Efficient XML and Entity Retrieval with PF/Tijah: CWI and University of Twente at INEX'08

Henning Rode[1], Djoerd Hiemstra[2], Arjen de Vries[1], Pavel Serdyukov[2]

[1]CWI Amsterdam, The Netherlands
[2]CTIT, University of Twente, The Netherlands

Abstract.

1 Introduction

PF/Tijah is a research prototype created by the University of Twente and CWI Amsterdam with the goal to create a flexible environment for setting up search systems. By integrating the PathFinder (PF) XQuery system [1] with the Tijah XML information retrieval system [2] it combines database and information retrieval technology. The PF/Tijah system is part of the open source release of MonetDB/XQuery developed in cooperation with CWI Amsterdam and the University of Tübingen.

PF/Tijah is first of all a system for *structured retrieval* on XML data. Compared to other open source retrieval systems it comes with a number or unique features [3]:

- It can execute any NEXI query without limits to a predefined set of tags. Using the same index, it can easily produce a "focused", "thorough", or "article" ranking, depending only on the specified query and retrieval options.
- The applied retrieval model, score propagation and combination operators are set at query time, which makes PF/Tijah an ideal experimental platform.
- PF/Tijah embeds NEXI queries as functions in the XQuery language. This way the system supports ad hoc result presentation by means of its query language. The efficiency task submission described in the following section demonstrates this feature. The declared function `INEXPath` for instance computes a string that matches the desired INEX submission format.
- PF/Tijah supports text search combined with traditional database querying, including for instance joins on values. The entity ranking experiments described in this article intensively exploit this feature.

With this year's INEX experiments, we try to demonstrate the mentioned features of the system. All experiments were carried out with the least possible pre- and post-processing outside PF/Tijah. Section 2 shows with the application of the system to the INEX efficiency track, how a wide range of different NEXI queries can be executed efficiently. Section 3 demonstrates how combined database and retrieval queries provide a direct solution to specialized tasks like entity ranking.

2 Efficiency

The INEX efficiency task combines retrieval quality and performance. In order to test the performance on a wide range of different queries, the task uses a query set of 568 structured queries combined from other tasks and collected over the last years. The queries vary with respect to the contained number of query terms and structural requirements. A subset for instance represents typical relevance feedback queries containing a considerable higher number of query terms.

The retrieval efficiency of PF/Tijah was improved in the last year with respect to several aspects, which we wanted to test by our submissions. The index structure, containment joins, and score computation had been changed [4] to improve the execution of simple query patterns such as

```
//tag[about(., term query)]
```

PF/Tijah creates a full-text index on top of Pathfinder's pre/post encoding of XML files [5]. Instead of assigning a pre-order value to complete text-nodes as done by the Pathfinder, the Tijah full-text index enumerates each single term. Both the Pathfinder encoding and the separate full-text index are held in database tables. An "inverted" table is created by clustering the table (pre-order values) on tag- and term ID.

PF/Tijah does not use top-k query processing strategies. Neither tag-term pairs nor scores are precalculated or indexed in order avoid redundancy on the one hand, and to allow at query time the application of arbitrary ranking functions on the other hand. The applied ranking function is specified in PF/Tijah for each single query. Furthermore, PF/Tijah's containment join operator relies on input sorted in document order. Node sequences sorted on score order as they are typically accessed in the top-k query processing framework do not match this requirement. PF/Tijah does not implement any caching strategy itself. However, the underlying database system tries to make use of the operating system's caching functionalities.

2.1 Submissions

We submitted in total 4 runs, 1 "article" ranking and 3 "thorough" element rankings. Since PF/Tijah does not support top-k query processing, all submitted runs return the complete set of the 1500 highest ranked elements for each query. The applied ranking function for all submissions follows the language modeling framework for retrieval. The so-called NLLR, normalized logarithmic likelihood ratio, compares for each query term its distribution within element and query model. The ranking aggregates single term scores on the level of scored elements. Query terms marked by a leading '-' to indicate that they should *not* occur in relevant elements were removed from the queries, since PF/Tijah currently does not support this feature. For the same reason, phrases were treated as normal query terms only.

For repeatability we report here the complete XQuery that was used to produce the ranking in PF/Tijah. The XQuery below was generated for Topic 856.

The individual queries only substitute the inside NEXI string accordingly. The costly function call producing the required INEX path string was omitted when running time measurements, since it does not reflect the retrieval performance itself:

```
declare function INEXPath($n as node()) as xs:string
{
   let $paths :=
       for $a in $n/ancestor-or-self::*
       where local-name($a) ne "docs"
       return if (local-name($a) eq "article")
          then concat(local-name($a),"[1]")
          else concat(local-name($a),"[",
               string(1 + count($a/preceding-sibling::*
               [local-name() eq local-name($a)])),"]")
   return string-join($paths, "/")
};

let $opt := <TijahOptions returnNumber="1500" ir-model="NLLR"
             prior="NO_PRIOR" txtmodel_returnall="FALSE"/>
let $nexi := "//article//body[about(.//section//p, State Park) and
             about(.//section//title, Geology) and
             about(.//section//title, Geography)]
             //figure[about(.//caption, Canyon)]"
return <topic id="856"> {
   for $res at $rank in tijah:queryall($nexi, $opt)
   return <result><file> {
      concat("",$res/ancestor-or-self::article/name/@id)}</file>
      <path>{INEXPath($res)}</path>
      <rank>{$rank}</rank></result> }
</topic>
```

For the *article* ranking we automatically created NEXI queries by substitution of the placeholder ?CO-TITLE? below with the content-only (CO) field of the query topic:

```
//article[about(., ?CO-TITLE?)]
```

The run should show how our XML retrieval system performs when used as a standard document retrieval system.

In contrast, the “thorough” element rankings use the content-and-structure (CAS) field of each query topic. The first “thorough” run, called *star-strict*, executes the unmodified CAS query as provided in the query topic. The final two runs perform a slight modification. Since the new PF/Tijah system is tuned towards queries starting with a tag-name selection rather than searching in all element nodes, we translated queries starting with the pattern

```
//*[about(., terms)]...
```

to

```
//(article|section|p)[about., terms)]...
```

The runs based on this modification are called *asp-strict* and *asp-vague*. The distinction between both is explained in the following.

Thinking in terms of XPath, the base of the NEXI language, the scoring predicates `[about(., terms)]` are first of all evaluated to a boolean value, causing those elements to pass that satisfy the query. If the predicates are translated to a scoring operator in the algebra tree, that only assigns scores to all elements, the predicate becomes obsolete as a filter and the side effect of the predicate evaluation, the score assignment, has become the primary aim. This is clearly not the only possible query interpretation. We can require that an element has to reach a certain score threshold in order to satisfy the predicate condition. The least strict setting of such a threshold would be to filter out all zero scored element. In other words, the `about` function would assign a `true` value to all elements that contain at least one of the query terms. For a query of the form

```
//article[about(., xml)]//p[about(.,ir)]
```

strict semantics will pass only those articles that match the keywords of the first about, whereas *vague* semantics also considers results of paragraphs about "ir" that are not occurring within articles about "xml". The two submitted runs, *asp-strict* and *asp-vague*, compare the different query interpretation with respect to retrieval quality and performance.

2.2 Results

Test System setup The test system used for all time measurements in this article was an INTEL Core2 Quad machine running on 2.4 Ghz with 8 GB main memory. The necessary index structures could hence be held in memory, but not in the considerably smaller CPU caches. Queries were executed sequentially. For time measurements, we omitted the generation of the INEXPath as mentioned above and stored only node identifiers instead. We measured the full execution of the query, including the query compilation phase.

run	avg time	sum time	min time	max time
article	0.702	399	0.327	11.814
star-strict	17.186	9762	0.324	330.495
asp-strict	2.306	1310	0.324	52.388
asp-vague	8.213	4665	0.444	1235.572

Table 1. Execution time overview in sec

Table 1 shows an overview on the execution times of the different runs. The article ranking is obviously faster on average than the three other runs evaluating the CAS query. Since some CAS queries in the query set issue a simple fielded search, it is not surprising that the minimal execution time stays almost the same for all runs. Looking at the average and maximal execution time for a single query, we observe, however, huge differences. Most of the time differences can be attributed to queries that contain the pattern `//*` in the NEXI path. If a posting list of a certain tagname is fetched from the inverted index, the system guarantees the pre-order sortedness of the list, which is required for the subsequent containment evaluation. Fetching the entire inverted index, however, will not return a pre-order sorted element list, and therefore requires a resorting of the entire node set. The difference becomes apparent when comparing the execution times of the two runs *star-strict* and *asp-strict*. Even the expensive substitute pattern selecting all `article`, `section`, and `p` nodes shows still a better performance.

Evidently, the application of *strict* query semantics yield a better query performance. The average total execution is around 4 times faster than in the case of a *vague* interpretation. The early filtering on intermediary result sets especially helps on highly structured queries. Consequently, we observe similar minimal execution times but clear differences when looking at the highest times measured for evaluating a single query. The differences of the two query interpretations needs to be studied as well in terms of retrieval quality. Unfortunately, the results here were not available at the time writing this article.

3 Entity Ranking

The INEX entity ranking task searches for entities rather than articles or elements with respect to a given topic. With entities we mean here unique instances of a given type, such as “Hamburg” and “München” being an instance of type “German cities”. For a given query topic such as “hanseatic league” and target entity type “German cities” a good entity retrieval system should return “Hamburg”, but not “München” since it is off topic, or “Novgorod” since it is not a German city.

The target type is given as a Wikipedia category in the INEX task. Furthermore, each retrieved entity needs to have its own article in the Wikipedia collection. Obviously, this decision is only suitable for entity ranking within an encyclopedia, where we can assume that most mentioned entities in fact have their own entry. In consequence, a baseline ranking is achieved by a straightforward article ranking on the Wikipedia corpus combined with an appropriate category filtering mechanism.

The INEX task further provides a few relevant example entities for each query topic. The given entities can be used as relevance feedback to improve the initial text query or to redefine the set of target categories. Another application for the example entities comes with the list completion task. This task asks to derive appropriate target categories automatically from the given relevant entities.

Our main aim for this year’s track participation was to express entity ranking queries completely in the XQuery language. Hence, we wanted to show that

PF/Tijah is “out of the box” able to express and evaluate complex entity ranking queries with a high retrieval quality. One preprocessing step, however, turned out to be unavoidable. The INEX wikipedia corpus comes without category tagging in the provided XML format. Instead, the categorization of all articles is provided by separate plain text files. In order to unify all given information, we integrated the category tagging in the XML corpus itself as shown in the following example:

```
<article><name id="13467">Hamburg</name>
    <body>....</body>
    <category id="5654">cities in germany</category>
    <category id="52414">port cities</category>
</article>
```

Next to the title keywords, target categories, and relevant entities provided with each search topic, we generated for each search topic an additional list of relevant derived categories. Namely those categories assigned to the specified relevant entities. The derived relevant categories are used as mentioned above for refinement of the target categories as well as for the list completion task:

```
for $topic in doc("topics.xml")//inex_topic
let $relevant_entities := $topic//entity/@id
return collection("wikipedia")//
       article[name/@id =
$relevant_entities]//category/@id
```

3.1 Submissions

We submitted in total 6 runs, 4 runs for the entity ranking task, and 2 list completion submissions. The submissions can also be divided into 3 runs based solely on a direct article ranking, and 3 other runs using also the scores of adjacent articles in the link graph.

We start by describing the direct article rankings. The ranking and category filtering is performed by a single XQuery, which is shown below. The fields fields ?QID?, ?QTERMS?, ?CATS?, ?DERIVEDCATS? were substituted according to the given query topic:

```
(: part1 - retrieval :)
let $query_num := "?QID?"
let $q_terms := tijah:tokenize("?QTERMS?")
let $opt := <TijahOptions ir-model="LMS" returnNumber="1000"
             collection-lambda="0.5"/>
let $nexi := concat("//article[about(.,", $q_terms, ")]")
let $tijah_id := tijah:queryall-id($nexi, $opt)
let $nodes := tijah:nodes($tijah_id)

(: part2 - determine target categories :)
let $targetcats := distinct-values(((?CATS?), (?DERIVEDCATS?)))
```

```
(: part3 - filtering and output generation :)
for $a at $rank in $nodes
let $score := if ($a//category/@id = $targetcats)
              then tijah:score($tijah_id, $a)
              else tijah:score($tijah_id, $a) * 0.0000001
order by $score descending
return string-join((string($query_num), "Q0", concat("WP",$a/name/@id),
       string($rank), string($score), "ER_TEC"), " ")
```

The presented XQuery ranks in the first part all articles of the Wikipedia collection according to the topic of the query. We applied here a standard language modeling retrieval model with the smoothing factor set to $\lambda = 0.5$. Moreover, the result set was limited to the top 1000 retrieved articles.

The second part determines the target categories. Whereas our first run *ER_TC* uses only the categories provided with the query topic, the second run *ER_TEC* refines the target category set by uniting the given and derived categories as shown in the query. The list completion *LC_TE*, on the other hand, uses only the derived but not the given categories.

The final part performs the actual filtering and required TREC-style output generation. Notice that the applied filtering in fact only performs a reordering and does not remove articles from the ranked list. Last year's experiments had clearly shown that the reordering comes with a higher recall compared to the filtering technique.

The other 3 runs *ER_TC_idg*, *ER_TEC_idg*, *LC_TE_idg* exploit the retrieval scores of adjacent nodes and follow otherwise a symmetrical experiment schema with respect to the used target categories. The underlying idea behind the exploitation of link structure is adopted from other entity ranking tasks such as expert finding, where we typically find a number of topical relevant documents that mention relevant entities, but entities do not have a textual description themselves. A sample cutout of such a graph is visualized in Figure 1. The edges here symbolize containment of entities within documents. Entities are then ranked by a propagation of scores from adjacent documents.

Although entity ranking on the Wikipedia corpus is different since entities are represented by their own articles and have a text description themselves, it still often occurs the articles outside the target category carry valuable information for the entity ranking. Recall the above given example query searching

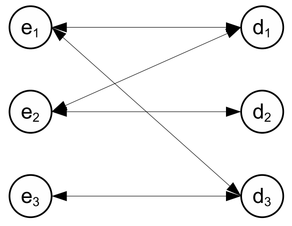
$$\begin{tikzcd} e_1 \arrow[r, leftrightarrow] \arrow[rdd, leftrightarrow] & d_1 \arrow[ld, leftrightarrow] \\ e_2 \arrow[r, leftrightarrow] & d_2 \\ e_3 \arrow[r, leftrightarrow] & d_3 \end{tikzcd}$$

Fig. 1. Part of a link graph containing entities e_i and other documents d_j

for German cities in the hanseatic league. We will find Wikipedia entries about the history of the hanseatic league listing and linking to all major participating cities. While such article remains outside the target category, the links to relevant city pages are of high value for the ranking. Especially, when a city's description itself does not reach far enough into history. We developed last year a ranking method matching this condition [6]. The personalized weighted indegree measure tries to combine the article ranking itself $w(e|q)$ with the ranking of other Wikipedia entries $w(e'|q)$ linking entity e:

$$PwIDG(e) = \mu w(e|q) + (1-\mu) \sum_{e' \in \Gamma(e)} w(e'|q) \tag{1}$$

A corresponding indegree score computation can be expressed as well in XQuery. The below shown query part substitutes the score computation in the previous entity ranking example and sets the parameter μ to 0.85:

```
for $a at $rank in $nodes
let $in_score := sum(
     for $l in $nodes//collectionlink[@*:href =
          concat($a/name/@id, ".xml")]
     let $source_article := exactly-one($l/ancestor::article)
     return tijah:score($tijah_id, $source_article)
)
let $score := if ($a//category/@id = $targetcats)
    then 0.85 * tijah:score($tijah_id, $a) + 0.15 * $in_score
    else (0.85 * tijah:score($tijah_id, $a) + 0.15 * $in_score)
         * 0.0000001
order by $score descending
return string-join((string($query_num), "Q0", concat("WP",$a/name/@id),
     string($rank), string($score), "1_cirquid_ER_TEC_idg"), " ")
```

Notice that each link between two entities is counted separately here. We tested before a version of the query that establishes only one link between two entities e_1 and e_2 even if e_1 links e_2 multiple times. Initial tests on last years data indicated, however, a higher retrieval quality for the above presented query.

3.2 Training

We trained the parameter μ on the data of last year's entity ranking task. For the chosen relevance propagation method a setting of $\mu = 0.85$ showed the best performance with respect to precision on top of the retrieved list as well as for mean average precision:

μ	0.8	0.85	0.9	0.95
MAP	0.3373	**0.3413**	0.3405	0.3349
P5	**0.4435**	**0.4435**	0.4304	0.4348
P10	0.3739	**0.3783**	0.3717	0.3630

3.3 Results

The results will be shown in the final version of this paper, but have not been evaluated at the time writing the notebook paper.

4 Conclusions

We demonstrated with this article the flexibility and effectiveness of the chosen approach to integrate the retrieval language NEXI with the database query language XQuery. The PF/Tijah system allows to express a wide range of INEX experiments without changes to the system itself. Often time consuming pre- and post-processing of data is not necessary or reduced to simple string substitutions of query terms for each given query.

Although PF/Tijah does not apply top-k query processing techniques, it shows a good performance on a wide range of NEXI queries. Future developments should address the currently bad supported retrieval on the entire node set, issued by `//*`-queries.

The INEX entity ranking task demonstrates how standard retrieval functions can be applied to non-standard retrieval tasks with the help of score propagation expressed on the XQuery level. A combined DB/IR system as PF/Tijah can demonstrate here its full advantage.

References

1. Boncz, P., Grust, T., van Keulen, M., Manegold, S., Rittinger, J., Teubner, J.: MonetDB/XQuery: a fast XQuery processor powered by a relational engine. In: SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2006) 479–490
2. List, J., Mihajlovic, V., Ramírez, G., de Vries, A., Hiemstra, D., Blok, H.E.: Tijah: Embracing Information Retrieval methods in XML databases. Information Retrieval Journal **8**(4) (2005) 547–570
3. Hiemstra, D., Rode, H., van Os, R., Flokstra, J.: PF/Tijah: text search in an XML database system. In: Proceedings of the 2nd International Workshop on Open Source Information Retrieval (OSIR), Seattle, WA, USA, Ecole Nationale Supérieure des Mines de Saint-Etienne (2006) 12–17
4. Rode, H.: From Document to Entity Retrieval. PhD thesis, University of Twente, CTIT (2008)
5. Grust, T., van Keulen, M., Teubner, J.: Accelerating XPath evaluation in any RDBMS. ACM Trans. Database Syst. **29** (2004) 91–131
6. Rode, H., Serdyukov, P., Hiemstra, D.: Combining Document- and Paragraph-Based Entity Ranking. In: Proceedings of the 31th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2008). (2008) 851–852