IMPROVING CONTENT-ORIENTED XML RETRIEVAL BY APPLYING STRUCTURAL PATTERNS

Philipp Dopichaj
University of Kaiserslautern
Gottlieb-Daimler-Str.
67663 Kaiserslautern, Germany

Keywords: XML retrieval, knowledge management, fuzzy logic.

Abstract: XML is the perfect format for storing (mostly) textual documents in a knowledge management system; its flexibility enables users to store both highly structured data and free text in the same document. For knowledge management, it is important to be able to search the free-text parts effectively; users need to find the information that helps them solve their problem without having to wade through much information that is not relevant for their problem. Content-oriented XML retrieval addresses this challenge: In contrast to traditional information retrieval, documents are not considered atomic units, that is, elements such as sections or paragraphs can be returned. One implication of this is that results can overlap (for example a paragraph and the surrounding section). Although overlapping results are undesirable in the final retrieval result as presented to the user, they can help to improve the quality of the final result: We take advantage of overlaps by applying patterns to small subtrees of the retrieval result (result contexts); matching patterns adjust the retrieval status values of the involved node in order to promote the best results. We demonstrate on the INEX 2005 test collection that this postprocessing can lead to a significant improvement in retrieval quality.

1 INTRODUCTION

XML is the perfect format for storing (mostly) textual documents in a knowledge management system; its flexibility enables users to store both highly structured data (like database records) and free text in the same document. The data-centric parts can be searched using query languages like XPath and XQuery, where exact conditions on the structure can be imposed. For knowledge management, however, it is important to be able to search the free-text parts effectively; users need to find the information that helps them solve their problem *without having to wade through much information that is not relevant for their problem.*

To this end, *content-oriented XML retrieval* can help: In content-oriented XML retrieval, documents are not considered atomic entities as they are in traditional text-based information retrieval: A retrieval result can not only contain complete documents, but also elements such as chapters or paragraphs. This is convenient for the user, who does not have to examine large chunks of irrelevant context information to find the possibly small units of information he seeks, but it gives rise to the new issue of handling overlap (Kazai et al., 2004). XML documents are hierarchical (like document–chapters–sections–paragraphs), so one matching result may be embedded in another one. For example, if a section contains the keywords from the user's query, the enclosing chapter and document also contain these keywords, so they also match the query (possibly to a lower degree).

To provide good results to the searcher, we aim at making sure that the results are as diverse as reasonably possible. In this paper, we discuss how this problem can be approached by examining small subtrees from the retrieval result for adjusting the scores to better match the searcher's intentions. The method makes no assumptions about the schemas of the documents, as long as the logical structure of the text corresponds to the tagged structure of the XML files. Because of this, it should be possible to use it on heterogeneous collections of textual documents that fulfill this basic requirement; we do not expect the method to be useful for data-centric XML.

In addition to presenting the general framework, we establish several example patterns and evaluate their performance on the INEX 2005 data set, showing significant improvements over the baseline.

2 RELATED WORK

Overlapping results are natural in the context of semi-structured retrieval, so XML retrieval engines have to deal with this.

On the one hand, overlapping results can be annoying for the user, so the retrieval engine should strive to deliver as little redundant content as possible; Kazai et al. (2004) discuss the overlap problem in detail. In practice, however, this was not always enforced by the evaluation metrics: Kekäläinen et al. (2005) submitted retrieval runs with varying degrees of overlap to the workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, and found that returning overlapping elements appears to improve the score.

To address this, INEX 2005 features a new task that explicitly disallows the submission of overlapping documents, CO.Focused (Malik et al., 2006). Our paper aims at improving results for this task by deciding which of parent or children to prefer.

Although it is not desirable to present overlapping results to the user, they can be used to improve the quality of the retrieval results. Several researchers make use of the context of an element in order to improve retrieval quality.

Before the advent of XML, Salton et al. (1993) use textual context for improving results for passage retrieval; this includes considering titles and section headings. Ogilvie and Callan (2005) use the context of an element by incorporating the parent's language model. Arvola et al. (2005) use the context of retrieval nodes by incorporating evidence from other nodes in the same document into the retrieval status value (RSV) of a node. Their approach differs from ours in that they use the scores of the enclosing elements to alter the score of a node; only the score of these nodes is taken into account, and only the ancestors are considered. Ramírez et al. (2006) use small elements with high RSVs to adapt the RSV of the enclosing element, for example, a match in the section title will increase the corresponding section's RSV. This requires preparation and knowledge of the documents' schema, because all these relationships must be modeled explicitly. The approach we present in the remainder of this paper is schema-independent.

3 CONTEXT-BASED SCORE ADJUSTMENT

For our approach, we assume that we get as input the results from a traditional information retrieval engine, where the textual contents of each XML element is considered a document. The implication is that each XML document is represented by a multitude of fragments corresponding to XML elements in the index.

Like the original document, the retrieval results are a set of trees, where each tree contains the result fragments from one XML document. Results from distinct documents cannot overlap, so we only consider the result tree from a single document in the following discussion. In a real search engine, the result documents can be processed separately (even concurrently), and the elements that form the tree nodes are later re-arranged in a list and sorted according to the updated retrieval status values (RSVs).

In order to keep the amount of processing reasonable, we examine small sub-trees (called *result contexts*) of each document tree and use the data contained therein to adjust the RSVs.

3.1 Result Context

For each non-leaf node, the result context consists of this node and the children that contain at least one hit, that is, children with a non-zero RSV. For each node, we consider the following information for deciding how to change the nodes' scores: The RSV, the length, and the position inside the parent node (this is zero if the parent and the child start with the same token). Both length and position are measured in tokens roughly corresponding to words (as determined by automatic analysis); they do not vary from retrieval run to retrieval run, so they can be stored in the index.

This information can be visualized in two dimensions, one for the lengths and positions of the text contents of the elements and the other for the RSV. Figure 1 shows an example XML fragment and how it can be visualized. The horizontal position of the left-hand side of each rectangle denotes the starting position in the text of the parent element, and its width corresponds to the length of the text it contains (this implies that the parent element occupies the width of the diagram). The parent element (in Figure 1, the root element /sec[1]) is the reference for the scale of the horizontal axis.

3.2 Context Patterns

While examining the result contexts of test retrieval runs, we found several distinctive patterns that war-

```
<sec>
Hello, <b>world!</b>
How <i>are</i> you?
</sec>
```

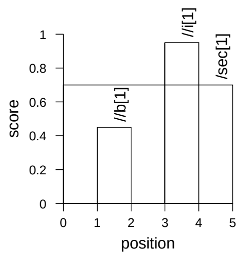
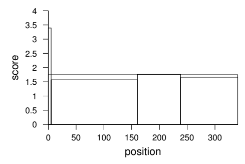
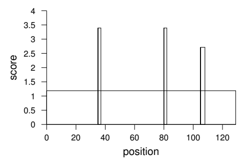


Figure 1: XML text and corresponding context diagram. The horizontal axis denotes the positions and lengths of the text fragments, and the vertical axis shows the RSV (in this case random numbers).



(a) Title pattern: The short peak at the very left is this subsection's title (an st element).



(b) Inline pattern: The peaks are italicized phrases two or three words long; the whole paragraph is 129 words long.

Figure 2: Title and inline pattern examples.

rant closer examination.

3.2.1 Title Hits

Longer texts are usually broken into smaller units, for example, sections, and these units often have a title. The titles contain a very brief summary of the text in the section, so a match in a title should promote the score of the corresponding section. Early web search engines (for HTML) made use of this by increasing the weighting of the terms occurring in title tags; this was later abandoned due to spamming. Spamming is not an issue for retrieval in digital libraries of XML documents—we assume that the documents come from trusted sources, for example, internal documents of an organization—, so in this context, making use of titles is still useful.

In order to make use of titles, we first need to recognize them. One approach would be to use the tag names (probably similar to title), but this would only work for a specific schema, and even if we fix the schema, it may not be totally accurate: For example, we found that authors occasionally use unstructured tags for something that would semantically be considered a title (for example, bold text at the beginning of a paragraph).

Instead, we propose to identify a title by its context; we assume that a title has the following properties (see Figure 2a for an illustration):

- Titles are short.
- Titles occur at the start of the parent element.[1]
- Titles are followed by long text.

It should be noted that the RSVs of the involved hits do not play a role, which implies that the degree to which a node is a title in its enclosing node could be precomputed. (But this has a negative impact on the index size and flexibility.)

[1]This need not be the case, like in HTML, but in that case, there is no section element to be returned, anyway.

3.2.2 Inline Children

Although very short elements, that is, elements that only contain a few words, are not useful retrieval results (Kamps et al., 2005), they can still provide valuable hints about their ancestor elements (although the short elements themselves are never returned). In document-centric XML, elements containing only a few words are likely to represent some form of emphasis (like printing a single word in bold or italic typeface), which in turn implies that the surrounding element is particularly relevant. Thus, a high-scoring match in a short child element should increase the parent's score. Figure 2b shows an example of a situation where the inline pattern matches. Again we do not rely on specific pre-defined tag names in order to be independent of the document's schema.

3.2.3 Good Neighborhood

If an element is surrounded by lower-scoring elements that are relevant to the query (their RSV is greater than zero), this may be an indication that the element is even more relevant than expected by the search engine (this is similar to the parent contextualization described by Arvola et al. (2005)).

4 STRUCTURAL PATTERNS

Now that we have established several structural patterns that occur frequently in retrieval results, we need to find a way to express these findings algorithmically. In the following sections, we will first discuss how patterns operating on a single result context can be implemented and then go on to describe in what way the complete list of preliminary results is processed.

4.1 Implementing Patterns using Fuzzy Logic

The pattern descriptions contain vague terms: What exactly makes an element "short", when is a score "significantly higher" than another score? It is obvious that there can be no clear-cut limits like "elements containing less than five tokens are short", instead, we should have degrees of shortness. This is a perfect fit for *fuzzy logic,* which extends Boolean algebra to cope with fuzzy sets. In the following sections, we will briefly describe the basics of fuzzy logic and then discuss how it can be applied to the problem at hand. The central notion of fuzzy logic is the fuzzy set: In traditional set theory, membership is binary (either a given item is in a set or it is not); with fuzzy sets, membership is given in degrees. Membership values of zero and one correspond to the binary notion of membership, but in addition, any real number between these extremes can be used, depending on the degree of membership.

Fuzzy logic also defines equivalents of Boolean operators like *and*, *or*, and *not*, which enables us to formulate rules like "if the element is very short and its score is higher than its parent's, then lower its score". The output of the condition is the degree to which this rule matches. This value is then used to determine to what degree the specified action should be executed (see below for details). For a more in-depth treatment of fuzzy logic, see Michalewicz and Fogel (2004).

For element e in the results, we specify the functions $\mathrm{len}(e)$, $\mathrm{rsv}(e)$, and $\mathrm{pos}(e)$ denoting length, RSV, and position (see Section 3.1).

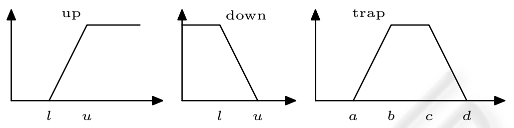
In our context, the following sets of membership functions are useful:

- Length of an element in tokens: (very) short, medium length, (very) long.
- Comparison of a score x in relation to a score y: (much) greater/less than, equal.

The output of each pattern is a proposed change for applicable elements' scores (a factor $>= 0$) along with a degree to which the pattern supports this score change (in the range $[0,1]$).

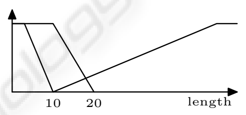
We use the following functions to define the membership functions; l, u, a, b, c, and d denote the bounds as indicated by the graphs:

$$\begin{aligned}
\mathrm{up}_{l,u}(x) &= \begin{cases} 0 & \text{if} \quad x < l \\ \frac{x-l}{u-l} & \text{if} \quad l \le x \le u \\ 1 & \text{if} \quad u < x \end{cases} \\
\mathrm{down}_{l,u}(x) &= 1 - \mathrm{up}_{l,u}(x) \\
\mathrm{trap}_{a,b}^{c,d}(x) &= \min\{\mathrm{up}_{a,b}(x), \mathrm{down}_{c,d}(x)\}
\end{aligned}$$



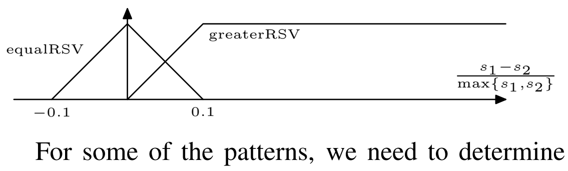
Using these functions, we can specify support membership functions operating on the context parameters. The following functions express properties of the lengths t of context elements:

$$\begin{aligned}
\mathrm{tiny}(t) &= \mathrm{down}_{3,10}(t) \\
\mathrm{short}(t) &= \mathrm{down}_{10,20}(t) \\
\mathrm{long}(t) &= \mathrm{up}_{10,50}(t)
\end{aligned}$$



It is also important to be able to compare the RSVs of to context nodes in order to decide which of two nodes has a higher score. In order to be independent of the magnitude of the RSVs, we divide the difference by the greater value (we assume that the scores are positive).

$$\begin{aligned}
d &= \frac{s_1 - s_2}{\max\{s_1, s_2\}} \\
\mathrm{equalRSV}(s_1, s_2) &= \begin{cases} 1 \quad \mathit{if} \quad s_1 = s_2 = 0 \\ \mathrm{trap}_{-0.1,0}^{0,+0.1}(d) \\ \qquad \mathit{otherwise} \end{cases} \\
\mathrm{greaterRSV}(s_1, s_2) &= \begin{cases} 0 \quad \mathit{if} \quad s_1 = s_2 = 0 \\ \mathrm{up}_{0,0.1}(d) \\ \qquad \mathit{otherwise} \end{cases}
\end{aligned}$$



For some of the patterns, we need to determine whether a count of n elements is "several" elements:

$$\mathrm{several}(n) = \mathrm{up}_{0,5}(n)$$

In addition to the membership functions, which are used to specify the membership value $F(y)$ of the patterns, we need a means of adjusting the score by setting the output value y. For this purpose, we define $\text{degrade}(e)$ and $\text{promote}(e)$ to reduce respectively increase the score of an element e. These functions should set y to values different from 1 ("no change"); we set $y = 0$ for degrade and $y = 2$ for promote.

With these functions, we can now define the conditions and actions for each pattern, where the input is the data we have given (see Section 3.1), and the output are rules to adapt the RSVs of any of the elements in the current context.

4.1.1 Title Hits

The conditions and actions for the title pattern are as follows: Let p be the parent, f the first child (that is, the child with the lowest position), and $\text{isZeroPos}(e) = 1$ if $\text{pos}(e) = 0$ and 0 otherwise.

$$\begin{array}{ll} \textbf{if} & \text{isZeroPos}(\text{pos}(f)) \\ \wedge & \text{short}(\text{len}(f)) \wedge \neg\,\text{short}(\text{len}(p)) \\ \wedge & \text{greaterRSV}(\text{rsv}(f), \text{rsv}(p)) \\ \Rightarrow & \text{promote}(p), \text{degrade}(f) \end{array}$$

4.1.2 Inline Children

The inline pattern does not use the same degree of membership for all nodes; instead, it first determines the degree of "inlineness" for each child separately:

$$\begin{array}{ll} \textbf{if} & \text{tiny}(\text{len}(c)) \\ \wedge & \text{greaterRSV}(\text{rsv}(c), \text{rsv}(p)) \\ \Rightarrow & \text{degrade}(c) \end{array}$$

These degrees are then aggregated to a count of high-scoring inline children. This is implemented in fuzzy logic as the sum of the children's degrees, which ensures that many children that more or less fulfill the requirements can replace a few children that are 100 % members. This fuzzy count n of inline children is then used to determine whether the parent p should be promoted:

$$\begin{array}{ll} \textbf{if} & \text{several}(n) \\ \Rightarrow & \text{promote}(p) \end{array}$$

4.1.3 Good Neighborhood

Given n the count of children in this context, b the element with the highest score among the children, S the set of its siblings, and a the average score of all children, the *good neighborhood* pattern is defined as follows:

$$\begin{array}{ll} \textbf{if} & \text{several}(n) \\ \wedge & \text{greaterRSV}(a, 0.25b) \\ \wedge & \text{greaterRSV}(b, 0.75a) \\ \Rightarrow & \text{promote}(b), \\ & \text{degrade}(s) \quad \forall s \in S \end{array}$$

4.2 Processing the Retrieval Results

The overall retrieval process for our method consists of three basic steps:

1. We perform our searches on an index containing an entry for each element on the original XML documents, which is then searched using traditional information retrieval techniques. Currently, Apache Lucene is used for this step.
2. We apply the structural patterns and perform the RSV updates.
3. We remove overlapping results by repeatedly choosing the result with the highest RSV in a tree and removing all overlapping nodes.
4. We re-rank the results according to the new RSVs and return them to the user.

The core part of our retrieval method occurs in step two. Apart from the result list obtained in step one, which gives us an RSV for each element, we also take the static features length and position into account for each element. Furthermore, we need the list of structural patterns p_i ($i \geq 1$). The algorithm for pattern-based score updates goes as follows:

1. (Build a result tree from the input list.)
   The XPaths of the results are used to reconstruct the nesting of the elements.
2. (Process each context with each pattern.)
   For each element f that is not a leaf node:
   (a) Get the set C of children c from the results.
   (b) Apply each pattern to the context (f, C). The output values y_i and degrees $F(y_i)$ are recorded; the RSV is *not* updated yet.
3. (Compute the final RSVs of the results.)
   For each element e from the results, set

$$\text{rsv}(e) \leftarrow \text{rsv}(e) \frac{\sum F(y_i) y_i}{\sum F(y_i)}.$$

The separation of the pattern applications and the actual RSV updates ensures that the order in which the patterns applied as well as the order in which the result tree is traversed is irrelevant.

A question that arises at this point is whether only the scores in the examined context should be adjusted, or if the adjustments should be propagated. If a paragraph contains many inline elements with matching keywords, its score should be increased. But what about the enclosing elements, like sections or the whole article? Obviously, these elements also contain the inlined keywords as descendants, so their scores should be increased, too, but the pattern only matches for parent and children.

One solution would be to simply apply patterns not only to a node and its children, but to include a node and its descendants on a given level in the result context. This is, however, not a viable solution: It increases the number of contexts that need to be examined, and it also gets harder to understand the meaning of the patterns. It is more feasible to simply propagate the RSV changes up from the parents and down from the children. We currently do not do this (so the changes are only applied to the nodes in the context), because it is unclear whether the propagation is sensible for all patterns (the *good neighborhood* pattern, for example, should be applied to a local context only) and if so, whether the degree should be reduced for propagated scores.

Preliminary tests with applying the promote of the title pattern to all ancestors instead of just the parent appear to indicate that this does not improve retrieval quality, but we will need to investigate this more thoroughly.

4.3 Example

We will illustrate the application of context patterns by looking at a concrete example: We're searching for "bash profile", and among the results of the first pass is the document fragment shown in Figure 3. By definition, each result context spans two levels in the document tree, so this fragment contains three contexts, one rooted at /article, the second one rooted at /article/body, and the third one rooted at /article/body/p[1].

When we apply the inline pattern to the result context /article/body/p[1], we get the degree of membership $F(y) = 1$ for the children emph3[1], emph3[2], and collectionlink[1] because all of them are tiny, and each score is greater than the parent's score. This yields a count of inline children $n = 3$, so the parent's degree of membership is $F(y) = \text{several}(3) = 0.6$.

Similarly, if we apply all patterns to all contexts, we get the result in Table 1.

In order to get the new RSV of each element, we need to apply center-of-area averaging of the factors assigned by the different patterns. For the root element and all leaf elements, we only have one output/degree pair per pattern, the inner nodes may have *two* such pairs because they can occur as either the parent or the child in a result context. The new RSV for the /article/body/p[1] element is calculated as follows:

⟨article⟩

⟨name⟩Iodised *salt*⟨/name⟩$^{l=2}_{s=0.79}$

⟨body⟩

⟨p⟩ ⟨emph3⟩Iodised *salt*⟨/emph3⟩$^{l=2}_{s=0.79}$ (⟨emph3⟩iodized *salt*⟨/emph3⟩$^{l=2}_{s=0.79}$) is table ⟨collectionlink⟩*salt*⟨/collectionlink⟩$^{l=1}_{s=1.26}$ mixed with a minute amount of ⟨collectionlink⟩iodine⟨/collectionlink⟩$^{l=1}_{s=0}$ *salt*s to help reduce the chance of iodine deficiency which can lead to disease of the ⟨collectionlink⟩thyroid⟨/collectionlink⟩$^{l=1}_{s=0}$ gland. Only tiny quantities of iodine are required in the ⟨unknownlink⟩diet⟨/unknownlink⟩$^{l=1}_{s=0}$ to prevent this disease, but there are many places around the world where natural levels of iodine in ⟨collectionlink⟩soil⟨/collectionlink⟩$^{l=1}_{s=0}$ are low and the iodine is not taken up by vegetables. ⟨/p⟩$^{l=71}_{s=0.28}$

⟨p⟩Iodised *salt* is a cheap and effective way of distributing the necessary iodine.⟨/p⟩$^{l=13}_{s=0.32}$

⟨p⟩Iodised *salt* is more common in the United States than Britain, as Britons generally drink iodised milk, while Americans do not.⟨/p⟩$^{l=21}_{s=0.24}$

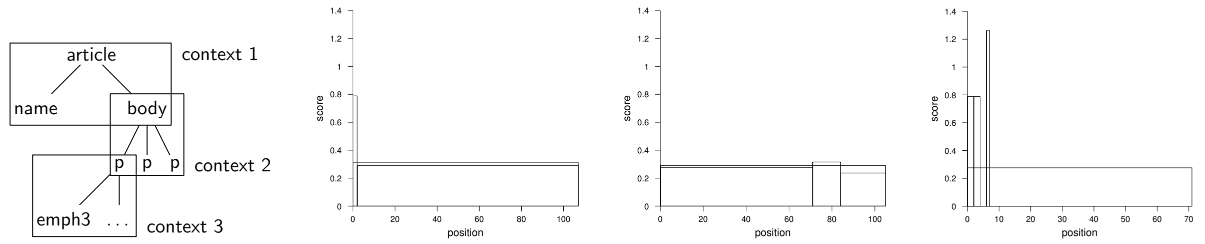
⟨/body⟩$^{l=105}_{s=0.29}$

⟨/article⟩$^{l=107}_{s=0.31}$

Figure 3: Example document fragment from the Wikipedia page on Iodized salt (adapted from the INEX 2006 document collection (Denoyer and Gallinari, 2006)). Appended to each closing tag is the RSV *s* and the length in tokens *l*. The matching keywords are italicized, and the RSV for an element is appended to the closing tag in superscript.

$$\begin{aligned} s_{\text{new}} &= s_{\text{old}} \frac{\sum F(y_i) y_i}{\sum F(y_i)} \\ &= s_{\text{old}} \cdot \frac{0 \cdot 0 + 0 \cdot 0 + 1 \cdot 2 + 0.6 \cdot 2}{0 + 0 + 2 + 0.6} \\ &= s_{\text{old}} \cdot 2 \approx 0.55 \end{aligned}$$

We can see that the RSV of the sect2 element has increased because of the hits in the title; the inline pattern did not have much effect because of the low degree. The net effect, however, is that this element now has a higher score than the descendant programlisting element, which is helpful because it contains more explanatory text.



(a) Overview of the result contexts

(b) Context graph for context 1

(c) Context graph for context 2

(d) Context graph for context 3

Figure 4: Result tree processing for the query "salt".

5 EVALUATION

To determine whether our new method can improve result quality, we participated in the INEX 2005 workshop Dopichaj (2006). The INEX (Initiative for the Evaluation of XML Retrieval) workshop (Fuhr et al., 2005) is a yearly event for evaluating the effectiveness of XML retrieval systems. It is comparable to TREC in traditional information retrieval. The test collection consists of more than 15 000 articles from the IEEE Computer Society's journals and transactions.

5.1 Setup

Every year, the INEX participants submit topics and assess the pooled retrieval results submitted by all participants. In INEX, there are two dimensions to relevance, *specificity* and *exhaustivity*. Specificity denotes what fraction of a retrieval result is relevant to the topic at hand (this is done by highlighting the relevant parts), and exhaustivity measures to what degree the information need is fulfilled (on a scale from 0 meaning "not exhaustive" to 2 for "highly exhaustive").

The evaluation of retrieval quality is performed using the extended cumulative gain (xCG) metric introduced in INEX 2005 (Kazai et al., 2004; Kazai and Lalmas, 2005), an extension of the cumulative gain (CG) metric by Järvelin and Kekäläinen (2002). The basic idea of CG is to obtain a graded relevance assessment for each retrieval result, and then to determine the quality of a given ranked list of retrieval results up to a specific rank by summation of the relevance assessment. For example, if the first three results had the relevance scores 3, 0, and 1, the CG values would be 3, $3+0=3$, and $3+0+1=4$ for the first three cut-off points. The normalized version nCG simply divides the CG values for a given system by the values of an ideal run constructed from the relevance assessments.

The CG metric requires a single relevance value, so xCG introduces quantization functions for calculating a combined value from the exhaustivity and the specificity: A strict version that only accepts the best results ($e = 2$, $s = 1$), and a generalized version that gives partial credit to less than perfect results by multiplying e and s.

Our method does not directly support queries with structural hints, so we only evaluate content-only (CO) topics whose queries consist of queries and phrases. We are interested in finding the best results without overlap, so we only present the outcome for the CO.Focused sub-task.

As a baseline, we used the Lucene retrieval engine[2] in version 1.4.3 with standard stemming and indexed each element as a separate document Eger (2005). Lucene performs ranking like method 2 in Lee et al. (1997), using *tf.idf* and normalization based on document length alone. Our index also contains inline elements only a few words long, which frequently get high RSVs in this baseline implementation, although they are generally of little interest to the user (Kamps et al., 2005). Because of this, both our baseline and our improved runs omit short results (less than 50 words long) from the results.

For the evaluation, we used the INEX 2005 CO topics with the corresponding relevance assessments and the EvalJ evaluation package[3].

5.2 Results and Discussion

For the statistical evaluation, we used the mean average nxCG (MAnxCG) for each topic and run. The MAnxCG value is calculating by averaging the nxCG values at all cut-off points from 1 to 1 500. The following runs are included in the evaluation: the baseline, each pattern applied in isolation, and all patterns applied at the same time.

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

[3]see http://evalj.sourceforge.net/

Table 1: Results of applying the patterns to the result contexts from Figure 4; y stands for the output value (i. e. the value to multiply with) and $F(y)$ for the corresponding degree. Omitted elements have an RSV of 0. The //title/para. element is the only element that occurs in both contexts, so it has entries in both parts of the table. Due to rounding, the result of s_{new} may be slightly different than expected.

		Context 1				Context 2				Context 3				
		Title		Inline		Title		Inline		Title		Inline		
Element	s_{old}	y	$F(y)$	y	$F(y)$	y	$F(y)$	y	$F(y)$	y	$F(y)$	y	$F(y)$	s_{new}
/article	0.31	2	1	2	1									0.63
/article/name	0.79	0	1	0	0.2									0
/article/body	0.29	–	–	0	0	2	0	2	0					0.29
/article/body/p[1]	0.28					0	0	0	0	2	1	2	0.6	0.55
/article/body/p[2]	0.32					–	–	0	0					0.32
/article/body/p[3]	0.24					–	–	0	0					0.24
/article/body/p[1]/emph3[1]	0.79									0	1	0	1	0
/article/body/p[1]/emph3[2]	0.79									–	–	0	1	0
/article/body/p[1]/collectionlink[1]	1.26									–	–	0	1	0

Table 2: MAnxCG values averaged over all topics for the different runs. A * indicates that the Wilcoxon signed rank test gave $p \leq 0.05$ (one-sided).

Run	MAnxCG	Compared to baseline
Baseline	0.2100	
All patterns	0.2289	+9 %
Title and inline	0.2386	+13 %*
Title	0.2356	+12 %*
Inline	0.2378	+13 %*
Neighborhood	0.1948	−7 %

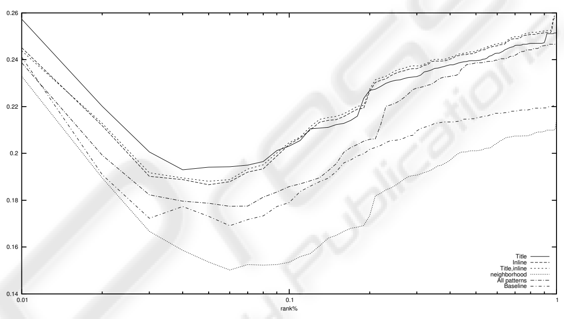


Figure 5: Evaluation (nxCG), focused, quantization generalized. We use a logarithmic scale for the x axis.

For significance testing, we determined the nxCG value for each topic separately and performed statistical tests for correlated values. A Friedman test on the nxCG values indicated that there may be a significant difference in at least one pair of runs. We determined the significance of the baseline versus each of the other runs. Table 2 shows the nxCG value over all topics and an indication whether the difference in the result is statistically significant, applying the paired Wilcoxon test. Note that although the *neighborhood* pattern actually decreases the MAnxCG value, it provides a gain for low cut-off-points.

Figure 5 shows a plot of the nxCG values at all cut-off points. We can see that the title and inline patterns yield a significant improvement over the baseline, both in isolation and combined, whereas the neighborhood pattern actually decreases result quality. For early precision, we see that the title pattern yields the most pronounced improvement compared to the baseline run, but combining it with the inline pattern does not lead to improvements. Several other patterns we tested failed to have a positive impact on the result quality, so we omitted them from the previous description. Overall, applying more than one pattern at the same time does not improve the score as much as we would have hoped; we will need to examine the cause for this.

Comparing the results presented here to our INEX 2005 results (Dopichaj, 2006), note that the implementation used for the runs we submitted to INEX differs from the version described in this paper: The scores are not updated independently, so the patterns applied later receive the scores already updated by earlier patterns, which may lead to unintended interactions.

Our INEX submissions for both the CO.Thorough and the CO.Focused sub-tasks outperform the other organizations' submissions at high precision (cut-off 10 and 25) with generalized quantization (but the results are not statistically significant). The relative performance drops dramatically after 50 results, indicating a problem with recall; as our baseline has the same problems, we assume that this is not caused by the patterns.

6 CONCLUSIONS

We have presented a framework that facilitates the use of overlapping results to our advantage in a post-processing step that can be applied to (almost) arbitrary retrieval results. We do this by way of structural patterns, a generic means of exploiting simple context information. We have seen in our evaluation that several patterns can lead to significant improvements of result quality.

We need to analyze further what the reasons are for the current shortcomings of our approach, in particular, the lack of improvement when applying several patterns simultaneously. We plan to investigate more patterns, and whether propagating the score changes further up or down the result tree can lead to improvements. As far as improvements of the method are concerned, we plan to incorporate additional information such as which query terms were matched by the context nodes.

All the patterns we described in this paper were created from manual inspection of context graphs, but of course it would be useful to have tools that analyze a document collection and some example queries to find candidate patterns.

REFERENCES

Arvola, P., Junkkari, M., and Kekäläinen, J. (2005). Generalized contextualization method for XML information retrieval. In *Proc. CIKM 2005*.

Denoyer, L. and Gallinari, P. (2006). The Wikipedia XML Corpus. *SIGIR Forum*, 40(1).

Dopichaj, P. (2006). The University of Kaiserslautern at INEX 2005. In Fuhr et al. (2006).

Eger, B. (2005). Entwurf und Implementierung einer XML-Volltext-Suchmaschine. Master's thesis, University of Kaiserslautern.

Fuhr, N., Lalmas, M., Malik, S., and Kazai, G., editors (2006). *Proc. INEX 2005*. Springer.

Fuhr, N., Lalmas, M., Malik, S., and Szlávik, Z., editors (2005). *Proc. INEX 2004*. Springer.

Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446.

Kamps, J., de Rijke, M., and Sigurbjörnsson, B. (2005). The importance of length normalization for XML retrieval. *Information Retrieval*, 8(4):631–654.

Kazai, G. and Lalmas, M. (2005). Notes on what to measure in INEX. In *Proc. of the INEX 2005 Workshop on Element Retrieval Methodology*.

Kazai, G., Lalmas, M., and de Vries, A. P. (2004). The overlap problem in content-oriented XML retrieval evaluation. In Sanderson, M., Järvelin, K., Allan, J., and Bruza, P., editors, *Proc. SIGIR 2004*, pages 72–79. ACM.

Kekäläinen, J., Junkkari, M., and Arvola, P. (2005). TRIX 2004 – struggling with the overlap. In Fuhr et al. (2005), pages 127–139.

Lee, D. L., Chuang, H., and Seamons, K. (1997). Document ranking and the vector-space model. *IEEE Software*, 14(2):67–75.

Malik, S., Kazai, G., Lalmas, M., and Fuhr, N. (2006). Overview of INEX 2005. In Fuhr et al. (2006).

Michalewicz, Z. and Fogel, D. B. (2004). *How to Solve It: Modern Heuristics*, chapter 13, pages 367–388. Springer, 2nd edition.

Ogilvie, P. and Callan, J. (2005). Hierarchical language models for XML component retrieval. In Fuhr et al. (2005), pages 224–237.

Ramírez, G., Westerveld, T., and de Vries, A. P. (2006). Using small XML elements to support relevance. In Efthimiadis, E. N., Dumais, S. T., Hawking, D., and Järvelin, K., editors, *Proc. SIGIR 2006*. ACM.

Salton, G., Allan, J., and Buckley, C. (1993). Approaches to passage retrieval in full text information systems. In *Proc. SIGIR 1993*, pages 49–58.