The TopX DB&IR Engine

Martin Theobald
Stanford University
theobald@stanford.edu

Ralf Schenkel
MPI Informatik
schenkel@mpi-inf.mpg.de

Gerhard Weikum
MPI Informatik
weikum@mpi-inf.mpg.de

ABSTRACT

This paper proposes a demo of the TopX search engine, an extensive framework for unified indexing, querying, and ranking of large collections of unstructured, semistructured, and structured data. TopX integrates efficient algorithms for top-k-style ranked retrieval with powerful scoring models for text and XML documents, as well as dynamic and self-tuning query expansion based on background ontologies.

Categories and Subject Descriptors: H.2.4 [Systems]: Textual Databases, H.3 [Information Storage and Retrieval]: General

General Terms: Performance

Keywords: retrieval, top-k, XML, text

1. INTRODUCTION

Non-schematic XML data that comes from many different sources and inevitably exhibits heterogeneous structures and annotations (i.e., XML tags) cannot be adequately searched using database query languages like XPath or XQuery. Often, queries either return too many or too few results. Rather the ranked-retrieval paradigm is called for, with relaxable search conditions, various forms of similarity predicates on tags and contents, and quantitative relevance scoring. These considerations also hold for applications that deal with structured data but face uncertainty in the data, for example, because the data is probabilistic or the data is derived from text, Web, and other sources by automatic information extraction methods or by heuristically reconciling many different databases that vary in their schemas and representations of entities. For example, we have converted the IMDB movie database, a structured dataset, and the Wikipedia encyclopedia, a hyperlinked text corpus, into XML as test cases for ranked retrieval of XML data.

The TopX engine provides powerful search and ranking functionality for this class of applications, addressing recent trends towards integrating DB and IR [1]. Example queries that TopX can efficiently handle with high-quality results are (in the W3C XPath Full-Text syntax)

```
/movie[.//location ftcontains "university"]
      [.//plot ftcontains "mathematics"]//casting
```

or

```
/article[.//topic ftcontains "travel"]
        //section[.//caption ftcontains "church"].
```

In both cases, ranking the search results becomes important if there are no results that perfectly match all query conditions, or if there are too many perfect matches of different quality, or if the user is willing to relax some of the content conditions, e.g., finding "college" or "Princeton" instead of "university", or some of the structural constraints, e.g., finding articles that contain only "category" tags instead of "topic" tags and not having any "section" elements but matching the other conditions.

While a good fraction of the TopX functionality was already supported in our earlier work on the XXL system [6], TopX has an improved scoring model for better precision and recall, and it is much more efficient and scalable. For scalability and performance, TopX employs various novel techniques: carefully designed index structures, probabilistic models as score predictors for efficient top-k query processing [10], judicious scheduling of index accesses [2, 8], and the incremental merging of index lists for on-demand, self-tuning query expansion based on integrated thesauri and background ontologies [7].

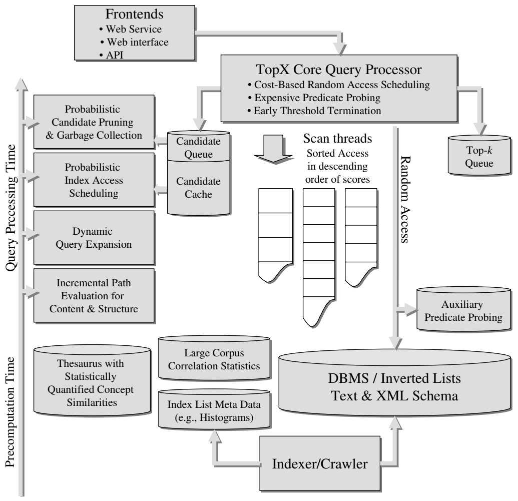
Frontends
• Web Service
• Web interface
• API
TopX Core Query Processor
• Cost-Based Random Access Scheduling
• Expensive Predicate Probing
• Early Threshold Termination
Probabilistic Candidate Pruning & Garbage Collection
Probabilistic Index Access Scheduling
Dynamic Query Expansion
Incremental Path 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
Query Prccessing Time
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

Figure 1: TopX Architecture

2. TOPX SYSTEM OVERVIEW

Figure 1 depicts the TopX main components. TopX comes with Web and file crawlers for indexing text and XML data in a relational schema on top of a database engine used as a storage platform. TopX currently uses Oracle 10g but could easily be ported to other DBMSs or a customized index using inverted files. The TopX core query processor provides the algorithms for efficient top-k query evaluation with early termination, with the option of plugging in specialized components for probabilistic candidate pruning, cost-oriented index access scheduling for further query acceleration, and dynamic query expansion for IR-style relaxed search. The query processor operates in a multi-threaded way with asynchronous disk I/O for best possible performance of sequential scans.

TopX provides both an interactive Web frontend for human users and a Web Service API to be used by other applications. A demo of the system is available at `http://infao5501.ag5.mpi-sb.mpg.de:8080/topx`. TopX is completely implemented in Java as a servlet and class library running in a Tomcat server, with a code base of approximately 37,500 lines of code in 198 classes. It is available as open-source package from `http://topx.sourceforge.net`.

3. CORE QUERY ENGINE

In order to find the top-k matches for multidimensional text or XML queries, score, and rank them, TopX uses extensions of the threshold algorithm (TA) [3], or more specifically, generalizations of its variants that use only sequential index accesses (NRA) or combine sequential scans with carefully scheduled random accesses. Our algorithms leverage the observation that sequential disk I/O is one or two orders of magnitude faster than random accesses, and we also benefit from high locality in the processor cache hierarchy.

At indexing time, TopX precomputes, for each indexable feature (i.e., text term, tag-term pair, attribute-value pair), an index list by which one can access data items in descending order of the "local" scores with regard to a given feature, e.g., the BM25-based probabilistic-IR scores [4], and stores it in a relational database. At query time, TopX scans all relevant index lists in an interleaved manner in large batches. In each scan step, when the engine sees the score for a data item in one list, it is hash-joined with the partial scores for the same item previously seen in other index lists and aggregated into a global score. The algorithm maintains a (worstscore, bestscore) interval for each item. Candidate items that may still qualify for the final top-k are kept in a priority queue of bounded length. Then testing the bestscore of the top candidate in the queue against the worstscore of the current rank-k item gives us a lightweight, anytime threshold test for early termination and early candidate pruning for efficient memory management [10].

In addition to sequential index scans, the query engine also has the option of performing random accesses to directly resolve score uncertainty of data items. Altogether, this entails the need for judiciously scheduling two types of disk accesses: 1) the prioritization of different index lists in the sequential accesses, and 2) the decision on when to perform random accesses and for which candidates. Both types involve probabilistic cost models that drive the scheduling decisions for result candidates. Our scheduling can substantially improve performance with no loss in result quality [2].

Since the user's intention behind a top-k query usually is not to find *exactly* the k best data items with regard to some ranking model, but rather to incrementally explore a topic and identify highly relevant and novel pieces of information, it is intriguing to allow approximate results. TopX provides an optional *approximate top-k* algorithm based on probabilistic predictions and with probabilistic guarantees about relative recall and precision (i.e., relative to the exact top-k result) [10]. When scanning index lists, the prediction model is used to determine when it is safe, with high probability, to prune candidates.

Compared to a straightforward implementation of TA or NRA, our algorithms and implementation techniques – priority queue management, judicious scheduling, and probabilistic pruning – yield a performance improvement of two to three orders of magnitude. For example, keyword queries like "kyrgyzstan united states relation" on the 25-million-documents TREC Terabyte corpus can be answered in 10 milliseconds and need to scan only 2 percent of the entries of the 4 relevant index lists.

4. QUERYING XML DATA

Query languages for XML IR like W3C XQuery and XPath Full-Text combine content conditions on elements with structural conditions like tag names and paths. TopX indexes occurrences of terms in XML elements together with the corresponding tags, i.e., maintains index lists for tag-term pairs. Local scores are computed using a variant of the BM25 model [4], extended to XML by considering element frequencies and element sizes. All elements within the same document are grouped together and form a contiguous block in the index lists, the blocks are then sorted by maximum element score in descending order. For structural conditions along XPath axes, we store an element's preorder, postorder, and level numbers with the corresponding index entry, using the XPath accelerator technique [5].

To answer a content-and-structure query, we have to find embeddings of the query tree in the document trees of the data collection. Score bounds for the threshold-driven top-k query algorithm are computed for each document, derived from the score of the best, partial or complete, embedding found so far. By default, TopX returns documents as query answers, but also has the option to show only the best embedding per document (i.e., subtree rooted at the target element of the query) or the best embeddings overall. The overall score of a document or subtree is the sum of its local scores for content and structural conditions.

For efficient testing of structural conditions, TopX first transitively expands all structural dependencies of the query. For example, in the query `//A//B//C[. ftcontains("t")]` an element with tag `C` (and content term `"t"`) has to be a descendant of both `A` and `B` elements. Branching path expressions can be expressed analogously. This way, the query forms a DAG with tag-term conditions as leaves, elementary tag conditions as interconnecting nodes between elements of a structural query, and all transitively expanded descendant relations as edges. This expansion of structural constraints allows *incremental testing* of path conditions [8].

The query processor primarily uses a threshold-driven top-k algorithm based on index lists, tests structural conditions based on the preorder/postorder encodings of XML elements whenever blocks of index entries are available in memory, and judiciously schedules random index accesses for additional tests of structural conditions.

5. QUERYING STRUCTURED DATA

TopX treats data with a structural schema (e.g., data from a relational database, but also schematic XML) as a special case of semistructured data, conceptually mapping non-XML data to virtual XML documents. Retrieval processes the XML-ified data and the underlying index lists, where queries are conjunctions of elementary conditions of the form `A=x` (for attribute A and value x), automatically mapped into simple XPath expressions. This query model does not support join queries, but it is at least as powerful as the search capabilities of Google Base and similar engines. In the presence of schema heterogeneity or if the user does not know the attribute names that are actually used in the data, TopX can leverage its capabilities for determining semantically similar concepts and names from its background ontologies in order to relax the query and produce ranked lists of approximate query matches.

6. DYNAMIC QUERY EXPANSION

TopX can utilize automatic query expansion to enhance result quality for difficult queries where good recall is a problem. For numerical or categorical attribute-value conditions that are not perfectly matched, the query processor considers "alternative" values in ascending order of some notion of similarity (e.g., years 1999 or 2001 for a query about year 2000). For difficult text queries like the ones in the TREC Robust track, e.g., queries for "transportation tunnel disasters" or "ship losses", TopX can integrate external knowledge to expand the query. To this end, the TopX Ontology Service provides unified access to general-purpose thesauri or ontology sources, such as WordNet or OpenCyc, or domain-specific taxonomies and ontologies. The same techniques can be applied to expand/relax tags for XML search or attribute names of queries on relational data.

Static query expansion in the IR tradition selects additional terms from a thesaurus based on a predefined similarity threshold. Thus it faces a difficult threshold tuning problem, may end up with expensive queries that contain hundreds of keywords, and runs the risk of topic drifts and result dilution. In contrast, our incremental query expansion method [7] avoids these pitfalls. The key to making query expansion efficient, scalable, and self-tuning is twofold: we avoid aggregating scores for multiple expansion terms of the same original query term, and we avoid scanning the index lists for all expansion terms. For example, when the term "disaster" in the query "transportation tunnel disaster" is expanded into "fire", "earthquake", "flood", etc., we do not count occurrences of several of these terms as additional evidence of relevance. Instead we use an aggregation function that counts only the best match of an item for a given set of expansion terms of the same original query term, weighted by the similarity of the expansion term to the original term. For efficiency, we open scans on the index lists for expansion terms as late as possible, namely, only when the best possible candidate item from a list can achieve a score contribution that is higher than the score contributions from the original terms list.

7. EVALUATION

TopX has been extensively evaluated with the two major benchmark series in IR, namely the largest available collections of the Text REtrieval Conference (TREC) on text IR and the Initiative for the Evaluation of XML Retrieval (INEX) on XML IR. For INEX, TopX performed especially well for content-and-structure queries, ranking among the top-5 of 25 in the 2005 benchmark, with a peak position 1 for two of the five official evaluation methods [9]. TopX has been the official host for the INEX 2006 topic development phase, and its Web Service interface is used by the INEX 2006 Interactive Track. During the topic development phase on an XML version of Wikipedia, more than 10,000 queries from roughly 70 different participants were processed.

8. ABOUT THE DEMO

The demo will allow interactive queries on the following XML corpora: the TREC Terabyte text corpus with 25 million Web pages, the DBLP bibliography, the IMDB movie database, the Wikipedia encyclopedia with structural tags such as "section", "category", "caption", etc., and a semantically enriched Wikipedia corpus with tags such as "person", "country", "singer", etc. (built in an automated manner from the Wikipedia structure itself). Background ontologies with different extents of knowledge can be optionally enabled for query expansion; these include ontologies based on WordNet and a recently compiled ontology derived from Wikipedia categories. Demo attendees can phrase their own queries, refine and browse results, and explore the powerful search and ranking capabilities that TopX offers.

As an example that demonstrates various features of TopX, consider a query about the life of the scientist Max Planck on the richly tagged XML version of Wikipedia. A keyword query about "life of scientist Max Planck" would mostly return articles about Max Planck institutes (i.e., research institutes that belong to the Max Planck Society), for example, in the area of life sciences. With TopX the query could be phrased much more specifically, for example as:

```
//article[./person ftcontains("Max Planck")]
        [.//category ftcontains("scientist")]//life
```

Note that this is not an interface for untrained end-users but meant as an API for program development. Using the built-in background ontology, this query finds the article about the person Max Planck and its section about his biography even if the category elements in this article mention only the term "physicist" and not "scientist" and if there is no element tagged "life" but only a tag "biography". TopX returns this article and its biography element as top-ranked result within half a second on a corpus with about 660,000 XML documents and more than 24,000,000 elements.

9. REFERENCES

[1] S. Amer-Yahia et al. Report on the DB/IR panel at SIGMOD 2005. *SIGMOD Record*, 34(4):71–74, 2005.

[2] H. Bast et al. IO-Top-k: Index-access optimized top-k query processing. In *VLDB*, 2006.

[3] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[4] D. A. Grossman and O. Frieder. *Information Retrieval.* Springer, 2005.

[5] T. Grust. Accelerating XPath location steps. In *SIGMOD*, 2002.

[6] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT*, 2002.

[7] M. Theobald, R. Schenkel, and G. Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In *SIGIR*, 2005.

[8] M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In *VLDB*, 2005.

[9] M. Theobald, R. Schenkel, and G. Weikum. TopX & XXL at INEX 2005. In *INEX, LNCS 3977*, 2006.

[10] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, 2004.