Relevance Feedback in XML Retrieval

Hanglin Pan

Max-Planck-Institut für Informatik,
D-66123 Saarbrücken, Germany
pan@mpi-sb.mpg.de

Abstract. Highly heterogeneous XML data collections that do not have a global schema, as arising, for example, in federations of digital libraries or scientific data repositories, cannot be effectively queried with XQuery or XPath alone, but rather require a ranked retrieval approach. As known from ample work in the IR field, relevance feedback provided by the user that drives automatic query refinement or expansion can often lead to improved search result quality (e.g., precision or recall). In this paper we present a framework for feedback-driven XML query refinement and address several building blocks including reweighting of query conditions and ontology-based query expansion. We point out the issues that arise specifically in the XML context and cannot be simply addressed by straightforward use of traditional IR techniques, and we present our approaches towards tackling them.

1 Introduction

1.1 Motivation

Ranked retrieval systems for heterogeneous XML data with both structural search conditions and keyword conditions have been developed recently across digital libraries, federations of scientific data repositories, and hopefully portions of the ultimate Web (XRank [8], XIRQL [6], XXL [13, 14], etc.). These systems are based on pre-defined similarity measures for elementary conditions and then use rank aggregation techniques to produced ranked results lists. Due to the users' lack of information on the structure and terminology of the underlying diverse data sources, users can often not avoid posing overly broad or overly narrow initial queries, thus facing either too many or too few results.

For the user, it is much more appropriate and easier to provide a relevance judgment on the best results of an initial query execution, and then refine the query, either interactively or automatically by the system. This calls for applying relevance feedback technology [1, 2, 9, 11, 12, 16] in the new area of XML retrieval. The key question is how to generate a refined query appropriately based on a user's feedback in order to obtain more relevant results among the top-k result list.

As an example, suppose we have the following portion of XML document collection (Figure 1) with research activities and bibliographic information after crawling and indexing some scientists' homepages.

The user may submit the following query in order to find researchers who work in Germany on the field of Information Retrieval (IR). The query is expressed in the XXL

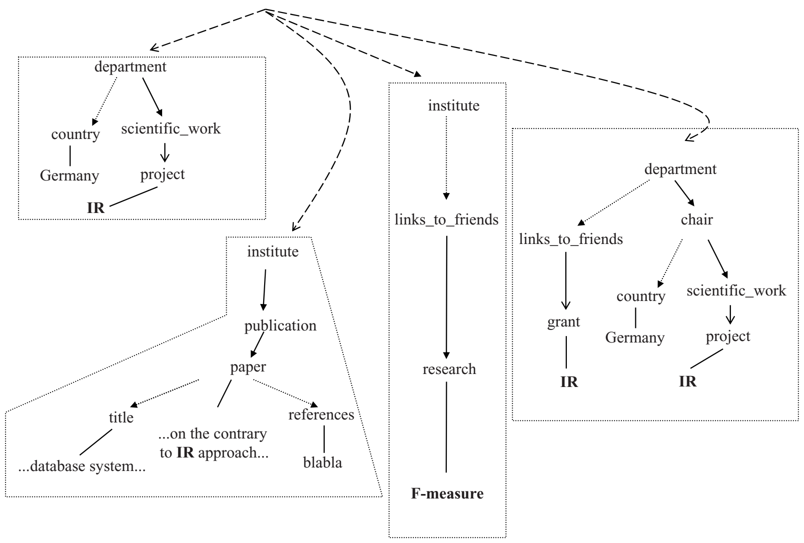

Fig. 1. Structurally diverse XML data graph

language [14], but could be posed in a similar way in other languages such as XIRQL [6] or XRank [8].

```
SELECT * FROM INDEX
WHERE ~department AS $H
AND $H/country = "Germany"
AND $H/#/~research AS $R
AND $R ~ "IR";
```

Here '/' stands for path concatenation, '#' for arbitrary paths of XML elements, '$\sim$' for semantic similarity conditions on XML element (or attribute) names as well as XML element (or attribute) contents, and '`$H`' for variable condition. We refer to [14] for details.

XXL queries contain exact and vague conditions. In the above query, the path-matching condition `$H/country` and the element-content condition =`"Germany"` are exact, the other four conditions are vague: 1) `#`, 2) $\sim$`department`, 3) $\sim$`research`, 4) $\sim$ `"IR"`. In the following we focus on the vague conditions.

The sample query is decomposed into four elementary sub-queries, each of which is binded with a $weight$ indicating its relative importance. (see weight assignment in section 2.1 and architecture of XXL in section 4.)

Each of elementary sub-queries is evaluated by system locally and scored based on IR-style $tf * idf$ measures for element contents and general ontological similarity measure for element names. The total *score* of an XML path with regard to the entire query is computed in a simple probabilistic manner as the product of the local scores, (for all conditions are combined in a conjunctive manner. See section 2.1 for how to combine *weights* and *scores*.) Here using *maximum* is one option to avoid result redundancy. To give an impression on how overall scores are computed based on local scores, we show a sample result list:

```
Result 1: (overall=0.8)
 /department;1/(scientific_work/project);0.8/IR;1

Result 2: (overall=0.7)
 /institute;1/publication/paper;0.7/IR;1

Result 3: (overall=0.6*1*0.9=0.54)
 /institute;1/links_to_friends;0.6/research;1/F-measure;0.9

Result 4: (overall=max(0.48,0.42)=0.48)
 /department/chair;0.6/(scientific_work/project);0.8/IR;1
 /department/links_to_friends;0.6/grant;0.7/IR;1
```

In the result set from the first round of query execution, the attached numbers to each blocks of meta-data from index are *scores* given by the engine, either as local measurements on element path (some are shown as rectangle in Figure 2) and element content (shown as circle), or as overall.

In such a ranked retrieval model, the aggregation of, possibly weighted, scores of sub-queries is often subjective to the user and thus should be personalized at runtime. Relevance feedback can help to automatically tune weights and other options of the query execution engine to the user's specific information needs.

In an XML setting we have much richer opportunities for relevance feedback than in a traditional text-only IR environment. Consider the sample results for our example query shown in Figure 2. At the document level, the user may mark results 1, 2, and 4 as positively relevant. At the element level, the user has much more fine-grained control over positive and negative feedback. For example, in results 1 and 4, the path `scientific_work/project` is positive. In result 2, after zooming into the content, the user may find out that the paper is mainly about "Database Systems" instead of "Information Retrieval", so it is assessed as negative. Finally, in results 3 and 4, the tag name `links_to_friends` is assessed as negative.

Our opportunity now is to exploit this kind of feedback for automatically refining the initial query into a better suited query such as:

```
SELECT * FROM INDEX
WHERE (department;1.0 | institute;1.0 | chair;0.8) AS $H ::1.0
AND $H/country = "Germany" ::0.6
AND $H/(research;1.0 | (scientific_work/project);0.95 | project;0.8 | grant;0.7
| paper;0.7)/# AS $R ::0.8
AND $R ∼ ("IR"|"F-measure") ::0.9;
```

Here the numbers after '::' that are attached to the various search conditions are *weights* that are used in the total scoring function (see section 2.1) to reflect the relative importance of the various conditions. The numbers after ';' are *weights* used for local

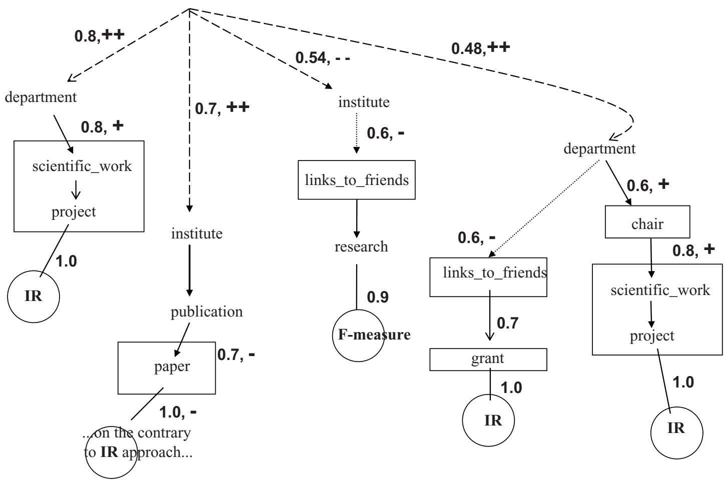

Fig. 2. Sample retrieval results, with computed similarity scores and user's feedback at both document level (marked with ++, −−) and element level (marked with +,−)

scoring function in query expansion. The symbol '|' denotes disjunction. Thus the new query uses query expansion to reach out for semantically relevant data that uses diverse terminology different from the terms in the original query. Query expansion is a standard technique in IR, but note that in our approach it is used not just for XML content terms, but also for XML element names and it is driven by query-specific ontological similarity measures [13].

1.2 Research Objectives

The objectives in this Ph.D. project are twofold:

Feedback Capturing: A systematic study is required on how to capture and exploit different kinds of feedback interactions like:

Binary feedback vs. non-binary feedback:

1. Binary feedback means (+), (−) are the only two values of feedback.
2. Non-binary feedback means using a multivalued relevance scale, e.g. (+2), (+1), and (−2) mean "highly relevant", "marginally relevant", and "not at all relevant" respectively.

Feedback for different granularities:

1. feedback only on entire documents,
2. feedback on documents as well as elements,
3. feedback on documents, elements, and also entire paths.

Query Modification: The aim is to develop a formal foundation for a comprehensive query refinement framework in XML retrieval, with an efficient implementation and an experimental evaluation of benefits, tradeoffs, and fine-tuning options. We currently focus on the following two techniques:

Reweighting of Elementary Query Conditions: Reweight on both exact conditions and vague ones after feedback. Furthermore, remove a elementary condition completely from the query if its weight is lower than a certain threshold.

Expansion of Elementary Query Conditions: Introduce local disjunctions for element content conditions (keywords or phrases) and element names based on personalized ontology. Another example is adding new variable condition as intermediate if necessary.

2 Query Refinement Model

We take the following steps in our current query refinement model:

1. Distinguish different types of elementary vague conditions:
   - C–conditions (*Content conditions*, e.g., $\sim$ "IR"),
   - T–conditions (*Tag conditions*, e.g., ~`research`),
   - P–conditions (*Path conditions*, e.g., #),
   - V–conditions (*Variable conditions*, e.g., `$H`).
2. Obtain feedback for each vague condition (identified by VagueID), with different granularities, e.g., single XML element level, block of path/tree/sub-graph level (over several XML elements), or document level.
3. Compute statistics over positive and negative docs (i.e., feedback on each result entry), and then adjust weights of conditions for the total scoring function and/or expand the query.
4. Refine query appropriately.

These steps are explained further in Sections 2.1 through 2.3.

2.1 Weight Adjustment

To compute an overall ranking score for each result document (or path within a document or even across multiple documents, which is supported by XXL [14]), individual scores of the elementary conditions in the query are combined and weighted by their relative importance, using the following scoring model or weighted sum or more such as in [5]:

$$Score(Q)_{overall} = \prod_{i=1}^{m+n} \left(\left(score(q_i, d) \right)^{weight(q_i)} \right) \tag{1}$$

where d is a candidate document or element/path/tree/sub-graph, and query $Q = (q_1, \ldots, q_m, q_{m+1}, \ldots, q_{m+n})$. Each q_i is an elementary condition, where $q_1, \ldots, q_m$ are *exact conditions* (each score is either 0 or 1) and $q_{m+1}, \ldots, q_{m+n}$ are *vague conditions* (each score is between 0 and 1). For the example in Section 1.1, m=2 and n=4. In the first round of iteration all conditions $q_1, \ldots, q_{m+n}$ are assigned the same $weight$ as 1.0. In subsequent rounds, based on positive and negative user feedback after each iteration, we will adjust each $weight(q_i)$, which will influence the overall score in the next iteration. Normalization is followed when necessary so that $\sum_{i=1}^{m+n}(weight(q_i)) = 1$ in certain scoring models. The weights are determined by using linear regression to minimize the error between the system-computed score and the user assessment.

2.2 Local Query Expansion

We do local query expansion on each sub-query by broadening the vague condition based on an ontology index and then trying to disambiguate the sub-query with the help of user feedback. We expand elementary vague conditions into local disjunctions with corresponding weights for each candidate.

For example, the content condition $\sim$ `"IR"` could be expanded into `( IR;1.0 | (Vector Space Model);0.7 | (Information Retrieval);1.0 )`, the content condition `"Germany"` as `( Germany;1.0 | Deutschland;1.0 )`, (see section 2.3 for variable conditions and path conditions), and the tag condition `~research` could be expanded into `( (scientific_work/project);0.95 | research;1.0 | project;0.8 ).`

As pointed out in [13, 15], the idea of ontology-driven query expansion is to identify strongly related concepts in a directed ontology graph. In more detail, consider an ontology graph with concepts as nodes, synonyms as separate explicit nodes, synonym edges with weight 1, and hypernym or hyponym edges with weights derived from correlation measures over corpus statistics between 0 and 1 (assume, for simpler presentation, that there are no polysems.). The personalized ontology graph is the source that gives us candidate terms for query expansion if the similarity weight is higher than a certain threshold.

Ontology-Based Query Expansion: Our query expansion approach is based on both ontological knowledge for initialization and human feedback for modification:

1. Evaluate the query with expansion driven by the original ontology from the Global Ontology Index (GOI), see architecture in section 4.
2. After the first round of feedback, construct a Query-specific Ontology Index (QOI) as a small portion of GOI by making a copy of all the necessary information.
3. Add the query as an extra node, and add edges with high weight (e.g., 1) to concepts with positive feedback, also add edges with low weight (e.g., 0) to concepts with negative feedback.
4. After each round of query re-execution plus user feedback, modify the QOI graph.

For the query `~research` and some sample results with feedback, the result of local query expansion might be adding positive edges: $(q - research; 1)$, $(q - project; 0.8)$, and negative edges: $(q - teaching; 0)$, $(q - lecture; 0)$, shown in Figure 3. This modified ontology graph will be used as the source for query expansion. Based on Figure 3, `~research` will be expanded as `(research;1 |scientific_work;1 |project;0.8 |grant;0.7 |fund;0.54 |teaching;0 |lecture;0).`

Terms that received negative feedbacks are still among the list for filter purposes, even if their weight is zero.

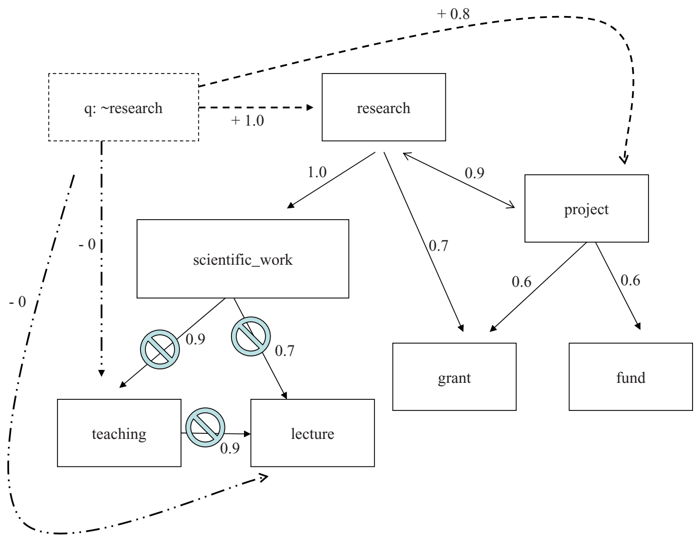

Fig. 3. Sample query-specific ontology (QOI) graph with an inserted node q: ∼research, all connections are directed and weighted

Further Issues on Expansion: In this subsection, we briefly point out some further questions on ontology-based query expansion among many.

1. How to choose the best candidates for expansion?
   Use thresholds on distance measures between original query terms and expansion candidates, and use correlation measures such as *Cosine measures* or *Euclidian distance* or *Kullback-Leibler divergence*.
2. How to do disambiguation, i.e., how to map query words onto concepts in the presence of polysems?
   Rely on word context (from query and documents with positive feedback) and concept context in ontology (e.g., descriptions of hyponyms, siblings, etc.), see [13] for more details.
3. How to keep direct and transitive weights consistent while updating the ontology graph?
   So far: $sim(x, y) = max\{sim(p)|$ all paths p from x to $y\}$, i.e., the strongest path with query-specific additional edges determines the similarity. This works fine for additional positive edges, but not for additional negative edges.

2.3 Global Query Expansion

Global query expansion includes adding new conjunctive conditions to the query. Using the example of section 1.1, a new elementary condition (`$H/city AS $X`) with initial weight 1.0 could be possibly added based on feedback information analysis, in order to catch user's information need. In the same example, the path condition (`/#`) could be possibly expanded into (`/#/Publications AS $P AND $P/#`).

2.4 Long-Term Profiling

As one of long-term goals, we plan to exploit query logs [4] and user profiles [3], etc., as they can be regarded as *implicit* user feedback. In general, the derived information about a user's interest profile can be used for adaptive ranking of search results. The information about a user's activities can be collected on the client side (e.g., using a browser-embedded ActiveX component) or on the server side (e.g., using session logs).

3 Interaction with Feedback Information and Guidance of Users

A good user interface is also important, in order to show ranked result lists, zoom in and zoom out on some portion of XML data, obtain the user's subjective feedback at the document level or element level (or others, such as path/tag/content combination), show the possible query expansion candidates, and provide implicit or explicit user guidance according to individual preferences.

At present we simply use a two-dimensional graph layout for showing the XML data of ranked results from XML retrieval engine, plus pop-up menu for relevance feedback. A next step could be studying richer visual techniques for navigation in large collections of XML data. For this purpose, the Cone Tree technique, or a commercial product named Inxight Star Tree (hyperbolic technique in [10]) could be of interest.

4 System Architecture

Figure 4 depicts the architecture of the XXL search engine and the extensions for relevance feedback that we are currently working on.

- XXL Applet handles the user interface.
- The Query Logs Manager (QLM) maintains implicit information of user's interest.
- The Scoring Model (SM) maintains the parameters of overall scoring function and the weights of all elementary query conditions.
- The Query-specific Ontology Index (QOI) maintains the candidate terms for query expansion.
- The Feedback Manager (FbM) collects all the information from user's feedback with different granularities, does statistical computation, maintains necessary information for query refinement, and updates QOI and SM, etc.
- The Query Processor (QP) is the kernel of a XML retrieval system [14]. It parsers and executes query, generates result lists based on local score evaluation (by accessing Element-Content-Index, Element-Path-Index and Global-Ontology-Index) and overall weighted score evaluation by accessing SM, refines query by accessing QOI and FbM, etc.

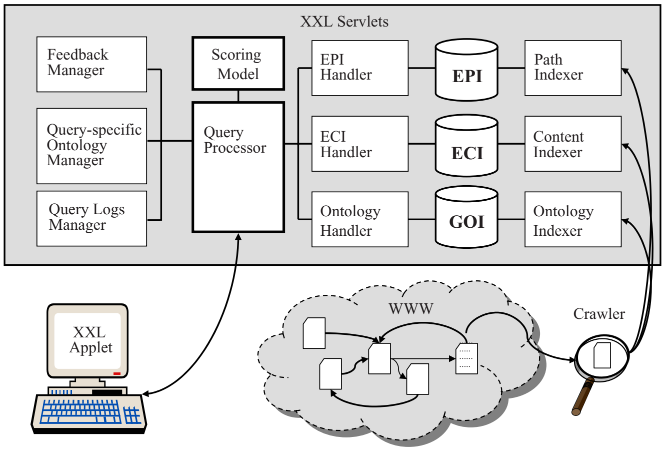

Fig. 4. Architecture of the XXL Search Engine with relevance feedback support

5 Evaluation Plan

Our evaluation plan is to participate in the proposed *Relevance Feedback Track* at INEX 2004 [7]. It will provide a benchmark for relevance assessment for further the relevance feedback process in XML retrieval, and handle both Content-Only (CO) queries and Content-And-Structure (CAS) queries.

Acknowledgments

I wish to express my gratitude to my advisor Gerhard Weikum for his guidance and encouragement, and to Ralf Schenkel for his helpful comments.

References

1. Ricardo Baeza-Yates and Berthier Ribeiro-Neto, editors. *Modern Information Retrieval.* Addison Wesley, 1999.
2. Chris Buckley and Gerard Salton. Optimization of relevance feedback weights. In *Proc. of the 18th ACM SIGIR*, pages 351–357. ACM Press, 1995.
3. Ugur Cetintemel, Michael J. Franklin, and C. Lee Giles. Flexible user profiles for large scale data delivery. Technical Report CS-TR-4005 (UMIACS-TR-99-18), University of Maryland, 1999.
4. Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Query expansion by mining user logs. *IEEE Transaction on Knowledge and Data Engineering*, 15(4):829–839, 2003.
5. Ronald Fagin and Edward L. Wimmers. Incorporating user preferences in multimedia queries. In Foto N. Afrati and Phokion G. Kolaitis, editors, *Database Theory – ICDT '97, 6th Int. Conf., Delphi, Greece, 1997, Proceedings*, volume 1186 of *LNCS*, pages 247–261. Springer, 1997.

6. Norbert Fuhr and Kai Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Research and Development in Information Retrieval*, pages 172–180, 2001.
7. Norbert Fuhr and Mounia Lalmas. Initiative for the evaluation of XML retrieval (INEX), 2003. `http://inex.is.informatik.uni-duisburg.de:2003/`.
8. Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *Proc. of the 2003 ACM SIGMOD Int. Conf. on Management of Data*, pages 16–27. ACM Press, 2003.
9. E. Ide. New experiments in relevance feedback. In Gerard Salton, editor, *The Smart Retrieval System: Experiments in Automatic Document Processing*, pages 337–354. Prentice Hall, 1971.
10. John Lamping and Ramana Rao. Laying out and visualizing large trees using a hyperbolic space. In *ACM Symposium on User Interface Software and Technology*, pages 13–14, 1994.
11. Michael Ortega-Binderberger, Kaushik Chakrabarti, and Sharad Mehrotra. An approach to integrating query refinement in SQL. In *Extending Database Technology, EDBT*, pages 15–33, 2002.
12. J. Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *The Smart Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. Prentice Hall, 1971.
13. Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Ontology-enabled XML search. In Henk M. Blanken, Torsten Grabs, Hans-Jörg Schek, Ralf Schenkel, and Gerhard Weikum, editors, *Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *LNCS*, pages 119–131. Springer, 2003.
14. Anja Theobald and Gerhard Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In Christian S. Jensen, Keith G. Jeffery, Jaroslav Pokorný, Simonas Saltenis, Elisa Bertino, Klemens Böhm, and Matthias Jarke, editors, *Advances in Database Technology – EDBT 2002, 8th Int. Conf. on Extending Database Technology, Prague, Czech Republic, Proceedings*, volume 2287 of *LNCS*, pages 477–495. Springer, 2002.
15. Martin Theobald, Ralf Schenkel, and Gerhard Weikum. Exploiting structure, annotation, and ontological knowledge for automatic classification of XML data. In Vassilis Christophides and Juliana Freire, editors, *WebDB: International Workshop on Web and Databases, San Diego, 2003*, 2003.
16. Xiang Sean Zhou and Thomas S. Huang. Exploring the nature and variants of relevance feedback. In *Proc. of the IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL'01)*, pages 94–101. IEEE CS Press, 2001.