Sound ranking algorithms for XML search in PF/Tijah

Djoerd Hiemstra[1], Stefan Klinger[2], Henning Rode[3], Jan Flokstra[1], and Peter Apers[1]

[1]University of Twente, [2]University of Konstanz, and [3]CWI
hiemstra@cs.utwente.nl, klinger@inf.unikonstanz.de,
henning@cwi.nl, flokstra@cs.utwente.nl, apers@cs.utwente.nl

Abstract

We argue that ranking algorithms for XML should reflect the actual combined content and structure constraints of queries, while at the same time producing equal rankings for queries that are semantically equal. Ranking algorithms that produce different rankings for queries that are semantically equal are easily detected by tests on large databases: We call such algorithms *not sound*. We report the behaviour of different approaches to ranking content-and-structure queries on pairs of queries for which we expect equal ranking results from the query semantics. We show that most of these approaches are not sound. Of the remaining approaches, only 3 adhere to the W3C XQuery Full-Text standard.

1 Introduction

Models for ranked retrieval of XML data should comprise four parts: 1) a model of the text, 2) a model of the structure, 3) a query language, and 4) a ranking algorithm. Ranking is of the utmost importance if an effective XML search system is needed. Some queries might match millions of elements from the text database, but users will only be able to inspect a few. Many of the early structured text retrieval models do not consider ranked retrieval results, or if they do only as an afterthought, i.e., by ranking the retrieval results using a text-only query disregarding the structural conditions in the query [5]. A simple but powerfull way to take the structure of the results into account is to apply a standard information retrieval model to the retrieved content, and then propagate or aggregate the scores based on the structure [6, 12]. In several of these approaches to ranking, the propagation is guided by weighting paths to elements differently by so-called augmentation weights [7, 8], to model for instance that a title element is more likely to contain important information than a bibliography item. Instead of propagating or aggregating the scores from the leaf nodes, *algebraic* approaches include the ranking functionality inside each operator of the query language [2, 15]. Ranking might also include relaxation of the queries' structural conditions, for instance by rewriting complex queries step-wise to simpeler queries [4]. The development of effective ranking algorithms for XML information retrieval is studied in the workshops of the Initiative for the Evaluation of XML retrieval (INEX) [13].

This paper studies mathematical properties of ranking algorithms. While we pursue *effective* algorithms as described above, in this paper we additional pursue *sound* ranking algorithms. Ranking algorithms for stuctured information retrieval are sound if the following two conditions are met:

1. Ranking should reflect the actual, combined content and structure constraints;
2. Two queries that are semantically equal (from a standard –unranked– XPath or XQuery perspective) should produce the same ranked results.

Dagstuhl Seminar Proceedings 08111
Ranked XML Querying
http://drops.dagstuhl.de/opus/volltexte/2008/1533

An example of a system that violates Condition 1 would be a system that first runs the query as a Boolean selection, and then ranks the resulting elements using a standard text retrieval model, i.e., a ranking algorithm that ignores the structure. Suppose we are looking for articles that talk about *ranked xml retrieval* which were supported in one way or another by *John Doe*. This might be formulated as follows using the NEXI query language [17] (Similar examples will be provided for XQuery Full-text [1] below).

```
//article[about(.//p, ranked xml retrieval) and about(.//ack, john doe)]        (1)
```

NEXI stands for Narrowed Extended XPath I, a version of XPath that only supports the descendant and self steps, but that is extended by a special `about()` function. The results of a NEXI query are not in document order, but ranked by estimated relevance to the `about()` parts. If the system first performs a Boolean selection, then it suffers from the well-known disadvantages of Boolean systems: if we interpret the `about()` function as a conjunctive query for which all three words *ranked*, *xml* and *retrieval* should occur in the document, then it is for long queries unlikely that any article matches the query (not because there are no relevant articles, but because they discuss for instance *probabilitic xml retrieval*, or *ranked structured retrieval*, or *ranked xml search*, etc. In that case the result would be empty. If we however interpret the `about()` function as a disjunctive query for which the matching of a single word suffices, then the ranking (i.e., a ranking that ignore the structure) would ignore the paragraphs and acknowledgements. In this case, the top document might very well discuss the holiday diary of John Doe, in which he acknowledges the top ranked XML systems for retrieval (i.e. it might be the paragraph the matches *john doe* and the acknowledgements that match *ranked xml retrieval*.

We believe a true XML retrieval system should meet Condition 1 above. Suppose such as system executes the following query.

```
//article[about(.//p1, xml)]|//article[about(.//p, xml)]        (2)
```

If Condition 1 is met then the system's ranking reflects the actual, combined content and structure constraints, so the ranking will reflect a match in the *p1* elements or a match in the *p* elements. These queries occur a lot in complex documents, such as the IEEE journal data used in the Initiative for the Evaluation of XML Retrieval (INEX) from 2002 to 2005 [13]. In this collection the elements *p1* and *p* both refer to types of paragraphs, as do the elements *p2*, *ip1*, *ip2*, etc. In queries, the user usually does not want to distinguish these different kinds of paragraphs, hence the query above. In fact, such cases were that frequent in INEX that the organisation introduced tag equivalence classes [13], and additional query syntax to ease the formulation of such queries (which is also allowed in XPath 2.0). The following NEXI query is equivalent to the query above:

```
//article[about(.//(p1|p), xml)]        (3)
```

Suppose the system ranks the returned articles for the second query differently than for the first query, resulting in 8 articles in the top 10 that were previously not in the top 10. In that case, the system violates Condition 2: Because the queries are semantically equal, they should result in the same ranking. In order for a ranking algorithm to be *sound* it should meet Condition 1 and Condition 2. We will show in this paper that for systems that meet Condition 1, it is not trivial to meet Condition 2 as well. In fact, we believe it might be impossible to come up with a ranking approach that meets Condition 2 in all cases, especially in the case of XQuery full-text for which there are many ways of formulating the same query.

In this paper, wel will investigate the soundness of ranking algorithms by systematically comparing the retrieval results of ranking algorithms that meet Condition 1 for two queries that are semantically equal. As a starting point of our study, we hypothesise that all ranking algorithms meet Condition 2 as well. Only if we find an example that violates Condition 2 we will drop the hypothesis. We will show that for almost all reasonable ranking algorithms, there are examples of two semantically equal queries and a data set for which the two queries produce a different ranking.

The paper is organised as follows. Section 2 describes the queries used for analysing the soundness of ranking algorithms, and how they are executed. Section 3 presents the test data used. In Section 5 the experimental results are presented. Finally, Section 6 concludes this paper.

2 The test queries

Our analysis of the problem follows that of Mihajlovic [14, Chapter 3], who identifies three requirements for scoring in structured retrieval models. XML ranking algorithms should provide:

Score computation: Given a text query and a set of nodes, compute the score of each node. This is provided by traditional information retrieval models.

Score propagation: This is needed for all XPath axis steps. To do an axis step from a node for which a score was computed, the scores need to propagate to the result nodes. For some axis steps, for instance the parent step, the score of several children needs to propagate to a single result node.

Score combination: Score combination is needed if the same set of nodes is scored multiple times and the final score should reflect the scores of the nodes in both sets.

As an example, consider the following XQuery Full-Text query, that ranks articles about *XML* that thank *John Doe*, similar to the query in Example 1 above:

```
for $d score $s in doc("test.xml")//ack
where ../article ftcontains "xml" and . ftcontains ("John", "Doe")        (4)
return $d order by $s desc
```

Here, score *computation* is needed for the `article` elements (scored by the similarity to *xml*) and for the `ack` elements (scored by the the similarity to *John Doe*). The query should rank `ack` elements, so the scores of the `article` elements needs to be *propagated* to their child `ack` elements. Finally, the two scores for each `ack` element need to be *combined* in a final score.

If the user wants to ranks articles about *XML* that thank *John Doe*, he/she might as well pose the following query.

```
for $d score $s in doc("test.xml")//article[. ftcontains "xml"]/ack
where . ftcontains ("John", "Doe")                                        (5)
return $d order by $s desc
```

In fact, several queries are possible that are semantically equal as meant by Condition 2 above. We define equality as follows: The XPath representation of a NEXI query is defined as the query produced by replacing every NEXI function `about(n, s)` with `fn:contains(fn:string(n), "s")`. Two NEXI queries are semantically equal if and only if their XPath representations are equal. Similarly, the XQuery representation of an XQuery Full-Text query is defined as the query produced by replacing every XQuery Full-Text function `n ftcontains "s"` by `fn:contains(fn:string(n), "s")`, and two Full-Text queries are semantically equal if their XQuery representations are equal. However, because of the properties of score computation, propagation, and combination, two semantically equal queries might produce different rankings, and might therefore return different (top) elements to the user, i.e., the ranking algorithm is not *sound.*

Case 1: The semantics of score computation

In our first case we look at the semantics of *score computation.* Score computation is the most important of Mihajlovic's requirements. Here, we only consider simple scoring, i.e., scoring of queries without using proximity or phrases (in NEXI, phrases can be marked as with double quotes), i.e., the following query.

```
//article[. ftcontains ("xml", "ir", "db")]        (6)
```

In XQuery Full-text, this query retrieves articles that match any of the terms, i.e., the standard behaviour is that of a Boolean OR query. Given these semantics, we expect scoring to be compositional, that is, if we select article containing *"xml"* and union those with articles containing *"ir", "db"* as shown in the following query, then we expect the same results as the query above.

```
//article[. ftcontains "xml"]|//article[. ftcontains ("ir", "db")]          (7)
```

Several alternative formulations are possible in XQuery Full-text, for instance the query `//article[. ftcontains "xml" ftor ("ir", "db")]` or `//article[. ftcontains "xml" or . ftcontains("ir", "db")]`. The alternative formulations select the exact same articles, and if simple scoring behaves as a Boolean OR query, then we expect a sound ranking approach to produce the same rankings for all these formulations.

However, from the user's point of view, we might argue that scoring should have the semantics of the Boolean AND: the best documents are the ones containing *all* three query terms, not just many occurrences of either query term. If simple scoring behaves as a Boolean AND query, then we expect a sound ranking of Query 6 approach to produce the same rankings as the following query:

```
//article[. ftcontains "xml"][. ftcontains("ir","db")]          (8)
```

or alternative formulations: `//article[. ftcontains "xml" ftand ("ir", "db")]`, or `//article[. ftcontains "xml" and . ftcontains ("ir", "db")]`. These queries (and similar NEXI queries) correspond to different query plans in the PF/Tijah XML search system [10]. These plans use so-called *score region algebra* to process these queries. Figure 1 contains the query trees for the three plans. Instead of putting the region algebra operators in the trees (the exact definition of algebraic operators in outside the scope of this paper), the figure contains the partial queries that represent the intermediate results at that stage of the query plan.

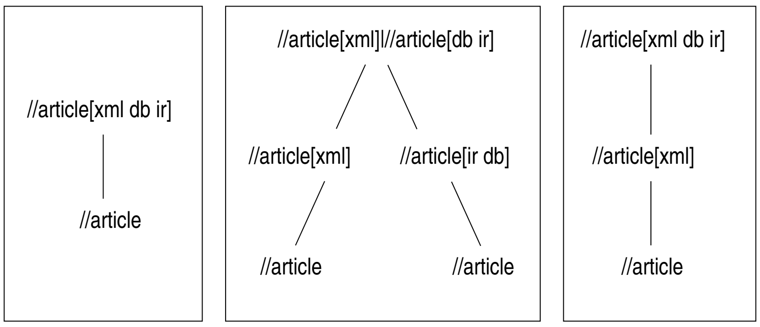

Figure 1: Query plans for Case 1

Following the line of reasoning of sound ranking algorithms presented above, the simple scoring plan shown in Figure 1a should either produce the same ranking as the disjunctive plan shown in Figure 1b, or it shouls produce the same ranking as the conjunctive plan shown in Figure 1c. If Plan 1a produces a result different from both Plan 1b and 1c, then the score computation is not sound: Users of retrieval systems should be able to understand the difference between OR-queries and AND-queries, however, it is hard to anticipate on semantics that is different from these two.

Case 2: Score propagation – downwards

In the second case we look at *downwards score propagation*: Suppose the user is interested in sections about *"databases"* from articles about *"xml"*. In this case, the scores of the article elements have to be propagated to the section elements. Such a query can be processed in two ways. Either first score all articles, propagate the scores to the contained sections, and score those, as shown in the following query:

```
//article[. ftcontains "xml"]//section[. ftcontains "db"]          (9)
```

... or, first score all sections, and then score the articles that contain these sections, as follows:

```
//section[. ftcontains "db"][./ancestor::article ftcontains "db"]          (10)
```

The query trees of the actual query plans are shown in Figure 2.

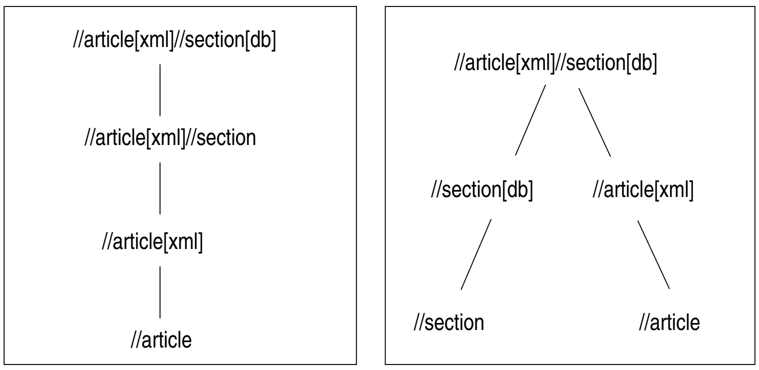

Figure 2: Query plans for Case 2

Following the line of reasoning of sound ranking algorithms presented above, there is no reason why both queries and both query plans above should not produce the exact same ranking of section elements.

Case 3: Score propagation – upwards

In the second case we look at *upwards score propagation*: Suppose the user that was interested in articles about *"xml"* with sections about *"databases"* now wants to retrieve the articles. In this case, the scores of the section elements have to be propagated upwards to the article elements. Again, the query can be processed in two ways. Either first score all articles, and then propagate the scores to the contained sections upwards as shown in the following query:

```
//article[. ftcontains "xml"][.//section ftcontains "db"]          (11)
```

... or, first score all sections and propagate the score to articles that contain these sections, as follows:

```
//section[. ftcontains "db"]/ancestor::article[. ftcontains "xml"]          (12)
```

The query trees of the actual query plans are shown in Figure 3. Again, a sound ranking approach would produce the exact same ranking for both queries.

Case 4: Score Combination – union

Case 4 looks at *score combination*, more specifically at score combination when the union of two node sets is taken. Suppose the user wants articles that mention *"xml"* in a section, *or* that mention *"db"* in the title:

```
//article[.//section ftcontains "xml" or .//title ftcontains "db"]          (13)
```

As discussed above, both XQuery/XPath Full-Text and NEXI support a union operator "|" that might be used as well. For instance, an alternative formulation of the query above would union

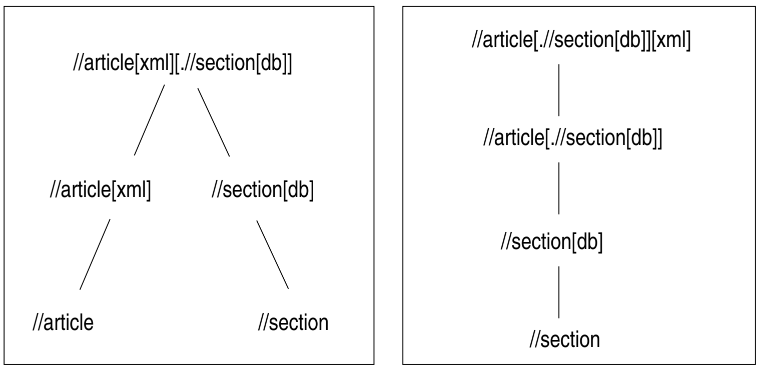
//article[xml][.//section[db]]
//article[xml]
//section[db]
//article
//section
//article[.//section[db]][xml]
//article[.//section[db]]
//section[db]
//section

Figure 3: Query plans for Case 3

two sets of article nodes, one of which the sections contain *"xml"* and another set of article nodes which titles contain *"db"*.

```
//article[.//section ftcontains "xml"]|//article[.//title ftcontains "db"]
```
(14)

For article nodes that are in both sets, the union operator should somehow combine the scores. The query trees of the actual query plans are shown in Figure 4. A sound ranking approach produces the exact same ranking for both queries.

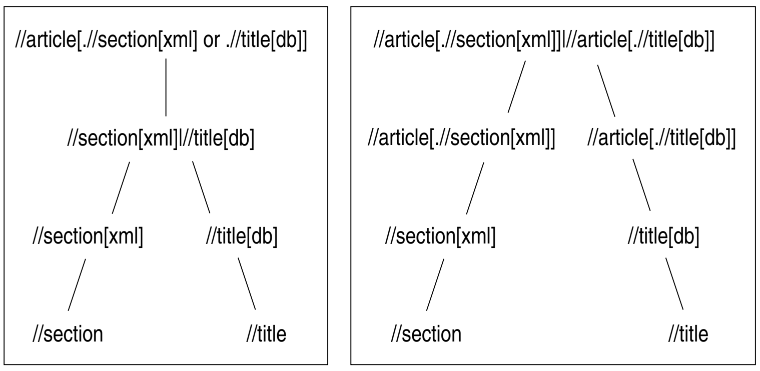
//article[.//section[xml] or .//title[db]]
//section[xml]|//title[db]
//section[xml]
//title[db]
//section
//title
//article[.//section[xml]]|//article[.//title[db]]
//article[.//section[xml]]
//article[.//title[db]]
//section[xml]
//title[db]
//section
//title

Figure 4: Query plans for Case 4

Mihajlovic' score region algebra [14] supports an intersection operator similar to the union operator above. Such an operator is not supported by the XQuery/XPath Full-Text and NEXI, since it is unnecessary in practice. Therefore, we will not consider score combination in the case of intersecting two node sets.

Case 5: XQuery Full-text scoring properties

The XQuery Full-text standard imposes very few restrictions on scoring: The numeric score computed by queries is *implementation-dependent*, i.e, scoring may differ between implementations; scoring is not specified by the W3C specification, and scoring is not required to be specified by the implementor for any particular implementation. The standard however imposes the following two restrictions on full-text contains expressions [1]:

A full-text contains expression returns a Boolean value.

So, a full-text contains expression always distinguishes the matching nodes from the non-matching nodes. In Mihajlovic's score region algebra [14], operators compute scores for *all* nodes, that is, all nodes always match the expression. One might argue that this follows the XQuery Full-text standard (each full-text expression always returns *true*), but at least it is not in the spirit of XQuery Full-text. We will call the semantics of the score region algebra operators *ranking semantics* and the semantics suggested by XQuery Full-text *matching semantics.* In practice, matching semantics is often required in practical systems. The current PF/Tijah implementation has default matching semantics. We investigate both matching semantics and ranking semantics to see which of the two is more likely to produce sound ranking.

Score values are of type xs:double in the range [0, 1].
This restriction is not imposed by score region algebra [14]. In fact, many well-performing ranking functions – for instance Okapi's BM25 [16] – produce scores greater than 1 in some cases, and even if they do not, the approach might produce scores greater than 1 after score propagation and score combination.

Case 5 does not define an extra set of queries. The soundness of rankings produced by the queries in Case 1 to 4 above are checked when using matching semantics and ranking semantics. Furthermore, we check if the scores of the results of the queries in Case 1 to 4 are in the range [0, 1].

3 The test data

We will evaluate the test cases on artificial data to ensure that we control the circumstances in which queries fail. Some ranking algorithms might be sound in most cases, for instance, they might be sound, unless the elements that are ranked are nested. In the examples above, we would expect every article to have only one acknowledgements section; which might actually be defined in a DTD or XML schema. The schema might also contain elements that have a many-to-one relation to `article` elements, such as `paragraph` elements, or elements that might be nested inside themselves, such as `section` elements (i.e., sections, subsection, and subsubsections might all be ambiguously referred to as `section`. When element scores are propagated through the document structure, they might have to be aggregated in case of many-to-one relations, or divided in case of one-to-many relations, or both aggregated and divided in case of nested relations. The following DTD contains several of such cases:

```
<!ELEMENT root (article | report)* >
<!ELEMENT article (title, section+) >
<!ELEMENT report (title, section+) >
<!ELEMENT section (heading, (section+ | paragraph+)) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT heading (#PCDATA) >
<!ELEMENT paragraph (#PCDATA | list)* >
<!ELEMENT list (item | list)* >
<!ELEMENT item (#PCDATA) >
```

We distinguish the following 5 cases in which elements from two node sets can be nested: 1:1, 1:n, 1:n where the elements of the second node set are nested, n:m where the elements of first node set are nested, and n:m where the elements of both sets are nested.

article vs. title	1 : 1
article vs. paragraph	1 : n
article vs. section	1 : n nested
section vs. paragraph	n nested : m
section vs. list	n nested : m nested

The queries presented in the cases above are all examples of the "1:n nested" type, i.e., the queries refer to `article` elements and `section` elements. Each query is run as in one of the above five types; so, `article` and `section` are replaced by: 1) `article` elements and `title`, 2) `article` elements and `paragraph`, 3) `article` elements and `section`, 4) `section` elements and `paragraph`, 5) `section` elements and `list`.

4 The ranking approaches

We test a total number of 200 ranking approaches for XML search. These approaches are for an important part the same as the approaches Mihajlovic [14] evaluated in oder to find effective ranking approach for XML search. So, our ranking approaches are motivated mostly by testing those approaches for which we expect good recall and precision values in standard information retrieval evaluations, such as those provided by INEX. The number of ranking approaches that can be defined for XML search is endless however, and we do not attempt in anyway to test a complete (sub-)set of all possible ranking functions.

The choices of our ranking approaches are restricted by PF/Tijah's algebraic approach to XML search. Each operators in score region algebra follows the same pattern: It operates on a context node set (context region set), and a target node sets (target region set). Both the context nodes and the target nodes might have scores already from previous operations. Each operator has to combine the score of target node with the scores of (possibly) multiple matching context nodes. For each target node, we distinguish three situations: 1) the target node does not match any context node: In this case the target node is not returned (matching semantics) or it is returned with a default score (ranking semantics); 2) the target node matches one and only one context node: In this case, the target node is returned with the *combined score* of the target node and the matching context node; 3) the target node matches more than one context node: In this case, scores of the matching context nodes are first *aggregated*, and then *combined* with the score of the target node. *Aggregation* and *combination* define two of the dimensions along which we define ranking approaches, the other dimensions are the *retrieval model*, and *ranking/matching semantics*:

Score combination

We test 5 different ways to combine the scores of the context node and the target node: adding, multiplying, maximum, minimum, and average.

Score aggregation

We test the same 5 different ways to aggregate the scores of the matching context nodes: sum, product, maximum, minimum, and average. This brings the total number of approaches to 25.

The retrieval model

We test four different retrieval models, bringing the total number of approaches to 100: LMS, a standard language model using linear interpolation smoothing [9]; LM, a standard language model without smoothing; NLLR: normalized log-likelihood ratio (a simple derivation of LMS that produces log-linear scores) [11]; BM25: Okapi's BM25 ranking formula [16].

Ranking semantics vs. matching semantics

We test each approach with ranking semantics (each operator returns all nodes with a score), and matching semantics (each operator returns a selection of the nodes), so 200 approaches in total over all four dimensions.

5 Investigating the soundness in practice

Using the DTD described in Section 3, a 100kB artificial test collection with articles and reports was generated. The text nodes were generated from a simple language model of three words (*"ir", "db", and "xml")*, where each word is generated by some probability. This way, almost every element will match a query to some extend, with scores similar to other elements, possibly resulting in different rankings. Each of the 200 ranking approaches from Section 4 was tested on a pair of queries from one of the four cases from Section 2, where each query followed one of the five ways in which data can be nested described in Section 3, defining in total 5000 queries. We summarize the results by reporting the most important lessons learned.

Lessons for Case 1: Score computation

In all cases that used the NLLR retrieval model, we detected unsound ranking behaviour on the test data. The NLLR retrieval model differs from the LMS retrieval model mainly because it uses query length normalization. Apparently, retrieval models that use some form of query length normalization are not sound. Other examples of retrieval models that uses query length normalization are vector space models that use the cosine similarity.

For almost all ranking approaches that use *matching semantics*, we detected unsound ranking behaviour when comparing Plan 1a to 1c. Apparently, matching semantics excludes the possibility to follow the semantics of AND-queries, which seems logical because most models retrieve elements even if they do not contain all query terms. An exception is the LM retrieval model, i.e., the language model without smoothing: this is the only model that does not produce unsound rankings behaviour when comparing Plan 1a to 1c.

Unsound ranking behaviour was occassionally detected when comparing Plan 1a to 1b. The ranking approaches tested are more likely to follow the semantics of OR-queries.

Lessons for Case 2: Score propagation – down

In most cases that used the BM25 retrieval model, we detected unsound ranking behaviour, except when score combination uses the product or the maximum. We believe the unsound behaviour can be explained by the fact that the BM25 retrieval model uses the number of documents as one of its parameters. This was implemented in the system as the size of the target node set (i.e., the size of the set that needs to be ranked). We believe implementing BM25's N (the number of documents) by taking the size of the set to be ranked is the only sensible thing to do. We cannot take a predifined N, because the set might be the result of a complex selection query, possible combining for instance article elements and report elements and then restricting them on some other criterion.

The size of the sets, however, differ depending on the query plan used. This is even the case in more simple queries such as `//article//section[. ftcontains "xml"]`: If the system first selects the sections that are contained by articles (excluding the sections contained by reports in our test data) then the size of the set to be ranked is obviously smaller than the size of the complete set of sections as represented by the query `//section[. ftcontains "xml"][./ancestor::article]` Interestingly, all *tf.idf* term weighting algorithms use the number of documents to be ranked in their definition. The results indicate that all such approaches would produce unsound rankings.

We did not detect unsound ranking for BM25 if score combination uses the product or the maximum. If the maximum is used for score combination, then the approach would often ignore the BM25 score, so this approach is useless in practice. We are unable to explain the behaviour when score combination uses the product: It might be an artefact of the data. This needs to be analysed in the future.

Lessons for Case 3: Score propagation – up

In almost all cases that used the BM25 retrieval model, we detected unsound ranking behaviour of the queries on the test data, except when score combination uses the maximum. As above, we have strong indication that this is due to the use of the size of the set that needs to be ranked in the definition of the model, which differs depending on the query plan used.

Lessons for Case 4: Score Combination – union

Unsound ranking was detected if score aggregation uses the average score of all matching context nodes. This might be due to the fact that taking the average function is not associative, and produces different values depending on the order in which it is evaluated.

Lessons for Case 5: XQuery FT properties

By design, the ranking approaches using ranking semantics (half of the approaches) do not adhere to the XQuery Full-text standard, or at least they are not in the spirit of the standard. A bigger problem might be restriction that scores should be between 0 and 1. The retrieval models NLLR and BM25 produced scores greater than 1 in all cases, the score aggregation that uses the sum of scores also produced scores greater than 1 in most cases. The score combination that uses the sum of scores produced scores greater than 1 in some cases.

Overall lessons learned

If we only consider ranking approaches that: 1) did not produce unsound rankings; 2) did never produce scores greater than 1; and 3) use matching semantics, then only 3 approaches remain: These three approaches use the language model without smoothing (LM), multiply for score combination and either product, minimum or maximum for score aggregation. Whereas approaches based on language models without smoothing *might* be sound, it is likely that the search quality of the systems is below average: It is well-known, that smoothing is important for getting high quality retrieval results.

If we drop the requirement that scores should never by greater than 1, then 4 ranking approaches remain. Again, all of them use the LM retrieval model.

If we however drop the requirement that scoring should uses matching semantics, then 13 ranking approaches remain, among which several approaches that use the language model with smoothing (LMS).

6 Conclusions

We report the behaviour of 200 ranking approaches to ranking content-and-structure queries on pairs of queries for which we expect equal ranking results from the query semantics. We show that most of these approaches are not sound, i.e., they fail to produce equal rankings in the cases studied. Of the remaining approaches, only 3 adhere to the W3C XQuery Full-Text standard, which requires so-called matching semantics, and which requires retrieval scores to be smaller or equal than 1 at all times. The difficulties in implementing effective and sound ranking for XQuery Full-Text might affect its acceptance as a standard in the future.

Is ranking really necessary?

The XQuery Full-Text standard was largely motivated by complex retrieval queries. The XQuery Full-Text Use Cases [3] discuss for instance querying accross element boundaries, wild cards, stop words, stemming, sensitivity to diacritics, cardinalities, existential quantification, proximity, implicit sentences and paragraphs, the use of thesauri, etc. Only a small part of the Use Cases consider scoring. So, maybe scoring is not really necessary in XML search?

Table 1 presents the average precision at 10 elements retrieved over 114 NEXI queries provided by the INEX 2006 evaluation. If we treat each `about()`-clause in NEXI as a Boolean OR (this is the default behaviour of XQuery Full-text), then 97 out of 114 queries do not find any relevant element in their top 10, the average precision at 10 being 0.053. The best language model and

Approach	P@10	queries failed?
Boolean OR	0.053	97
Boolean AND	0.200	59
LMS/MULT/MAX/Matching	0.361	22
BM25/ADD/MAX/Matching	0.379	18
LMS/MULT/MAX/Matching/prior	0.439	15

Table 1: Retrieval quality on 114 INEX 2006 content-and-structure queries

BM25 approaches tested in this paper reduce the number of failed queries to respectively 22 and 18 (about five times less errors) and respectively 0.361 and 0.379 average precision at 10 (about 500% performance increase). If we add an element length prior to the language modelling approach (longer elements are more likely to be relevant; this was not tested for soundness in this paper), then the number of failed queries is down to 15 and the average precision is up to 0.439 (more than 700% improvement in performance). Clearly, a system without ranking is useless compared to the system that includes high quality ranking algorithms.

References

[1] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, M. Holstege, J. Melton, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 full-text 1.0. Technical report, World Wide Web Consortium, May 2008. http://www.w3.org/TR/xpath-full-text-10/.

[2] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. Texquery: a full-text search extension to XQuery. In *Proceedings of the 13th international World Wide Web conference*, 2004.

[3] S. Amer-Yahia and P. Case. XQuery and XPath Full Text 1.0 use cases. Technical report, World Wide Web Consortium, May 2008. http://www.w3.org/TR/xpath-full-text-10-use-cases/.

[4] S. Amer-Yahia, L.V.S. Lakshmanan, and S. Pandit. Flexpath: flexible structure and full-text querying for XML. In *Proceedings of the 2004 ACM SIGMOD international conference*, 2004.

[5] F.J. Burkowski. Retrieval activities in a database consisting of heterogeneous collections of structured text. In *Proceedings of the 15th ACM Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 112–124, 1992.

[6] D. Carmel, Y.S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML documents via XML fragments. In *Proceedings of the 26th ACM Conference on Research and Development in Information Retrieval (SIGIR'03)*, pages 151 – 158, 2003.

[7] N. Fuhr and K. Grossjohann. XIRQL: A query language for information retrieval in XML. In *Proceedings of the 24th ACM Conference on Research and Development in Information Retrieval (SIGIR'01)*, pages 172–180, 2001.

[8] T. Grabs. Generating vector spaces on-the-fly for flexible XML retrieval. In *Proceedings of the SIGIR workshop on XML and Information Retrieval*, pages 4–13, 2002.

[9] D. Hiemstra and W. Kraaij. Twenty-One at TREC-7: Ad-hoc and cross-language track. In *Proceedings of the seventh Text Retrieval Conference TREC-7*, pages 227–238. NIST Special Publication 500-242, 1998.

[10] D. Hiemstra, H. Rode, R .van Os, and J. Flokstra. PF/Tijah: text search in an XML database system. In *Proceedings of the 2nd International Workshop on Open Source Information Retrieval*, 2006.

[11] W. Kraaij. *Variations on Language Modeling for Information Retrieval.* PhD thesis, University of Twente, 2004.

[12] M. Lalmas. Dempster-shafer's theory of evidence applied to structured documents: modelling uncertainty. In *Proceedings of the 20th ACM Conference on Research and Development in Information Retrieval (SIGIR'97)*, pages 110 – 118, 1997.

[13] M. Lalmas and A. Tombros. Evaluating XML retrieval effectiveness at INEX. *SIGIR Forum*, 41(1):40–57, 2007.

[14] V. Mihajlovic. *Score Region Algebra: A flexible framework for structured information retrieval.* PhD thesis, University of Twente, 2006.

[15] V. Mihajlovic, H.E. Blok, D. Hiemstra, and P.M.G. Apers. Score region algebra: Building a transparent XML-IR database. In *Proceedings of the 14th International Conference on Information and Knowledge Management (CIKM)*, pages 12–19, 2005.

[16] S.E. Robertson, S. Walker, M.M. Beaulieu, M. Gatford, and A. Payne. Okapi at TREC-4. In *Proceedings of the 4th Text Retrieval Conference (TREC-4)*, pages 73–97. NIST Special Publication 500-236, 1995.

[17] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, pages 16–40, 2004.