TopX – AdHoc Track and Feedback Task

Martin Theobald, Andreas Broschart, Ralf Schenkel, Silvana Solomon,
and Gerhard Weikum

Max-Planck-Institut für Informatik
Saarbrücken, Germany
{mtb,abrosch,schenkel,solomon,weikum}@mpi-inf.mpg.de
http://www.mpi-inf.mpg.de/departments/d5/

Abstract. This paper describes the setup and results of the Max-Planck-Institut für Informatik's contributions for the INEX 2006 AdHoc Track and Feedback task. The runs were produced with the TopX system, which is a top-k retrieval engine for text and XML data that uses a combination of BM25-based content and structural scores.

1 System Overview

TopX [10,11] aims to bridge the fields of database systems (DB) and information retrieval (IR). From a DB viewpoint, it provides an efficient algorithmic basis for top-k query processing over multidimensional datasets, ranging from structured data such as product catalogs (e.g., bookstores, real estate, movies, etc.) to unstructured text documents (with keywords or stemmed terms defining the feature space) and semistructured XML data in between. From an IR viewpoint, TopX provides ranked retrieval based on a relevance scoring function, with support for flexible combinations of mandatory and optional conditions as well as text predicates such as phrases, negations, etc. TopX combines these two aspects into a unified framework and software system, with emphasis on XML ranked retrieval.

Figure 1 depicts the main components of the TopX system. It supports three kinds of front-ends: as a servlet with an HTML end-user interface (that was used for the topic development of INEX 2006), as a Web Service with a SOAP interface (that was used by the Interactive track), and as a Java API (that was used to generate our runs). TopX currently uses Oracle10g as a storage system, but the JDBC interface would easily allow other relational backends, too.

The *Indexer* parses and analyzes the document collection and builds the index structures for efficient lookups of tags, content terms, phrases, structural patterns, etc. An *Ontology* component manages optional ontologies with various kinds of semantic relationships among concepts and statistical weighting of relationship strengths; we used WordNet [2] for some of our runs.

At query run-time, the *Core Query Processor* decomposes queries and invokes the top-k algorithms. It maintains intermediate top-k results and candidate items in a priority queue, and it schedules accesses on the precomputed index lists in a multi-threaded architecture. Several advanced components provide means for run-time acceleration:

N. Fuhr, M. Lalmas, and A. Trotman (Eds.): INEX 2006, LNCS 4518, pp. 233–242, 2007.

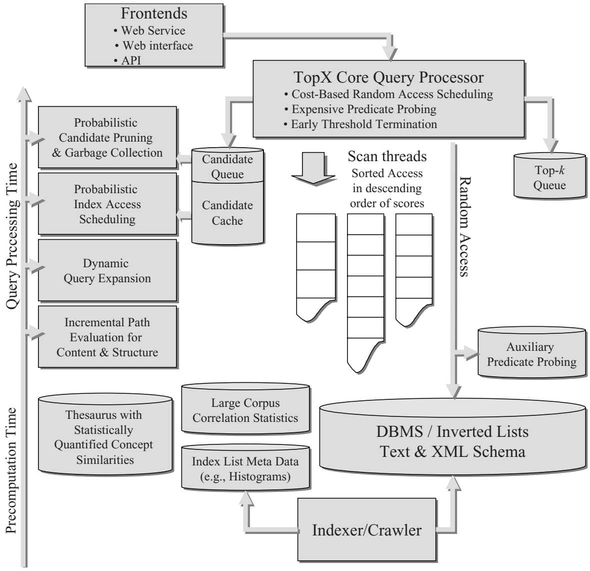
Frontends
• Web Service
• Web interface
• API
TopX Core Query Processor
• Cost-Based Random Access Scheduling
• Expensive Predicate Probing
• Early Threshold Termination
Query Processing Time
Probabilistic Candidate Pruning & Garbage Collection
Probabilistic Index Access Scheduling
Dynamic Query Expansion
Incremental Path Evaluation for Content & Structure
Candidate Queue
Candidate Cache
Scan threads
Sorted Access in descending order of scores
Random Access
Top-k Queue
Auxiliary Predicate Probing
Precomputation Time
Thesaurus with Statistically Quantified Concept Similarities
Large Corpus Correlation Statistics
Index List Meta Data (e.g., Histograms)
DBMS / Inverted Lists Text & XML Schema
Indexer/Crawler

Fig. 1. TopX architecture

- The *Probabilistic Candidate Pruning* component [12] allows TopX to drop candidates that are unlikely to qualify for the top-k results at an early stage, with a controllable loss and probabilistic result guarantees.
- The *Index Access Scheduler* [1] provides a suite of scheduling strategies for sorted and random accesses to index entries.
- The *Incremental Path Evaluation* uses additional cost models to decide when to evaluate structural conditions like XML path conditions, based on specialized indexes for XML structure.
- The *Dynamic Query Expansion* component [9] maps the query keywords and/or tags to concepts in the available ontology and incrementally generates query expansion candidates.

2 Data Model and Scoring

We refer the reader to [11] for a thorough discussion of the scoring model. This section shortly reviews important concepts.

2.1 Data Model

We consider a simplified XML data model, where idref/XLink/XPointer links are disregarded. Thus every document forms a tree of nodes, each with a *tag* and a related *content*. We treat attributes nodes as children of the corresponding element node. The content of a node is either a text string or it is empty;

typically (but not necessarily) non-leaf nodes have empty content. With each node, we associate its *full-content* which is defined as the concatenation of the text contents of all the node's descendants in document order.

2.2 Content Scores

For content scores we make use of element-specific statistics that view the full-content of each element as a bag of words:

1) the *full-content term frequency*, $ftf(t, n)$, of term t in node n, which is the number of occurrences of t in the full-content of n;
2) the *tag frequency*, N_A, of tag A, which is the number of nodes with tag A in the entire corpus;
3) the *element frequency*, $ef_A(t)$, of term t with regard to tag A, which is the number of nodes with tag A that contain t in their full-contents in the entire corpus.

The score of an element e with tag A with respect to a content condition of the form `*[about(., t)]` is then computed by the following BM25-inspired formula:

$$score(e, \texttt{*[about(., t)]}) = \frac{(k_1+1)\ ftf(t,e)}{K + ftf(t,n)} \cdot \log\left(\frac{N_A - ef_A(t) + 0.5}{ef_A(t) + 0.5}\right) \quad (1)$$

$$\text{with } K = k_1\left((1-b) + b\frac{\sum_{s \in \ full\ content\ of\ e} ftf(s,e)}{avg\{\sum_{s'} ftf(s', e') \mid e'\ with\ tag\ A\}}\right)$$

For a query content condition with multiple terms, the score of an element satisfying the tag constraint is computed as the sum of the element's content scores for the corresponding content conditions, i.e.:

$$score(e, \texttt{*[about(.,}\ t_1 \dots t_m\texttt{)]}) = \sum_{i=1}^{m} score(e, \texttt{*[about(.,}\ t_i\texttt{)]}) \quad (2)$$

TopX provides the option to evaluate queries either in conjunctive mode or in "andish" mode. In the first case, all terms (and, for content-and-structure queries, all structural conditions) must be met by a result candidate, but still different matches yield different scores. In the second case, a node is already considered a match if it satisfies at least one content condition.

Orthogonally to this, TopX can be configured to return two different granularities as results: in *document mode*, TopX returns the best documents for a query, whereas in *element mode*, the best target elements are returned, which may include several elements from the same document.

2.3 Structural Scores

Given a query with structural and content conditions, we transitively expand all structural query dependencies. For example, in the query `//A//B//C[about(., t)]` an element with tag `C` has to be a descendant of both `A` and `B` elements. Branching path expressions can be expressed analogously. This process yields a *directed acyclic graph* (DAG) with tag-term conditions as leaves, tag conditions as inner nodes, and all transitively expanded descendant relations as edges.

Our structural scoring model essentially counts the number of navigational (i.e., tag-only) conditions that are satisfied by a result candidate and assigns a small and constant score mass c for every condition that is matched. This structural score mass is combined with the content scores. In our setup we have set $c = 1$, whereas content scores are normalized to $[0, 1]$, i.e., we emphasize the structural parts.

3 AdHoc Track Results

There were two major changes in this year's AdHoc Track: The queries were run against the Wikipedia collection instead of the old IEEE collection, and there was only a single dimension of relevance (i.e., specificity) instead of both exhaustivity and specifitiy. As a consequence of this, smaller elements should be favored over larger elements (e.g., complete articles) at least for the Thorough subtask. Our scoring functions do not take this into account as they are still tuned towards the old bidimensional relevance (with exhaustivity and specificity).

For each subtask, we submitted at least the following four types of runs:

- `CO_{subtask}_baseline`: a CO run that considered the terms in the title of a topic without phrases and negations, limiting tags of results to `article`, `section`, and `p`.
- `CO_{subtask}_exp`: a CO run that considered terms as well as phrases and negations (so-called *expensive predicates*), again limiting tags of results to `article`, `section`, and `p`.
- `CAS_{subtask}_baseline`: a CAS run that considered the castitle of a topic if it was available, and the title otherwise. The target tag was evaluated strictly, whereas support conditions were optional; phrases and negations were ignored.
- `CAS_{subtask}_exp`: a CAS run that additionally considered phrases and negations.

3.1 Thorough Task

We submitted six runs to the Thorough task. In addition to our four standard runs, we submitted

- `TOPX_CO_Thorough_all`: a CO run that allowed all tags in the collection instead of limiting the tags to `article`, `section`, and `p`

- TOPX_CAS_Thorough_ex_incr: a CAS run that included expanding terms using WordNet

Table 1 shows the results for our runs. It turns out that all CO runs outperform the CAS runs that suffer from the strict evaluation of the target tag. Among the CO runs, the run that allows all result tags is best; this is not surprising as the other runs exclude many relevant results that have the 'wrong' tag. We see a slight advantage for runs that include phrases and negations, and a slight disadvantage for the run that expanded terms with WordNet. Overall, the performance of TopX is good (with a peak rank of 20), taking into account the limited amount of tuning that we did. Being a top-k engine, we expect that TopX would, like last year, perform even better for early cutoff points; however, they were unfortunately not measured this year.

Table 1. Results for the Thorough Task

run	**rank**	**MAep**
TOPX_CO_Thorough_all	20	0.0253
TOPX_CO_Thorough_ex	26	0.0190
TOPX_CO_Thorough_baseline	32	0.0178
TOPX_CAS_Thorough_ex	61	0.0103
TOPX_CAS_Thorough_baseline	62	0.0101
TOPX_CAS_Thorough_ex_incr	75	0.0081

3.2 Focused Task

Our runs for the focused task were produced by postprocessing our AdHoc runs to remove any overlap. For each such AdHoc run, we kept an element e if there was no other element e' from the same document in the run that had a higher score than e and had a path that overlapped with e's path. This simple, syntactic postprocessing yielded good results (shown in Table 2). Especially for the early cutoff points, TopX performed extremely well with peak ranks 3 and 4. Interestingly, the CO run that considered phrases and negation did slightly better than its counterpart without expensive predicates.

3.3 BestInContext Task

To produce the runs for the BestInContext task, we ran TopX in document mode. This yielded a list of documents ordered by the highest score of any element within the document, together with a list of elements and their scores for each document. To compute the best entry point for a document, we simply selected the element with highest score from each document and ordered them by score. The results (Tables 3 and 4) show that this gave good results, with a peak rank of 1.

Table 2. Results for the Focused Task with the nxCG metric at different cutoffs (ranks are in parentheses), with overlap=on

run	nxCG[5]	nxCG[10]	nxCG[25]	nxCG[50]
TOPX_CO_Focused_ex	0.3769 (3)	0.3154 (4)	0.2431 (10)	0.1916 (14)
TOPX_CO_Focused_baseline	0.3723 (4)	0.3051 (10)	0.2432 (9)	0.1913 (15)
TOPX_CAS_Focused_baseline	0.3397 (16)	0.2792 (21)	0.2017 (31)	0.1524 (39)
TOPX_CAS_Focused_ex	0.3339 (20)	0.2790 (22)	0.1985 (33)	0.1501 (41)
TOPX_CAS_Focused_ex_incr	0.2909 (40)	0.2341 (50)	0.1640 (59)	0.1232 (59)

Table 3. Results for the BestInContext Task with the BEPD metric (ranks are in parentheses)

run	A=0.1	A=1	A=10	A=100
TOPX-CO-BestInContext-baseline	0.1280 (22)	0.2237 (11)	0.3685 (5)	0.5715 (5)
TOPX-CO-BestInContext-exp	0.1189 (28)	0.2074 (20)	0.3451 (11)	0.5384 (11)
TOPX-CAS-BestInContext-baseline	0.0718 (54)	0.1361 (53)	0.2272 (53)	0.3780 (53)
TOPX-CAS-BestInContext-exp	0.0653 (57)	0.1254 (56)	0.2131 (57)	0.3594 (54)

Table 4. Results for the BestInContext Task with the EPRUM-BEP-Exh-BEPDistance metric (ranks are in parentheses)

run	A=0.1	A=1	A=10	A=100
TOPX-CO-BestInContext-exp	0.0260 (13)	0.0604 (7)	0.1241 (3)	0.2081 (1)
TOPX-CO-BestInContext-baseline	0.0258 (17)	0.0607 (5)	0.1231 (4)	0.2050 (3)
TOPX-CAS-BestInContext-exp	0.0163 (42)	0.0394 (38)	0.0764 (26)	0.1422 (29)
TOPX-CAS-BestInContext-baseline	0.0160 (44)	0.0388 (40)	0.0748 (33)	0.1380 (32)

4 Structural Query Expansion

Our feedback framework aims at generating a content-and-structure query from a keyword query, exploiting relevance feedback provided by a user for some results of the keyword query. This section gives a very brief summary of our approach; for a more detailed and formal description, see [8].

We consider the following classes of candidates for query expansion from an element with known relevance:

- all terms of the element's content (C candidates),
- all tag-term pairs of descendants of the element in its document (D candidates),
- all tag-term pairs of ancestors of the element in its document (A candidates), and
- all tag-term pairs of descendants of ancestors of the element in its document, together with the ancestor's tag (AD candidates).

To weight the different candidates c, we apply an extension of the well-known Robertson-Sparck-Jones weight [5] to element-level retrieval in XML, applying it to elements instead of documents:

$$w_{RSJ}(c) = \log \frac{r_c + 0.5}{R - r_c + 0.5} + \log \frac{E - ef_c - R + r_c + 0.5}{ef_c - r_c + 0.5}$$

Here, for a candidate c, r_c denotes the number of relevant elements which contain the candidate c in their candidate set, R denotes the number of relevant elements, E the number of elements in the collection, and ef_c the element frequency of the candidate.

To select the candidates to expand the query, we use the Robertson Selection Values (RSV) proposed by Robertson [4]. For a candidate c, its RSV has the form $RSV(c) = w_{RSJ}(c) \cdot (p - q)$, where $p = r_c/R$ is the estimated probability of the candidate occurring in a relevant element's candidate set and q is the probability that it occurs in a nonrelevant element's set. We ignore candidates that occur only within the documents of elements with known relevance as they have no potential to generate more relevant results outside these documents, and we ignore candidates that contain a query term. We choose the top b of the remaining candidates for query expansion (b is a configurable parameter).

Using these top-b candidates, we generate a content-and-structure query from the original keyword query, where each additional constraint is weighted with the normalized RSJ weight of its corresponding candidate (see [8]). The expansion itself is actually rather straightforward; the generated query has the following general structure:

```
//ancestor-tag[A+AD constraints]//*[keywords+C+D constraints]
```

As an example, if the original query was "XML" and we selected

- the A candidate `//ancestor::article[about(., IR)]`,
- the AD candidate `//ancestor::article[about(.//bib, index)]`,
- the D candidate `//descendant:p[about(., index)]`, and
- the C candidate `about(., database)`,

the expanded query (omitting the weights) would be

```
//article[about(., IR) and about(.//bib, index)]//*[about(., XML)
and about (., database) and about(.//p, index)].
```

5 Feedback Task Results

INEX 2006 introduced a new relevance measure, specificity, that replaced the two dimensions of relevance, exhaustivity and specificity, used before. This happened mainly for two reasons: First, to make assessments easier, and second, because correlation analyses had shown that comparing systems in the AdHoc track yields a result when using specificity only that is sufficiently similar to the result with specificity and exhaustivity.

However, this new measure does not reflect the relevance of an element from a user's point of view. It is unlikely that a user would greatly appreciate seeing a single `collectionlink` element or, even worse, an isolated `xlink:href` attribute in a result list. It is therefore questionable if specificity alone can be used for automated feedback.

5.1 Evaluation of Feedback Runs

We discussed different evaluation modes in our paper at last INEX [7]. There is still no common agreement on one mode that should give the 'best' results. We shortly review the modes here and introduce a new mode, *resColl-path*.

- Simply comparing the results of the baseline run with the results generated from feedback (we denote this as *plain*) is commonly considered as illegal, as feedback includes the advantage of knowing some relevant results and hence can yield a better performance.
- With rank freezing, the rank of results with known relevance is frozen, thus assessing only the effect of reranking the results with unknown relevance. We label this approach *freezeTop* as usually the top-k results are used for feedback and hence frozen. This has been the standard evaluation mode for the INEX relevance feedback task.
- With the residual collection technique, all XML elements with known relevance must be removed from the collection before evaluation of the results with feedback takes place. Depending on which elements are considered as having known relevance, a variety of different evaluation techniques results:
  - *resColl-result*: only the elements for which feedback is given are removed from the collection,
  - *resColl-desc*: the elements for which feedback is given and all their descendants are removed from the collection,
  - *resColl-anc*: the elements for which feedback is given and all their ancestors are removed from the collection,
  - *resColl-doc*: for each element for which feedback is given, the whole document is removed from the collection, and
  - *resColl-path*: for each element for which feedback is given, the element itself, its ancestors and its descendants are removed from the collection.

The most natural evaluation mode is *resColl-path*, as it removes all elements for which the feedback algorithm has some knowledge about their potential relevance. We evaluate our approach with all seven evaluation techniques in the following section and try to find out if there are any differences.

5.2 Official Results

Due to time constraints, we submitted only a single official run where we used only content-based feedback from the top-20 elements of our baseline run

TOPX_CO_Thorough_all. In the official evaluation, this setting performed well, with an absolute improvement of 0.0064 in MAep over the baseline run for the *Generalised* quantisation, which corresponds to a relative improvement of about 25%. Both the t-test and the Wilcoxon signed-rank test show that the improvement is significant (with $p \approx 0$).

5.3 Additional Results

Besides our official run that used only content-based feedback, we tried different combinations of candidate classes. We measured only MAP and precision at different cutoffs. Due to time constraints, we consider only the first 49 topics that have assessments (topics 289-339, excluding topics 299 and 307), runs with 100 elements, and feedback for the top-20 results of our baseline run TOPX_CO_Thorough_all with the *Generalised* quantization. Our experiments use the top-10 candidates for feedback. We tested the significance of our results with the t-test and the Wilcoxon signed-rank test [6].

Table 5. MAP values for different configurations and different evaluation modes. Runs shown in **bold** are significantly better than the baseline under the WSR test ($p < 0.01$), runs shown in *italics* are significantly better than the baseline under the t-test ($p < 0.01$).

evaluation	baseline	C	D	C+D	A	AD	A+AD
plain	0.0188	***0.0364***	***0.0328***	***0.0344***	0.0187	0.0164	***0.0228***
freezeTop	0.0188	***0.0284***	***0.0248***	***0.0256***	**0.0189**	0.0181	***0.0216***
resColl-result	0.0108	***0.0264***	***0.0212***	***0.0218***	0.0106	0.0106	**0.0171**
resColl-anc	0.0101	***0.0246***	***0.0194***	***0.0201***	0.0102	0.0102	**0.0169**
resColl-desc	0.0049	***0.0087***	0.0078	***0.0081***	0.0048	0.0045	0.0056
resColl-doc	0.0041	***0.0077***	**0.0067**	***0.0072***	0.0040	0.0038	0.0048
resColl-path	0.0046	***0.0085***	0.0072	***0.0079***	0.0044	0.0043	0.0056

Table 5 shows the results for our additional runs. Unlike our results from last year with the IEEE collection, content-only feedback outperformed all other combinations, and A and AD candidates alone often could not improve result quality significantly. At this time, we do not have a well-founded explanation for this behaviour. However, there are some major differences of the new Wikipedia collection to the old IEEE collection:

- Wikipedia documents do not have a clear structure with front and back matter. For the old IEEE collection, especially A and AD candidates could exploit things like authors of a document, authors of a cited document, or journal names.
- The unidimensional relevance measure penalizes large elements towards the root of a document. This is a natural disadvantage for using D candidates that tend to add results near the root element.

– Our old experiments used only the *Strict* quantization where the best elements typically were sections or paragraphs. With the new relevance measure and quantization, the best elements and attributes are small (like `collectionlink` or `xlink:href`) which do not contribute many candidates to the candidate pool.

Our future work in this area will focus on using other measures of relevance like the one proposed for HiXEval [3]. This may additionally pave the way for feedback that exploits the granularity of results (e.g., to derive tags for a keyword-only query). We will additionally examine how to choose a threshold for the element frequency of candidates that are considered, and which other candidate classes could be used.

References

1. Bast, H., Majumdar, D., Theobald, M., Schenkel, R., Weikum, G.: IO-Top-k: Index-optimized top-k query processing. In: Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006), pp. 475–486 (2006)
2. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
3. Pehcevski, J., Thom, J.A.: HiXEval: Highlighting XML retrieval evaluation. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 43–57. Springer, Heidelberg (2006)
4. Robertson, S.E.: On term selection for query expansion. J. Doc. 46, 359–364 (1990)
5. Robertson, S.E., Sparck-Jones, K.: Relevance weighting of search terms. J. Am. Soc. Inform. Sci. 27, 129–146 (1976)
6. Savoy, J.: Statistical inference in retrieval effectiveness evaluation. Inform. Process. Manag. 33(4), 495–512 (1997)
7. Schenkel, R., Theobald, M.: Relevance feedback for structural query expansion. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 344–357. Springer, Heidelberg (2006)
8. Schenkel, R., Theobald, M.: Structural feedback for keyword-based XML retrieval. In: Proceedings of the 28th European Conference on IR Research (ECIR 2006), pp. 326–337 (2006)
9. Theobald, M., Schenkel, R., Weikum, G.: Efficient and self-tuning incremental query expansion for top-k query processing. In: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 242–249 (2005)
10. Theobald, M., Schenkel, R., Weikum, G.: An efficient and versatile query engine for TopX search. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005), pp. 625–636 (2005)
11. Theobald, M., Schenkel, R., Weikum, G.: TopX & XXL INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 282–295. Springer, Heidelberg (2006)
12. Theobald, M., Weikum, G., Schenkel, R.: Top-k query evaluation with probabilistic guarantees. In: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004), pp. 648–659 (2004)