Feedback-Driven Structural Query Expansion for Ranked Retrieval of XML Data

Ralf Schenkel and Martin Theobald

Max-Planck-Institut für Informatik, Saarbrücken, Germany
{schenkel, mtb}@mpi-inf.mpg.de

Abstract. Relevance Feedback is an important way to enhance retrieval quality by integrating relevance information provided by a user. In XML retrieval, feedback engines usually generate an expanded query from the content of elements marked as relevant or nonrelevant. This approach that is inspired by text-based IR completely ignores the semistructured nature of XML. This paper makes the important step from content-based to structural feedback. It presents an integrated solution for expanding keyword queries with new content, path, and document constraints. An extensible framework evaluates such query conditions with existing keyword-based XML search engines while allowing to easily integrate new dimensions of feedback. Extensive experiments with the established INEX benchmark show the feasibility of our approach.

1 Introduction

1.1 Motivation

With the proliferation of XML as a document format, information retrieval on XML data has recently received great attention. XML search engines employ the ranked retrieval paradigm for producing relevance-ordered result lists rather than merely using XPath or XQuery for Boolean retrieval. An important subset of XML search engines uses keyword-based queries [3, 9, 28], which is especially important for collections of documents with unknown or highly heterogeneous schemas.

Relevance Feedback is an important way to enhance retrieval quality by integrating relevance information provided by a user. In XML retrieval, existing feedback engines usually generate an expanded keyword query from the content of elements marked as relevant or nonrelevant. This approach that is inspired by text-based IR completely ignores the semistructured nature of XML. This paper makes the important step from content-based to structural feedback. We extend the well-established feedback approach by Rocchio [21] to expand a keyword-based query with additional structural constraints on result elements and on documents in which result elements reside, in addition to "standard" content-based query expansion. The resulting expanded query has weighted structural and content constraints and can be fed into a full-fledged XML search engine like our own TopX engine [24].

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 331–348, 2006.

However, as there are many keyword-only search engines [3, 9] and some other engines allow only unweighted structural constraints, this approach is not generally applicable. Additionally, the correct choice of weights and the way the query is expanded depend on the underlying scoring model of the engine (like Rocchio's method initially requires that the vector space model is applied). To overcome these problems, this paper presents an extensible framework to extend existing keyword-based XML search engines with structure-based feedback. It reranks results of keyword-only queries according to additional scores induced by the structure of results that are marked as relevant (or nonrelevant).

This paper makes the following important contributions: (1) It presents a formal framework to integrate different dimensions of feedback, beyond content-based feedback, into XML retrieval, (2) it presents two structural query expansion techniques as important instances of new dimensions for query expansion, and (3) it shows how to effectively implement the framework, including structural constraints, with keyword-based XML search engines. We show that structural query expansion gives a huge gain in effectiveness with the established INEX benchmark [13].

To the best of our knowledge, this is the first paper that considers user relevance feedback for structural query expansion. The primary goal of this paper is to show that structural feedback helps to enhance result quality. The paper does not claim to present the ultimately best implementation of structural feedback, but opens a whole design space and presents variants that give reasonably good results.

1.2 Related Work

Relevance feedback has already been considered for document retrieval for a long time, starting with Rocchio's query expansion algorithm [21]. Ruthven and Lalmas [22] give an extensive overview about relevance feedback for unstructured data, including the assessment of relevance feedback algorithms.

Relevance feedback in XML IR is not yet that popular. Of the few papers that have considered it, most contentrate on query expansion based on the content of elements with known relevance [7, 15, 23, 27]. Some of these focus on blind ("pseudo") feedback, others on user feedback. Pan et al. [17, 18] apply user feedback to recompute similarities in the ontology used for query evaluation.

Even fewer papers have considered structural query expansion [8, 10, 11, 16, 19, 20]. Mihajlovič et al. [16, 19, 20] proposed deriving the relevance of an element from its tag name, but could not show any significant gain in retrieval effectiveness. Additionally, they considered hand-tuned structural features specific for the INEX benchmark (e.g., the name of the journal to which an element's document belongs), but again without a significant positive effect. In contrast, we propose a general approach for feedback that can be applied with INEX, but does not rely on any INEX-specific things.

Hlaoua and Boughanem [10] consider common prefixes of relevant elements' paths as additional query constraints, but don't provide any experimental

evaluation of their approach. Our path-based feedback supports path prefixes as one of six classes of path features.

Gonçalvez et al. [8] use relevance feedback to construct a restricted class of structured queries (namely field-term pairs) on structured bibliographic data, using a Bayesian network for query evaluation, but did not consider semistructured data like XML.

The work of Hsu et al. [11] is closest to our approach. They use blind feedback to expand a keyword-based query with structural constraints derived from a neighborhood of elements that contain the keywords in the original query. Our approach considers the whole document instead of only a fragment, can generate constraints with negative weight, and integrates also path- and content-based constraints.

1.3 Formal Model and Notation

We consider a fixed corpus of XML documents. For such a document d, $E(d)$ denotes the set of elements of the document; for an element e, $\text{tag}(e)$ denotes its tag name and $D(e)$ the document to which it belongs.

The *content* $c(e)$ of an element e is the set of all terms (after stopword removal and optional stemming) in the textual content of the element itself and all its descendants. (Note that XML retrieval engines usually use this content model, while boolean languages like XPath or Xquery typically only use the content of the element itself.) For each term t and element e, we maintain a weight $w_e(t)$. This can be a binary weight ($w_e(t) = 1$ if the term occurs in e's content and 0 otherwise), a tf-idf style [14] or a BM25-based [1, 26] weight that captures the importance of t in e's content.

We represent elements in the well-known vector space model. Formally, with $T = \{t_1, ..., t_{|T|}\}$ the set of all terms occuring in the contents of elements, our vector space is $\mathcal{V} = \mathbb{R}^{|T|}$. Each element e is assigned a vector $\boldsymbol{e} \in \mathcal{V}$ where $\boldsymbol{e}_i = w_e(t_i)$ corresponds to the weight of term t_i in e. Analogously, a query $\boldsymbol{q} \in \mathcal{V}$ is also a vector with nonzero entries for the requested keywords. The score of an element with respect to a query can then be defined as the cosine similarity [2]

$$s(\boldsymbol{q}, e) = \frac{< \boldsymbol{q}; \boldsymbol{e} >}{\|\boldsymbol{q}\| \cdot \|\boldsymbol{e}\|}$$

of the query and the element's vector (or any other distance measure), and the result to a query is then a list of elements sorted by descending score.

Note that, even though this paper uses the vector space model, the techniques for query expansion presented here can be carried over to other retrieval models, as long as they allow queries with structural constraints.

2 Dimensions for Query Expansion

In text-based IR and mostly also in XML IR, feedback has concentrated on the content of relevant documents or elements only. We propose to extend this

text-driven notion of feedback with new structural dimensions for feedback that are more adequate to the semistructured nature of XML. Our framework for feedback supports the following three feedback dimensions:

- *content constraints* that impose additional conditions on the content of relevant elements,
- *path constraints* that restrain the path of relevant elements, and
- *document constraints* that characterize documents that contain relevant elements.

Another possible dimension for feedback is the quality of ontological expansions that is considered in [18].

In the remainder of this section, we give motivating examples for each of the dimensions and show how they can be formally expressed in the vector space model. The evaluation of queries that are expanded along these dimensions is presented in the following section.

For the remainder of this section, we consider a keyword query $\boldsymbol{q} \in \mathcal{V}$ for which we know a set $E^+ = \{e_1^+, \ldots, e_r^+\}$ of relevant elements and a set $E^- = \{e_1^-, \ldots, e_n^-\}$ of nonrelevant elements, e.g., from user feedback. We assume boolean feedback, i.e., a single element can be relevant or nonrelevant, but not both. The models and formulae presented in the following can be extended to support weighted feedback (to capture vague information like 'somewhat relevant' or weighted relevance like in the different INEX quantizations), e.g. using the notion of probabilistic sets, but this is beyond the scope of this paper.

2.1 Content Constraints

Content-based feedback is widely used in standard IR and has also made its way into XML retrieval [15, 23]. It expands the original query with new, weighted keywords that are derived from the content of elements with known relevance. As an example, consider the keyword query `"multimedia information" retrieval` (this is topic 178 from the INEX topic collection). From the feedback of a user, we may derive that elements that contain the terms 'brightness', 'annotation', or 'rgb' are likely to be relevant, whereas elements with 'hypermedia' or 'authoring' are often irrelevant. An expanded query could include the former terms with positive weights and the letter terms with negative weights.

Formally, to expand our keyword query $\boldsymbol{q}$ with new keywords, we apply a straight-forward extension of the well-known Rocchio method [21] to XML: We add new weighted keywords to the query that are taken from the contents of result elements for which we know the relevance. The weight $w(t)$ for a term t is computed analogously to Rocchio with binary weights:

$$w_c(t) = \beta_c \cdot \frac{\sum_{e \in E^+} w_e(t)}{|E^+|} - \gamma_c \cdot \frac{\sum_{e \in E^-} w_e(t)}{|E^-|}$$

with adjustable tuning parameters β_c and γ_c between 0 and 1. We set $\beta_c = 0.5$ and $\gamma_c = 0.25$ which gave good results in our experiments. A term has a high

positive weight if it occurs in many relevant elements, but only a few nonrelevant elements, so adding it to the query may help in dropping the number of nonrelevant results. Analogously, terms that are frequent among the nonrelevant elements' contents, but infrequent among the relevant elements', get a high negative weight. The expanded query $\boldsymbol{q^c}$ is then a combination of $\boldsymbol{q}$ with weighted terms from the candidate set, i.e., $q_i^c = \alpha \cdot q_i + w_c(t_i)$ for all $t_i \in T$.

Among the (usually many) possible expansion terms, we choose the n_c with highest absolute weight, with n_c usually less than 10. If there are too many with the same weight, we use the mutual information of the term's score distribution among the elements with known relevance and the relevance distribution as a tie breaker, which prefers terms that have high scores in relevant elements, but low scores (or are not present at all) in nonrelevant elements, or vice versa. If E^+ is empty, i.e., all elements with feedback are nonrelevant, mutual information cannot distinguish good from bad expansion terms as it is zero for all terms. We use the term's element frequency (the number of elements in which this term occurs) for tie breaking then, preferring terms that occur infrequently.

2.2 Path Constraints

Elements with certain tag names are more likely to be relevant than elements with other tag names. As an example, a keyword query may return entries from the index of a book or journal with high scores as they often contain exactly the requested keywords, but such elements are usually not relevant. Additionally, queries may prefer either large elements (such as whole articles) or small elements (such as single paragraphs), but rarely both. However, experiments show that tag names alone do not bear enough information to enhance retrieval quality, but the whole path of a result element plays an important role. As an example, the relevance of a paragraph may depend on whether it is in the body of an article (with a path like `/article/bdy/sec/p` from the root element), in the description of the vitae of the authors (with a path like `/article/bm/vt/p`), or in the copyright statement of the journal (with a path like `/article/fm/cr/p`).

As element tag names are too limited, but complete paths may be too strict, we consider the following seven classes of *path fragments*, with complete paths and tag names being special cases:

- P_1: prefixes of paths, e.g., `article/#`,`/article/fm/#`
- P_2: infixes of paths, e.g., `#/fm/#`
- P_3: subpaths of length 2, e.g., `#/sec/p/#`
- P_4: paths with wildcards, e.g, `#/bm/#/p/#`
- P_5: suffixes of paths, e.g., `#/fig`, `#/article`
- P_6: full paths, e.g, `/article/bdy/sec`

Mihajlovič et al. [16] used a variant of P_5, namely tag names of result elements, but did not see any improvement. In fact, only a combination of fragments from several classes leads to enhancements in result quality.

Formally, we consider for an element e its set $P(e)$ of path fragments that are computed from its path. For each fragment that occurs in any such set, we compute its weight

$$w_p(p) = \beta_p \cdot \frac{|\{e \in E^+ : p \in P(e)\}|}{|E^+|} - \gamma_p \cdot \frac{|\{e \in E^- : p \in P(e)\}|}{|E^-|}$$

with adjustable tuning parameters β_p and γ_p between 0 and 1; we currently use $\beta_p = 1.0$ and $\gamma_p = 0.25$. This is a straightforward application of Rocchio's formula to occurrences of path features in result elements.

As we created new dimensions, we cannot use our initial vector space $\mathcal{V}$ to expand the initial query $\boldsymbol{q}$. Instead, we create a new vector space $\mathcal{V}_p = \mathbb{R}^{|P|}$ where $P = \{p_1, \ldots, p_{|P|}\}$ is the set of all path features of elements in the corpus. The components of the vector $\boldsymbol{e^p} \in \mathcal{V}_p$ for an element e are 1 for path features that can be computed from e's path and 0 otherwise. The extended query $\boldsymbol{q^p}$ is then a vector in $\mathcal{V}_p$ with $q_i^p = w_p(p_i)$.

2.3 Document Constraints

Unlike standard text retrieval where the unit of retrieval is whole documents, XML retrieval focuses on retrieving parts of documents, namely elements. Information in other parts of a document with a relevant element can help to characterize documents in which relevant elements occur. A natural kind of such information is the content of other elements in such documents.

As an example, consider again INEX topic 178 (`"multimedia information" retrieval`). We may derive from user feedback that documents with the terms 'pattern, analysis, machine, intelligence' in the journal title (i.e., those from the 'IEEE Transactions on Pattern Analysis and Machine Learing') are likely to contain relevant elements. The same may hold for documents that cite papers by Gorkani and Huang (who are co-authors of the central paper about the QBIC system), whereas documents that cite papers with the term 'interface' in their title probably don't contain relevant elements (as they probably deal with interface issues in multimedia applications).

Formally, we consider for a document d its set $S(d)$ of *structural features*, i.e., all pairs $(\mathrm{T}(e), t)$ where $\mathrm{T}(e)$ is the tag name of an element e within d and t is a term in the content of e. For each feature s that occurs in any such set, we compute its weight

$$w_s(s) = \beta_s \frac{|\{e \in E^+ : s \in S(D(e))\}|}{|E^+|} - \gamma_s \frac{|\{e \in E^- : s \in S(D(e))\}|}{|E^-|}$$

with adjustable tuning parameters β_s and γ_s between 0 and 1; we currently use $\beta_s = \gamma_s = 1.0$. This is a straight-forward application of Rocchio weights to structural features. Note that we are using binary weights for each structural feature, i.e., a feature is counted only once per document. We experimented with using scores here, but the results were always worse than with binary weights; however, a more detailed study of different score functions is subject to future work. We select the n_s features with highest absolute score to expand the query. If there are too many with the same weight, we use the mutual information of

the feature's score distribution among the documents with known relevance and the relevance distribution as a tie breaker (like we did with content features). If there are no positive exampes and mutual information is zero for all features, we use the tag-term pair's document frequency (the number of documents in which this term occurs) for tie breaking then, preferring tag-term pairs that occur infrequently.

As with path constraints, we have to consider a new vector space $\mathcal{V}_s = \mathbb{R}^{|S|}$ where $S = \{s_1, ..., s_{|S|}\}$ is the set of all structural features of documents in the corpus[1]. Each element e is assigned a vector $\boldsymbol{e^s} \in \mathcal{V}_s$ such that $e^s_i = 1$ if $D(e)$ contains the structural feature s_i and 0 otherwise. The extended query $\boldsymbol{q^s}$ is then a vector in $\mathcal{V}_s$ with $q^s_i = w_s(s_i)$.

Other possible structural features include twigs, occurence of elements with certain names in a document, or combination of path fragments with terms. Further exploration of this diversity is subject to future work.

3 Expanding and Evaluating Queries

Once we have generated additional content, path, and document constraints, we want to evaluate the query and retrieve better results. There are three different ways to do this: (1) Evaluate the query in a combined vector space, (2) convert the generated constraints to the query language used in an existing XML retrieval engine (e.g., the engine used to compute the initial results) and evaluate the expanded query with that engine, (3) evaluate some part of the expanded query with an existing engine, the remaining part in the vector space model, and combine the resulting scores.

3.1 Combined Vector Space

As we have expressed each feedback dimension in a vector space, it is straightforward to combine the individual vector spaces and evaluate the query in the resulting combined vector space $\mathcal{V}' = \mathcal{V} \times \mathcal{V}_P \times \mathcal{V}_S$. The score of an element for an expanded query is then the similarity of the element's combined vectors and the corresponding combined vectors of the expanded query, measured for example with cosine similarity.

While this is an elegant solution from a theory point of view, it is not as elegant in practise. Even though we have a probably well-tuned XML search engine used to generate the initial set of results, we cannot use it here, but have to reimplement a specialized search engine for the new vector space. It is unsatisfactory to develop two independent, full-fledged search engines just to allow user feedback.

3.2 Engine-Based Evaluation

Standard feedback algorithms for text retrieval usually generate an expanded weighted keyword query that is evaluated with the same engine that produced

[1] Note that Carmel et al. [5] introduced a similar vector space with (tag, term) features for XML retrieval, but not for feedback.

the initial results. XML retrieval is different as expanded queries consist of more than simple keyword conditions. To benefit from the three dimensions of query expansion introduced in this paper (content, path, and document constraints), an XML search engine must support queries with weighted content and structural constraints that should be evaluated in a disjunctive mode (i.e., the more conditions an element satisfies, the better, but not conditions have to be strictly satisfied). Given such an engine with its query language, generating an expanded query is almost trivial.

As an example, consider again INEX topic 178 (`"multimedia information" retrieval`) already discussed in the previous section. We use INEX's query language NEXI [25] to explain how an expanded query is generated. NEXI basically corresponds to XPath restricted to the `descendants-or-self` and `self` axis and extended by an IR-style `about` predicate to specify conditions that relevant elements should fulfil. The initial query can be reformulated to the semantically identical NEXI query `//*[about(.,"multimedia information" retrieval)]`, i.e., find any element that is "about" the keywords. Let's assume that the best content-based features were 'brightness' with weight 0.8 and 'hypermedia' with weight -0.7, and that the best document-based features (i.e., tag-term pairs) were 'bib[Gorkani]' with weight 0.9 and 'bib[interface]' with weight -0.85. Here 'bib[Gorkani]' means that the keyword 'Gorkani' occurs in an element with tag name 'bib'. Expanding the query with these features yields the following weighted query, formulated in NEXI with extensions for weights:

```
//article[0.9*about(.//bib,"Gorkani")
        -0.85*about(.//bib,'interface)]
      //*[about(.,0.8*brightness -0.7*hypermedia "multimedia
                  information" retrieval)]
```

This query extends the initial keyword query with additional weighted constraints on the content of relevant elements. To specify document constraints, we have to somewhat abuse NEXI: We add an article element that has to be an ancestor of results; as the root of each INEX document is an article element, this is always true. We need it to specify the document constraints within its predicate. Such an expanded query can be submitted to any search engine that supports NEXI, possibly without the weights as they are not part of standard NEXI yet.

Path constraints cannot be easily mapped to a corresponding NEXI query as we cannot easily specify components of paths to result elements in NEXI. We evaluate path constraints therefore not with the engine, but externally; details are shown in the following subsection.

3.3 Hybrid Evaluation

The most promising approach for evaluating an expanded query is to evaluate as much of the expanded query with the existing search engine, evaluate the remaining part of the query separately, and combine the partial scores of each

element. The result of the expanded query is then a ranked list of elements, sorted by combined score. Besides reusing the existing engine, this approach has the additional advantage that we can use different similarity measures for the different constraint dimensions and are not fixed to a single one (as with the extended vector space). On top of that, we can easily integrate new constraint dimensions.

As an example, assume we decide to evaluate the content-based part of the expanded query (i.e., $\boldsymbol{q^c}$ in the notation of Section 2.1) with the engine and the remaining parts $\boldsymbol{q^p}$ and $\boldsymbol{q^s}$ in their corresponding vector spaces. The combined score of an element e is then the sum of its score $S_c(e)$ computed by the engine for $\boldsymbol{q^c}$, its score $S_p(e)$ for $\boldsymbol{q^p}$ and its score $S_s(e)$ for $\boldsymbol{q^s}$, where each of the additional scores is normalized to the interval $[-1.0, 1.0]$.

The score $S_S(e)$ is computed using standard cosine similarity of $\boldsymbol{e^s}$ and $\boldsymbol{q^s}$ in $\mathcal{V}_S$ (the same would hold for a content-based score if we decided to evaluate $\boldsymbol{q^c}$ not with the engine, but separately). We could also use cosine similarity to compute $S_P(e)$; however, experiments have shown that using the following score function consistently gives better results:

$$S_P(e) = \frac{\sum q_i^p \cdot e_i^p}{|\{i : q_i^p \cdot e_i^p \neq 0\}|}$$

In contrast to cosine similarity, this normalizes the score with the number of dimensions where both vectors are nonzero. The rationale behind this is that path fragments that do not occur in either the query (i.e., in elements with known relevance) or in the current element should not modify the score.

This scoring model can easily integrate new dimensions for feedback beyond content, path and document constraints, even if they use a completely different model (like a probabilistic model). It only requires that the relevance of an element to a new feedback dimension can be measured with a score between -1 and 1. It is simple to map typical score functions to this interval by normalization and transformation. As an example, the transformation rule for a probability p, $0 \leq p \leq 1$, is $2 \cdot p - 1$.

4 Architecture and Implementation

4.1 Architecture

Figure 1 shows the high-level architecture of our extensible feedback framework. Each feedback dimension is implemented with a standard interface that allows a simple integration of new dimensions. For each dimension, there are methods to compute constraints, select the most important constraints, and compute the score of an element with respect to these constraints.

The initial results of a query are presented to the user who gives positive or negative feedback to some of the results. This feedback is sent together with the query and its initial results to the feedback framework which forwards them to the available feedback dimensions. Each dimension computes a number of

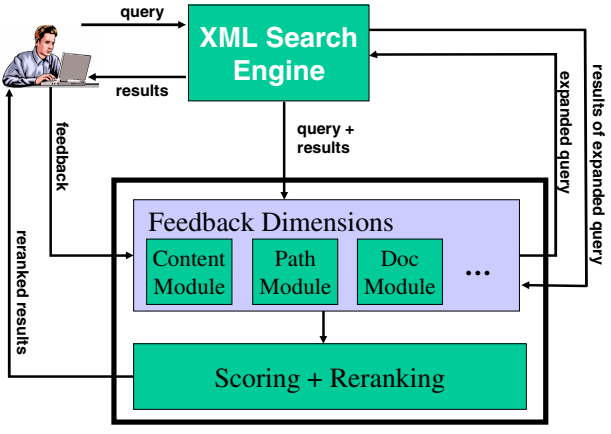

Fig. 1. Architecture of the feedback engine

constraints from the results with feedback. If the constraints for some dimensions are evaluated with the XML search engine, they are collected and transformed to an expanded query which is then evaluated with the engine; the results of this query are then the input for the remaining dimensions. For each element in the result of the expanded query, a score in the remaining dimensions is computed; the scores for each element are then aggregated, the list is sorted and returned to the user. The user may now again submit feedback for some of the new results, triggering another feedback cycle.

To facilitate an automatic assessment, the system can import queries and results from INEX (see Section 5.1) and automatically generate feedback for the top-k results, using the existing INEX assessments.

4.2 Implementation

We have implemented the framework in Java with content, path, and document constraints as examples for feedback dimensions and our TopX Search Engine [24]. For constraint generation we implemented the approaches presented in Section 2. Our implementation requires that important information about elements is precomputed:

- unique identifiers for the element (`eid`) and its document (`did`),
- its tag,
- its terms (after stemming and stopword removal), together with their score

This information is stored in a database table with schema `(did,eid,tag,term, score)` that contains one tuple for each distinct term of an element. We can reuse an existing inverted file of a keyword-based search engine that captures similar information, possibly after some transformation. On the database side, we provide indexes on `(did,tag,term,score)` (to efficiently collect all tag-term pairs of an element) and `(eid,term,score)` (to efficiently collect all distinct

terms of an element). Inverse element frequency of terms and inverse document frequency of tag-term pairs that are used for tie breaking are derived from the base table if they are not available from the engine already.

Our current prototype reuses the `TagTermFeatures` table of TopX (see [24] that already provides did, eid, tag, term, and score, together with information on structural relationships of elements (like pre and post order) that is not used here.

The implementations of content-based and document-based feedback first load the content of elements with known relevance or the tag-term pairs of the corresponding documents, respectively, and compute the best features with respect to the score functions and tie breaking methods presented in the previous section. If content feedback is evaluated with the engine, an expanded query is created; otherwise, the implementation computes the additional content score of all candidate elements (i.e., the elements in the initial result list provided by the engine) with a database query. Analogously, the implementation of document-based feedback either builds an expanded query or computes the scores for documents itself, again using the database.

Unlike the others, path feedback does not require any information from the database; features and scores are computed only from the paths of elements. As this can be done very quickly and the scoring model for path-based feedback cannot be easily mapped to an expanded query for the engine, we always evaluate path-based scores within the framework.

5 Experimental Results

5.1 Settings

We use the well-known *INEX* [13] benchmark for XML IR that provides a set of 12,107 XML documents (scientific articles from IEEE CS), a set of queries with and without structural constraints together with a manually assessed set of results for each query, and an evaluation environment to assess the effectiveness of XML search engines. INEX provides a Relevance Feedback Track [12, 6] that aims at assessing the quality of different feedback approaches. As this paper concentrates on keyword-based queries (*content-only topics* or *CO* for short in INEX), we used the set of 52 CO queries from the 2003 and 2004 evaluation rounds with relevant results together with the *strict* quantization mode, i.e., an element was considered as relevant if it exactly answers the query. A *run* is the result of the evaluation of all topics with a search engine; it consists of 1500 results for each topic that are ranked by expected relevance. The measure of effectiveness is the *mean average precision* (MAP) of a run. Here, we first compute for each topic the average precision over 100 recall points (0.01 to 1.00) and then take the macro average over these topic-wise averages. Note that absolute MAP values are quite low for INEX (with 0.152 being the best MAP value of any participating engine in 2004). In addition to MAP values, we also measured precision at 10 for each run.

Table 1. Precision at k for the baseline TopX run

k	1	3	5	8	10	13	15	18	20
prec@k	0.269	0.237	0.227	0.204	0.190	0.176	0.168	0.155	0.152

To assess the quality of feedback algorithms, we use the residual collection technique [22] that is also used in the INEX Relevance Feedback Track. In this technique, all XML elements that are used by the feedback algorithm, i.e., those whose relevance is known to the algorithm, must be removed from the collection before evaluation of the results with feedback takes place. This includes all k elements "seen" or used in the feedback process regardless of their relevance. Under INEX guidelines, this means not only each element used or observed in the relevance feedback process but also all descendants of that element must be removed from the collection (i.e., the residual collection, against which the feedback query is evaluated, must contain no descendant of that element). All ancestors of that element are retained in the residual collection.

For all experiments we used our TopX Search Engine [24] that fully supports the evaluation of content-and-structure queries with weighted content constraints. The baseline for all experiments is a TopX run for all 52 INEX topics, with 1500 results for each topic. Table 1 shows the macro-averaged precision for this run for the top-k ranked elements per topic, for different k.

The experiments were run on a Windows-based server with two Intel Xeon processors@3GHz, 4 gigabytes of RAM and a four-way RAID-0 SCSI disk array, running an Oracle 9i database on the same machine as the feedback framework.

5.2 Choice of Parameters

Choice of Path Constraints. We conducted experiments to assess the influence of the different classes of path constraints on the result quality. To do this, we examine how the MAP value and the precision at 10 of the baseline run change if we rerank the results based on feedback on the top-20 results with different combinations of path constraints enabled.

Table 2 shows the results of our experiments. It is evident that the best results are yielded with a combination of P_1,P_3, and P_4 (i.e., prefixes of paths, subpaths of length 2, and paths with wildcards), while adding infixes (P_2) is slightly worse.

Table 2. MAP and precision@10 values for some combinations of path fragments with the baseline TopX run

classes	**MAP**	**prec@10**
none	0.0214	0.0706
$P_1 - P_6$	0.0231	0.0745
$P_1 - P_4$	0.0276	0.0824
P_1,P_3,P_4	**0.0281**	**0.0843**
P_5	0.0154	0.0569
P_6	0.0157	0.0588

The absolute gain in MAP of about 0.0067 on average is small, but significant, as it corresponds to a relative gain of 31.3%. Using tag names alone (P_5) does not help and even hurts effectiveness (which was already shown in [16]), as does using the complete path (P_6). A similar influence of the different types of path constraints can be seen for the precision values.

Number of content and document constraints. We made some initial experiments to determine how many content and document constraints should be selected for expanding queries. Varying the number of content and/or document constraints from 1 to 10, we found that the MAP value didn't change a lot as soon as we had at least 3 content and 3 document constraints, even though results got slightly better with more constraints. For our experiments we choose the top 5 content and/or top 5 document constraints.

Tuning parameters. To calibrate the different tuning parameters introduced in Section 2, we made some initial experiments with varying parameter values. The best results were yielded with $\beta_c = 0.5$ and $\gamma_c = 0.25$ for content feedback, $\beta_p = 1.0$ and $\gamma_p = 0.25$ for path feedback, and $\beta_s = \gamma_s = 1.0$ for structural feedback. We used these values for our main series of experiments presented in the following subsections.

5.3 Engine-Based Feedback with TopX

We assessed the quality of document-based feedback with and without additional content-based feedback with our baseline run as input. Using relevance information for the top-k results for each topic from this run (for varying k), we computed the best five content-based and/or the best five document-based features features for each topic, created the corresponding expanded queries and evaluated them with TopX again.

Figure 2 shows the effect of content- and document-based feedback with the baseline `TopX` run, with known relevance of a varying number of elements from the top of the run. Our baseline is the TopX run on the corresponding residual collection (light dashed line in Figure 2),i.e., the run where the elements with known relevance and their descendants are virtually removed from the collection; its MAP value is not constant because the residual collection get smaller when elements are removed. Our implementation of content feedback (dark solid line) yields an absolute gain of about 0.02 in MAP which corresponds to roughly 70% relative gain when the relevance of at least the top-8 elements is known. Adding document-based constraints (dark dashed line) yields similar gains, but already with feedback for the top-ranked element. The combination of both approaches (light solid line) is best, yielding an absolute gain of up to 0.03 (corresponding to a relative gain of more than 100% for top-20 feedback).

The improvements for precision at 10 are less spectacular with document-based feedback (see Figure 3), especially compared to content-based feedback. On the other hand, this behaviour is not surprising as adding document-based constraints to a query gives a higher score to all elements in a matching document, regardless of their relevance to the query; hence document-based feedback

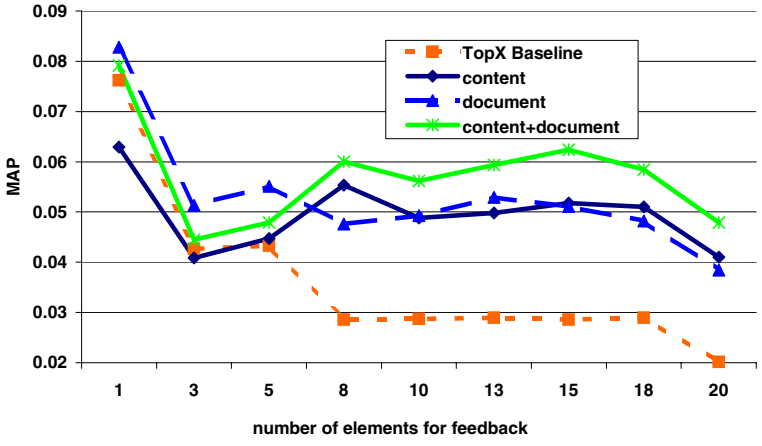

Fig. 2. MAP for document-based feedback on TopX

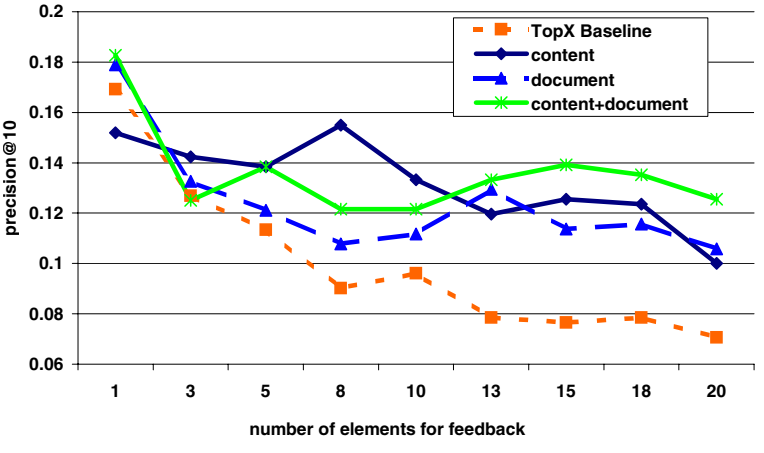

Fig. 3. Precision@10 for document-based feedback on TopX

is better in improving recall than precision. The combination of content-based and document-based feedback again beats the single dimensions, giving high scores to elements with good content in good documents.

Figure 4 shows the effect of additional path-based feedback in combination with document- and content-based feedback. The effect of path-based feedback alone (dark solid line in Figure 2) is mixed: While it slightly increases MAP for a medium number of elements with feedback, it downgrades results with feedback for a few or many elements. A similar effect occurs for the combination of path-based feedback with either content-based or document-based feedback; we omitted the lines from the chart for readability. However, combined with both content- and document-based feedback (dark dashed line), path constraints do result in a small gain in MAP.

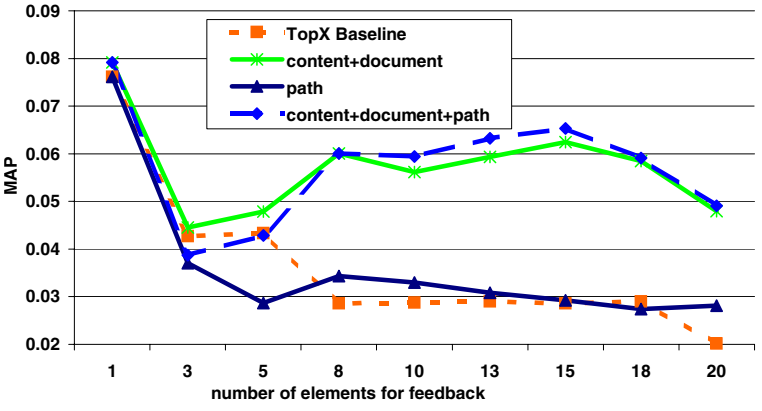

Fig. 4. MAP values for path-based feedback with TopX

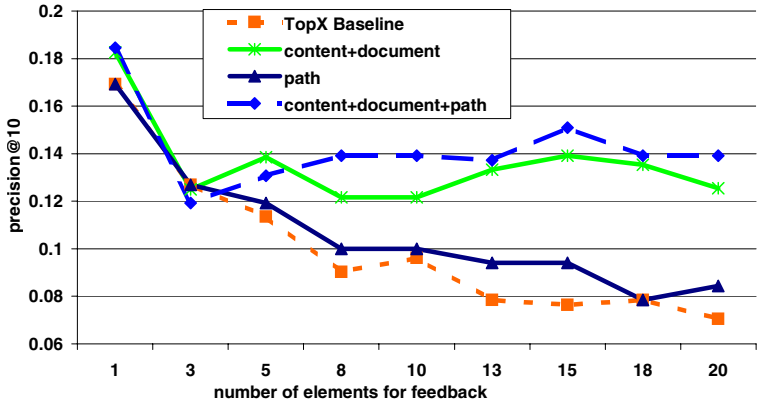

Fig. 5. Precision@10 values for path-based feedback with TopX

Regarding precision at 10 (shown in Figure 5), we find that path-based feedback alone always slightly increases precision over the baseline. The same holds for path feedback in combination with either document or content feedback (we omitted the lines in the chart for the sake of readability); the combination of all three dimensions of feedback yields the best precision if the relevance of at least the top-8 elements is known. The peak improvement (for top-20 feedback with all three dimensions) is about 100% over the baseline without feedback.

5.4 Hybrid Evaluation

To show that hybrid evaluation is a viable way of integrating structural feedback with keyword-based engines, we evaluated the effectiveness of content, path and document constraints with our hybrid evaluator and the TopX engine, using TopX only to compute the baseline run.

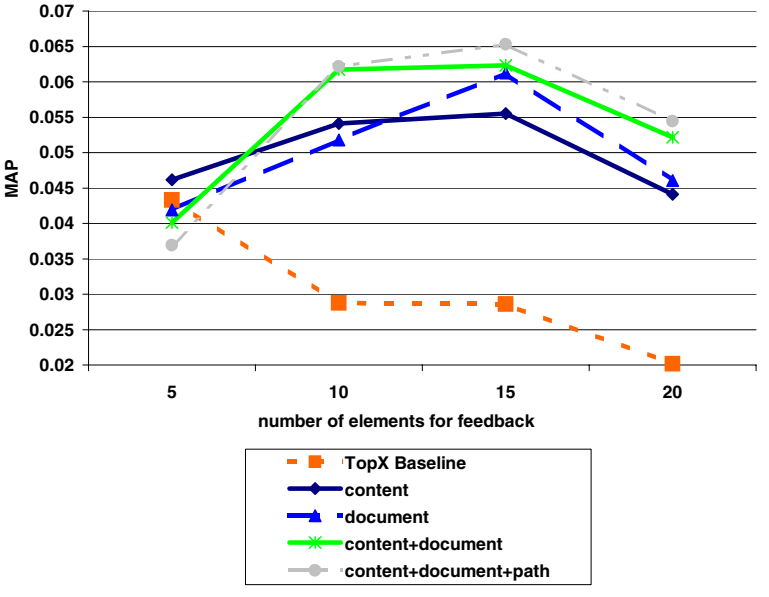

Fig. 6. MAP values for hybrid evaluation

Figure 6 shows the resulting MAP values for important combinations of feedback dimensions and the TopX baseline without feedback, measured with different numbers of elements with known relevance. The general trends are the same as with engine-based evaluation, which shows that our approach to evaluate feedback separately from the engine is viable. Interestingly, content feedback is slightly better than document-based feedback for small numbers of elements with known relevance, and slightly worse for large numbers; with engine-based evaluation we observed the inverse orders. Path based feedback in combination with a single, second dimension – which is again omitted from the chart – slightly outperformed the same dimension without path-based feedback. The combination of all feedback dimensions again outperforms all the others. The absolute MAP values are often slightly higher with hybrid evaluation; we attribute this to the fact that scores for each dimensions are normalized to the interval $[-1; 1]$ before aggregating them. The same effect could be obtained with engine-based evaluation if weights are selected more carefully; we plan to study this in future work. We also measured precision at 10 and got similar results that are omitted from the paper.

Even though execution times are not the focus of this paper, we measured the time needed for computing the best features for query expansion and reranking the list of 1500 results. On average, this took not more than 20 seconds per query with our preliminary, database-backed implementation, all three feedback dimensions and feedback for the top-20 results; we expect that this time will decrease a lot with a more performance-oriented implementation.

6 Conclusion and Future Work

This paper has made important steps from content-based to structural feedback in XML retrieval. It presented an integrated solution for expanding keyword queries with new content, path, and document constraints as a part of an extensible framework and showed huge performance gains with the established INEX benchmark of up to 150% for MAP and up to 100% for precision@10 under the evaluation method used in the INEX relevance feedback track.

Our future work will contentrate on adding new classes of document constraints (like paths, twigs, and the existence of elements or paths in a document) and integrating this work with our previous work on ontological query expansion [18]. We will also consider pseudo relevance feedback with paths and document constraints. We plan to evaluate our approach with the INEX Relevance Feedback Track where we will also examine the effect of feedback on queries with content and structural constraints.

References

1. G. Amati, C. Carpineto, and G. Romano. Merging XML indices. In *INEX Workshop 2004*, pages 77–81, 2004. available from `http://inex.is.informatik.uni-duisburg.de:2004/`.
2. R. A. Baeza-Yates and B. Riberto-Neto, editors. *Modern Information Retrieval*. Addison Wesley, 1999.
3. A. Balmin et al. A system for keyword proximity search on XML databases. In *VLDB 2003*, pages 1069–1072, 2003.
4. H. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors. *Intelligent Search on XML Data*, volume 2818 of *LNCS*. Springer, Sept. 2003.
5. D. Carmel et al. Searching XML documents via XML fragments. In *SIGIR 2003*, pages 151–158, 2003.
6. C. Crouch. Relevance feedback at the INEX 2004 workshop. *SIGIR Forum*, 39(1):41–42, 2005.
7. C. J. Crouch, A. Mahajan, and A. Bellamkonda. Flexible XML retrieval based on the extended vector model. In *INEX 2004 Workshop*, pages 149–153, 2004.
8. M. A. Gonçalves, E. A. Fox, A. Krowne, P. Calado, A. H. F. Laender, A. S. da Silva, and B. Ribeiro-Neto. The effectiveness of automatically structured queries in digital libraries. In *4th ACM/IEEE-CS joint conference on Digital libraries (JCDL04)*, pages 98–107, 2004.
9. L. Guo et al. XRANK: ranked keyword search over XML documents. In *SIGMOD 2003*, pages 16–27, 2003.
10. L. Hlaoua and M. Boughanem. Towards context and structural relevance feedback in XML retrieval. In *Workshop on Open Source Web Information Retrieval (OSWIR)*, 2005. `http://www.emse.fr/OSWIR05/`.
11. W. Hsu, M. L. Lee, and X. Wu. Path-augmented keyword search for XML documents. In *ICTAI 2004*, pages 526–530, 2004.
12. INEX relevance feedback track. `http://inex.is.informatik.uni-duisburg.de:2004/tracks/rel/`
13. G. Kazai et al. The INEX evaluation initiative. In Blanken et al. [4], pages 279–293.

14. S. Liu, Q. Zou, and W. Chu. Configurable indexing and ranking for XML information retrieval. In *SIGIR 2004*, pages 88–95, 2004.
15. Y. Mass and M. Mandelbrod. Relevance feedback for XML retrieval. In *INEX 2004 Workshop*, pages 154–157, 2004.
16. V. Mihajlovič et al. TIJAH at INEX 2004 modeling phrases and relevance feedback. In *INEX 2004 Workshop*, pages 141–148, 2004.
17. H. Pan. Relevance feedback in XML retrieval. In *EDBT 2004 Workshops*, pages 187–196, 2004.
18. H. Pan, A. Theobald, and R. Schenkel. Query refinement by relevance feedback in an XML retrieval system. In *ER 2004*, pages 854–855, 2004.
19. G. Ramirez, T. Westerveld, and A. de Vries. Structural features in content oriented xml retrieval. Technical Report INS-E0508, CWI,Centre for Mathematics and Computer Science, 2005.
20. G. Ramírez, T. Westerveld, and A. P. de Vries. Structural features in content oriented XML retrieval. In *CIKM 2005*, 2005.
21. J. Rocchio Jr. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, pages 313–323. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.
22. I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review*, 18(1), 2003.
23. B. Sigurbjörnsson, J. Kamps, and M. de Rijke. The University of Amsterdam at INEX 2004. In *INEX 2004 Workshop*, pages 104–109, 2004.
24. M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In *VLDB 2005*, pages 625–636, 2005.
25. A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). available at `http://www.cs.otago.ac.nz/postgrads/andrew/2004-4.pdf`, 2004.
26. J.-N. Vittaut, B. Piwowarski, and P. Gallinari. An algebra for structured queries in bayesian networks. In *INEX Workshop 2004*, pages 58–64, 2004.
27. R. Weber. Using relevance feedback in XML retrieval. In Blanken et al. [4], pages 133–143.
28. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD 2005*, pages 537–538, 2005.