves to penalise repeated applications of an operation. The idea behind this is that certain operations (e.g. deletions) have only a minor impact on a path if they are performed once or twice, but a significant impact, if they are performed many times. Thus we increase the penalty for an operation each time it is applied to a path. An even more elaborated approach would be to determine the repetition factor not only per operation, but also have it depend on the length of the path (because a long path is more tolerant to repeated operations than a short one). We restrain from this, however, for the sake of simplicity until our baseline path matching strategy is found useful in practical experiments. A generic cost function implementing these concepts then looks as follows (with $\omega \in \Omega$ being an arbitrary edit operation and $n \in \mathbb{N}^{\geq 0}$ the count of its applications to a given path):

$$\text{editcost}_{\text{gen}}(\omega, n) = \text{editcost}_{\text{base}}(\omega) \cdot \text{editcost}_{\text{rep}}(\omega) \cdot n \tag{4.5}$$

For substitutions we further refine this function: As mentioned above, there may exist a semantic relationship between a path component to be substituted (p) and the path component substituting it (q). In fact, if there is no such relationship we believe substitution should not take place (or at least be performed at a very high cost). For example, if a path component referencing a `<book>` element is substituted by one referencing a `<publication>` element, there may be a slight change in the semantics, but the intention of the original path is likely still upheld. If, on the other hand, it is substituted by a component representing something entirely different (say `<animal>`), this is likely to severely affect the semantics of the match. Therefore we adapt our cost function accordingly. To accomplish this, we need a measure to approximate semantic similarity. For this purpose we employ the namesim function which we have introduced in the preceding section in the context of name matching to solve exactly the same problem. The following equation shows the adapted cost function definition for substitutions:

$$\text{editcost}_{\text{sub}}(p, q, n_{\text{sub}}) = (1 - \text{namesim}(p, q)) \cdot \text{editcost}_{\text{gen}}(\omega_{\text{sub}}, n_{\text{sub}}) \tag{4.6}$$

The idea is that the cost of a substitution decreases as the similarity of p and q increases. To attain the desired effect (i.e. making non-similar substitutions very costly), we also have to strongly increase the base cost of ω_{sub} compared to the value given in table 4.3. Setting $\text{editcost}_{\text{base}}(\omega_{\text{sub}}) = 10.0$ may be appropriate, for example. Please note that for ease of readability we treat the path components as mere strings in the above formula; more precisely p should be replaced with something like $\text{tostring}(p)$.

The resulting overall dynamic cost function thus now looks like this:

$$\text{editcost}(\omega, p, q, N) = \begin{cases} \text{editcost}_{\text{gen}}(\omega_{\text{ins}}, n_{\text{ins}}) & \text{if } \omega = \omega_{\text{ins}} \\ \text{editcost}_{\text{gen}}(\omega_{\text{del}}, n_{\text{del}}) & \text{if } \omega = \omega_{\text{del}} \\ \text{editcost}_{\text{sub}}(p, q, n_{\text{sub}}) & \text{if } \omega = \omega_{\text{sub}} \\ \text{editcost}_{\text{gen}}(\omega_{\text{swp}}, n_{\text{swp}}) & \text{if } \omega = \omega_{\text{swp}} \end{cases} \tag{4.7}$$

To use it for scoring we still need to define a total cost function $\text{editcost}_{\text{total}} : \mathcal{P}_{\text{query}} \times \mathcal{P}_{\text{match}} \longrightarrow [0,1]$ which calculates the minimal cost for transforming a path $\mathcal{P}_{\text{query}} \subset \mathcal{P}$ into a path $\mathcal{P}_{\text{match}} \subset \mathcal{P}$ using an arbitrary sequence of operations in Ω. Lowrance and Wagner [LW75] have defined such a cost function in the context of string matching. It uses the same set of operations as our approach, so we can easily apply it to our problem by thinking of path components as characters. Their algorithm guarantees an algorithmic complexity proportional to $|\mathcal{P}_{\text{query}}| \cdot |\mathcal{P}_{\text{match}}|$, if the following condition holds for all $p \in \mathcal{P}_{\text{query}}$, $q \in \mathcal{P}_{\text{match}}$, and arbitrary N:

$$2 \cdot \text{editcost}(\omega_{\text{swp}}, p, q, N) \geq \text{editcost}(\omega_{\text{ins}}, p, q, N) + \text{editcost}(\omega_{\text{del}}, p, q, N) \tag{4.8}$$

In order to meet this condition, we introduce a custom cost function for swap operations (analogously to $\text{editcost}_{\text{sub}}$) which slightly refines the cost calculation for ω_{swp} as follows:

$$\text{editcost}_{\text{swp}}(p, q, N) = \max\left(\text{editcost}_{\text{gen}}(\omega_{\text{swp}}, n_{\text{swp}}), \frac{C_{\text{ins\&del}}}{2}\right) \tag{4.9}$$

with

$$C_{\text{ins\&del}} = \text{editcost}_{\text{gen}}(\omega_{\text{ins}}, n_{\text{ins}}) + \text{editcost}_{\text{gen}}(\omega_{\text{del}}, n_{\text{del}}) \tag{4.10}$$

To transform the total cost into a score s we can now simply define $s = 1 - \text{editcost}_{\text{total}}$. Hence these refinements complete our path matching solution (as far as path edit distances are concerned). It enables us to cope with SR-1 and SR-2 as well as the trivial case SR-0. The special case of equivalent modellings differing in terms of relevance (SR-3) still needs to be addressed, though. We can easily integrate SR-3, however, by increasing the base cost values and repetition factors shown in table 4.3. This increases the overall edit cost to match different paths and thus penalises different modellings more strongly. Which exact values are feasible depends on the scenario where the matching is performed. The differentiation of edit operations regarding the cost function and parameter values used provides us with a flexible framework to handle a broad range of scenarios.

Indexing

So far we have only addressed missing, excessive, and substituted path components as well as path inversion. The equivalence of element-based and attribute-based content modellings could also be handled in the same manner, that is, by path edit operations. This would require having operations to perform substitutions of path delimiters (e.g. substitute child axis by attribute axis) in addition to the path component-based operations discussed above. Enlarging the set of edit operations reduces the matching performance, though. Also, unlike the edit operations discussed above, substituting an element-subelement relationship by an element-attribute relationship (and vice versa) does not introduce the danger of matching arbitrary paths even when it is repeatedly applied to a path; thus we need no cost metric for this case. Therefore we prefer not to extend the set of edit operations, but instead model this aspect via index structures. To accomplish this, the index used should treat element-attribute relationships as element-subelement relationships. This is possible because we consider both modellings as equivalent and consequently our query language CAS-QL does not even allow to explicitly specify conditions on *attributes*; hence we do not need to distinguish the two. This approach is commonly employed to address this problem, e.g. in [TW02].

Handling element vs. reference equivalences also corresponds to the substitution of a path delimiter. This is non-trivial, however: Firstly, references (e.g. XPointer references [DJG+07]) may point to elements in different documents, thus we construct "virtual" paths which may cross document boundaries. Secondly, references may lead to cycles, so the resulting virtual paths may have an infinite length. In particular, if a path contains many references, matching it thus becomes very complex. To still provide a feasible solution we propose the following approximation: Like attributes, references are also handled at indexing time. The indexing logic follows a reference and treats the referenced element and its children as child elements of the referencing element. For example, an actual path `//books->book/author@name` would therefore be indexed as `//books/book/author/name`. Matching the "reference" thus obviously becomes trivial, but it comes at the cost of introducing redundancies in the indexed data. This may be troublesome in particular for referenced elements which have many descendants. We thus propose to restrict this indexing mechanism by introducing two parameters: A fixed maximum number of references resolved per path $N_{\text{max-refs}} \in \mathbb{N}^{\geq 0}$ and a maximum depth of descendant elements considered per reference $N_{\text{max-ref-depth}} \in \mathbb{N}^{\geq 0}$. If we set $N_{\text{max-refs}} = 2$, for example, only two references occurring in a path will be resolved, whereas all other references are ignored. The maximum depth $N_{\text{max-ref-depth}}$ controls how many levels of descendant elements of a referenced elements are redundantly indexed below the referencing element. When setting $N_{\text{max-ref-depth}} = 0$, only the referenced element itself is indexed redundantly, with $N_{\text{max-ref-depth}} = 1$ its direct children are also indexed redundantly, and so on. We assume, that in most scenarios paths used in a query are very short (even if we substitute the descendant axis by the corresponding child elements) and that they only contain few references in relation to other axes. Thus by employing the restrictions described, we believe that we can achieve a good result quality in most collections while keeping the index size down to a reasonable limit. If a collection contains very broad structures (i.e. elements have many children), we can additionally introduce a limit on the number of child elements considered per level; to limit complexity we restrain from doing so for the initial solution, however.

The problem of having an attribute value which corresponds to a tag name as shown in listing 4.2.1 consists of two major issues which we have to address: detection and interpretation. Detection means that we have to identify elements in the document collections used, which have an attribute related to their tag name; we will discuss this in more detail shortly. (For ease of expression we refer to such attributes as a *Tag Name Candidate Attributes* or TACAs, for short.) Independently of this, we can interpret TACAs either as corrective or additional information: In the first case, the attribute value of a TACA replaces the actual tag name of the element it belongs to; in the second case, we keep the actual tag name and use the TACA as additional metadata for matching. We believe that the corrective interpretation only applies to very few cases; for example, a generic element like `<entity>` may be assigned its semantics exclusively

Table 4.4.: Sample rules to identify TACAs

No.	Tag Name	Attribute	Value
1	publication	type	book
2	entity	type	*
3	*	type	person
4	*	class	*

by a TACA like "class" or "type". In most cases, however, the TACA only refines the actual tag name (e.g. in the example from listing 4.2.1). We therefore adopt the latter interpretation. To implement TACA detection we have two general alternatives at hand: indexing-time detection and runtime detection. If we perform the TACA detection at indexing time, we essentially need a manually defined rule set to base a heuristic on. This manual rule set may be a list of $\langle t, a, v \rangle$ tuples, for example, where t is a tag name, a an attribute name, and v the attributes value. To illustrate this, consider the rules listed in table 4.4. The first rule expresses the following: "All `<publication>` elements with an attribute named "type" whose value is "book" should be treated equivalent to `<book>` elements." The second and third rule use wild cards to be more flexible. The obvious drawback of this approach is that the manual definition of such rules is only possible in very confined scenarios with a small number of explicit schemas. Beyond that only very general rules can be generated which are likely error-prone. One such example is the fourth rule shown in table 4.4: It defines that all elements having an attribute named "class" are assigned their semantics by this attribute's value.

An alternative is to perform TACA detection at runtime, more specifically when matching tag names. The advantage of doing this is that the tag name to match is known (as it is defined in the query) so we do not depend on predefined rules. We thus further refine the cost function for our substitution operator ω_{sub} as follows: Let p be an arbitrary path component contained in the query which is to be substituted by a different path component q. Then $\text{editcost}_{\text{sub}}(p, q, n_{\text{sub}})$ will return a lower cost value, if q has an attribute whose value is p. We could combine the this solution with the rule-based solution discussed above. For example, only attributes with certain names may be considered. This would introduce the same difficulties as for index-time matching, however, so we restrain from doing so.

We expect this solution to perform well for a broad range of scenarios. Nonetheless, we need to conduct practical experiments to determine whether this solution is actually feasible in terms of performance and whether this approach is beneficial regarding retrieval quality. One aspect we still need to discuss are the requirements regarding feasible index implementations which result from our proposals: Earlier in this thesis we have proposed to simply store element-attribute relationships as element-sub-element relationships in the index. To fully support the ω_{sub} operation, however, the index will also have to indicate the axis type for each relationship; more precisely, the index has to track the relationship types element-sub-element, element-attribute, and element-referenced-element. Additionally, it has to provide a means for looking up the value of an attribute and these look-ups need to be fast. Please note that both requirements regarding feasible index implementations only exist, if the TACA logic is desired. For all other improvements we propose, an index ignorant to attributes and references is sufficient. Apart from these aspects, a suitable index implementation has to meet the requirement we have defined in section 2.3 regarding updates: The index must be able to handle insertions, modifications, and deletions of entire documents as well as fragments and it must be able to handle these operations at runtime without impacting the IR system's performance.

4.3. Implicit Structural Hints

So far we have only addressed explicit (that is, user-provided) query conditions. In this section we turn to implicit conditions and mainly discuss term proximities as one promising approach; also we briefly cover techniques based on element length.

4.3.1. Term Proximity

In flat Information Retrieval a common relevance indicator is *term proximity* [RS03]: Let A, B be two arbitrary terms used as query keywords, then informally their term proximity measures how far apart they appear in a document. For example, if the query is `information retrieval`, a document containing these terms right next to each other is usually assumed to be more relevant than one where both terms appear at very different positions. On a similar note, *term order* is often also considered: Matches containing "retrieval information" are likely less relevant than such containing "information retrieval".

Proximity Measures

We can distinguish two groups of proximity measures [TZ07], [Bei07]: distance aggregation measures and span-based measures. (The latter are often also referred to as interval-based methods.) Distance aggregation measures, calculate pairwise distances of the occurrences of terms used as keywords; span-based measures operate on sets of terms and calculate proximities related to the length of document parts containing all terms in a set. Both approaches generally apply to both flat retrieval and – with some adaptions – XML Retrieval. Based on experiments performed by Tao and Zhai [TZ07], we will give a short overview of several common measures of each group and how they perform in flat IR contexts.

Let K be the set of terms used as keywords in a query. In section 3.5.2 we have defined the function $\text{occ}(A, X) = \{a_1, a_2, \ldots, a_n\}$ to return all occurrences of a term $A \in K$ in an arbitrary set of term occurrences X. For term proximities we apply this function to the recursive full content of a document's root element, i.e. $\text{occ}(A, \text{cont}_{\text{rf}}(\text{root}(d)))$, to assemble the set of all term occurrences (including tag names, attributes names, etc.) in that document in document order. For ease of notation we write $\text{occ}(A, d)$

instead of $\mathrm{occ}(A, \mathrm{cont}_{\mathrm{rf}}(\mathrm{root}(d)))$. We also define the minimal distance function mindist : $K \times K \longrightarrow \mathbb{N}^{\geq 0}$ to calculate the number of terms separating the closest occurrences of two terms[9]. If a term does not occur in the document, mindist evaluates to the document length. For example, if $a_1, b_1, c_1, a_2, d_1, e_1$ are occurrences of terms $A, B, C, D, E \in K$ in a document, $\mathrm{mindist}(A, B)$ is 0 (as their closest occurrences are neighbouring) and $\mathrm{mindist}(A, E)$ is 1. Distance aggregation measures first calculate the minimal distance for each pair of terms; then an aggregation function is applied map the resulting set of distance values onto a single value. In [TZ07], the minimum, maximum, and average function are proposed for aggregation. For ease of expression, we will refer to these approaches (i.e. distance aggregation with the respective aggregation function) as MinAggDist, MaxAggDist, and AvgAggDist, respectively.

As opposed to distance aggregation measures, span-based proximity measures operate on the entire set K, i.e. not just pairs of terms. Two such measures are the Span measure and the MinCover measure [TZ07]: Span calculates the number of terms of the shortest interval in a document which contains *all* occurrences of each term in K; MinCover calculates the number of terms of the shortest interval in a document which contains *at least one* occurrence of each term in K. Thus, informally, Span also includes all repeated occurrences of terms whereas MinCover does not. To improve retrieval effectiveness, both measures can be normalised by the number of terms in the interval and the number of unique terms in the interval, respectively [TZ07]. We call the normalised versions of these measures NormSpan and NormMinCover.

Tao and Zhai [TZ07] state that a comparison of these measures on various TREC[10] test collections yields that the distance aggregation approach using the minimum function for aggregation (i.e. MinAggDist) has the strongest correlation with relevance judgements and performs best in terms of mean average precision (MAP). Other IR quality metrics have not been evaluated, though. (We discuss quality metrics in chapter 6.)

Proximities in XML Retrieval

A naive strategy to measure term proximities in XML Retrieval is to just ignore document structure and simply regard the recursive content of the root element as flat document. Any flat retrieval proximity measure can then be applied as-is and the scores it generates awarded either to the document itself or to the element to whose direct content a particular set of terms belongs. However, we believe such an approach to perform very poorly, because the physical distance of terms in an XML document (i.e. how far apart two terms are in document order) may differ strongly from their logical distance. Consider the sample XML fragment shown in listing 4.17; then the following issues have to be addressed by an effective XML term proximity measure:

- What is the proximity of two terms occurring within the direct content of an element, e.g. the proximity of `content` and `structure` in the example?

[9]Please note that Tao and Zhai [TZ07] define the minimum distance function slightly different: Their function always evaluates to mindist +1, i.e. neighbouring term occurrences have a distance of 1.
[10]`http://trec.nist.gov`

- What is the proximity of two terms, if one occurs in direct content of an element and the other within the direct content of one of that elements children, e.g. `retrieval` and `structure` or `XML` and `retrieval`? The latter case is of particular interest, because the terms "XML Retrieval" obviously are very closely coupled in the example.

- What is the proximity of two terms, if both occur in the direct content of different child elements of a single parent, e.g. `XML` and `structure`?

- What is the proximity of two terms, if one occurs in the recursive content of an element and the other in an attribute value of that element, e.g. `CAS` and `XML`?

- What is the proximity of two terms occurring in the direct content of adjacent siblings, e.g. the last word of one paragraph and the first word of the next? Here XML is particularly interesting, as two term occurrences can be very close regarding their "physical" position in a document, but very far apart considering the document's structure [EL00]. For example, the paragraphs mentioned above may even belong to different chapters and these chapters may cover completely different topics.

- What should the resulting score be awarded to? For example, if two terms occur in different elements, but have a high proximity, which element's score benefits from this?

```
<sec keywords="CO, CAS, introduction">
  <p>
    <link url="...">XML</link> Retrieval is commonly
       distinguished in <i>Content-only</i> and <i>Content and
       Structure</i> Retrieval.
  </p>
</sec>
```

Listing 4.17: Example of term proximities in XML

Related Work

A lot of publications on XML Retrieval *mention* the use of term proximities (e.g. [AYLP04], [FG04], [KMdRS06], [ST06]) and state that they believe it to be beneficial. However, only very few of these actually describe how proximities are determined and to what extent they are beneficial. The vast majority of these publications seem to apply one of the following two approaches: within-element term proximities or edge-counting. Within-element term proximities are proximity measures known from flat retrieval applied to either the direct content or the recursive content of an element without regarding the document structure at all. One such approach is described in [Feg04]: They apply the MinCover strategy to the recursive content of elements while ignoring the crossing of element boundaries. Edge-counting strategies ignore the position of term occurrences within an element's content, but *solely* operate on document structure. Most often they calculate the number of parent-child edges on the shortest path relating the elements which contain the term occurrences in question. For example, if the terms A and B occur within the same element, they have a count of zero, if A is in the parent element of B, they have a count of 1, and so on. Some proposal count the elements themselves, instead, or apply similar variations of this approach. Publications employing edge-counting-like strategies include [BW01], [CMKS03], and [AVF06].

We will now discuss some extensions proposed to these approaches. Abbaci et al. [AVF06, sec. III C] suggest to combine edge-counting with a semantic distance measure of tag names. Doing this their goal is to approximate the meaning of the document structure relating multiple terms. However, they have not yet devised a ranking model which actually takes this into account. Hristidis et al. [HPB03] also employ edge-counting, but include element references (e.g. IDREFs) as another axis besides the parent-child relationship. (Cycles are assumed not to exist.) Both axes are treated as equivalent, so the shortest path between two elements can include parent-child relationships, references, or both. Guo et al. [GSBS03] propose a two-dimensional proximity measure: One dimension describes the distance between two terms across different elements, the second dimension the distance of these elements to the result element which is returned. The latter is measured by counting the the number of parent-child edges that relate the element containing a term t and the result element which is actually returned to the user. When the element contains t itself, the count (denoted as c) is set to 1, if its parent contains t, c set to 2, and so on. Let e be a result element returned to the user, s_{init} its initial score, and o an occurrence of t in e or its descendants, then the proximity-adjusted score of that element is calculated by the following function:

$$\mathrm{s}(o, e) = s_{\text{init}} \cdot d^{(c-1)} \tag{4.11}$$

The so-called decay factor $d \in [0, 1]$ controls how strongly a decrease in proximity is penalised. If t occurs multiple times in e or its descendants, Guo et al. suggest to calculate e's score as the maximum of the scores of each occurrence, i.e. the best score is used.

Beigbeder [Bei07] adapts a distance aggregation measure to XML and takes into account the document structure. Their original measure models the so-called influence of term occurrences: Each term occurrence influences surrounding term occurrences up to a certain distance. The maximum influence is exercised at the position of the term itself and then linearly decreases to the left and right of the term until it reaches zero. Assuming that terms are used as conjunctive keywords, the document score is then calculated as the sum of the influence areas where occurrences of all keywords overlap. (We ignore the disjunction case here for the sake of simplicity.) To port this to XML, they assume a very simple document structure only consisting of nested sections and section titles and define the following rules: If a term occurs in the direct content of a section, its influence is limited to that section. If a term occurs in the direct content of a section title, its influence extends over the recursive content of the section it belongs to. The scores of elements are then calculated like in the flat retrieval case. This approach actually does take into account both term distances (in the narrow sense) *and* document structure. However, we believe it to be difficult to generalise this approach to arbitrary document structures and to cover the issues we have raised above.

Proximity Levels in XML

To our knowledge, none of the XML term proximity approaches proposed so far take into account the questions we have raised above. We therefore propose a novel strategy for term proximity handling in XML in this section. Namely, we propose to combine within-element proximity metrics and edge-counting metrics by introducing the concept of proximity levels. Before we define the term proximity level and describe our solution in detail, we first introduce several notational elements for ease of expression. Let

- $d \in \mathcal{D}$ be an arbitrary XML document,
- $A, B \in K$ with $A \neq B$ two terms used as keywords,
- $a \in \text{occ}(A, d)$, $b \in \text{occ}(B, d)$ occurrences of these keywords in d, and
- e_1, e_2 two not necessarily different elements in d with $a \in \text{cont}_\text{d}(e_1)$ and $b \in \text{cont}_\text{d}(e_2)$.

For the moment we also assume that $|\,\text{occ}(A, d)| = |\,\text{occ}(B, d)| = 1$, i.e. a and b occur only once in the document. A *proximity level* represents one particular hierarchical relation of a pair of term occurrences $\langle a, b\rangle$ in an XML document. Each pair of term occurrences belongs to exactly one proximity level and proximity levels are placed in a strict total order. Namely, we distinguish the following proximity levels (in the given order) in an XML document:

1. *Same Element*: $e_1 = e_2$, that is a and b belong to the direct content of the same element.
2. *Parent/Short Child*: e_2 is a child element of e_1 with a very short length. More precisely it holds that $|\,\text{cont}_\text{d}(e_2)| \leq M$, where $M \in \mathbb{N}^{\geq 1}$ is an arbitrary length limit, say 3.
3. *Adjacent Sibling*: e_1 and e_2 are both children of the same parent element p with e_1 being the i-th child of p and e_2 the j-th child of p with $j = i + 1$. For example, e_1 might a paragraph and e_2 the next paragraph.
4. *Non-adjacent Sibling*: This level equals the previous one with the only difference that $j \geq i + 2$.
5. *Parent/Long Child*: This level equals the *Parent/Short Child* level, only that $|\,\text{cont}_\text{d}(e_2)| > M$, i.e. e_2 is not short.
6. *Other*: None of the previous levels applies.

We do not explicitly include attribute names and values here. Instead, we treat attribute like child elements at indexing time as we have proposed in section 4.2.3. If our proximity level approach proves successful, refining attribute handling may be a viable tuning knob.

We now define a classification function $\mathrm{pl} : \mathrm{occ}(A, d) \times \mathrm{occ}(B, d) \longrightarrow L = \{1, \ldots, n\}$ which returns for two arbitrary term occurrences the number of their proximity level, with 1 being *Same Element*, 2 *Adjacent Sibling*, and so on. Now we define a proximity function $\mathrm{proxim} : \mathrm{occ}(A, d) \times \mathrm{occ}(B, d) \longrightarrow [0, 1]$; $\mathrm{proxim}(a, b) = 1$ denotes identity (i.e. $a = b$) and $\mathrm{proxim}(a, b) = 0$ denotes that either A, B or both terms do not occur in d at all. For this function it shall hold that

$$\forall a, b, c, d \text{ with } \mathrm{pl}(a, b) > \mathrm{pl}(c, d) : \mathrm{proxim}(a, b) < \mathrm{proxim}(c, d) \tag{4.12}$$

Informally, this means that a pair of term occurrences $\langle a, b\rangle$ which have a higher proximity level than another pair $\langle c, d\rangle$ always receives a lower score. One approximation of such a scoring function could be the following: Let the function $\mathrm{proximlb} : L \longrightarrow [0, 1]$ define lower bounds for the scores of each proximity level as follows:

$$\mathrm{proximlb}(l) = \frac{1}{l + 1} \tag{4.13}$$

This results in the $(0.5, 1]$ interval for scores of PL 1, $(0.33, 0.5]$ for PL 2, and so on. Thus proximities in low levels are weighted much stronger, than proximities in higher levels. Figure 4.2 visualises the resulting intervals. The idea behind this definition is to

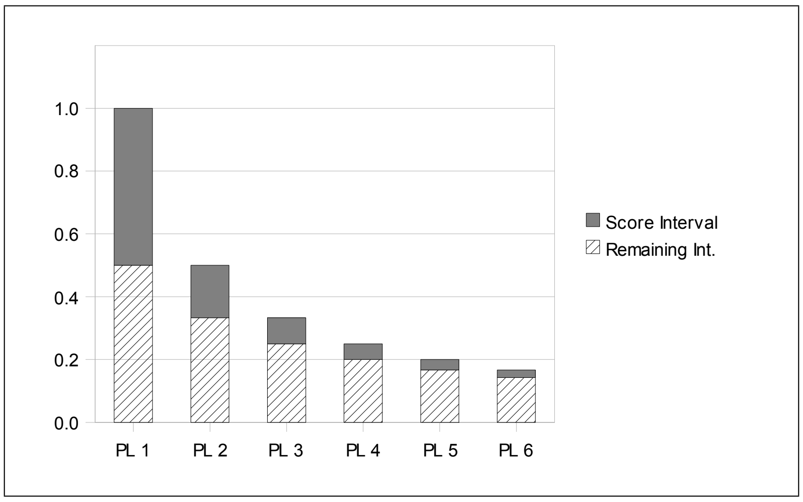

Figure 4.2.: Score intervals for proximity levels

perform coarse-grained scoring based on the above proximity levels while still enabling fine-grained scoring within each level. For example, within PL 1 proximities can be calculated like in flat retrieval by using the minimum distance aggregation function, whereas in PL 6 an edge-counting approach appears well-suited. The overall proximity function thus looks like this

$$\text{proxim}(a, b) = \begin{cases} \text{proxim}_{\text{PL1}}(a, b) & \text{if pl}(a, b) = 1 \\ \text{proxim}_{\text{PL2}}(a, b) & \text{if pl}(a, b) = 2 \\ \dots \\ \text{proxim}_{\text{PL6}}(a, b) & \text{if pl}(a, b) = 6 \end{cases} \tag{4.14}$$

where $\text{proxim}_{\text{PLi}}$ denotes a level-specific proximity function which returns results in the desired interval.

Now we drop the assumption that each term only can only occur once in a document, i.e. we now allow for $|\,\text{occ}(A, d)| \geq 1$ and/or $|\,\text{occ}(B, d)| \geq 1$. First of all, we then need to adapt the occ function to XML elements (instead of just documents). So let $\text{occ}(A, e)$ be a function which returns the set of occurrences of a term $A \in K$ in the recursive content of an element e ($\text{cont}_{\text{r}}(e)$). For ease of readability, also define the following auxiliary construct: Let $P_{\text{A,B}}$ be the set of proximities of all pairs of occurrences of A and B in $\text{cont}_{\text{r}}(e)$; more precisely: $P_{\text{A,B}} = \{\text{proxim}(a, b) : a \in \text{occ}(A, e), b \in \text{occ}(B, e)\}$. Now, we can easily define an aggregation-based proximity function for terms proxim : $K \times K \times \mathcal{E} \longrightarrow [0, 1]$ as follows:

$$\text{proxim}(A, B, e) = \begin{cases} \max(P_{\text{A,B}}) & \text{if occ}(A, e) \neq \emptyset \text{ and occ}(B, e) \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Informally, $\text{proxim}(A, B, e)$ calculates the term proximity of A and B as the proximity of their closest occurrences in e. This is analogous to the minimum distance aggregation (MinAggDist) discussed for flat retrieval. However, as we are concerned with fragment-oriented retrieval (as opposed to document retrieval), we need the fragment's root element e as an additional parameter besides A and B to define a scope. By setting $e = \text{root}(d)$, we can still obtain a term proximity for the entire document, though.

We now have a tool at hand that allows us calculate proximities for any pair of term occurrences $\langle a, b\rangle$ in an XML document. We have have also defined a way to aggregate this to a term-based proximity measure in equation 4.15 (i.e. to calculate the proximity of $\langle A, B\rangle$). To exploit this for XML Retrieval, however, we still need to define an according scoring mechanism.

Proximity-based Scoring

In this section we describe, how the proximity measure which we have devised above can be used for scoring. A naive solution is to adjust the document's score based on $\text{proxim}(A, B, \text{root}(d))$ for all $A, B \in K$. Doing this, we would only partially exploit the document structure, though: We would employ structure-aware proximities, but do document-level scoring. We thus need to devise a strategy how to assign element-specific scores based on proximities. One alternative to the naive solution is to calculate $\text{proxim}(A, B, e)$ for all keyword pairs and every element e and then adjust e's initial score based on the result. This is essentially a lowest common ancestor (LCA) approach (confer e.g. [LYWS06]), that is, relevance-enhancing properties of an element pair $\langle e_1, e_2\rangle$

are not attributed to the elements themselves, but to the first higher-level element which is an ancestor of both e_1 and e_2. This is fine, as long as e_1 and e_2 are not considered as possible result elements. If they are returned themselves, however, they do not benefit from term proximities[11]. As in XML Retrieval generally any element may become a result element, this solution is therefore not yet satisfactory.

We should thus devise a scoring approach which adjusts the scores of both e_1 and e_2. This can either be done symmetrically or asymmetrically. In the first case, both elements receive the same score. Albeit being a very simple solution, we argue that the containedness of elements should be considered: If neither element contains the other (i.e. because e_1 and e_2 are siblings), symmetrical scoring is appropriate. Otherwise, however, the element containing the other (say e_1) also actually contains (in its recursive content) both of the terms whose proximity is considered, whereas the other element does not. We thus propose the following asymmetric scoring function: Let $\mathrm{lca}(e_1, e_2)$ be a function which returns the lowest-common ancestor of $e_1, e_2 \in \mathcal{E}$ (and $\perp$, if the elements do not belong to the same document). We propose the following rules for scoring e_1, e_2 regarding the terms A and B:

- If e_1 and e_2 are identical, $e_1 = e_2$ receives the full proximity value of A and B in $\langle e_1, e_1 \rangle$ as proximity score.
- If e_1, e_2 are siblings, both receive half the proximity value of A and B in $\langle e_1, e_2 \rangle$ as proximity score.
- If e_2 is a descendant of e_1, e_1 receives the full proximity as score (as it contains both terms) and e_2 half the proximity.
- Vice versa, if e_1 is a descendant of e_2.
- In all other cases neither element contains the other (like in the sibling case), and thus e_1 and e_2 each receive half the proximity value as score.

The resulting scoring function $\mathrm{s}_{\mathrm{proxim}} : \mathcal{E} \times \mathcal{E} \longrightarrow [0,1] \times [0,1]$ is defined in equation 4.16; please note that for ease of readability we write $p(e)$ instead of $\mathrm{proxim}(A, B, e)$.

$$\mathrm{s}_{\mathrm{proxim}}(e_1, e_2, A, B) = \begin{cases} \langle \mathrm{p}(e_1), \mathrm{p}(e_1) \rangle & \text{if } e_1 = e_2 \\ \left\langle \mathrm{p}(e_1), \frac{\mathrm{p}(e_1)}{2} \right\rangle & \text{if } e_2 \in \mathrm{desc}(e_1) \\ \left\langle \frac{\mathrm{p}(e_2)}{2}, \mathrm{p}(e_2) \right\rangle & \text{if } e_1 \in \mathrm{desc}(e_2) \\ \left\langle \frac{\mathrm{p}(\mathrm{lca}(e_1,e_2))}{2}, \frac{\mathrm{p}(\mathrm{lca}(e_1,e_2))}{2} \right\rangle & \text{otherwise} \end{cases} \tag{4.16}$$

This way of scoring is consistent, because the full score is awarded, if an element contains both terms in its recursive content, whereas only half the score is awarded if an element only contains one term. Please also note that we do not need to consider the hierarchy

[11]The elements e_1 and e_2 may actually indirectly receive higher scores due to term proximities, if downwards score propagation (cf. section 5.2) is performed. This still does not guarantee a "fair" scoring regarding term proximities, however.

relation of e_1, e_2 (except for the differentiation of siblings and descendants), because it is already reflected in the proximity value itself. If in the same document as e_1 and e_2 there exists a third element e_3 (with $e_3 \neq e_1$ and $e_3 \neq e_2$), we need to calculate the proximity scores (for $\langle A, B \rangle$) for every pair of elements. In this case every element receives more than one proximity score for a single pair of terms, so we choose the *best* one as actual score.

As already mentioned above, term proximity scoring is innately context-aware and thus conflicts with the propagation of scores. Consider the following example to illustrate this: Assume that e_1 is the parent element of e_2; then one common score propagation mechanism is to reward e_1 a score bonus for the relevance of its child element. If both elements have received a higher score based on the proximity of a pair of terms they contain, and then e_2's score is propagated to e_1, e_1 will be overly rewarded due to score duplication. Thus we need to ensure that proximity-based scores are *not* propagated and therefore classify them as non-propagatable scores.

4.3.2. Length-based Heuristics

Apart from term proximities, another means to implement implicit support conditions are heuristics based on the length of elements. There are two major ways in which such techniques are currently used in XML Retrieval: One is to bias result selection towards elements of a certain length, the other is to apply heuristics based on element length (among other factors) to adjust elements' scores. We discuss result selection-related techniques in chapter 5 and thus concentrate on the latter kind in this section. Unlike for term proximities there already exist proposals for exploiting length information which we deem promising (cf. [Dop05], [RWdV06]). Therefore we only provide a brief overview of these proposals and then concentrate on how these techniques integrate with our retrieval framework. Also, we introduce the concept of Relevance Influence Factors as a minor improvement.

Related Work

All of the following length considerations are based on the recursive content length of elements, or more precisely the recursive content length of the fragment whose root element the element in question is (i.e. $|\,\mathrm{cont_r}(\mathrm{root}(f \in \mathcal{F}))|$). The adjectives "long" and "short" serve to indicate that the recursive content length is greater than some length threshold $M \in \mathbb{N}^{\geq 1}$ or less than or equal to this threshold[12], respectively.

Dopichaj [Dop05] proposes to identify patterns in a document tree based on the length, position, and score of elements. Two such patterns which have shown to improve retrieval quality are the title pattern in the inline pattern [Dop07]: The title pattern matches for long elements whose first child element is short and highly relevant. As the name

[12]Some proposals like [Dop07] distinguish more than two classes of lengths (e.g. "tiny", "short", "long") and also use more sophisticated means for classifying length values. This is orthogonal to the aspects which are of interest to us, however. Therefore we assume the use of a simple threshold value for ease of understanding and without loss of generality.

suggests, this is typically the case, for example, for `<section>` elements containing a `<section-title>` element. The inline pattern aims at long elements which contain several short, highly relevant child elements whose positions are distributed throughout their parent. This is intended to match `<paragraph>` elements, for example, which contain terms in special formatting such as `<i>` for italics. If one of these patterns matches, Dopichaj increases the score of the containing element (i.e. the `<section>` and the `<paragraph>`, respectively) and decreases the score of the child elements involved in the match (i.e. `<section-title>` and `<i>`).

Ramírez et al. [RWdV06] pursue a very similar idea (i.e. using small elements to adapt the scores of their ancestors), but chose a different path to implement it. Instead of a heuristic based on element length, position, and score, they perform statistical analyses of assessed document collections to derive rules for propagating short elements' scores: First, they manually select tag names of elements which are typically short. Then they identify per tag name which level of ancestors of an element with that tag name is most frequently relevant. (Level 0 is the element itself, level 1 its parent, and so on.) For example they identified that in the INEX 2005 test collection, figure captions (`<fgc>`) typically have a relevant grandparent, whereas section titles (`<st>`) have a relevant parent. Based on this, they establish rules (so-called *support links*) for increasing the score of elements based on the scores of short descendant elements.

The approach of Dopichaj [Dop05] has the advantage of not requiring the existence of collection-specific rule sets[13]. On the downside, his pattern detection mechanism involves the scores of elements, thus making it mandatory to separate pattern detection from the initial scoring; this is not a general problem, but in our model it implies that besides pre-propagation via non-propagatable scores and the propagation of direct scores (cf. section 5.2 for details) we essentially need yet another mechanism for propagation. Another disadvantage of this approach is that (for efficiency reasons) it only takes into account parent-child relationships as opposed to arbitrary ancestor-descendant relationships. The proposal of Ramírez et al. [RWdV06] features the complementary properties: It does require collection-specific rule sets, but it is independent of initial scores and allows to integrate arbitrary levels of ancestor-descendant relationships without impairing efficiency. Unlike the former approach, we can easily integrate their mechanism into the Relevance Influence Model which we introduce as part of our score propagation logic in chapter 5.

Relevance Influence Factors

One disadvantage equally applies to both proposals we have discussed in the previous section: They assume that all small elements equally affect the scores of surrounding elements. We expect, however, that the relevance of some small elements such as `<section-title>` and `<bold>` have a much stronger impact on the scores of surrounding elements than for example the relevance of e.g. `<small>` and `<footnote>`. Hence we suggest to introduce a *Relevance Influence Factor* (RIF for short) $r \in [0, 2]$ which

[13]We expect that patterns as defined in [Dop05] also depend on the collections used, albeit to a smaller degree.

Table 4.5.: Example of Relevance Influence Factor (RIF) assignments

Tag Name	r
`<title>`	1.5
`<bold>`	1.2
`<section>`	1.0
`<small>`	0.8
`<tiny>`	0.2

Table 4.6.: Example of an extended RIF rule set based on TACAs

No.	Tag Name	Attribute	Value	r
1	`<title>`	*	*	1.5
2	`<bold>`	*	*	1.2
3	`<section>`	*	*	1.0
4	`<small>`	*	*	0.8
5	`<tiny>`	*	*	0.2
6	`<font>`	color	red	1.5
7	`<font>`	color	orange	1.2
8	`<font>`	color	*	0.8

controls how strongly an element with a particular tag name influences the relevance of its ancestors. An element not matching any of the small element heuristics proposed in [Dop05] and [RWdV06] has $r = 1.0$ and thus no impact on surrounding elements' scores. An element like `<section-title>`, on the other hand, might be assigned a rather high RIF value like $r = 1.5$. Table 4.5 shows some sample RIF assignments to illustrate the idea. We multiply the RIF by the direct score of the element in question when propagating it. (For details on how we perform the actual score propagation see section 5.2.)

The RIF approach requires the existence a manually defined, collection-specific rule set. It therefore nicely integrates with the approach proposed in [RWdV06], but would destroy the main advantage of the approach proposed in [Dop05] (i.e. *not* requiring such a rule set). This is in line with the other compatibility criteria we have discussed in the previous section, so we recommend combining our solution with that proposed by Ramírez et al. [RWdV06]. In some cases the RIF definitions we have provided are not yet sufficient: Consider, for example, the element `<font color="red">`. Like for `<bold>` we assume it to have a positive impact on the score to be propagated. Unfortunately its tag name is neutral, that is, its impact on score propagation is solely based on a particular attribute value. To address a similar problem we have introduced the concept of TACAs (Tag Name Candidate Attributes) in section 4.2. We thus refine our RIF assignments into a TACA-based structure. Table 4.6 illustrates this based on the above examples.

4.4. Summary

In this chapter we have discussed the evaluation of explicit (i.e. user-provided) and implicit (i.e. system-provided) query conditions. As a foundation we have first introduced our concept for scoring XML fragments. The overall score of a fragment reflects the relevance of its root element's direct content (*direct score*), the relevance of other elements in its root element's context such its sibling and descendant elements (*context score*), and how well it matches target conditions (*target score*). The calculation of the context score is an activity in the result set generation phase which we discuss in chapter 5; for certain query evaluation techniques such as term proximities, the resulting scores innately reflect the relevance of context elements, however. Therefore we complement the direct score with a *non-propagatable score* which must not be propagated to other elements. We refer to the resulting logical structure as the *extended score tuple* which consists of the direct, non-propagatable, target, and context score.

When evaluating query conditions we have to face various problems most of which are due to *semantic relativism.* This means that we can model an arbitrary real-world phenomenon (e.g. an entity or a relationship) in different ways using XML. Thus conditions on the logical document structure have to match different *modelling variants* of the same phenomenon. Other problems include determining the semantic equivalence of names and coping with typographic errors in query conditions. Our solution for the evaluation of explicit query conditions is three-fold: Firstly, we integrate approximate string matching and substitution groups (i.e. rules for name equivalences) to address name matching; this integration is dynamic and uses name length as an indicator of which techniques are applicable. Secondly, we adapt string edit distances to XML paths to handle certain problems related to semantic relativism; this solution includes a fine-grained cost model to allow for scoring based on edit distances. Thirdly, we prose an index-based solution to handle the remaining problems such as references and attributes which influence the semantics of a tag name (*tag name candidate attributes*).

For implicit query conditions we have focussed on the adaption of term proximities to XML documents. We have analysed existing solutions for non-XML contexts and how they have been applied in the context of XML so far. As a result we classify existing approaches as within-element strategies or edge-counting strategies and propose an integrated solution based on *proximity levels.* Our solution approximates the logical proximity of terms in an XML document and provides a solid model for transforming proximity values into element-specific scores. Apart from term proximities, we have also briefly covered existing heuristics based on the length of elements and proposed a minor improvement in the form of *Relevance Influence Factors.*

5. Result Set Generation

Result selection determines which parts of the document collections are returned in answer to the user's query. In flat Information Retrieval, this activity is trivial: From a set of independently scored documents, the ones with scores greater than some threshold are returned. In XML Retrieval, however, one of the chief aims is to enable fragment-oriented retrieval as opposed to document retrieval. Result candidates are thus elements in document trees, so several challenges arise: Unlike documents, elements cannot be scored independent of each other, but strongly influence one another. For example, if an element is marked irrelevant (i.e. receives a score of zero), can its child elements then be relevant? Also, as a consequence of their nesting, result candidates *contain* other result candidates which is the reason why we have introduced the concept of XML fragments in addition to XML elements. This leads to the question, whether overlapping results may be returned, and – if not –, how to select the best result from a set of overlapping ones. Finally, the user can declare particular elements as desired results by including target conditions in his query.

We now refine the XML Retrieval process to provide a conceptional framework for addressing these issues. Remember that in the query evaluation phase which precedes result selection content and support conditions are evaluated. After query evaluation every element in the element space has been assigned a score tuple $S_{\text{EST}} = \langle \mathrm{s}_{\text{direct}}, s_{\text{np}}, s_{\text{tgt}}, \mathrm{s}_{\text{ctx}} \rangle$ which consists of a direct score $\mathrm{s}_{\text{direct}}$, a non-propagatable score s_{np} (which may be null), a target score, and a context score. Thus the input to use for result selection consists of these tuples and the target conditions which the user has supplied. We divide the result selection phase into three activities:

1. **Target Candidate Determination:** We identify which elements in the element space are potential elements of the result set to return to the user. This decision is based on the evaluation of target conditions and/or implicit information such as statistics, rule sets, and so on.

2. **Score Propagation:** For every element in the element space we calculate its context score based on the direct scores of its context elements and then its overall final score. During score propagation, we use the set of result candidates as guidance as to which elements are preferably contained in the result set.

3. **Result Selection:** We apply predefined rule sets to determine which elements are actually returned to the user based on the scores resulting from the previous activities. These rules may include aggregation and pruning techniques which, for example, remove overlapping elements.

In the remainder of this chapter, we will discuss each activity in detail. Our aim is to devise a consistent solution for selecting result elements.

5.1. Target Candidate Determination

In flat Information Retrieval, target candidates are typically not needed as the IR system simply returns the most relevant documents. XML Retrieval, on the other hand, enables the user to specify which fragments he deems useful by using target conditions. If no user-provided target conditions are available, the IR system either has to generate target conditions automatically or operate solely based static rule sets to select the right fragments to retrieve. We will not cover automatic target condition generation in this thesis as our focus lies on user-provided target conditions. As a fallback in case no target conditions have been provided by the user, however, we will ensure that our scoring mechanism still enables our result selection logic to work; we accept that without target conditions a lower retrieval quality is achieved, though. In the following sections we will discuss our interpretation of target conditions as well as the preliminaries for target condition-based scoring. Based on this we then devise a strategy for scoring target conditions.

5.1.1. Interpretation of Target Conditions

Like all conditions, we can, of course, interpret target conditions either strictly and vaguely. Strict evaluation means that only elements matching at least one[1] target condition may be returned. Vague evaluation, on the other hand, treats target conditions as mere hints which do not have to be observed. As already discussed in section 2.1.1, we believe that strict evaluation is infeasible in most scenarios, because the user has little or no knowledge of the actual document structure and thus can only provide hints of limited reliability. In particular, we must therefore ensure that an element not matching any target condition (i.e. an element with a target score of zero) can be still contained in the result set.

The kind of target condition interpretation we are chiefly concerned with in this section is a different one, however. We differentiate two general interpretations of target conditions: One focuses on granularity, the other on content. A granularity-focussed condition expresses how "fine-grained" the user prefers result items to be, but does not impose any restrictions on the content of result items. (We will discuss the notion of granularity in more detail, shortly.) A content-focussed condition complements this notion: It does not aim at fragments of any particular granularity, but expresses a

[1]Even for the strict evaluation of target conditions, there are several alternative implementations: Besides the common "at least one" semantics, it might be desired that all target conditions are fulfilled or that a combination of logical operators determines which conditions must be fulfilled. The latter two variants conflict with our definition of target conditions, however: If a target condition comprises no path information, but only a tag name (or granularity), having a single element match several target conditions is not possible.

preference towards fragments with a certain content. Our query language CAS-QL supports two ways of specifying a target condition: by means of a tag name (e.g. `tgt:section`) and by means of a granularity indicator (e.g. `tgt:0.5`). The latter one obviously corresponds to the granularity-focussed interpretation. The former one, however, can actually express either one interpretation depending on the type of the tag name that is used. In the subsequent sections, we will discuss these element type dependencies in detail and then devise a metric for measuring granularity. Before that will now briefly address the issue of Semantic Relativism (cf. section 4.2.1) in the context of target conditions.

For non-content-focussed conditions, we can safely ignore Semantic Relativism: It may indirectly be of interest, because varying XML modellings of a real-world phenomenon can cause variations of the document structure. These variations only have a minor impact on the granularity measures we define below, however. If we use tag-name-based target conditions, on the other hand, together with the content-focussed interpretation, Semantic Relativism is of concern. For example, the elements which correspond to a target condition (e.g. `tgt:section`) may have been modelled as attribute values. In our example, an element `<struct type="section">` should match our condition just like `<section>`. Nonetheless we do not have to specifically address Semantic Relativism at this point, as the approaches to handle it are exactly the same in the context of target conditions as they are for support conditions.

Element Type Dependencies

The semantics of a tag name-based target condition depend on the element type of target elements. If a user asks for `<abstract>` elements in a collection of books, he defines a particular desired granularity *and* a particular kind of content: He wants to retrieve fragments with several paragraphs in length at most (granularity) which give a concise overview of a piece of writing (content). Having the IR system include `<section>` elements in the result set is likely unwanted, although they may have the desired granularity, because a section does not feature the content properties of an abstract. If he asks for `<section>` elements, on the other hand, the user will probably also want to receive long subsections and short chapters. Thus he only defines a granularity condition. We therefore believe, that the right way of interpreting a target condition depends on the on the type of the respective element (cf. section 3.4). We thus derive handling rules for the different element types below. As an element may belong to more than one type, these rules must not contradict each other.

Entity elements (like `<author>`) may be selected as target elements by the user or the IR system. They do not hint towards granularity, but have strong implications on their content. Thus relevant candidate fragments are all semantically related tag names. Matches are penalised based on the semantic distance of their tag name to the requested tag name. For example, if the user requests `<author>` elements, `<author>` and `<aut>` elements might be matched with a score of 1.0, `<person>` elements with 0.7, and `<vehicle>` elements with 0.0. If the target element is a structuring element (e.g. a `<section>`), we believe that it hints towards a particular granularity as opposed to particular contents.

More precisely we claim that if a target condition references a tag name which is only classified as a structuring element (i.e. an element that neither describes an entity nor a particular formatting), this condition only expresses the preference for elements of one particular granularity. The relevance of a fragment is thus judged based on how closely it matches the desired granularity, whereas the semantics of its tag name are ignored. For example, if the target condition is `tgt:sec`, it does not matter if the result is a `<sec>`, `<section>`, `<scene>` or `<clause>` element (provided that all of these are only structuring elements and that all have the same granularity). It is often observed in practical experiments, however, that it does, in fact, matter which element is returned albeit all elements have the same granularity (cf. [LR04, p. 433], [KMdRS05, p. 7]). We believe that this is due to the rareness of sole structuring elements: i.e. all `<section>`, `<scene>`, and `<clause>` can additionally be classified as entity elements and thus are related to their content. In these cases, both granularity and semantic distance should thus be used for scoring. Another aspect we need to discuss is the following: One fundamental assumption we make regarding the granularity of element types is that every structuring element actually corresponds to *one* particular granularity, although it may, for example, occur at different positions in the document hierarchy. For example, we claim that although a paragraph may be located below a chapter as well as a subsection, it always has roughly the same granularity. If this assumption actually holds or if further refinement of element types is necessary, we need to determine by practical evaluations as described in part III of this thesis.

Formatting elements (like `<b>`) are only of limited use for target element selection; they do not correspond to a particular granularity (like structuring elements) or a particular kind of content (like entity elements). One of the rare cases in which using them as target elements appears useful is the Re-finding use case described in section 2.3.2. If formatting elements are used in target conditions, however, vaguely matching them is very hard: Relevant elements are other formatting elements which lead to a similar formatting result – a fact which we believe is almost impossible to detect for an IR system. As this may also apply to not directly formatting-related elements like `<title>`, they should also be classified as formatting elements. For example, if the user requests `<b>` elements, `<b>`, `<textbf>`, and `<title>` might have a score of 1.0, `<i>` and `<large>` a score of 0.5 and `<text-normal>` a score of 0.0. We thus propose a two-fold solution: If the environment of the IR system permits this, the system administrator can manually define groups of substitutable formatting elements (i.e. substitution groups). This is the case, for example, in a corporate intranet environment with a very limited number of explicitly available XML schemas. For generic tag names such as `<font>` whose substitutability depends on attribute values (e.g. `<font color="red">` vs. `<bold>`), we propose to extend our substitution group approach by using TACAs analogous to our RIF extension described in section 4.3.2. Table 5.1 illustrates an according TACA-based substitution group for the formatting element examples we have used above. If the environment does not allow for manually-defined substitution groups (e.g. in a heterogeneous environment), we propose to use semantic distances as a fallback. For example, the IR system may substitute `<small>` by `<small-font>` or `<tiny>` (with a small penalty) at a reasonable level of confidence. We do not believe, however, that

Table 5.1.: Example of a substitution group for formatting elements

No.	Tag Name	Attribute	Value	Score
1	`<b>`	*	*	1.0
2	`<bold>`	*	*	1.0
3	`<bold-font>`	*	*	1.0
4	`<font>`	color	red	1.0
5	`<font>`	color	*	0.8
6	`<title>`	*	*	0.8
7	`<i>`	*	*	0.6
8	`<italics>`	*	*	0.6
9	`<subtitle>`	*	*	0.4

textual similarity measures (e.g. edit distances) are beneficial for formatting elements, as formatting tag names are typically to small to guarantee satisfactory results. To illustrate this, consider an element `<b>`, for example, which is a lot closer to `<s>` (for "strike-through font") in terms of edit distances than to `<bold>`.

Granularity Measures

In order to evaluate target conditions aiming at the granularity of results, we need a metric to define and measure granularity. We require this metric to have at least an interval scale (i.e. interval or ratio) to enable quantifiable distance calculations. In order words, we want to be able to calculate how much more fine-grained an element e_1 is compared to an element e_2.For ease of notation we define a granularity function $\text{gran} : \mathcal{E} \longrightarrow \mathbb{R}^{\geq 0}$ which calculates the granularity of an arbitrary element $e \in \mathcal{E}$ in an arbitrary, but linear and deterministic way. Based on that, we define a normalised granularity function $\text{gran}_{\text{norm}} : \mathcal{E} \longrightarrow [0, 1]$ which enables comparison of distance values across different collections; $\text{gran}_{\text{norm}}(e) = 1.0$ means that the element is very fine-grained, whereas $\text{gran}_{\text{norm}}(e) = 0.0$ represents a very coarse granularity. Let $e \in \mathcal{E}$ be an arbitrary element and $G_{\max}$ the maximum granularity of all elements in the same collection as e; then we calculate e's normalised granularity as follows[2]:

$$\text{gran}_{\text{norm}}(e) = \frac{\text{gran}(e)}{G_{\max}} \tag{5.1}$$

Because we require our granularity metrics to be linear and can normalise it to the $[0, 1]$ interval, we can now define a feasible distance function $\text{grandist} : \mathcal{E} \times \mathcal{E} \longrightarrow [0, 1]$ in a very simple manner as shown in the following equation:

$$\text{grandist}(e_1, e_2) = \text{abs}(\text{gran}_{\text{norm}}(e_1) - \text{gran}_{\text{norm}}(e_2)) \tag{5.2}$$

We now discuss several alternative implementations of the gran function:

[2]We assume that the minimum granularity is zero in all collections and that we can obtain the maximum granularity per collection (e.g. by regular generation of collection statistics).

- *Specificity:* One way of defining how fine-granular a fragment is consists in using its specificity: A fragment which covers (almost) exclusively topics which are of interest to the user would then be fine-grained, whereas fragments also covering other topics are coarse-grained. This may lead to very large fragments being returned, as, for example, an entire book may also be very specific. An advantage of using the specificity is that it can be applied to documents which are varying widely regarding their structure. Because a user explicitly requesting a section is unlikely to regard a book as relevant, though, we consider specificity an inappropriate granularity measure.

- *Absolute Hierarchy Depth:* Another granularity indicator may be how deep an element is located in the XML tree of its document. A subsection, for example, is usually located beneath a section and thus more fine-grained. For some elements (e.g. paragraphs) this does not always hold, as a `<paragraph>` element may be the child of a `<subsection>` as well as of a `<chapter>`. Also the number of elements which are not directly related to a hierarchy, but located above the elements of interest in the document tree (like `<mainmatter>` and `<body>`, for example) may vary across documents. This holds, in particular, when the we consider multiple collections.

- *Relative Hierarchy Depth:* To improve the applicability of hierarchy depth as a granularity measure across different document collections, we can slightly modify it: Instead of counting hierarchy levels starting at the document root element, we can define custom hierarchies on arbitrary sets of elements. For example, the elements `<chapter>`, `<section>`, `<subsection>`, and `<subsubsection>` may be defined as forming a hierarchy (in the given order) with `<chapter>` as the hierarchy root; thus `<chapter>` would be located at depth 0, `<section>` at depth 1, and so on. This still leaves us with the problem of how to handle elements which are related to granularity, but not necessarily to a hierarchy. This is the case, for example, for the `<paragraph>` element mentioned above.

- *Average Recursive Content Length:* A very simple yet elegant solution is using the recursive content length of elements to measure their granularity. (For the definition of an element's recursive content length cf. section 2.2.) It is based on the assumption that for every element we can statistically determine an average length which we assume to be relatively stable even across different document collections (maybe corrected by some constant factor per collection). The recursive nature of our length definition guarantees that the recursive content length of an element is always greater than the recursive content length of any of its descendants. Thus the average recursive content length of an element with a hierarchical nature is greater than that of other elements deeper in the hierarchy. So, by using the average element length, we can define a non-total order of all elements (or optionally only hierarchical elements) which integrates well with our distance metric (i.e. the normalised difference between two elements' average lengths).

5.1.2. Target Scoring

We refer to the scoring of elements based on target conditions as *target scoring*. The resulting score is the target score s_{tgt} which we have already introduced as a constituent of our extended score tuple S_{EST} in section 4.1. Unlike scores resulting from support conditions, target scores must never be propagated, because otherwise they might interfere with result selection in an incorrect manner. To illustrate this, consider the following example: The system processes a query containing the condition `tgt:paragraph`. To highlight the relevant effect, we ignore non-target conditions. The documents used follow a fairly regular structure as shown in listing 5.1. Then, obviously, the `<paragraph>` elements are most relevant regarding the target condition and thus assigned the highest scores (say 1.0) whereas other elements receive lower scores (say 0.0). A common score propagation strategy is to propagate score from child elements to their parent, optionally down-weighted by some constant factor α (say 0.8).

```
<article>
  <body>
    <section>
      <paragraph>...</paragraph>
      <paragraph>...</paragraph>
      <paragraph>...</paragraph>
    </section>
    <section>
      <paragraph>...</paragraph>
      <paragraph>...</paragraph>
      <paragraph>...</paragraph>
    </section>
  </body>
</article>
```

Listing 5.1: Simplified excerpt from an XML document

For the sake of simplicity, we use the formula shown in equation 5.3 to calculate a parent element's score. It has been proposed by Gövert et al. in [GAFG02, sec. 2]; we will discuss it detail in section 5.2 among other, more flexible score propagation approaches. Also, we ignore score normalisation (i.e. the scores calculated below are not necessarily in the $[0, 1]$ interval) for the same reason.

$$\mathrm{s}(p) = \mathrm{s}_{\text{direct}}(p) + \sum_{e \in C} \alpha \cdot \mathrm{s}(e) \tag{5.3}$$

If we apply this strategy to the example document, then the `<section>` elements are scored with 2.4 and thus higher than the `<paragraph>` elements, and the `<body>` element is again scored higher then the `<section>` elements. As we can see, target conditions reward particular elements (e.g. elements with a certain granularity) and particularly *not* their ancestors (they would even imply to penalise ancestors); score propagation, on the other hand, rewards the ancestors and thus overrules and contradicts the intention of target scoring. Hence we only apply score propagation to scores resulting from content and support conditions and not for target scores.

Related Work

Most XML Retrieval proposals handle target scoring analogously to content and support scoring without distinction. We consider this as insufficient as target conditions (and thus the resulting scores) play a special role in result selection. Also, we have introduced granularity as a main concept in the handling of target conditions and consequently need to account for it in target scoring. Some proposals regard target scoring as an integral part of score propagation (e.g. [FG04], [Gev05], [SHB05]); we will discuss these proposals separately in section 5.2. Finally there exist a number of proposals which consider element length as a major factor in result selection. Albeit not necessarily being implemented by means of target scoring, we consider these approaches to be – at least to some extent – related to our idea of granularity. Therefore we first discuss the proposal of Kamps et al. [KMdRkS04] as one representative of length-based techniques before we describe our own solution.

Kamps et al. [KMdRkS04] have found that element length plays an important role in XML Retrieval: While, in the collection analysed, the number of very small elements (i.e. elements containing only a few words) is far greater than that of long elements, the probability of an element to be relevant increases overproportionally to the element length. This results in a strong unbalance in retrieval results, if elements of all lengths are considered equally relevant. They propose two alternative approaches to take this into account: an adaption of the scoring model and a restriction of the index. In the first case, they use a language model which is biased towards generating higher relevance probabilities for long elements. This could also easily be adapted to a scoring method not based on language models, e.g. by assigning a score bonus for long elements. The alternative approach is to restrict the element index used to only contain elements above a certain length threshold. Kamps et al. experimentally compared both approaches with the result that the scoring model adaption performs far better than the index cut-off and that the combination of both approaches yields the best results. We will refer to this combined strategy as *static length biasing*.

However, the length/relevance correlation has only been studied so far by analysing the manual relevance judgements conducted as part of the annual INEX workshop. To our knowledge, no broad research has yet been conducted to verify, if these findings apply to XML Retrieval in general. We believe, that there are contexts where either different or no length biases apply. For example, in the Re-finding use case described in section 2.3.2 the user is not likely to favour longer result fragments. Apart from the environmental context, we also think that a length bias is only applicable to particular element types: If the user searches for an entity element like `<author>`, its length supposably plays a minor role. We argue that we can regard element length as one particular representative of element granularity. Hence, the element type dependencies for result selection which we have described in section 5.1.1 do also apply here. In consequence, we believe that static length biasing should not be applied to other elements than structuring elements. On top of this restriction, two other issues still remain to be solved: The first one is to find the optimal length bias for an arbitrary collection and ideally doing so without manual involvement; the second one is the benefit of length biasing when target conditions are specified by the user. As for the second issue, static length biasing even counteracts the aims of the user: By specifying a (structuring) target condition he requests elements of a particular granularity, so a length bias would not be helpful. We therefore consider this approach (and other length-based approaches) to not be feasible for our settings.

Solution Strategy

Our strategy for target scoring is based on the simple idea to score elements based on how closely they match target conditions. Thus we require a mechanism to calculate the distance of an arbitrary element to a target condition. In the previous section we have defined a normalised granularity function $\mathrm{gran}_{\mathrm{norm}}$ which maps a tag name to a granularity value in the $[0, 1]$ interval. Thus to obtain an element's granularity, we can simply apply $\mathrm{gran}_{\mathrm{norm}}$ to its tag name. So, we only need to map a target condition to a granularity value in order to calculate their distance using the grandist function. If the target condition to evaluate is specified in the granularity format (e.g. `tgt:0.4`), we can use this value straight away. If the condition uses the tag name-based format, on the other hand, we calculate its granularity as the average granularity of that tag name across all collections involved. The target condition is matched more closely, the smaller the granularity distance is. Thus obtaining a target score is very simple: Let $g_{\mathrm{tgt}} \in [0, 1]$ be the normalised granularity of a target condition and $g_{\mathrm{elem}} \in [0, 1]$ the normalised granularity of the element we want to match against it; then we calculate the element's target score using the following function[3]:

$$\mathrm{s}_{\mathrm{tgt}}(g_{\mathrm{tgt}}, g_{\mathrm{elem}}) = 1 - \mathrm{grandist}(g_{\mathrm{tgt}}, g_{\mathrm{elem}}) \tag{5.4}$$

If the query contains several target conditions, we use the *best* target score. The reason behind this is that unlike content and support conditions, target conditions generally exclude each other[4]. For example, an element can either closely match `tgt:main-matter` or `tgt:abstract`, but not both conditions. Hence we treat all target conditions disjunctively and consequently for each element only regard the best-matching one. As grandist implementation we use the average recursive content length metric as it best suits our needs.

We have discussed in the previous section that for certain element types a target condition not only expresses the preference for a particular granularity, but also the preference for a particular content. Namely, we believe this to be the case for entity and formatting elements. Hence, for elements of these types, we calculate a semantic distance measure in addition to the granularity distance. To approximate the semantic distance

[3]We implicitly assume to have a grandist function which operates directly on granularity values. Equation 5.2 can easily be adapted accordingly.

[4]An exception is a query containing a tag name condition and a granularity condition which both correspond to the same granularity. Such a query essentially contains redundant conditions which we therefore ignore.

of an element's content to a target condition, we calculate the semantic distance of its tag name and the tag name used as target condition by an according distance function semdist : $\mathcal{T} \times \mathcal{T} \longrightarrow [0,1]$. As semdist implementation we propose to use common approaches such as WordNet [MBF+90] as we have described in section 4.2. Obviously, we only calculate the semantic distance, if there exists at least one tag name-based target condition. If there exists more than one such condition, we use the best semantic distance as score analogously to the granularity distance. To integrate the semantic distance in the above scoring function, we subtract the semdist value from 1 and combine it with the score from equation 5.4 by an arbitrary convex weighting scheme.

As we have stated above already, we believe that our strategy complements static length biasing: If the query contains at least on structuring target condition, that serves as an indicator of the result granularity the user desires for his present information need; our approach handles this more precisely than the statistics-based static length biasing and can be seen as a dynamic length biasing approach (provided we use a length-based granularity measure as suggested above). If the query contains no structuring target condition, but any other target conditions, static length biasing is undesired as we have discussed above, whereas, our combination of granularity distance with semantic distance enables effective handling of these conditions as well. If no target conditions are defined, however, and if the collection used is known to have a length/relevance correlation, we expect static length biasing to be preferable over our approach.

5.2. Score Propagation

Score propagation aims at calculating a context score for every element in the element space based on the elements in its context. We strictly separate this from the decision which elements to actually include in the result set. Besides reducing the complexity, our motivation for doing so is that we believe that each of these two activities pursue different goals: Score propagation is solely aimed at obtaining a reasonable overall relevance approximation for each fragment in the document collection. Result selection decides based on these relevance assignments how to construct a result set which suits the user's needs. In this decision non-relevance-related aspects play a major role (such as the removal of overlap or aggregation of results to make them handier for the user). So we *can*, of course, combine both score propagation and result selection in a single model, but this increases the model's complexity while greatly decreasing its flexibility. Hence the strict separation.

In this section will first provide an overview of related work. Most of this work does not observe the proposed separation of score propagation from result selection, but contains interesting ideas; thus at the end of this overview we will classify these ideas in such related to score propagation and such related to result selection. After that, we will briefly discuss the generalities of designing a score propagation concept and then introduce our propagation model.

5.2.1. Related Work

In XML Retrieval literature, a wealth of strategies for score propagation has been proposed. In this section, we will discuss a subset of these proposals which we believe to be a representative sample. We classify score propagation approaches in *downwards propagation strategies* (only ancestors are considered when calculating an element's score), *upwards propagation strategies* (only descendants are considered), and *bi-directional propagation strategies* (both ancestor and descendant elements are considered).

A very simple downwards propagation strategy is used by Theobald and Weikum [TW00] in their XXL retrieval engine. They obtain the context score of an element by multiplying all its parent elements' scores by the element's own score. Assuming that scores are in the $[0, 1]$ interval, this implies that a parent element is *always* assigned a score greater than or equal to that of any of its child elements. This contradicts the aim of fragment-oriented retrieval, however, as the root element always receives the highest score [Gev04], [GAFG02]. Arvola et al. [AKJ05] propose four alternative downwards propagation approaches (so-called contextualisation schemata). They all calculate an element's context score by using the weighted average of its own score and the scores of a subset of its ancestors. They differ, however, in which ancestors they use: The first schema uses the root element, the second one the parent element, the third one all ancestors, and the fourth one both the parent and the root element. Although offering more flexibility, this approach (regardless of the contextualisation schema used) still favours the root element over its descendants and therefore opposes fragment-oriented retrieval. To address this problem, Gövert et al. [GAFG02] employ an augmentation factor $\alpha \in [0, 1]$: When a score is propagated from an element e to its parent p, it is multiplied by α and then added to the parent's direct score. Thus a parent element is still positively affected by the scores of its children, but it is only scored higher than them, if it has either a high direct score or if several of its children are relevant. An according scoring function is shown in the following equation where $\mathrm{s}_{\mathrm{direct}}$ returns p's direct score:

$$\mathrm{s}(p) = \mathrm{s}_{\mathrm{direct}}(p) + \sum_{e \in \mathrm{chld}(p)} \alpha \cdot \mathrm{s}(e) \tag{5.5}$$

Many approaches to score propagation are rule-based. Piwowarski and Lalmas [PL04], for example, state that if an element is relevant, its parent must (to some extent) be relevant as well. Geva [Gev05] renders this idea more precisely by defining the following two rules: If an element has only a single relevant child element, the child element is ranked higher than the parent; if, on the other hand, an element has more than one relevant child element, each of the child elements is ranked lower than its parent. An actual implementation of such rules is provided by the augmentation factor approach discussed above: The augmentation factor controls the number of relevant children needed for the parent element to receive a higher score, i.e. the greater the augmentation factor, the less relevant children are needed. Another rule set is described in [MM03]. It is aimed at selecting the best result candidates instead of propagating scores, but the rules can easily be applied for propagation as well. Most rules are similar to the ones described above; however, they additionally introduce the following interesting rule (called `SingleChild`): If most relevant descendant elements of an element e are descendants of a single one of e's children, this child is scored higher than both e and all of e's other children[5]. This aims at elegantly handling concentrations of relevant elements in one particular subtree. Sauvagnat et al. [SHB05] calculate direct scores only for leaf nodes in a document tree. This is based on the (dangerous) assumption, that only leaf nodes contain textual content; nonetheless, this approach stresses the importance of score propagation, as score propagation serves as the sole means for obtaining non-leaf node scores. Their propagation function calculates the score for a non-leaf node e mainly based on the following rules:

- The more leaf nodes with non-zero relevance e has, the higher its score is.
- The larger the distance (counted as parent-child edges) between e and a descendant leaf node l is, the less l contributes to e's score.
- The smaller the content length[6] of l is, the more it contributes to e's score (i.e. short leaf nodes are propagated more strongly).

They also consider document relevance when propagating scores. Doing so is a special case of downwards propagation. As we will discuss shortly, we believe downwards propagation to have a negative impact on result quality rather than a positive one.

The works we have presented contain numerous interesting ideas. On the downside, they do not distinguish between score propagation and result selection. As a consequence, they often do not follow a clear strategy as to which factors to use to model propagation and how these factors are combined. In the following section we thus start out with a superset of potential influencing factors, based on which we then devise our score propagation approach.

5.2.2. Relevance Influence Model

If we generalise and supplement ideas stated in related works, we obtain the following list factors which potentially influence score propagation:

- For both e and each element $c \in \mathrm{ctx}(e)$:
  - its direct score
  - its direct content length
  - its recursive content length
  - its element type
- For each element $c \in \mathrm{ctx}(e)$:
  - c's distance to e

[5]Please note that we have slightly simplified the original rule and adapted it to score propagation.

[6]For leaf nodes the content length equals the recursive content length, so we can substitute the (implicit) length definition in [SHB05] by either one of our definitions.

- The number of e's relevant child elements C; here we have several alternatives at hand to define C:
  - all child elements with a direct score greater than or equal to some absolute threshold $T_{\text{abs}} \in [0, 1]$, that is $C = \{c \in \text{chld}(e) : \text{s}_{\text{direct}}(c) \geq T_{\text{abs}}\}$
  - all child elements with an direct score greater than or equal to some relative threshold $T_{\text{rel}} \in [0, \infty)$ regarding e's direct score, that is $C = \left\{c \in \text{chld}(e) : \frac{\text{s}_{\text{direct}}(c)}{\text{s}_{\text{direct}}(e)} \geq T_{\text{rel}}\right\}$
- The number of e's relevant child elements compared to the number of all of e's child elements

The main factors we integrate in our score propagation model are the direct scores of context elements and their distance to the element in question. (The use of direct scores is a direct consequence of the scoring framework which have introduced in section 4.1; the importance of distances we will explain in detail shortly.) Content lengths and element types are potential candidates for fine-tuning the model; until our baseline model has proven its benefit in experimental evaluations, we restrain from integrating them, however, to keep the complexity as low as possible. The various factors concerned with the number of relevant children aim at controlling the result set composition based on existing scores. Therefore they are prime candidates to be considered in result selection, but not in score propagation. With these decisions in mind, we now define our *Relevance Influence Model* (RIM for short) for score propagation based on the following two principles:

1. Score propagation is generally performed upwards, not downwards.
2. The further apart (in terms of the document hierarchy) an element is from an element in its context, the less its relevance is influenced by it.

The first principle is based on the idea that XML structures feature containment hierarchies: All descendants of an element e are actually contained in this element. When we return e as part of the result set, all of its descendants are innately also returned. Thus a decision to include e in the results should also be based on the relevance of the elements it contains. Downwards propagation, on the other hand, (that is, allowing for e to influence its descendants relevance) lacks sufficient justification. Why should, for example, a chapter which may have little or no direct content be used to adjust the scores of sections and paragraphs contained therein? Remember that we have already handled specific cases where such behaviour is actually of use (e.g. the propagation of title element scores to their respective content elements) by means of pre-propagation. Also, downwards propagation comes at the price of diffusing relevance scores – which defeats the purpose of scoring (i.e. to provide a clear distinction of elements in terms of their relevance). The rationale behind the second principle is twofold: Firstly, the hierarchy provides us with strong indications of how closely related two elements are semantically. If, for example, a `<chapter>` element contains a `<paragraph>` nested deep down in its descendant tree, we believe that this paragraph should have a weaker impact on the chapter's relevance than a paragraph which is its direct child element. Secondly, we believe that the user is less likely to look at a descendant the further down it is "hidden" in the descendant tree.

Due to our decision for upwards score propagation, we can choose from two general strategies to design our model: global and local strategies. A global strategy calculates the propagation score for an element $e \in \mathcal{E}$ based on the direct scores of the the entire set of e's context elements. A local strategy, on the other hand, calculates a propagation score based on the *context* scores of its neighbouring elements. Consider the example XML document tree in figure 5.1 to illustrate this. For simplicity, let us assume that the context elements only contain the descendant elements. Then to calculate the propagation score for an element a, a global strategy considers the direct scores of all other nodes in the document tree, namely b, c, …, g. A typical local strategy, on the other hand, first assigns the context scores of the leaf nodes d, e, f, and g to equal their direct scores. Then it calculates b's context score based on the context scores of d and e and c's context score based on f and g. Finally it obtains a's score based on the context scores of b and c.

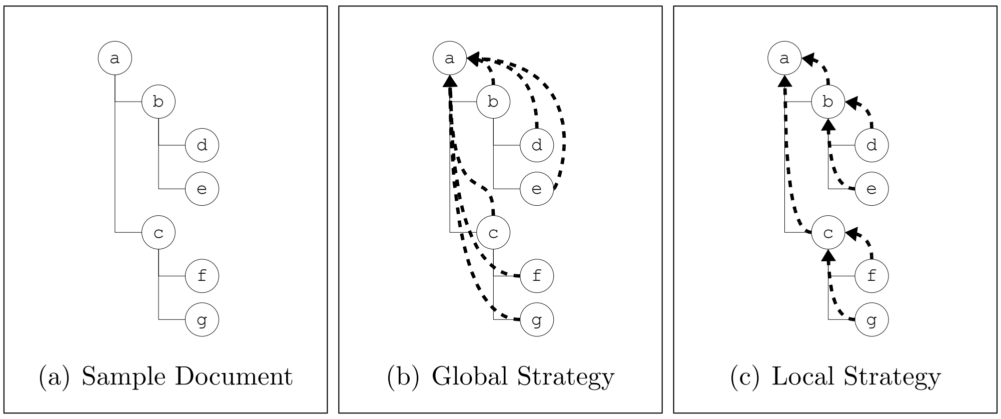

Figure 5.1.: Score propagation strategies

The main advantages of local strategies are computational efficiency and the reduction of complexity (by only considering a very small subset of context elements). This only works, though, if the context only consists of a single axis (e.g. only descendants). If all context element axes (i.e. descendants, ancestors, siblings, etc.) need to be considered, however, they either need several passes through the document tree or must be combined with global strategies which both decreases efficiency and increases complexity. Another disadvantage of local strategies is that distance-based decay factors are hard to implement: If, for example, for an element e the influence of a different element $c \in \text{ctx}(e)$ on e's propagation score shall decrease, the higher the distance of e and c in the document hierarchy is, this is non-trivial to compute locally. Global strategies can accommodate advanced scoring features (like distance-based decay factors) more easily. Implementing them efficiently is more challenging, however. We thus resort to a global strategy to devise our propagation concept, whereas implementations of this concept should presumably use local variants instead.

To account for the distance principle, we need fine-grained control over how strongly elements in the context influence the relevance of an element e. This is supported by our decision to use a global propagation strategy. We base the RIM on descendants (both children and other descendants) and siblings which is only a subset of an element's context elements: The ancestors are not included following the first principle. For simplicity reasons, we do not include referencing and referenced elements, either. In chapter IV we will briefly outline, though, how references can be used to refine our model. The siblings should also be excluded when strictly applying the containment paradigm. Nonetheless, we still include siblings in the RIM context as the following seems to be a reasonable heuristic: If an element's sibling is relevant, there is a great likelihood of that element being also relevant. Whether this heuristic really works, only practical experiments can show, however.

We define a function $\text{ctx}_{\text{RIM}} : \mathcal{E} \longrightarrow \mathcal{E}^n$ to return the subset of context elements used by the RIM, that is all descendants and siblings. This may still leave us with a huge set of elements, however, to consider when calculating an element's context score. (Think of the root element, for example.) To increase computational efficiency, we thus need to limit this set even further. Here we can exploit the second principle: As the influence of the elements in $\text{ctx}_{\text{RIM}}(e)$ decreases with decreasing proximity of elements to e, we can introduce a proximity threshold t_{RIM} beyond which context elements are ignored. (We will discuss shortly how to measure proximities and how to choose this threshold.) We redefine $\text{ctx}_{\text{RIM}}(e)$ to only return descendants and siblings with a proximity greater than or equal to this threshold and refer to the set of elements returned by $\text{ctx}_{\text{RIM}}(e)$ as e's *influence region*[7].

We employ the proximity model which we have proposed in the context of term proximities as a conceptional foundation for reasoning about how strongly a context element should affect the context score. Therefore we adapt the pl function defined in section 4.3.1 to operate on elements (as opposed to term occurrences); we refer to the adapted function as pl_{RIM}. We can then define an element proximity function for to arbitrary elements $e_1, e_2 \in \mathcal{E}$ in accordance to equations 4.13 and 4.14 as follows:

$$\text{proxim}_{\text{RIM}}(e_1, e_2) = \begin{cases} \text{pl}_{\text{RIM}}(e_1, e_2)^{-1} & \text{if } \text{pl}_{\text{RIM}}(e_1, e_2) \leq 5 \\ \text{proxim}_{\text{RIM,6,norm}}(e_1, e_2) & \text{otherwise} \end{cases} \tag{5.6}$$

This results in 1, if e_1 and e_2 are on proximity level 1 (identical), 0.5, if they are on level 2, 0.33 for level 3, and so on. For level 6 (descendant which is no child), we use a more fine-grained calculation:

$$\text{proxim}_{\text{RIM,6}}(e_1, e_2) = \frac{1}{\text{edgecount}(e_1, e_2)} \tag{5.7}$$

The auxiliary function $\text{edgecount} : \mathcal{E} \longrightarrow \mathbb{N}^{\geq 0}$ calculates the number of edges separating two elements. Thus the above function returns a higher value the smaller the edge count

[7]Formally, we define $\text{ctx}_{\text{RIM}}(e)$ to return the set $\{c \in \text{ctx}(e) : \text{proxim}_{\text{RIM}}(e, c) \geq t_{\text{RIM}} \in [0, 1)\}$. For the definition of $\text{proxim}_{\text{RIM}}$ see below.

is; to integrate with equation 5.6, this value must be normalised to the $(\frac{1}{7}, \frac{1}{6}]$ interval. We omit the normalisation to ease readability, but indicate the fact by calling the function $\mathrm{proxim}_{\mathrm{RIM},6,\mathrm{norm}}$. (Please note that the edge count for two elements on proximity level 6 is always at least 2; thus the above function is always defined and the normalisation should assume $\frac{1}{2}$ as the maximum value.)

We can now easily define the influence threshold as an arbitrary value $t_{\mathrm{RIM}} \in [0, 1)$. To fully exploit the above proximity features, the value should be chosen from the $(\frac{1}{7}, \frac{1}{6})$ interval which includes children, siblings, and non-child descendants up to a maximum edge count. The function to calculate the context score of an element e is the following:

$$\mathrm{s}_{\mathrm{ctx}}(e) = \frac{\sum_{c \in \mathrm{ctx}_{\mathrm{RIM}}(e)} \mathrm{s}_{\mathrm{direct}}(c) \cdot \mathrm{proxim}_{\mathrm{RIM}}(e, c)}{\frac{1}{2}|\,\mathrm{ctx}_{\mathrm{RIM}}(e)|} \tag{5.8}$$

This sums up the direct scores of all elements in e's influence region (down-weighted by their proximity to e) and then normalises the resulting score to the $[0, 1]$ interval. Please note that as e is not included in its own influence region, the maximum score any element in $\mathrm{ctx}_{\mathrm{RIM}}(e)$ can obtain is 0.5 (as the elements have at least proximity level 2). Therefore we normalise the result by *half* the length of $\mathrm{ctx}_{\mathrm{RIM}}(e)$ as opposed to the full length to ensure that the resulting scores can not only be in $[0, 0.5]$, but actually in $[0, 1]$.

5.2.3. Final Score Computation

After all conditions (including target conditions) have been evaluated and score propagation has determined how strongly an element's score is influenced by its context, the final scoring step is to compute an overall score for each element. Remember that we have an extended score tuple $S_{\mathrm{EST}} = \langle \mathrm{s}_{\mathrm{direct}}, s_{\mathrm{np}}, s_{\mathrm{tgt}}, \mathrm{s}_{\mathrm{ctx}} \rangle$. Before score propagation, only the first three components have been defined, whereas $\mathrm{s}_{\mathrm{ctx}}$ is only defined after score propagation has been performed. Please also recall that s_{np} is defined, but may be null ($\perp$), if the element in question does not have a non-propagatable score. In the following we devise a model to compute the final score based on S_{EST}. This model strongly depends on the other activities in the result selection phase and therefore includes target scoring, score propagation, final score computation, and result selection as individual activities. So, conceptionally, it is not a final score computation model, but rather an overall model for the entire result set generation phase. Hence we will assume now that we are actually at the *start* of the result selection phase, that is, we have $S_{\mathrm{EST}} = \langle \mathrm{s}_{\mathrm{direct}}, s_{\mathrm{np}}, \uparrow, \uparrow \rangle$.

For clarity, we start out with an initial model and then in a second step refine it into our actual model. Our initial model comprises the followings steps:

1. *Computation of target scores*: For this, any of the target scoring means described in section 5.1.2 can be employed.

2. *Preselection of target candidates*: All target elements with $s_{\mathrm{tgt}} \geq t$ are selected where $t \in [0, 1]$ is an arbitrary threshold. We refer to these elements as *target candidates*. The idea behind this is to reduce the large number of elements which

match a target condition to some extent. We do this for two reasons: Firstly, the efficiency of the subsequent steps increases, the smaller the set of target elements is; secondly, this enables us to configure the strictness of the IR system regarding target conditions. (The extreme cases are setting $t = 1$ which only allows for perfect target matches and setting $t = 0$ to allow for any element to be included in the result set.)

3. *Score propagation*: For each target candidate e, we identify its influence region. Then we perform score propagation, that is, we calculate e's context score $\mathrm{s}_{\mathrm{ctx}}$. The context scores are only calculated for target candidates. Because of the threshold described above, we expect this to significantly reduce the cost of score propagation.

4. *Calculation of final scores*: We compute a final score s_{final} for the target candidates based on their extended score tuples. There are two variants of how to do this: The first one is to base the final score on the direct score, non-propagatable score, and context score, only. This essentially excludes target conditions from relevance scoring, so their purpose is reduced to the preselection of result elements. The second variant is to use all scores (i.e. including the target score) in the computation. We believe that (at least tag-name-based) target conditions do have an influence on the relevance of an element's content, apart from controlling its inclusion in the result set. Hence we consider the latter variant as preferable.

5. *Result selection*: Finally, we perform result selection based on the set of target candidates (as opposed to the entire document tree). In this step, another threshold may be applied to filter the remaining elements which is based on the final score. In this case, eventually only such elements remain which have a certain minimum target score and a minimum final score.

The above model reflects the special role of target conditions compared to other conditions. However, it does not fulfil our requirement that an element which does not match *any* target condition can still be contained in the result set. Also, it only works, *if* the user has provided target conditions in his query. We hence call this model the *authoritative model* to express the role target conditions play. As a refinement, we propose the following *semi-authoritative model*. It is designed in such a way that it is still biased towards elements matching target conditions, but that elements with very high non-target scores can still make it into the result set regardless of their target scores. The semi-authoritative model is composed as follows:

1. *Computation of target scores* (Analogous to the authoritative model.)

2. *Preselection of target candidates (optional)*: We do not want to confine the result set to elements matching target conditions, only. Thus we obviously cannot apply the target score-based preselection of the authoritative model. If, on the other hand, we do not perform *any* preselection, we expect score propagation to become a lot more expensive, as it has to be preformed for every element in the document

tree. Therefore we propose a heuristic approach which selects all elements which are either above a target score threshold (like in the authoritative model) or have a high direct or non-propagatable score (which we model via two additional thresholds). This is based on the assumption that elements where these (readily available) scores are high have a high likelihood of also having a high final score. There are cases which this presumably does not work well: For example, a `<chapter>` element which has a target score of zero, little direct and non-propagatable scores (because it does not have any direct content), but features many highly relevant child elements. In this case, the score would be boosted strongly by score propagation without the other scores giving an indication of this beforehand. We believe, however, that by adjusting the various thresholds we can find a feasible configuration for most scenarios. If the environment really does not permit the heuristic proposed, we can omit preselection at the cost of the score propagation becoming computationally more expensive.

3. *Score propagation* (Analogous to the authoritative model)

4. *Calculation of final scores*: We calculate the final score s_{final} based on the entire extended score tuple, i.e. including the target score. The focus of this step lies on the influence of target conditions in relation to all other conditions. To underline this, we first for each element combine its direct, non-propagatable, and context scores into a single score called *content score*. This combination can adhere to arbitrary weighting schemes as discussed in section 3.5. For example, the convex weighting scheme shown in equation 5.9 illustrates this (with $\alpha, \beta, \gamma \in [0, 1]$ and $\alpha + \beta + \gamma = 1$).

$$s_{\text{content}} = \alpha \cdot \mathrm{s}_{\text{direct}} + \beta \cdot s_{\text{np}} + \gamma \cdot \mathrm{s}_{\text{ctx}} \tag{5.9}$$

We then calculate the final score as

$$s_{\text{final}} = \begin{cases} 0 & \text{if } \delta \cdot s_{\text{content}} = 0 \\ \delta \cdot s_{\text{content}} + (1 - \delta) \cdot s_{\text{tgt}} & \text{otherwise} \end{cases} \tag{5.10}$$

with $\delta \in (0, 1)$. The condition ensures that a fragment with a zero content score also receives a zero final score regardless of whether it matches target conditions. This is analogous to the ideas we have discussed in section 3.5 regarding condition weighting. Which δ values lead to good results we will have to determine by practical test runs; however, we consider δ values around 0.75 promising, as they result in the following scoring behaviour:

- With a target score of zero, the maximum score that an element can reach is 0.75.
- An element with a target score of 1.0 and a content score of 0.33 will receive a score of ≈ 0.5; so does an element with a content score of 0.66 and a target score of zero.

5. *Result Selection* (Analogous to the authoritative model.)

Ergo, in the semi-authoritative model for an element with a target score of zero it is more difficult to become a result element than for an element with $s_{\text{tgt}} > 0$. Unlike in the authoritative model, it is still *possible*, however.

5.3. Result Selection

After all scores have been computed, we must decide which fragments to return to the user in answer to his query. We refer to these fragments as the *result set* $\mathcal{R}$. As the result set is typically ordered descendingly by the scores of the fragments' root elements, we also refer to the result set as the *ranking*. In this section we will first provide an overview of related work and then propose a custom result selection logic aimed at integration and configurability.

5.3.1. Related Work

The most straightforward way to perform result selection is to select the elements with the highest final scores just like in flat Information Retrieval. There are two variants of this approach: We can select all elements e with a score greater than some threshold; formally: $\mathcal{R} = \{e \in \mathcal{E} : \mathrm{s}(e) > s_{\min}\}$ where $s_{\min} \in [0, 1)$. By setting $s_{\min} = 0$ we can include all relevant elements in the result[8]. Additionally, we can restrict the result set to a fixed maximum size by returning only the k most relevant fragments (with $k \in \mathbb{N}^{\geq 1}$) which is commonly known as the Top k approach (cf. section 2.3).

In XML Retrieval, these means are insufficient due to the nesting structuring of XML documents. Nonetheless, XML Retrieval literature typically does not discuss result selection as such. Instead publications address specific aspects related to it, often without distinction from target scoring and score propagation. One such aspect which is discussed in numerous publications is overlap removal. We classify the solution approaches for overlap handling in two categories: overlap handling by score adaption and overlap handling by filtering the result set. To illustrate this, we briefly introduce a representative of each category: Clarke [Cla05] proposes a technique to control the overlap of fragments in the result set. He assumes that overlap is per se undesired. He does not remove overlapping elements entirely, however. Instead he decreases the scores of elements which are contained in other elements that are a part of the ranking. Sigurbjörnsson et al. [SKdR04] rank the result set according to relevance; then they optionally process it sequentially from top to bottom by removing all fragments which either contain or are contained in other fragments that are located higher in the ranking. Thus they select the most relevant fragment from each set of overlapping fragments.

[8]To keep the result selection algorithms as simple as possible, our modelling does not allow to only include perfect matches in the result (i.e. setting $s_{\min} = 1$). We believe that this is a very rare case. If it should indeed be needed, the definition of $s_{\min}$ and and the subsequent algorithms can easily be altered accordingly, though.

A second aspect which is often discussed is result set compression. It selects the best representative fragments for each subtree in a document. It complements overlap handling as it primarily aims at handling non-overlapping fragments (e.g. siblings) and situations where overlap is allowed. We have already introduced the according mechanisms when discussing work related to our score propagation approach. In particular, preferring a parent element over its children based on their count and relevance (as in [Gev05]) and vice versa (e.g. [MM03]) are such proposals. We will thus not discuss them in detail again here.

5.3.2. Result Selection Logic

We believe that the related work discussed above introduces various ideas which are helpful in performing result selection. However, to our knowledge none of the proposals actually integrates all of the aspects we have come across. Also, most publications assume one particular environment and hence only allow very limited configuration of the system's behaviour. For example, [Cla05] assume that overlap is always undesired and [Gev05] always replace a set of relevant siblings with their parent element. We therefore identify individual aspects of result selection behaviour and then integrate them into a single, freely configurable solution. We refer to these aspects as *result compaction* techniques. Namely, we distinguish the following three aspects:

Overlap Removal If *overlap removal* is enabled, it ensures that the result set does not contain any overlapping fragments. That is, the result set must not contain any two fragments whose root elements have an ancestor-descendant relationship. Formally: $\forall f_\mathrm{p} \in \mathcal{R} : \nexists f_\mathrm{c} \in \mathbb{R}$ with $\mathrm{root}(f_\mathrm{c}) \in \mathrm{desc}(\mathrm{root}(f_\mathrm{p}))$. Whether overlap removal is desirable, depends on the scenario [KLdV04], [Dop06][9].

Aggregation *Aggregation* means, that instead of returning several relevant child elements of the same parent, only the parent element is returned. In the Book Search use case this is desired, for example: If most sections belonging to a chapter are relevant, the user wants the chapter to be retrieved instead of the individual sections. In contrast to that, in the Re-finding use case the user wants to retrieve the individual sections and *not* the chapter – unless, of course, the chapter is relevant itself.

Specialisation *Specialisation* is the opposite to aggregation: If a parent element has one child element which is substantially more relevant than both its parent and its siblings, specialisation returns this child element instead. Again, this may be useful for Book Search (to point the user to those parts of the book which provide the highest benefit), but not for Re-finding. Please note that although aggregation and specialisation are exclusive regarding a single fragment, they can (and should)

[9]Dopichaj [Dop06] finds that in a practical application he analyses overlap is not removed and thus asks whether users really mind overlap in real-world scenarios. Our use cases suggest that there are even scenarios where overlap is actually desired.

both be supported by a result selection algorithm; the decision which one to use should then be made on a per-fragment basis. Nonetheless we need to be able to configure, if and when each technique may be used.

We propose an algorithm which integrates all of the above techniques (overlap removal, aggregation, and specialisation) as well as the general features introduced at the beginning of the previous section (relevance threshold, Top k processing). Its pseudocode is shown in listing 5.2. The algorithm defines a function which returns a set of elements based on the following input variables:

- An arbitrary fragment's root element $p = \mathrm{root}(f \in \mathcal{F})$.
- The *relevance threshold* s_{min}: Only elements with a score higher than this threshold are considered relevant.
- The *set of child elements* $C = \mathrm{chld}(p)$.
- The *set of relevant child elements* $C_{\mathrm{rel}} = \{c \in \mathrm{chld}(p) : \mathrm{s}(c) > s_{\mathrm{min}}\}$. Please note that C and C_{rel} are only auxiliary constructs that may, of course, be calculated by the algorithm based on the other variables. We include them as parameters, however, to keep the pseudo code as simple as possible.
- The *aggregation threshold* $t_{\mathrm{agg}} \in (0, 1] \cup \{\perp\}$: If the ratio of p's relevant child elements to all of p's child elements is greater than this threshold (i.e. $|C_{\mathrm{rel}}| \cdot |C|^{-1} \geq t_{\mathrm{agg}}$), then p's fragment is contained in the result set instead of its children's fragments and regardless of p's own score. If we set $t_{\mathrm{agg}} = \perp$, no aggregation is performed. Please note that if the aggregation rule applies, even the fragment of an element with a score *below* the relevance threshold (i.e. $\mathrm{s}(p) \leq t_{\mathrm{agg}}$) may be included in the result set.
- The *specialisation threshold* $t_{\mathrm{spec}} \in (0, 1] \cup \{\perp\}$: If there exists a child element c of p which has a score that is far greater than that of both p and all of c's siblings, c is returned instead. Formally we can define this condition as $\exists c \in C \forall d \in (C \cup \{p\}) - \{c\} : t_{\mathrm{spec}} \cdot \mathrm{s}(c) \geq \mathrm{s}(c)$. This means that when we set $t_{\mathrm{spec}} = 0.2$, for example, c's parent and siblings must have less than 20% of c's score for specialisation to occur. Again, we can disable this feature by setting $t_{\mathrm{spec}} = \perp$.
- An *overlap removal switch* o: If we set o to `TRUE`, overlap is to be removed, otherwise overlap is allowed.

The algorithm must be applied bottom-up (i.e. starting with the leaf elements) to all element in the the element space.

```
/**
 * p      := Arbitrary element
 * smin   := Relevance threshold
 * tagg  := Aggregation threshold
 * tspec := Specialisation threshold
 * o     := TRUE, iff overlap removal is desired
 * C     := Set of all child elements of p
 * Crel  := Set of child elements of p with a score > smin
 */
Set getPartialResultSet(p, smin, tagg, tspec, o, C, Crel)
{
  // first of all, check if there is any element we can
      specialise to
  var smax = 0;      // highest child score
  var snext = 0;     // second-highest child score
  var cspec = null;  // child with highest score
  if (tspec != null)
  {
    for (c in Crel)
    {
      // find specialisation candidates
      if (score(c) > smax)
      {
        snext = smax;
        smax = score(c);
        cspec = c;
      }
    }

    if (mult(tspec, smax) < snext || mult(tspec, smax) < score(p
       ))
      // specialisation threshold is not fulfilled
      cspec = null;
  }

  if (score(p) <= smin)
  {
    // p is not relevant, so there is no overlap

    if (count(Crel) == 0)
      // no relevant children: return empty set
      return {};
    else if (div(count(Crel), count(C)) < tagg)
      // only few relevant children: no aggregation needed
      return Crel;
    else
      // many relevant children: force aggregation
      return {p};
  }
  else
  {
    // p is relevant, so there may be overlap

    if (count(Crel) == 0)
      // there are no relevant children: return parent
      return {p};
    else if (cspec != null)
      // specialisation to relevant child possible: specialise
      return {cspec};
    else if (div(count(Crel), count(C)) < tagg)
    {
      // not enough relevant children to aggregate
      if (o == true)
        // overlap must be removed: force aggregation
        return {p};
      else
        // no compaction necessary: return parent and children
        return union({p}, Crel);
    }
    else
      // we are above the aggregation threshold: aggregate
      return {p};

  }
}
```

Listing 5.2: Result selection algorithm

5.4. Summary

In this chapter we have discussed the result set generation phase of the XML Retrieval process. We divide this phase into three activities: target candidate determination, score propagation, and result selection. Target candidate determination identifies which elements the user might want to see included in the result set. To make this decision, we evaluate target conditions (*target scoring*) and optionally consider implicit information such as document statistics. Target conditions we can either interpret as content-focussed or as granularity-focussed. The former notion uses tag names to reason about an element's likely semantics and content; the latter notion requires granularity metrics to assign granularity values to elements and query conditions. Which interpretation is feasible depends on the element type among other factors.

Score propagation calculates an element's context score based on its context elements. We have introduced the *Relevance Influence Model* (RIM) to define how an element's score is influenced by the direct scores of other elements and hence how score propagation is to be performed. The RIM is based on the principles that only upwards score propagation is performed (i.e. we ignore ancestor elements) and that the influence of a context element decreases as its proximity to the element in question decreases. To implement this, we have reused the proximity level approach which we have introduced in the previous chapter. After score propagation all scores in the extended score tuple (that is, the direct, non-propagatable, target, and context score) have been calculated. Consequently, we then transform the score tuple into a single overall score called *final score* for each element. For this transformation we have devised a semi-authoritative model which favours elements with high target scores, but also assigns non-zero final scores to elements not matching any target condition.

Based on the final scores we then decide which fragments to return to the user in the result selection activity. How result selection is to be performed strongly depends on the assumed user model which varies widely across different scenarios. Hence our result selection logic (RSL) features several parameters to enable broad configurability. Apart from a relevance threshold to control the minimum relevance of result items and a maximum result size, our RSL supports three result compaction techniques: overlap removal, aggregation, and specialisation. *Overlap removal* ensures that no element in the result set contains any other result item as its descendant. *Aggregation* returns a parent element instead of several relevant children. *Specialisation* returns a single highly relevant child element in place of its parent and siblings. To demonstrate that these techniques can indeed be integrated, we have finally provided the pseudo-code of a sample RSL implementation.

Part III.

Evaluation

6. Evaluation Framework

Throughout this thesis we have devised a conceptional framework for XML Retrieval. Based on this, we have analysed to what extend the various aspects are covered by state of the art solutions and proposed custom solutions for further improvement. However, our solutions are only conceptional constructs so far which still lack practical evaluation to prove their effectiveness. The focus of this part is thus to specify a strategy how this evaluation should be performed. (Due to the limited scope of this thesis we have to leave the implementation of this strategy to subsequent works.) To achieve this, we first discuss the foundations of evaluating XML Retrieval systems and devise an evaluation framework suitable for our needs in this chapter. Equipped with this, we then lay out an actual evaluation plan for our improvement proposals in the following chapter.

6.1. Evaluation Approaches

There are several ways in which we can evaluate the proposals made in this thesis. A central question in this context is whether to evaluate a composite solution which comprises all improvement proposals or to evaluate the aspects one by one. The former approach enables us to judge whether the individual aspects integrate seamlessly and lead to an overall improvement in retrieval quality. The latter approach, on the other hand, enables us to analyse in detail how each individual aspect affects an IR system. Obviously both results are necessary. We expect, however, that a fair amount of fine tuning is needed before the individual improvement proposals actually lead to the desired effect. Therefore we concentrate on aspect-by-aspect evaluation for now and aspire toward experiments with composite solutions only in the long run.

For aspect-centred evaluation we consider two non-exclusive techniques: benchmarking and statistical correlation testing. *Benchmarking* means that we compare a baseline IR system implementation with an implementation that is identical except that it has been enhanced by a particular improvement feature[1]. For example, we may compare the results of an IR system not exploiting term proximities to a modified version which does. This obviously requires a functional IR system implementation which we can access at the source code level and a metric to measure result quality. Also, a suitable collection of test documents and an according set of IR queries are needed. By statistical *correlation testing* we check whether a particular variable correlates with another variable at a statistically significant level. In the above example, we might check whether the relevance

[1]Obviously we can also benchmark one IR system implementation against a different one. This generally only provides rather vague pointers to the likely *cause* of the observed benchmarking result so we do not consider this helpful in our context.

values assigned to elements in a test collection positively correlate with the proximity of terms used in the query. This enables us to predict the improvement potential of the corresponding improvement aspect or analyse the cause of effective improvements, respectively. The advantage of this technique is that it does not require a fully functional IR system implementation; instead we can apply it to mere data such as a test collection and its corresponding relevance assessments, potentially with the help of tools such as a parser or parts of an IR system implementation. Unlike benchmarking, though, we cannot actually measure factual improvements by testing correlations. In our evaluation strategy we will use both benchmarking and correlation testing.

6.2. Measuring XML Retrieval Quality

In order to assess our improvement proposals we also need metrics to quantify an XML Retrieval system's quality. Based on the definition given in [KL06], we define retrieval quality as a function of retrieved relevant fragments and all fragments in a collection[2]. In other words, we compare the scores which an IR system assigns to a fragment to the fragments' relevance. A fundamental problem with this approach is that relevance is always relative to one particular information need and highly subjective: For example, for the same information need, user A may judge a result item highly relevant, whereas user B considers it to be irrelevant. Hence organisations such as TREC and INEX provide normative relevance assessments [KL06]. To obtain these assessments, a group of users manually judges the results of a given query on a particular document collection. Typically they do not assign relevance values directly, but use other metrics to judge the results which are then aggregated into a single relevance value. In early INEX workshops, for example, assessors used a two-dimensional metric [GK02]: The first dimension (*exhaustivity*) describes to what extent a fragment covers an information need, the second dimension (*specificity*) describes how focussed the fragment is on the information need, i.e. if it also covers other topics; they than map these assessments to a relevance value in the $[0, 1]$ interval by employing a so-called quantisation function. More recently, INEX uses a simplified metric: Assessors highlight relevant portions of text; the ratio of highlighted to non-highlighted text in a fragment is then treated as its specificity [PT05] and relevance is solely judged based on specificity [LP06]. In the following, we discuss metrics for measuring retrieval quality (as opposed to relevance assessment metrics) and select suitable ones. At the end of this section, we then complement this by defining which test collections are feasible for our evaluation.

[2]Please note that our definition of retrieval quality is solely based on a systems's input and output; we deliberately do not consider how the output is produced and thus ignore performance aspects, in particular. We consider performance as an equally important second dimension of retrieval system quality in a broader sense; this dimension is not in the focus of this thesis, though, and therefore not considered when discussing evaluation.

6.2.1. Retrieval Quality Metrics

A straightforward approximation of our view of quality is the notion of precision and recall in traditional Information Retrieval: *Precision* is the ratio of relevant to irrelevant result items returned by the IR system, *recall* is the ratio of relevant result items returned to all result items in the collection [MRS08, chpt. 1]. These metrics are not well-suited for XML Retrieval, though, due to the additional challenges it introduces. These challenges include overlap (which we have already discussed in section 5.3) and near-misses [KL06]. *Near-misses* are fragments in the result set which are imperfect matches, but close to perfect matches in terms of proximity.

Over the last few years numerous metrics have therefore been proposed to measure XML Retrieval quality. In the following we provide a brief overview[3] of such proposals to identify metrics suitable for our needs. One of the first metrics used for XML Retrieval evaluation was inex-eval. It was introduced by INEX in 2002 [GK02, sec. 5]. For the inex-eval metric, every fragment in the fragment space is assigned a relevance value in the $[0, 1]$ interval based on manual assessments in a two-dimensional criteria space. Based on this value the metric calculates the probability that a fragment in the result list is relevant, given that the following to conditions hold: Firstly, the user views all fragments linearly in order of decreasing relevance (i.e. the ranking produced by the IR system); secondly, he only views a fixed maximum number of relevant fragments [GK02, sec. 5]. The major drawback of this metric is its ignorance of the fact that XML fragments are *not* independent as they are nested [KL06].

Järvelin and Kekäläinen [JK02] have introduced a more flexible family of retrieval quality metrics called *cumulated gain-based metrics* (CG, for short). They are targeted at flat Information Retrieval, yet can be adapted to XML Retrieval as well as we will see shortly. Let us assume (like for inex-eval) that the user views results linearly in order of decreasing relevance. Then the *direct cumulated gain* at an arbitrary position i in the result ranking is the sum of the relevance values of all items in the result ranking up to the i-th position. In other words, this aims at the gain a user has by looking at the first $i + 1$ items compared to the first i items. Let $f_j \in \mathcal{R}$ be the fragment at the j-th position in a result ranking; then the following equation illustrates the CG calculation:

$$\mathrm{cg}(i) = \sum_{j=1}^{i} \mathrm{rv}(f_j) \tag{6.1}$$

Based on this idea, Järvelin and Kekäläinen [JK02] further refine this metric by considering the position of an item in the ranking and by normalizing the gain values. The first aspect means that we down-weight an item's relevance value the further down it is located in the ranking. The rationale behind this is the assumption that even a relevant item provides less gain to the user, if it is located at a low rank, because the user is less likely to look at it. The second aspect (normalisation) enables the comparison of different result values: To obtain the absolute gain value at each rank is divided by the

[3]For a comprehensive overview of XML Retrieval quality metrics cf. [KL06], for example.

ideal gain value at this rank as shown in the following equation.

$$\mathrm{cg}_{\mathrm{norm}}(i) = \frac{\mathrm{cg}(i)}{\mathrm{cg}_{\mathrm{ideal}}(i)} \tag{6.2}$$

To calculate the ideal gain (i.e. $\mathrm{cg}_{\mathrm{ideal}}(i)$) we first need to introduce the concept of an *ideal result list* $\mathcal{R}_{\mathrm{ideal}} = \{f \in \mathcal{F} : \mathrm{rv}(f) > 0\}$. This is the set of all fragments in the fragment space with a non-zero relevance, ordered by decreasing relevance. Please note that if there exist at least two fragments in $\mathcal{F}$ with the same relevance, the ideal result list is ambiguous; therefore we transform $\mathcal{R}_{\mathrm{ideal}}$ into the *ideal gain vector* I_{ideal} by only regarding the relevance value at each position. The ideal gain vector is thus unambiguous and the ideal gain at an arbitrary rank i is simply i-th value in the vector I. The resulting measure is known as *normalised cumulated gain* (*nCG*, for short).

Kazai and Lalmas [KLdV04] propose the *eXtended cumulated gain* (XCG) family of metrics as a CG adaption to XML Retrieval (more precisely, the INEX Ad hoc Focussed Task [MTLF06]) which takes into account the nesting of result items. To achieve this, they modify $\mathcal{R}_{\mathrm{ideal}}$ by filtering out all but one of each set of overlapping fragments. The filtering logic uses criteria based on exhaustivity and specificity and makes assumptions on the desired result selection behaviour (that is, all deviations from this behaviour are penalised). Therefore only one fragment per path is contained in the ideal result list, and thus only perfect matches (i.e. matches contained in the ideal result list) are taken into account by the metric so far. Returning near-misses (such as ancestors or descendants) of an ideal fragment, on the other hand, is not rewarded at all, as they are not contained in $\mathcal{R}_{\mathrm{ideal}}$. Therefore Kazai and Lalmas propose another extension to the CG metric in [KL06, sec. 5.2.2] to also account for relevance values of elements not contained in the ideal result list: To ensure that a system returning several near-misses instead of the ideal match does not receive a higher nxCG score, they employ a normalisation function to all relevance values. Let near-misses : $\mathcal{F} \longrightarrow \mathcal{F}^n$ be a function which returns the set of all near-misses of an ideal match m_{ideal}. Then the normalisation function $\mathrm{rv}_{\mathrm{norm}} : \mathcal{F} \times [0,1] \longrightarrow [0,1]$ ensures that the relevance value used for a fragment adheres to the following condition[4]:

$$\forall m \in \text{near-misses}(m_{\mathrm{ideal}}) : \mathrm{rv}_{\mathrm{norm}}(m) + \sum_{n \in M} \mathrm{rv}_{\mathrm{norm}}(n) < \mathrm{rv}(m_{\mathrm{ideal}}) \tag{6.3}$$

where $M = \text{near-misses}(m_{\mathrm{ideal}}) - m$ is the set of all near-misses of m_{ideal} except for m. In other words, even if a system returns *all* near-misses of an ideal match, their cumulated gain must be lower than that of the ideal match; thus a system returning the ideal match is always preferred over one that does not.

The composition of the set of near-misses (i.e. the specification of our near-misses function) is deliberately left undefined in [KL06], as it depends on the assumed user preferences. This enables us to integrate the XCG approach with our proximity model.

[4]Please note that we have made some slight corrections to the corresponding equation proposed in [KL06]. This does not affect the underlying idea, however.

For example, we may define the near-misses function to return all fragments up to a certain level of proximity to m_{ideal}. Additionally, we can adapt rv_{norm} to penalise near-misses relative to their proximity. The ideal result selection behaviour Kazai et al. [KLdV04], [KL06] assume to be fixed[5], however: They assume that the behaviour is invariant across all scenarios and hence does not need to be configurable. Their assumption contradicts our result selection proposal in section 5.3 which is based on entirely different selection rules and additionally features various configuration parameters. One such parameter even allows to control whether overlap is permitted or not; in case we permit overlap, both overlap filtering and the handling of near-misses are obviously superfluous.

Thus applying the XCG metric unchanged to an IR system which implements our result selection logic (RSL, for short) would not yield meaningful results in any case. We can, of course, build an XCG-like metric featuring the rules we have proposed for result filtering. Such a metric would only be necessary though, if we wanted to evaluate our other improvement proposals (that is, all but the RSL proposal) in an implementation featuring this RSL. Instead of doing this, we may just as well use a XCG-compliant RSL implementation to test our other features, as they are independent of result selection. For evaluating the RSL *itself* the use of a custom metric is pointless, as we will discuss in the subsequent chapter; therefore we will not define an XCG adaption (or any other metric) specific to our RSL proposal.

Because nxCG only provides a value for an individual rank, several derived measures have been proposed to ease broader evaluation. Two such measures are *MAep* and *ep/gr* which have been established as the official measures of the INEX workshop for the so-called thorough and focussed tasks in 2005 [LKK+06]. Both measures are based on the concepts of *effort-precision* (EP) and *gain-recall* (GR). EP defines how much effort a user needs to spend to attain a particular gain value $g \in \mathbb{R}^{\geq 1}$ compared to the effort needed in an ideal system; the effort is measured as the number of ranks the user has to look at (i_{run} and i_{ideal}, respectively). The following equation shows the EP calculation for an arbitrary gain value [LT07]:

$$\text{ep}(g) = \frac{i_{\text{ideal}}}{i_{\text{run}}} \in (0, 1] \tag{6.4}$$

An EP value of 1.0 corresponds to an ideal system.

Let

- $\text{xg} : \mathbb{N}^{\geq 0} \longrightarrow \mathbb{R}^{\geq 0}$ be an XML-adapted gain function which calculates the gain for a given rank $i \in \mathbb{N}^{\geq 0}$,
- $\text{xg}_{\text{ideal}} : \mathbb{N}^{\geq 0} \longrightarrow \mathbb{R}^{\geq 0}$ a function which calculates the ideal gain for i, and
- n be the number of fragments in the fragment space which provide a non-zero gain.

Then GR is calculated as [LT07]:

$$\text{gr}(i, n) = \frac{\sum_{j=1}^{i} \text{xg}(j)}{\sum_{j=1}^{n} \text{xg}_{\text{ideal}}(j)} \in [0, 1] \tag{6.5}$$

[5]For the exact rules Kazai and Lalmas propose to filter the result set cf. [KLdV04, sec. 4.1].

This corresponds to the ratio of cumulated gain that is obtained at rank i to the *total* ideal gain that can be archived for the fragments in the collection used. Because of the division by the total ideal gain, the resulting values are normalised to the $[0, 1]$ interval. By using the GR values as input for the EP function, we can plot a graph (the so-called ep/gr graph) with both the x and y axis normalised to $[0, 1]$; it depicts the performance of a retrieval system over the entire result set. Finally, the MAep measure (for *mean average effort-precision*[6]) condenses the retrieval quality of a system into a single number. It averages the EP values for every rank i with a non-zero gain (i.e. $xg(i) > 0$).

6.2.2. Test Data Sets

Apart from quality metrics we need sets of test data in order to benchmark retrieval approaches. By *data* we mean the actual collections of XML documents as well as defined test queries and corresponding relevance assessments. To our knowledge, the only broad source of such test data for XML Retrieval to the present day is the INEX workshop. Hence we now give a brief overview of the organisation of INEX and analyse to what extend we can use its evaluation-related aspects.

INEX consists of several *tracks* which focus on a particular field of XML Retrieval each. The main track is called the *Ad hoc Track* which is also most relevant to us. It deals with processing varying information needs on a stable collection of documents [MTLF06]. The Ad hoc Track consists of four so-called *tasks*. Each of these tasks models one particular aspect which an XML IR system may be confronted with [MTLF06, sec. 4]:

- The *Thorough Task* demands an IR system to locate all relevant fragments in a collection and rank them according to relevance. No result selection techniques like overlap removal or aggregation are to be performed. Semantically we can interpret this as the intermediate result of an IR system which is passed on to other parts of the system (or components outside the system) for further processing; in our XML Retrieval process this corresponds to the result of the score propagation activity just before we would normally perform result selection. Alternatively, we can go through all phases of the XML Retrieval process and configure our result selection algorithm with $s_{\text{min}} = 0$, $t_{\text{agg}} = \perp$, and $o = \texttt{FALSE}$, meaning that all fragments with a non-zero relevance are to be returned without performing either overlap removal or compaction.

- The *Focussed Task* corresponds to the thorough task plus additional result selection. For the selection, one particular behaviour is requested: Overlap must be removed and the elements to be returned shall be as fine-grained as possible while still exhaustively addressing the user's query. In our result selection algorithm, the configuration most closely meeting these requirements is the following one:

[6]More precisely, MAep stands for *non-interpolated* mean average effort-precision. For details on interpolation cf. [LT07].

$s_{\min} = 0$, $t_{\text{agg}} = \perp$, and $o =$ `TRUE`. This is the same configuration as we have stated for the thorough task, except that overlap removal is enabled. However, albeit emulating the given requirements, these settings do not reflect the ideas which lead to the design of our result selection concepts. Hence we deem the focussed task to not be of use for our purposes.

- For the *Relevant in Context Task* a document-oriented view is pursued: Per document in the collection the (non-overlapping) fragments best representing the relevant document parts are sought for. This endeavour only partially corresponds to our scope: By using the aggregation feature of our result selection algorithm (i.e. by setting the t_{agg} parameter to a finite value), we can model the "clustering" aspect of finding fragments which cover the relevant parts of all sub trees. This solution is still fragment-oriented and only produces a per-document listing of results. Hence this task is not useful either for our evaluation.

- Finally, the *Best in Context Task* aims at finding the best fragment to start reading a document for each document in the collection. Hence it also takes a document-oriented view and therefore is not of use for this thesis.

Apart from the ad hoc track, the *Heterogeneous Collections Track* and the *Use Case Track* (partially) address aspects which are also of interest to us. The heterogeneous track deals with searching over heterogeneous collections, i.e. collections containing different kinds of contents and featuring a different schema each. It currently assumes that there is a comparatively small number of collections, only, and that these collections are all known in advance and feature an explicit schema [FL06]. Thus, at present, the track corresponds to our Corporate Intranet Search use case rather than our XML Web Search use case (cf. section 2.3). We believe that the long term goals of this track rather focus on the latter scenario, though, as for example aspects like data source selection are already discussed today (cf. [FL06]). A drawback of this INEX track regarding its use for our evaluation is its current status: According to [FL06], in 2007 the track's participants have laid administrative foundations for performing evaluations (such as creating test collections and sets of queries), but have not yet conducted relevance assessments. Therefore at the time of writing, the INEX Heterogeneous Track is not yet an option for evaluating heterogeneity-related aspects; we expect that will provide a valuable basis for doing so in the future, however. The INEX Use Case Track mentioned above aims at devising sound models of user expectancies and likely use cases for XML Retrieval. This is relevant to us in the context of XML Retrieval requirements and designing an appropriate system behaviour (cf. chapters 2 and 5, respectively), but only indirectly[7] relates to our evaluation. The remaining INEX tracks which, for example, focus on user interaction and natural language processing address issues outside of our scope and are thus not relevant for our evaluation purposes.

Therefore, the Ad hoc Thorough Task is the only part of INEX which we expect to be directly beneficial for the evaluation of our improvement proposals *at present.*

[7]For a brief discussion of user models in the context of retrieval evaluation see page 138.

As mentioned above, INEX officially uses the ep/gr and MAep measures for this task which we have generally found to be suitable for our needs. Hence we plan to use the instruments provided by INEX for this task to enable benchmarking; specifically we will use metrics, tools, and test collections. The metrics we have already discussed above. Tools are, for example, libraries and scripts for calculating the various measures for a given result set and visualising them. A test collection consists of XML documents, queries, and relevance assessments. The documents in a collection typically belong to a particular genre such as scientific papers or lexicon entries. The queries express specific information needs related to the documents used and contain additional documentation of the intended results to ease relevance assessments. The assessments are performed per query and XML fragment as described above; they serve as a normative source of relevance assignments for evaluation. We decided to employ two test collections based on the data provided by INEX:

- C_{hom} represents a collection with a large number of homogeneous articles that all adhere to a common schema. As instances of this collection we use the INEX Wikipedia and the IEEE collections. The former contains roughly 1.5 million articles from Wikipedia[8] which have been transformed to XML using a single explicit schema. It has been established as official INEX collection in 2006. The latter contains about 16,000 XML transformations of scientific articles published by the IEEE Computer Society[9]; it was employed by INEX in the 2002 to 2005 period, with several updates of the data. Like the Wikipedia collection all documents in the IEEE collection adhere to a single explicit schema. Please note that each test run using C_{hom} has to be conducted twice (i.e. once with the Wikipedia collection and once with IEEE), as otherwise it would not represent a *homogeneous* collection. The reason to use both collections in combination is simply to provide a broader basis for our evaluation runs and hence make them more representative.

- C_{het} is also a collection of collections. Unlike C_{hom}, however, we use all of its sub-collections at the same time instead of sequentially. The individual document collections cover different kinds of content (e.g. news items, scientific articles, bibliographical records) and each adhere to an explicit, but different schema. All of the individual collections contained in C_{het} share a common set of queries, though, to enable heterogeneous retrieval. The collection is still under construction, but intended to be used by the Heterogeneous Collection Track of future INEX workshops.

For detailed descriptions of the three collections, please refer to [DG06], [GK02, sec. 4], and [FL06], respectively.

[8]http://www.wikipedia.org
[9]http://www.computer.org

6.3. Evaluation Methodology

So far, we have constructed a framework for conducting evaluations: We have chosen evaluation approaches (correlation testing and benchmarking), appropriate metrics (ep/gr ad MAep), and data sets (C_{hom} and C_{het} of the INEX Ad hoc and Heterogeneous Tracks). In addition to these constituents, we now introduce a methodological instrument: the evaluation task. An *evaluation task* addresses the evaluation of one concise aspect, typically an improvement proposal; it defines how to test the aspect (i.e. the evaluation approach to use and the conceptional details of how to use it), which test collections to run the evaluation on, and what the expected outcomes are. Each evaluation task consists of one or more evaluation runs; an *evaluation run* is one execution of the selected queries (typically: all) on a particular test collection using defined implementation and configuration settings. Table 6.1 shows the standardised template we use to define an evaluation task.

6.4. Summary

In this chapter we have devised a framework to evaluate our improvement proposals for XML Retrieval. The focus of this evaluation lies on retrieval quality which we define as a function of retrieved relevant fragments and all relevant fragments in a collection; this equates to a comparison of scores generated by an IR system to some normative relevance assessment. We have discussed various metrics for measuring XML Retrieval quality. This showed that the *eXtended cumulated gain* (xCG) family of metrics [KLdV04] and commonly used derived measures such as mean average effort precision (MAep) and effort-precision/gain-recall (ep/gr) are feasible for our evaluation and integrate well with our retrieval framework.

As general evaluation techniques we use benchmarking and statistical correlation testing. The former compares a baseline IR system implementation to a modified IR system variant featuring one or more of our improvement proposals. The latter tests whether two arbitrary variables (e.g. relevance and proximity level) exhibit a statistically significant correlation. Both techniques require the existence of test data, that is, collections of XML documents, defined test queries, and according normative relevance assessments. We employ test data provided by the INEX Ad Hoc Thorough Task and the INEX Heterogeneous track albeit the latter does not yet include relevance assessments, so actually conducting the evaluation will have to be partially postponed. We represent this test data by the logical test collections C_{hom} and C_{het} which represent a homogeneous and a heterogeneous collection, respectively.

To describe the details of how to evaluate an arbitrary aspect (e.g. one improvement proposal), we have introduced the concept of *evaluation tasks*. Each evaluation task defines which test data and technique to use and what results we expect. It may consist of multiple *evaluation runs*, such as benchmarking runs with different configurations of an IR system. As a notation for evaluation tasks we use a standardised template.

Table 6.1.: Template for evaluation tasks

Name	A short but informative name of the evaluation task
Test Type	Benchmarking or Correlation Testing
Data	The collection(s) to test on (e.g. C_{hom}) and restrictions regarding the documents and/or queries to use.
Prerequisites	Implementation parts, tools, and manual activities which are needed to conduct the evaluation task.
Details	The conceptional details of how to apply the evaluation approach. For a correlation test, this section defines which correlation of measures to look at. For a benchmark, we provide a list of evaluation runs (each using a different implementation variant) to conduct; the first evaluation run is always the baseline run which the others are then compared to. For example: • Baseline: Description of the baseline run. • Evaluation run 2: Description of the second run, which is compared with the baseline. • Evaluation run 3: …
Hypotheses	In this section we state which outcomes of the evaluation runs we expect. For example: 1. Evaluation run 1 performs better than evaluation run 2. 2. Evaluation run 3 yields good results whenever the sun is shining. After the evaluation task has been performed, we have to compare the expected outcomes with the actual ones and analyse potential deviations.

7. Evaluation Strategy

The focus of this chapter lies on determining *what* exactly to evaluate and on how to do this. For we summarise the improvement proposals described throughout this thesis, analyse their dependencies and for each aspect define how to evaluate it using the evaluation task template introduced in the previous chapter.

7.1. Evaluation Tasks

7.1.1. Element Type Classification

In section 3.4 we have proposed to distinguish various types of elements in an XML document (namely: formatting, structuring, and entity elements); we can assign each element to one or more of these types. This classification is not a stand-alone improvement proposal, but supports other aspects. Currently we only employ element type classification for the evaluation of target conditions; provided the classification is sound, however, element types are of potential use for many other aspects, e.g. term proximities, name matching, reference handling, and even weighting. Therefore we want to measure the benefit of type classification directly. We thus propose two evaluations to be performed: In table 7.1 we define various statistical analyses of element type classifications; table 7.2 describes the benchmarking of various modes of target condition evaluation.

7.1.2. Weighting Strategies

In section 3.5.2 we have defined several strategies to weight query conditions. The simplest approach was static condition-type based weighting (cf. equation 3.7) where we assign fixed weights α, β, and γ to content, support, and target conditions, respectively. We believe that the exact configuration of these weights is scenario-dependent. Hence the evaluation task which we define in table 7.3 only aims at obtaining a rough idea on how different $\alpha/\beta/\gamma$ values affect the retrieval results based on two sample scenarios. The other approaches are orthogonal to the static one. Thus we propose to use the best-working $\alpha/\beta/\gamma$ configuration (with regard to the collection used) obtained in the above evaluation task as a baseline. We then refine the weighting by using each statistics-based weighting approach in turn on top of the static weighting. We provide the details of this evaluation task in table 7.4. Please note that we expect the performance of some of these approaches to strongly depend on whether the collection used is homogeneous or heterogeneous (cf. section 3.5.2). As the C_{het} collection has not yet been assessed by INEX, we can only evaluate the homogeneous perspective.

Table 7.1.: Evaluation Task: Relevance/Element Type Correlation

Name	Relevance/Element Type Correlation
Test Type	Correlation
Data	$C_{\text{hom}}, C_{\text{het}}$
Prerequisites	Element type classification heuristic or manual type classification; Statistical tools
Details	• What percentage of elements in the schema belongs to which combination of element types? In particular it is interesting to analyse, if for each element type there is a substantial number of elements belonging to this type, *only*. • We should then analyse if there is a statistically significant correlation between assignments to certain element types. If, for example, all structuring elements are typically also classified as entity elements, there is little point in making the distinction and instead we should identify different types. • Is there a significant correlation between the type of the target elements used in test queries and the relevance assessments of elements with the respective type? Does this correlation also hold, if we only regard matches of the same type as the target condition, but not identical to it?
Hypotheses	1. For each of the element types we have defined there exists a substantial number of elements which belong to this element type, only. 2. There exists no statistical correlation between assignments to particular element types. 3. If a query contains tag name-based target conditions, elements with the same element type as the target condition are significantly more relevant than other elements. 4. The latter also holds, if we only regard elements which are not identical to the target condition, but have the same element type.

Table 7.2.: Evaluation Task: Type-specific Target Scoring Benchmarking

Name	Type-specific Target Scoring Benchmarking
Test Type	Benchmarking
Data	C_{hom}
Prerequisites	Element type classification heuristic or manual type classification; Type-aware index; Standard query language and processor; Target scoring logic
Details	• Baseline: No evaluation of target conditions (we assign every element a target score of zero) • Evaluation as defined for structuring elements in section 5.1 regardless of the element type • Evaluation as defined for formatting elements regardless of the element type • Evaluation as defined for entity elements regardless of the element type • Evaluation according to the type-specific rules For all of these runs, we use the target score as the overall score of fragments to highlight results.
Hypotheses	1. The baseline implementation performs worst. 2. Provided that the previous evaluation task yields that our type classification is reasonable, we expect the last benchmarking run to be clearly superior to the other runs.

Table 7.3.: Evaluation Task: Condition Type-based Weighting Benchmarking

Name	Condition Type-based Weighting Benchmarking
Test Type	Benchmarking
Data	C_{hom}
Prerequisites	Standard query language and processor; Condition type-based weighting
Details	• Baseline: Uniform weighting, that is $\alpha = \beta = \gamma = \frac{1}{3}$ • 2:1:1 weighting: Three runs with a $\frac{1}{2} : \frac{1}{4} : \frac{1}{4}$ distribution (with each weight being at $\frac{1}{2}$ in of the runs) • 1:2:2 weighting: Three runs with a $\frac{1}{5} : \frac{2}{5} : \frac{2}{5}$ distribution (with each weight being at $\frac{1}{5}$ in of the runs) • Content-only weighting: One run with $\alpha = 1.0$ and $\beta = \gamma = 0$
Hypotheses	1. Content conditions have the strongest influence on relevance; thus the 2:1:1 run with content conditions at $\frac{1}{2}$ performs best. 2. Ignoring structural conditions has a negative impact on the result; thus the content-only run performs worse than the mixed runs.

Table 7.4.: Evaluation Task: Statistics-based Weighting Benchmarking

Name	Statistics-based Weighting Benchmarking
Test Type	Benchmarking
Data	C_{hom}
Prerequisites	Standard query language and processor; Condition type-based weighting; Statistics-based weighting extensions
Details	• Baseline: The optimal (collection-specific) condition-type based weighting identified in the first evaluation task (cf. table 7.3) • Tag name ITF on top of baseline • Tag name IDF on top of baseline • Relationship rareness on top of baseline • Relationship IDF on top of baseline
Hypotheses	1. Both tag name ITF and relationship rareness improve baseline performance on both collections, as they are homogeneous. 2. Tag name IDF and relationship IDF decrease baseline performance for the same reason.

7.1.3. Explicit structural hint matching

Name Matching

The evaluation of name matching is difficult using the existing sources of test data: Test queries used in C_{hom} are typographically correct and formulated with a single, explicitly available schema in mind. We thus expect the majority of relevant result elements to be exact matches of a query condition. The correction-oriented parts of our name matching proposal (that is, edit distances, abbreviation matching, semantic distances, and so on) are hence of little value in this scenario. What we need instead are either queries formulated by schema-unaware users or queries targeted at heterogeneous collections such as C_{het} – for which unfortunately no relevance assessment has been performed yet. Assuming that feasible data will be available in the future, we formulate the evaluation task shown in table 7.5 to answer to following questions:

- Does length-specific name matching improve matching quality?
- What is a good length threshold L_{short}?
- Does the concept of increasing the confidence as name length increases work?
- Which string matching techniques works well for short and long names, respectively?

By this first evaluation we aim to achieve a general understanding of the performance of various name matching approaches in XML Retrieval. Based on this, we can then formulate further evaluation tasks to answer more focussed questions such as:

- Which *combinations* of string matching techniques perform well for XML Retrieval?
- What is an optimal function to map name length to confidence and penalty factors (such as P_{RFM}), respectively?

Path Matching

For the evaluation of path matching, the same constraints regarding the test data apply as for name matching: To effectively test how well our solution approaches cope with the different modellings listed in section 4.2.1, we need modelling discrepancies either between the query and the test collection or between individual documents in the test collection. C_{hom} does not account for this, whereas C_{het} potentially will. Aspects we can evaluate on C_{hom} are missing and excessive elements which we address by path edit distances. Hence we define a first evaluation task in table 7.6 which deals with these aspects. On the positive side, this restriction to the handling of missing and excessive elements causes the evaluation to become independent of name matching, whereas substitutions, in particular, strongly depend on it. Thus we can isolate the path matching evaluation at least partially from the quality of name matching. All

Table 7.5.: Evaluation Task: Name Similarity Benchmarking

Name	Name Similarity Benchmarking
Test Type	Benchmarking
Data	Potentially C_{het}
Prerequisites	Standard query language and processor; Adaptions to accommodate individual name matching logic
Details	The performance of the following variants of name matching implementations should be compared when using them for all name matching tasks (i.e. target conditions and support conditions on elements and attributes): • Exact matching • Uniform matching with traditional edit distances, recursive field matching, and (adequately configured) substitution groups, respectively • Length-specific matching with the different techniques used for $\text{namesim}_{\text{short}}$ and $\text{namesim}_{\text{long}}$ at varying length thresholds $L_{\text{short}} \in \{1, 2, \ldots, 10\}$
Hypotheses	1. Exact matching performs worst. 2. There exists an L_{short} value where length-specific matching performs substantially better than all other runs.

Table 7.6.: Evaluation Task: Simple Path Matching Benchmarking

Name	Simple Path Matching Benchmarking
Test Type	Benchmarking
Data	C_{hom}
Prerequisites	Standard query language and processor; Adaptions for custom path matching logic
Details	• Baseline: IR system implementation which ignores path conditions • Simplified path edit distances: Modified baseline implementation using $\text{editcost}_{\text{gen}}$ (equation 4.5) and the parameters defined in table 4.3 for all operations
Hypotheses	The path edit distance approach performs better than the baseline.

other aspects demand for more flexible test data, though. We therefore cover them with a separate evaluation task (cf. table 7.7) which is more vaguely defined, however.

A problem which remains unanswered in both evaluation tasks is that of identifying an optimal configuration of the cost functions used (i.e. the base cost and repetition factors for the various path edit operations). We presume that looking for such a configuration is pointless as long as the basic questions regarding our improvement approaches have not yet been answered. Hence we postpone this aspect until after the above evaluations have been performed and analysed.

7.1.4. Implicit structural hint matching

Term Proximities

For our term proximity proposal (cf. sec. 4.3.1), the first aspect to evaluate is whether proximity levels as we have defined them actually correlate to relevance. Ideally, proximity level 1 corresponds to the highest relevance, level 2 to a lower relevance, and so on. We describe an according evaluation task in table 7.8. If they do not, we have to analyse whether changing the order of proximity levels or the addition and/or removal of proximity levels affects this result. In addition we propose to benchmark various proximity implementations as defined in table 7.9. Even in case we do not detect a significant correlation in the first evaluation task, this benchmarking should still yield interesting results concerning the comparison of various approaches to term proximity.

Table 7.7.: Evaluation Task: Enhanced Path Matching Benchmarking

Name	Enhanced Path Matching Benchmarking
Test Type	Benchmarking
Data	Potentially C_{het}
Prerequisites	Standard query language and processor; Adaptions for custom path matching logic; Index implementation supporting our reference handling solution; TACA detection logic
Details	• Baseline: The simplified path edit distance implementation from table 7.6 • Enhanced path edit distances: Modified implementation using the enhanced cost function shown in equation 4.7 with the paremeters as suggested on page 77 • Simple indexing: Modified implementation which additionally uses an index which does not differentiate child elements and attributes • Enhanced indexing: Modified implementation which additionally indexes references; here various $N_{\text{max-refs}}$ and $N_{\text{max-ref-depth}}$ values (1, 5, 10) should be used. • TACA detection: Modified implementation which uses an enhanced index that enables runtime TACA detection and adequate detection rule sets
Hypotheses	Matching quality improves with each modification.

Table 7.8.: Evaluation Task: Relevance/Proximity Level Correlation

Name	Relevance/Proximity Level Correlation
Test Type	Correlation
Data	C_{hom} (Queries with more than one keyword)
Prerequisites	Statistical tools; Proximity level calculation logic
Details	Test, if there exists a statistical correlation of relevance assessments and proximity levels
Hypotheses	There is a significant correlation between a fragment's relevance and the proximity level of pairs of keywords used in the query: Fragments containing term pairs on proximity level 1 have the highest relevance, fragments with pairs on level 2 the second-highest relevance, and so on.

Table 7.9.: Evaluation Task: Proximity Benchmarking

Name	Proximity Benchmarking
Test Type	Correlation
Data	C_{hom}
Prerequisites	Standard query language and processor; Adaptions for custom term proximity logic
Details	• Baseline: Implementation not regarding term proximities at all • Document-level proximity: Implementation treating XML documents as flat and employing MinCover proximity of term across the entire document; scores are attributed equally to all elements in the document (i.e. does not help to choose result fragment within a document, but only to rank fragments of different document against each other). • Within-element proximity: Implementation using MinCover proximity within an element's recursive content; the score is attributed to the element whose recursive content is looked at. • Edge-counting proximity: Implementation using edge-counting proximity; scoring is done according to equation 4.11. • Level-based proximity: Implementation using our proximity level-based measure and scoring as defined on page 88.
Hypotheses	1. The baseline approach performs worse than all other approaches (potentially with the exception of document-level proximity). 2. Within-element and edge-counting proximity each perform better than baseline and document-level proximity. 3. Level-based proximity performs best.

Table 7.10.: Evaluation Task: RIF Benchmarking

Name	RIF Benchmarking
Test Type	Benchmarking
Data	C_{hom}
Prerequisites	Standard query language; Particular query processor support for length-based heuristics; Adaptions for custom RIF logic; TACA detection logic
Details	• Baseline: IR engine described in [Dop07] using title and inline structural patterns. • RIF variant: Modified baseline engine using RIFs to finetune scores
Hypotheses	The RIF variant performs significantly better than the baseline approach.

Length-based heuristics (RIFs)

The Relevance Influence Factor (RIF) proposal in section 4.3.2 is comparatively difficult to evaluate: We need a baseline implementation featuring a length-based heuristic and an appropriate configuration of collection-specific RIF factors; for example, the implementation of support links in [RWdV06] appears suitable. In the following we thus assume that this implementation is used for evaluation. Additionally we need to analyse the collection used to define RIF factors for elements matching the support links. Based on this, we can then adapt the baseline implementation to modify scores using RIF factors and benchmark the modified implementation against the baseline. Table 7.10 summarises this evaluation task.

7.1.5. Result Selection

Testing the result selection logic (RSL) we have defined in section 5.3 is a special problem: If we use common existing metrics such as the XCG metric, we cannot obtain meaningful results, as the metric assumes one particular result selection model – which is different from ours – to be ideal. If we use a custom metric adhering to our RSL rules, on the other hand, we cannot obtain meaningful results, either, as this reduces evaluation to a mere functional test. In consequence, an actual evaluation of RSL is impossible *per se*. Instead, scenario-specific user models need to be created, based on which we can then configure or adapt our RSL. This goes beyond the scope of this thesis, however, so we restrict ourselves to referring to on-going work[1] in this area.

[1]The INEX workshop, for example, pursues such endeavours (on a broader scale) in its use case track; cf. [WGE06] for pointers.

7.2. Summary

In this chapter we have discussed which aspects of our XML Retrieval concept require evaluation and how this evaluation is to be performed. We have employed the evaluation framework which we have introduced in the previous section and consequently defined a set of evaluation tasks. Element types we evaluate by testing their correlation with relevance assignments and by benchmarking variants of our type-specific target scoring proposal. To evaluate condition types we also conduct benchmarks with various weighting schemes both for our fixed static and fixed dynamic weighting strategies. In the context of explicit structural hints we benchmark our name matching solution against simplified name matching approaches and define two benchmarking tasks to evaluate path matching. For implicit structural hints we first evaluate the correlation of our proximity levels to relevance assignments; this serves to finetune our set of proximity levels and their order. We then benchmark several term proximity proposals against our own solution to determine whether our solution improves retrieval quality. In the same manner we evaluate our improvement proposal for length-based heuristics.

The only aspect we cannot evaluate is our result selection logic (RSL). Unlike other aspects, evaluating result selection requires to have a normative user model to define the "right" system behaviour. This model is a part of the quality metric as well as a part of the logic which the model is used to evaluate. This reduces the evaluation of any RSL to a mere functional test, if the same model is, indeed, used for both the RSL and the quality metric, and otherwise produces meaningless results, only. Hence we restrain from proposing an evaluation task for our RSL and instead refer to the ongoing research on appropriate user models.

Part IV.

Conclusions

8. Conclusions & Future Work

8.1. Conclusions

The overall goal of this thesis was to explore how information on the document structure can be exploited to improve XML Retrieval quality. To operationalise this goal, we have defined three subordinate goals: the design of a conceptional framework, the proposal of improvements, and – partially – their evaluation. The first goal is motivated by a general problem we perceive in current XML Retrieval research: There exist numerous publications addressing XML Retrieval which contain many interesting ideas. Many of these originate in the INitiative for the Evaluation of XML Retrieval (INEX) which provides a forum for researchers and infrastructure such as a query language, test collections, and quality metrics. Unfortunately, most of these publications try to address XML Retrieval as a whole and without a clear conceptional model. We believe that this entails two issues: Firstly, due to the lack of distinct, agreed-upon conceptional features, it is difficult to compare, reuse, combine, and extend existing ideas on XML Retrieval. Secondly, as there is no up-front restriction of the scope, researchers tend to build core aspects of their proposals on very specific assumptions which only hold in particular scenarios; thus their approaches become inflexible which makes it difficult to generalise existing ideas.

Hence we have devised the XML Retrieval process as a conceptional model for XML Retrieval. It features a sequence of logical phases, each of which we have refined into clearly distinct activities. This addresses the first of the two issues named above (lack of distinct features). Nonetheless, we do not claim this model to be the perfect solution. On the contrary, we intend our model to be a first step which provokes critical discussion and the creation of alternative models. To address the second issue (unjustified assumptions), we have identified a set of environmental properties which are of concern to XML Retrieval systems. We have then defined several use cases to explore likely configurations of these properties. Based on this, we have finally decided which general assumptions we can make without impacting the applicability of our concepts and which aspects must be configurable. As a consequence, we believe that our concepts apply to a broader range of scenarios than most existing proposals.

For our second goal, the proposal of improvements, our conceptional framework also provided the foundation. The distinction of logical phases and their refinement into activities corresponds to a divide-and-conquer strategy. This strategy enables us to analyse activity by activity and assess its status quo in XML Retrieval research. By doing so we identify improvement potentials and derive according solution proposals. The resulting proposals each address specific improvement potentials and build on existing ideas as opposed to reinventing the wheel. Our overall framework ensures that the various proposals integrate well and complement each other. However, which of our proposals actually do provide a benefit regarding retrieval quality will have to be determined by experimental evaluations. We even expect that most proposals perform rather poorly at first and that several rounds of refinements – based on the insights gained by evaluation runs – are required to adjust them well. As one example, consider our term proximity approach: We believe that the concept of having multiple proximity levels plus within-level refinements is far superior to existing techniques; we also believe, however, that our initial set of proximity levels and their order is unlikely to be the optimal (or even a good) choice.

Therefore only with our third goal things fall into place: We describe which improvement proposals we need to evaluate and which aspects of each proposal we must consider, in particular. By working through the evaluation tasks which we have defined and checking them against our hypotheses we should be able to generate a sufficient data base to adjust our proposals. One challenge still remaining, though, is to assemble a prototype IR system implementation which accommodates our proposals: We have merely stated requirements for crucial implementation aspects such as indexing and only marginally accounted for efficiency, so these aspects still have to be addressed in the future. Apart from that, the course of our work has inspired a far greater number of ideas than we were able to thoroughly elaborate within the limits of our scope and hence omitted. Also, for most of our improvement proposals further extensions come to mind. In the remainder of this chapter we thus provide a brief summary of ideas which we consider worthwhile to tackle in the future.

8.2. Future Work

We have introduced two interpretations of target conditions: a name-based and granularity-based interpretation. This already goes beyond the name-centred point of view XML Retrieval research has grown used to taking. Yet we propose to analyse in a broader manner what other factors apart from a tag name (or maybe derived from it) the user might want to express. Similarly, we should reconsider, if the three classes of element types we have defined are sufficient, supported by the insights gained by according evaluation runs. For example, a fourth class for referencing elements (like `<ref>`, `<link>`, `<url>`, `<figure>`) may be useful to account for the additional metadata they provide. To practically use element types we also need an (automated) classification technique. For this we should also consider to support context-aware classifications. A `<title>` element, for example, can have quite a different meaning when used as a child element of a `<section>` or a `<person>` element: In the former case it would likely contain a section heading, in the latter case a person's academic degrees.

Another idea which we deem useful to integrate is schema-level matching: Instead of (or in addition to) matching names and paths of each element in a document on the instance level, we can try to perform parts of the matching logic on the schema level. This may be feasible for name matching and path matching, in particular. Schema matching techniques could also be valuable in automatically generating substitution groups for homogeneous collections with several schemas. Another idea in this context is to define meta-modellings: A meta-modelling describes one particular real-world phenomenon; for example, on different levels of abstraction, such phenomena may be an n:m relationship, a value range, or a section title. At indexing time we can analyse element structures which belong to one particular modelling variant and assign them to the corresponding meta-modelling. For queries we can either do the same (detect the meta-modelling based on a given specific modelling) or design a query language which solely operates on the meta-level. We can also use schema-level information for weighting and other activities. The main aim is to increase processing efficiency.

Regarding term proximities, we have already proposed to further refine our proximity level approach in section 4.3. In particular, terms occurring in attributes and attribute values should be handled explicitly. Also the consideration of element length and type may lead to improvements; we could weight the crossing of proximity levels based on length and type, for example. Similar ideas apply for our Relevance Influence Model (RIM): We may define the size of influence regions dynamically based on factors such as element lengths, types, and maybe target scores. Another idea is to introduce a link-based model similar to web search engines: Elements which have many other elements referencing them may have a greater influence region or receive a score bonus. Scoring in general we may enhance by considering statistical correlations of content and structure; for example, if a keyword term x predominantly occurs in elements with the tag name y, we might be able to exploit this information.

Other ideas are inspired by issues we expect to occur in practical XML IR systems: One such issue is error handling. We have assumed documents to be always well-formed and potentially even valid regarding some schema. As we know from traditional IR domains such as web search, documents (e.g. HTML pages) commonly contain syntactic errors; we believe that the same kinds of errors are likely to occur in XML documents as well. Hence we need techniques for exploiting structural properties even for erroneous documents. Another issue is that content in XML documents is often not purely data-centric or document-centric, but mixed. So far we have only taken into account the document-centric parts. If we manage to identify data-oriented structures (such as the header information in a book or article), we may be able to use it as a source of additional metadata on the contents of documents or even fragments. This also holds for comments used in XML files which we can potentially use as metadata as well. Beyond this there are many more issues (such as atomic value handling, string representations of XML fragments, ...) which we believe can help us in further improving XML Retrieval. In other words, there are many interesting challenges still awaiting to be approached!

Appendix

A. Ehrenwörtliche Erklärung (In German)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Christoph R. Hartel
Kaiserslautern, im November 2007

B. Notations

Table B.1.: Overview of formal entities

Entity	**Definition**
Multi-Set	A *multi-set* is an arbitrary collection of values.
Set	A *set* is a duplicate-free multi-set; we denote it by curly brackets, e.g. $\{x, y, z\}$.
List	A *list* is an ordered multi-set of values. We employ the same notation as for sets.
Tuple	A *tuple* is a list of values which has an arbitrary but fixed length; tuples are denoted by angle brackets, e.g. $\langle x, y, z\rangle$.
Interval	An *interval* is a subset of $\mathbb{R}$ which contains every value in between $a \in \mathbb{R}$ and $b \in \mathbb{R}$: (a, b) denotes the *open interval* from a to b (i.e. the interval excluding a and b), $[a, b]$ denotes the *closed interval.* *Half-open intervals* are denoted by $(a, b]$ and $[a, b)$, respectively.
Function	We distinguish the *declaration* and *specification* of functions: The former only defines the name, domain, and range of the function, whereas the latter specifies how input values are transformed into output values. We denote a function definition in arrow notation: For example, $\mathrm{div} : \mathbb{N}^{\geq 1} \times \mathbb{N}^{\geq 1} \longrightarrow \mathbb{R}^{\geq 1}$ defines a function named "div" which takes two parameters with the domain $\mathbb{N}^{\geq 1}$ and maps them to a value in the range $\mathbb{R}^{\geq 1}$. If the range is not a single value, but multi-set of values in X, we denote this as X^n. A function specification we either provide in textual form or denote it as an equation; in the above example, the specification might be $\mathrm{div}(a, b) = \frac{a}{b}$.

Table B.2.: Overview of mathematical sets

Set	**Definition**
$\mathbb{N}^{\geq 0}$	The set of all natural numbers greater than or equal to zero, i.e. $\{0, 1, 2, \ldots\}$. We restrain from using $\mathbb{N}$ as notation to avoid confusion over the inclusion of zero.
$\mathbb{N}^{\geq 1}$	The set of all natural numbers greater than or equal to one, i.e. $\{1, 2, \ldots\}$.
$\mathbb{R}$	The set of all real numbers.
$\mathbb{R}^{\geq 0}$	The set of all real numbers greater than or equal to zero, i.e. $\{r \in \mathbb{R} : r \geq 0\}$.
$\mathbb{R}^{\geq 1}$	The set of all real numbers greater than or equal to one, i.e. $\{r \in \mathbb{R} : r \geq 1\}$.

Table B.3.: Overview of symbols and general notations

Symbol	**Definition**
$\downarrow$	A variable which has been *defined*, that is, it has been assigned a value.
$\uparrow$	A variable which is still *undefined*. For example, a variable which we have declared, but to which we have not yet assigned a value.
$\perp$	A *null value*. Please note that we distinguish between variables being undefined and a variables having a null value: To illustrate this, consider the real-world example of a database storing user records. Each record includes the user's telephone number among other contact details. If we add a new user without entering any data, the telephone number is undefined ($\uparrow$), that is, we do not *know* the user's telephone number. Once we enter the user's data, all fields including the telephone number are defined ($\downarrow$). If the user in question does not *have* a telephone number, however, we use the null value ($\perp$) to indicate this.
$x + y$	The *addition* of two arbitrary numbers $x, y \in \mathbb{R}$. For example, $41 + 1 = 42$.
$x - y$	The *subtraction* of two arbitrary numbers $x, y \in \mathbb{R}$. For example, $44 - 2 = 42$.
$x \cdot y$	The *multiplication* of two arbitrary numbers $x, y \in \mathbb{R}$. For example, $21 \cdot 2 = 42$.
$x : y$	The *division* of two arbitrary numbers $x, y \in \mathbb{R}$ with $y \neq 0$. For example, $84 : 2 = 42$.
$\text{abs}(x)$	The *absolute value* of an arbitrary number $x \in \mathbb{R}$. For example, $\text{abs}(-42) = \text{abs}(42) = 42$.
$\|A\|$	The *length* of an arbitrary collection A. For example, $\|\{1, 2, \ldots, 42\}\| = 42$.
$A - B$	The *subtraction* of two arbitrary sets A and B. For example, $\{42, 43, 44\} - \{43, 44\} = \{42\}$.

C. Query Language Grammars

In section 3.3 we have defined the CAS-QL query language and CAS-QLX as an extension to it. In the following two sections we provide the context-free grammars $G_{\text{CAS-QL}}$ and $G_{\text{CAS-QLX}}$ for CAS-QL and CAS-QLX, respectively. We state a context-free grammar as a four-tuple $\langle V, \Sigma, s, R\rangle$ consisting of a set of variables V, a set of terminal symbols Σ, a start symbol $s \in V$, and a set of production rules R [Wik07].

C.1. CAS-QL Grammar

- $V_{\text{CAS-QL}}$ = { ⟨query⟩, ⟨cond⟩, ⟨keyword-cond⟩, ⟨support-cond⟩, ⟨target-cond⟩, ⟨first-path-expr⟩, ⟨path-expr⟩, ⟨path-delim⟩, ⟨target-expr⟩, ⟨term⟩, ⟨letter⟩, ⟨digit⟩ }
- $\Sigma_{\text{CAS-QL}} = \{\text{a}, \text{b}, \ldots, \text{z}, 0, \ldots, 9, /, \texttt{-}, \texttt{:}, \texttt{>}, \texttt{,}\ , \texttt{.}\}$
- $s_{\text{CAS-QL}}$ = ⟨query⟩
- $R_{\text{CAS-QL}}$ is given by the following rules:

  ⟨query⟩ → ⟨cond⟩ [␣ ⟨query⟩]
  ⟨cond⟩ → (⟨keyword-cond⟩ | ⟨support-cond⟩ | ⟨target-cond⟩) [, w e i g h t = (0 . (1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) | 1 . 0)]
  ⟨keyword-cond⟩ → ⟨term⟩
  ⟨support-cond⟩ → s u p : ⟨first-path-expr⟩ | s u p p o r t : ⟨first-path-expr⟩
  ⟨target-cond⟩ → t g t : ⟨target-expr⟩ | t a r g e t : ⟨target-expr⟩
  ⟨first-path-expr⟩ → / ⟨term⟩ [⟨path-expr⟩]
  ⟨path-expr⟩ → ⟨path-delim⟩ ⟨term⟩ [⟨path-expr⟩]
  ⟨path-delim⟩ → /
  ⟨target-expr⟩ → ⟨term⟩ | (0 | 1) . ⟨digit⟩
  ⟨term⟩ → ⟨letter⟩ | ⟨digit⟩ | - | ⟨term⟩
  ⟨letter⟩ → a | b | . . . | z
  ⟨digit⟩ → 0 | 1 | . . . | 9

C.2. CAS-QLX Grammar

- $V_{\text{CAS-QLX}} = V_{\text{CAS-QL}} \cup$ { ⟨atomic-cond⟩, ⟨single-cond⟩, ⟨op⟩, ⟨preds⟩, ⟨pred⟩, ⟨comp-op⟩ }

- $\Sigma_{\text{CAS-QLX}} = \Sigma_{\text{CAS-QL}} \cup \{$=, <, @, (,), [,], &, |$\}$
- $s_{\text{CAS-QLX}} = s_{\text{CAS-QL}}$
- $R_{\text{CAS-QLX}}$ contains the rules defined for $R_{\text{CAS-QL}}$; in addition, the following rules replace and extend the according rules in $R_{\text{CAS-QL}}$:

  ⟨cond⟩ → ⟨atomic-cond⟩ | (⟨atomic-cond⟩)
  ⟨atomic-cond⟩ → ⟨single-cond⟩ | ⟨single-cond⟩ ␣ ⟨op⟩ ␣ ⟨single-cond⟩
  ⟨single-cond⟩ → (⟨keyword-cond⟩ | ⟨support-cond⟩ | ⟨target-cond⟩)
  ⟨op⟩ → & & | | |
  ⟨first-path-expr⟩ → / ⟨term⟩ [⟨preds⟩] [⟨path-expr⟩]
  ⟨path-expr⟩ → ⟨path-delim⟩ ⟨term⟩ [⟨preds⟩] [⟨path-expr⟩]
  ⟨preds⟩ → [⟨pred⟩ [␣ ⟨preds⟩]]
  ⟨pred⟩ → ⟨comp-op⟩ ⟨term⟩
  ⟨comp-op⟩ → =| < | >| > =| < =
  ⟨path-delim⟩ → / | // | @| - >

D. Glossary

Aggregation	*Result compaction* technique; returns the parent element instead of several relevant children.	112
Aggregation threshold	Controls when aggregation is performed based on the ratio of relevant child elements to all child elements.	113
Attribute	Consists of a name and optional value; describes an XML *element.*	13
Authoritative model	Model for result set generation which only includes elements with a non-zero target score in the result set. An alternative is the *semi-authoritative model.*	109
Axis	Relationship between two XML elements, e.g. parent-child; standard axes are defined by XPath [CD99].	14
Base cost	Initial cost value for path edit operations; the base cost is modified by a *repetition factor* among other factors.	76
Benchmarking	Experimental comparison of a baseline IR system implementation with a modified implementation. Also see *correlation testing.*	118
Bi-directional propagation	Score propagation technique which combines *upwards propagation* and *downwards propagation*, i.e. considers both ancestor and descendant elements.	102
Collection	Arbitrary set of documents; used by an IR system to evaluate queries on.	15

Constraint	A query condition which is evaluated strictly.	11
Content score	Aggregates *direct score*, *non-propagatable score*, and *context score*; computed during final score calculation.	110
Content-and-Structure Retrieval	Most enhanced XML Retrieval variant; considers *keywords*, *support conditions*, and *target conditions*. Also see *Content-only Retrieval*.	19
Content-only Retrieval	Simplified XML Retrieval variant which only considers *keywords*, but no *support conditions* or *target conditions*.	19
Context elements	An element's ancestors, descendants, siblings, and all elements it references or is referenced by.	56
Context score	The score which an element receives based on its context elements' scores, but independently of its direct score.	56
Correlation testing	Experimental evaluation technique to test whether certain variables exhibit a statistical correlation.	118
Data cleaning	Process of identification and correction of anomalies in a given data set [Har06]; involves the handling of *duplicates*.	25
Data-centric fragment	An XML fragment which describes entities in a key/value fashion; also see *document-centric fragment*.	23
Direct content	The textual content of an element excluding its tag name, attribute names and values and its descendant elements.	15
Direct full content	The tag name of an element, its attribute names and their respective values, and its direct content, in this order.	15

Direct score	The score which an element receives exclusively based on its direct content.	56
Document ID	Uniquely identifies a document.	23
Document order	Order of nodes in an XML document tree; defined by the order in which the corresponding elements appear in an XML document.	13
Document space	The set of all documents of an arbitrary IR query on which the query is evaluated; denoted as $\mathcal{D}$.	12
Document-centric fragment	An XML fragment which is dominated by textual content, e.g. an article; also see *data-centric fragment*.	23
Downwards propagation	Score propagation strategy which considers only ancestor elements; also see *upwards propagation*.	102
Duplicate	One of several instances of a single logical document in a collection.	25
Dynamic cost function	Calculates the path edit cost; the cost of operations depends on characteristics of the paths not just on the operators used.	75
Effort-precision	Measures how much effort a user needs to spend in order to attain a particular gain value. Also see *eXtended cumulated gain*.	122
Element	Entities in an XML document; may contain other elements, attributes, and textual content.	13
Element space	Set of all elements in the *fragment space*; denoted as $\mathcal{E}$.	15
Element type	Group of *elements* with common properties such as the tag name semantics.	47
Evaluation run	One execution of selected queries on a particular test *collection* using a defined implementation and particular configuration settings.	125

Evaluation task	Addresses the evaluation of one concise aspect of XML Retrieval, typically an improvement proposal; defines how to test the aspect, which test collections to run the evaluation on, and what the expected outcomes are.	125
Excessive element	Path component which occurs in a support condition, but not in an element which matches it.	64
Expanded name	An element or attribute name including a reference to its namespace.	72
Explicit condition	A query condition which is supplied by the user, e.g. as part of the query string; also see *implicit condition.*	18
eXtended cumulated gain	Adaption of cumulated gain metrics to XML Retrieval; focusses on the cumulated benefit a user can attain by looking at the first i ranks of a result set.	121
Extended score tuple	Tuple consisting of the *direct score*, *non-propagatable score*, *target score*, and *context score.* Also see *simplified score tuple.*	58
eXtensible Markup Language	Popular generic document format which features an explicit logical structure and metadata.	13
Fragment	Arbitrary subtree of an XML document; unlike an *element* it reflects the nesting structure of XML documents.	15
Fragment space	Set of all fragments belonging to documents in the *document space.*	15
Gain-recall	Ratio of cumulated gain that is obtained at a particular rank to the total ideal gain that can be archived for the fragments in the collection used. Also see *eXtended cumulated gain.*	122

Granularity	Determines how "fine-grained" an XML fragment is; there exist various granularity measures.	27
Heterogeneity	Property of a document collection which expresses that documents in the collection contain the various kinds of content; opposite of *homogeneity.*	25
Hint	A query condition which is evaluated vaguely.	11
Homogeneity	Property of a document collection which expresses that all documents in the collection contain the same kind of content; opposite of *heterogeneity.*	25
Ideal gain vector	Vector of gain values which are obtained by a perfect result set. Also see *eXtended cumulated gain.*	121
Ideal result list	Set of all fragments in the fragment space with a non-zero relevance, ordered by decreasing relevance. Also see *ideal gain vector.*	121
Implicit condition	A query condition which is generated by the IR system; also see *explicit condition.*	18
Influence region	Set of elements which we consider to calculate an element's context score; relies on *proximity levels.*	107
Information Retrieval	Process of informing a user on the existence (or non-existence) and whereabouts of documents and/or document parts.	10
Instability	The change frequency of a document collection.	26
Keyword	Condition on the content of documents; a keyword consists of exactly one term in the *term space.*	18

Local name	Part of a name in XML which remains after stripping the namespace reference; also see *expanded name.*	73
Missing element	Path component which we have to insert into the path of a support condition to match a particular element's path.	64
Modelling variant	One particular way of modelling a given real-world phenomenon.	59
Namespace	Provides a vocabulary of element and attribute names to enable reuse and ease the creation of uniform XML documents.	14
Near-misses	Fragments in the result set which are imperfect matches, but have a high proximity to perfect matches.	119
Non-propagatable score	A score assigned to an element e which must not be propagated to other elements.	57
Normalised cumulated gain	An *eXtended cumulated gain* value which has been normalised to the $[0,1]$; enables comparison of gain values across different documents.	121
Overlap	The containment of fragments in other fragments.	19
Overlap removal	*Result compaction* technique; ensures that no overlapping fragments are returned.	112
Path component	Part of an element path which corresponds to one node in an XML document tree.	68
Pre-propagation	Scoring of an element which cannot be restricted to an element's direct content and thus results in a *non-propagatable score.*	57

Precision	Ratio of relevant to irrelevant result items returned by the IR system.	119
Proximity level	Represents one particular hierarchical relation of a pair of term occurrences in an XML document.	85
Query	System of conditions on the documents (or document parts) to be retrieved.	11
Ranking	Synonym to *result set*; stresses the relevance-induced ordering of results commonly employed.	111
Recall	Ratio of relevant result items returned to all result items in the collection.	119
Recursive content	The direct content of an element collated with the direct contents of all its descendants, in document order.	15
Recursive full content	The full content of an element collated with the full contents of all its descendants in document order.	15
Reference	A logical link from an XML element to a different element; elements may be located in the same document or in different documents.	14
Relevance	The degree $r \in [0, 1]$ to which a document fragment satisfies an information need.	12
Relevance Influence Model	Model for score propagation based *upwards propagation* and *proximity levels.*	105
Relevance threshold	Only elements with a score higher than this threshold are considered relevant.	113

Repetition factor	Penalises repeated applications of an edit operation to a path; modifies the *base cost* of an edit operation.	76
Result compaction	Techniques to compact the result set; includes *aggregation*, *specialisation*, and *overlap removal.*	112
Result set	Set of fragments which we return to the user in answer to his query; typically ordered descendingly by the scores of the fragments' root elements.	111
Retrieval quality	Describes how well the results generated by an IR system satisfy the user's information need; formally, retrieval quality is a function of retrieved relevant fragments and all fragments in a collection.	8
Scenario	One specific configuration of a set of environmental properties of an IR system.	22
Schema	Defines constraints regarding XML documents; a document conforming to a schema is *valid.*	14
Score	An IR system's estimation of a fragment's *relevance.*	12
Score adjustment factor	Expresses the semantic equivalence of names used in a *substitution group.*	66
Score duplication	Assigning an element an overly high score due to counting a single relevance-affecting entity (e.g. a term occurrence) multiple times.	57
Score propagation	The calculation of an element's context score $\mathrm{s}_{\mathrm{ctx}} \in [0, 1]$ based on the direct scores of its context elements.	58

Semi-authoritative model	Model for result set generation which also includes elements with a zero target score in the result set. An alternative is the *authoritative model.*	109
Simplified score tuple	Tuple consisting of the *direct score* and the *context score.* Also see *extended score tuple.*	56
Specialisation	*Result compaction* technique; returns a single highly relevant child element instead of several relevant children or the parent element.	112
Specialisation threshold	Controls when specialisation may occur based on the score of sibling and parent elements.	113
Stability	The opposite of *instability.*	26
Static length biasing	Modification of scores to reward elements of a certain length; a more enhanced solution is the use of *granularities.*	100
Substituted element	Path component which we have to substitute by a different component to match a particular element's path.	64
Substitution group	Contains all names which may be substituted for a given name together with a *score adjustment factor* for each name.	66
Support conditions	Expresses that certain patterns in the document structure will render the corresponding part of the document helpful regarding the user's information need; consists of a path with optional predicates.	18
Tag name candidate attribute	Attribute which refines the semantics of the corresponding element's tag name.	79

Target condition	Defines which parts of a document to return to the user; consists of a tag name or a granularity value.	18
Target score	Score resulting from the evaluation of target conditions.	58
Target scoring	Scoring of elements based on target conditions.	98
Term space	Set of all terms (e.g. words, numbers) which occur in any document in the *document space.*	15
Top k approach	Restriction of the *result set* to a fixed maximum size by returning only the *k* most relevant elements.	27
Upwards propagation	Score propagation strategy which considers only descendant elements. Also see *downwards propagation.*	102
Validity	An XML document is valid, if and only if it conforms to a given schema.	14
Well-formedness	An XML document is well-formed, if and only if its nesting structure observes the rules of the XML specification.	14
XML document	Ordered, labelled tree of XML *elements* which is *well-formed.*	13
XML Retrieval	*Information Retrieval* over *XML documents*; its main potentials are the explicitness and genericness of the logical document structure and fragment-oriented retrieval.	17

E. Bibliography

[AKJ05] Paavo Arvola, Jaana Kekäläinen, and Marko Junkkari. Query Evaluation with Structural Indices. In Fuhr et al. [FLMK06], pages 134–145.

[AVF06] Faiza Abbaci, Jean-Baptiste Valsamis, and Pascal Francq. Index and Search XML Documents by Combining Content and Structure. In Hamid R. Arabnia, editor, *International Conference on Internet Computing*, pages 107–112. CSREA Press, 2006.

[AYBB+07] Sihem Amer-Yahia, Chavdar Botev, Stephen Buxton, Pat Case, Jochen Doerre, Mary Holstege, Jim Melton, Michael Rys, and Jayavel Shanmugasundaram, editors. *XQuery 1.0 and XPath 2.0 Full-Text 1.0*, 2007. W3C Working Draft 18 May 2007; Available from: `http://www.w3.org/TR/2007/WD-xpath-full-text-10-20070518`.

[AYKM+05] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and Content Scoring for XML. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB 2005 Proceedings*, pages 361–372. ACM, 2005.

[AYL06] Sihem Amer-Yahia and Mounia Lalmas. XML search: languages, INEX and scoring. *SIGMOD Record*, 35(4):16–23, 2006.

[AYLP04] Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Shashank Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In Gerhard Weikum, Arnd Christian König, and Stefan Deßloch, editors, *SIGMOD 2004 Proceedings*, pages 83–94. ACM, 2004.

[BCF+07] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon, editors. *XQuery 1.0: An XML Query Language*, 2007. W3C Recommendation 23 January 2007; Available from: `http://www.w3.org/TR/2007/REC-xquery-20070123`.

[Bei07] Michel Beigbeder. Structured Content-Only Information Retrieval Using Term Proximity and Propagation of Title Terms. In Fuhr et al. [FLT07], pages 29–36.

[BHLT06] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin, editors. *Namespaces in XML 1.0 (Second Edition)*, 2006. W3C Recommendation 16 August 2006; Available from: http://www.w3.org/TR/2006/REC-xml-names-20060816.

[Bla07] Paul E. Black, editor. *Dictionary of Algorithms and Data Structures (Online), "Levenshtein distance"*. U.S. National Institute of Standards and Technology, 2007. From: Mikhail J. Atallah, editor. Algorithms and Theory of Computation Handbook, CRC Press, 1999. Available from: http://www.nist.gov/dads/HTML/Levenshtein.html (Accessed: 21 August 2007).

[BPM04] Paul V. Biron, Kaiser Permanente, and Ashok Malhotra, editors. *XML Schema Part 2: Datatypes – Second Edition*, 2004. W3C Recommendation 28 October 2004; Available from: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.

[BPSM+06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau, editors. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, 2006. W3C Recommendation 16 August 2006, edited in place 29 September 2006; Available from: http://www.w3.org/TR/2006/REC-xml-20060816.

[BW01] Michael Barg and Raymond K. Wong. Structural Proximity Searching for Large Collections of Semi-Structured Data. In *CIKM 2001 Proceedings*, pages 175–182. ACM, 2001.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1999.

[CCPC+06] Jennifer Chu-Carroll, John Prager, Krzysztof Czuba, David Ferrucci, and Pablo Duboue. Semantic search via XML fragments: a high-precision approach to IR. In *SIGIR 2006 Proceedings*, pages 445–452, New York, NY, USA, 2006. ACM Press.

[CD99] James Clark and Steve DeRose, editors. *XML Path Language (XPath) – Version 1.0*, 1999. W3C Recommendation 16 November 1999; Available from: http://www.w3.org/TR/1999/REC-xpath-19991116.

[CEL+02] D. Carmel, N. Efraty, G. Landau, Y. Maarek, and Y. Mass. An Extension of the Vector Space Model for Querying XML Documents via XML Fragments. In R. Baeza-Yates, N. Fuhr, and Y. Maarek, editors, *Second Edition of the XML and IR Workshop*, volume 36, 2002.

[Chi02] Boris Chidlovskii. Schema extraction from XML collections. In *JCDL 2002 Proceedings*, pages 291–292. ACM, 2002.

[CKK+02] Sara Cohen, Yaron Kanza, Yakov Kogan, Yehoshua Sagiv, Werner Nutt, and Alexander Serebrenik. EquiX – A search and query language for XML.

Journal of the American Society for Information Science and Technology, 53(6):454–466, 2002.

[Cla05] Charles L. A. Clarke. Controlling overlap in content-oriented XML retrieval. In Ricardo A. Baeza-Yates, Nivio Ziviani, Gary Marchionini, Alistair Moffat, and John Tait, editors, *SIGIR 2005 Proceedings*, pages 314–321. ACM, 2005.

[CLM+01] R. C. Connor, D. Lievens, P. Manghi, S. Neely, and F. Simeoni. Extracting Typed Values from XML Data. In *OOPSLA 2001 Workshop on Objects, XML and Databases*, 2001.

[CMKS03] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSEarch: A Semantic Search Engine for XML. In *VLDB 2003 Proceedings*, pages 45–56, 2003.

[CMM+03] David Carmel, Yoëlle S. Maarek, Matan Mandelbrod, Yosi Mass, and Aya Soffer. Searching XML documents via XML fragments. In *SIGIR 2003 Proceedings*, pages 151–158. ACM, 2003.

[CP02] Paolo Ciaccia and Wilma Penzo. Adding Flexibility to Structure Similarity Queries on XML Data. In Troels Andreasen, Amihai Motro, Henning Christiansen, and Henrik Legind Larsen, editors, *FQAS 2002 Proceedings*, volume 2522 of *LNCS*, pages 124–139. Springer, 2002.

[CRF00] Donald D. Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In Suciu and Vossen [SV01], pages 1–25.

[CRF03] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In Subbarao Kambhampati and Craig A. Knoblock, editors, *IIWeb 2003 Proceedings*, pages 73–78, 2003.

[CT94] William B. Cavnar and John M. Trenkle. N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[DG06] Ludovic Denoyer and Patrick Gallinari. The Wikipedia XML Corpus. In Fuhr et al. [FLT07], pages 12–19.

[DJG+07] Steven DeRose, Ron Daniel Jr., Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh, editors. *XML Pointer Language (XPointer)*, 2007. W3C Working Draft 16 August 2002; Available from: `http://www.w3.org/TR/2002/WD-xptr-20020816`.

[Dop05] Philipp Dopichaj. The University of Kaiserslautern at INEX 2005. In Fuhr et al. [FLMK06], pages 196–210.

[Dop06] Philipp Dopichaj. Element Retrieval in Digital Libraries: Reality Check. In Trotman and Geva [TG06].

[Dop07] Philipp Dopichaj. Improving Content-oriented XML Retrieval by Applying Structrual Patterns. In *ICEIS 2007 Proceedings*, 2007.

[EDHJ06] Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin, editors. *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006).* ACM, 2006.

[EL00] Daniel Egnor and Robert Lord. Structured Information Retrieval using XML. In *Working Notes of the ACM SIGIR Workshop on XML and Information Retrieval*, 2000.

[Feg04] Leonidas Fegaras. XQuery Processing with Relevance Ranking. In Zohra Bellahsene, Tova Milo, Michael Rys, Dan Suciu, and Rainer Unland, editors, *XSym 2004 Proceedings*, volume 3186 of *LNCS*, pages 51–65. Springer, 2004.

[FG00] N. Fuhr and K. 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.* ACM, 2000.

[FG01] Norbert Fuhr and Kai Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel, editors, *SIGIR 2001 Proceedings*, pages 172–180. ACM, 2001.

[FG04] Norbert Fuhr and Kai Großjohann. XIRQL: An XML query language based on information retrieval concepts. *ACM Trans. Inf. Syst.*, 22(2):313–356, 2004.

[FGKL02] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors. *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX 2002)*, 2002.

[FHM05] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[FL06] Ingo Frommholz and Ray Larson. The Heterogeneous Collection Track at INEX 2006. In Fuhr et al. [FLT07], pages 312–317.

[FLM03] Norbert Fuhr, Mounia Lalmas, and Saadia Malik, editors. *Proceedings of the Second INEX Workshop (INEX 2003)*, 2003.

[FLMK06] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Gabriella Kazai, editors. *Advances in XML Information Retrieval and Evaluation, 4th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2005), Revised Selected Papers*, volume 3977 of *LNCS*. Springer, 2006.

[FLMS05] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Zoltán Szlávik, editors. *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2004), Revised Selected Papers*, volume 3493 of *LNCS*. Springer, 2005.

[FLT07] Norbert Fuhr, Mounia Lalmas, and Andrew Trotman, editors. *Comparative Evaluation of XML Information Retrieval Systems, 5th International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX 2006), Revised Selected Papers*, volume 4518 of *LNCS*. Springer, 2007.

[FML03] Norbert Fuhr, Saadia Malik, and Mounia Lalmas. Overview of the Initiative for the Evaluation of XML retrieval (INEX) 2003. In Fuhr et al. [FLM03], pages 1–11.

[GAFG02] Norbert Gövert, Mohammad Abolhassani, Norbert Fuhr, and Kai Großjohann. Content-oriented XML retrieval with HyRex. In Fuhr et al. [FGKL02], pages 26–32.

[Gev04] Shlomo Geva. GPX – Gardens Point XML IR at INEX 2004. In Fuhr et al. [FLMS05], pages 211–223.

[Gev05] Shlomo Geva. GPX – Gardens Point XML IR at INEX 2005. In Fuhr et al. [FLMK06], pages 240–253.

[GHT06] Shlomo Geva, Marcus Hassler, and Xavier Tannier. XOR – XML Oriented Retrieval Language. In Trotman and Geva [TG06].

[GK02] Norbert Gövert and Gabriella Kazai. Overview of the Initiative for the Evaluation of XML retrieval (INEX) 2002. In Fuhr et al. [FGKL02], pages 1–17.

[Goo02] Michel Goossens. Writing Documentation Using DocBook – Using DocBook at CERN, 2002. Available from: `http://xml.web.cern.ch/XML/goossens/dbatcern` (Accessed: 08 October 2007).

[GSBS03] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors, *SIGMOD 2003 Proceedings*, pages 16–27. ACM, 2003.

[Har06] Christoph R. Hartel. Data Cleaning und Record Matching, 2006. Seminar Information Integration, Research Group Databases and Information Systems, TU Kaiserslautern, Available from: `http://wwwdvs.informatik.uni-kl.de/courses/seminar/SS2006/DokumenteExtern/Ausarbeitung08_Hartel.pdf`, In German.

[HHW$^+$04] Arnaud Le Hors, Philippe Le H'egaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne, editors. *Document Object Model (DOM) Level 3 Core Specification – Version 1.0*, 2004. W3C Recommendation 07 April 2004; Available from: `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407`.

[HPB03] Vagelis Hristidis, Yannis Papakonstantinou, and Andrey Balmin. Keyword Proximity Search on XML Graphs. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *ICDE 2003 Proceedings*, pages 367–378. IEEE Computer Society, 2003.

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[KL06] Gabriella Kazai and Mounia Lalmas. eXtended cumulated gain measures for the evaluation of content-oriented XML retrieval. *ACM Trans. Inf. Syst.*, 24(4):503–542, 2006.

[KLdV04] Gabriella Kazai, Mounia Lalmas, and Arjen P. de Vries. The overlap problem in content-oriented XML retrieval evaluation. In *SIGIR 2004 Proceedings*, pages 72–79, New York, NY, USA, 2004. ACM Press.

[KLP04] G. Kazai, M. Lalmas, and B. Piwowarski. INEX 2004 Relevance Assessment Guide. In Fuhr et al. [FLMS05], pages 241–248.

[KMdRkS04] Jaap Kamps, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. Length normalization in XML Retrieval. In Mark Sanderson, Kalervo Järvelin, James Allan, and Peter Bruza, editors, *SIGIR 2004 Proceedings*, pages 80–87. ACM, 2004.

[KMdRS02] Jaap Kamps, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. The Importance of Morphological Normalization for XML Retrieval. In Fuhr et al. [FGKL02], pages 41–48.

[KMdRS05] Jaap Kamps, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. Structured queries in XML retrieval. In Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken, editors, *CIKM 2005 Proceedings*, pages 4–11. ACM, 2005.

[KMdRS06] Jaap Kamps, Maarten Marx, Maarten de Rijke, and Börkur Sigurbjörnsson. Articulating information needs in XML query languages. *ACM Trans. Inf. Syst.*, 24(4):407–436, 2006.

[LKK$^+$06] Mounia Lalmas, Gabriella Kazai, Jaap Kamps, Jovan Pehcevski, Benjamin Piwowarski, and Stephen Robertson. INEX 2006 Evaluation Measures. In Fuhr et al. [FLT07], pages 20–34.

[LP06] M. Lalmas and B. Piwowarski. INEX 2006 Relevance Assessment Guide. In Fuhr et al. [FLT07].

[LPT06] Miro Lehtonen, Nils Pharo, and Andrew Trotman. A Taxonomy for XML Retrieval Use Cases. In Fuhr et al. [FLT07], pages 267–272.

[LR04] Mounia Lalmas and Thomas Rölleke. Modelling Vague Content and Structure Querying in XML Retrieval with a Probabilistic Object-Relational Framework. In Henning Christiansen, Mohand-Said Hacid, Troels Andreasen, and Henrik Legind Larsen, editors, *FQAS 2004 Proceedings*, volume 3055 of *LNCS*, pages 432–445. Springer, 2004.

[LRM05] Wei Lu, Stephen E. Robertson, and Andrew MacFarlane. Field-Weighted XML Retrieval Based on BM25. In Fuhr et al. [FLMK06], pages 161–171.

[LT07] M. Lalmas and A. Tombros. INEX 2002 – 2006: Understanding XML Retrieval Evaluation. In *DELOS Conference on Digital Libraries*, 2007.

[LW75] Roy Lowrance and Robert A. Wagner. An Extension of the String-to-String Correction Problem. *J. ACM*, 22(2):177–183, 1975.

[LYWS06] Xiaoguang Li, Ge Yu, Daling Wang, and Baoyan Song. Evaluating Interconnection Relationship for Path-Based XML Retrieval. In Karl Aberer, Zhiyong Peng, Elke A. Rundensteiner, Yanchun Zhang, and Xuhui Li, editors, *WISE 2006 Proceedings*, volume 4255 of *LNCS*, pages 506–511. Springer, 2006.

[MAYKS05] Amélie Marian, Sihem Amer-Yahia, Nick Koudas, and Divesh Srivastava. Adaptive Processing of Top-K Queries in XML. In *ICDE 2005 Proceedings*, pages 162–173. IEEE Computer Society, 2005.

[MBF$^+$90] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[ME96] Alvaro E. Monge and Charles Elkan. The Field Matching Problem: Algorithms and Applications. In *Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[MHB06] Vojkan Mihajlovic, Djoerd Hiemstra, and Henk Ernst Blok. Vague Element Selection and Query Rewriting for XML Retrieval. In *Proceedings of the sixth Dutch-Belgian Information Retrieval Workshop (DIR 2006)*, pages 11–18, 2006.

[MM03] Y. Mass and M. Mandelbrod. Retrieving the most relevant XML components. In Fuhr et al. [FLM03], pages 53–58.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008. Draft of 13 September 2007; Available from: `http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html`.

[MTLF06] Saadia Malik, Andrew Trotman, Mounia Lalmas, and Norbert Fuhr. Overview of INEX 2006. In Fuhr et al. [FLT07], pages 1–11.

[Nav01] Gonzalo Navarro. A Guided tour to Approximate String Matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[OT03] R. O'Keefe and A. Trotman. The Simplest Query Language That Could Possibly Work. In Fuhr et al. [FLM03], pages 167–174.

[Pan04] Hanglin Pan. Relevance Feedback in XML Retrieval. In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and Athena Vakali, editors, *EDBT 2004 Workshops*, volume 3268 of *LNCS*, pages 187–196. Springer, 2004.

[Peh06] J. Pehcevski. Relevance in XML retrieval: the user perspective. In Trotman and Geva [TG06].

[PL04] Benjamin Piwowarski and Mounia Lalmas. Providing consistent and exhaustive relevance assessments for XML retrieval evaluation. In David Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *CIKM 2004 Proceedings*, pages 361–370. ACM, 2004.

[PMM07] Eugen Popovici, Gildas Ménier, and Pierre-François Marteau. SIRIUS XML IR System at INEX 2006: Approximate Matching of Structure and Textual Content. In Fuhr et al. [FLT07], pages 121–133.

[Pok93] Jaroslav Pokorný. Semantic Relativism in Conceptual Modeling. In *Proceedings of the 4th International Conference on Database and Expert Systems Applications (DDEXA 1993)*, pages 48–55. Springer, 1993.

[PT05] Jovan Pehcevski and James A. Thom. HiXEval: Highlighting XML Retrieval Evaluation. In Fuhr et al. [FLMK06], pages 43–57.

[RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs, editors. *HTML 4.01 Specification*, 1999. W3C Recommendation 24 December 1999; Available from: `http://www.w3.org/TR/1999/REC-html401-19991224`.

[RS03] Yves Rasolofo and Jacques Savoy. Term Proximity Scoring for Keyword-Based Retrieval Systems. In Fabrizio Sebastiani, editor, *ECIR 2003 Proceedings*, volume 2633 of *LNCS*, pages 207–218. Springer, 2003.

[RWdV06] Georgina Ramírez, Thijs Westerveld, and Arjen P. de Vries. Using small XML elements to support relevance. In Efthimiadis et al. [EDHJ06], pages 693–694.

[Sch01] Torsten Schlieder. Similarity Search in XML Data using Cost-Based Query Transformations. In *WebDB 2001 Proceedings*, pages 19–24, 2001.

[SHB05] Karen Sauvagnat, Lobna Hlaoua, and Mohand Boughanem. XFIRM at INEX 2005: ad-hoc, heterogeneous and relevance feedback tracks. In Fuhr et al. [FLMK06], pages 88–103.

[Sin01] Amit Singhal. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43, 2001.

[SK05] Börkur Sigurbjörnsson and Jaap Kamps. The Effect of Structured Queries and Selective Indexing on XML Retrieval. In Fuhr et al. [FLMK06], pages 104–118.

[SKdR04] Börkur Sigurbjörnsson, Jaap Kamps, and Maarten de Rijke. Mixture Models, Overlap, and Structural Hints in XML Element Retrieval. In Fuhr et al. [FLMS05], pages 196–210.

[SM00] Torsten Schlieder and Holger Meuss. Result Ranking for Structured Queries against XML Documents. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, 2000.

[SMGL06] Ismael Sanz, Marco Mesiti, Giovanna Guerrini, and Rafael Berlanga Llavori. *rHeX*: An Approximate Retrieval System for Highly Heterogeneous XML Document Collections. In Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Böhm, Alfons Kemper, Torsten Grust, and Christian Böhm, editors, *EDBT 2006 Proceedings*, volume 3896 of *LNCS*, pages 1186–1189. Springer, 2006.

[ST06] Ralf Schenkel and Martin Theobald. Structural Feedback for Keyword-Based XML Retrieval. In Mounia Lalmas, Andy MacFarlane, Stefan M. Rüger, Anastasios Tombros, Theodora Tsikrika, and Alexei Yavlinsky, editors, *ECIR 2006 Proceedings*, volume 3936 of *LNCS*, pages 326–337. Springer, 2006.

[SV01] Dan Suciu and Gottfried Vossen, editors. *The World Wide Web and Databases, Third International Workshop (WebDB 2000), Selected Papers*, volume 1997 of *LNCS*. Springer, 2001.

[TBMM04] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, editors. *XML Schema Part 1: Structures – Second Edition*, 2004. W3C Recommendation 28 October 2004; Available from: `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028`.

[TG06] Andrew Trotman and Shlomo Geva, editors. *Proceedings of the SIGIR 2006 Workshop on XML Element Retrieval Methodology*, 2006.

[TI06] Krishnaprasad Thirunarayan and Trivikram Immaneni. Flexible Querying of XML Documents. In Floriana Esposito, Zbigniew W. Ras, Donato Malerba, and Giovanni Semeraro, editors, *ISMIS 2006 Proceedings*, volume 4203 of *LNCS*, pages 198–207. Springer, 2006.

[TL05] Andrew Trotman and Mounia Lalmas. The Interpretation of CAS. In Fuhr et al. [FLMK06], pages 58–71.

[TL06] Andrew Trotman and Mounia Lalmas. Strict and vague interpretation of XML-retrieval queries. In Efthimiadis et al. [EDHJ06], pages 709–710.

[TPL06] Andrew Trotman, Nils Pharo, and Miro Lehtonen. XML-IR Users and Use Cases. In Fuhr et al. [FLT07], pages 274–286.

[TS04a] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In Fuhr et al. [FLMS05], pages 16–40.

[TS04b] Andrew Trotman and Börkur Sigurbjörnsson. NEXI, Now and Next. In Fuhr et al. [FLMS05], pages 41–53.

[TW00] Anja Theobald and Gerhard Weikum. Adding Relevance to XML. In Suciu and Vossen [SV01], pages 105–124.

[TW02] Anja Theobald and Gerhard Weikum. The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking. In Christian S. Jensen, Keith G. Jeffery, Jaroslav Pokorný, Simonas Saltenis, Elisa Bertino, Klemens Böhm, and Matthias Jarke, editors, *EDBT 2002 Proceedings*, volume 2287 of *LNCS*, pages 477–495. Springer, 2002.

[TZ07] Tao Tao and ChengXiang Zhai. An Exploration of Proximity Measures in Information Retrieval. In Fabian Kaiser, Holger Schwarz, Mihály Jakob, Wessel Kraaij, Arjen P. de Vries, Charles L. A. Clarke, Norbert Fuhr, and Noriko Kando, editors, *SIGIR 2007 Proceedings*, pages 295–302. ACM, 2007.

[vR79] C. J. van Rijsbergen. *Information Retrieval.* Dept. of Computer Science, University of Glasgow, 2nd edition, 1979. Available from: `http://www.dcs.gla.ac.uk/Keith/Preface.html`.

[vZ05] Roelof van Zwol. ^{3}-SDR and Effective Use of Structural Hints. In Fuhr et al. [FLMK06], pages 146–160.

[WGE06] Alan Woodley, Shlomo Geva, and Sylvia L. Edwards. What XML-IR Users May Want. In Fuhr et al. [FLT07], pages 423–431.

[Wik07] Wikipedia. Context-free grammar — Wikipedia, The Free Encyclopedia, 2007. Available from: `http://en.wikipedia.org/w/index.php?title=Context-free_grammar&oldid=164265715` (Accessed: 13 October 2007).

[XP05] Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD 2005 Proceedings*, pages 527–538, New York, NY, USA, 2005. ACM Press.

[XX07] Guangming Xing and Zhonghang Xia. Classifying XML Documents Based on Structure/Content Similarity. In Fuhr et al. [FLT07], pages 342–353.

[ZD96] J. Zobel and P. Dart. Phonetic String Matching: Lessons from Information Retrieval. In *SIGIR 1996 Proceedings*, pages 166–172, 1996.