TopX – Efficient and Versatile Top-k Query Processing for Text, Semistructured, and Structured Data

Martin Theobald Ralf Schenkel Gerhard Weikum
{mtb, schenkel, weikum}@mpi-inf.mpg.de
Max-Planck-Institut für Informatik

Abstract: This paper presents a comprehensive overview of the TopX search engine, an extensive framework for unified indexing and querying large collections of unstructured, semistructured, and structured data. Residing at the very synapse of database (DB) engineering and information retrieval (IR), it integrates efficient scheduling algorithms for top-k-style ranked retrieval with powerful scoring models, as well as dynamic and self-throttling query expansion facilities.

1 Introduction

The Web increasingly moves away from the collection of unstructured text it was 10 years ago. Nowadays, the Web is a huge pile of documents that are not only heterogeneous in their content, but also in the level of annotation and structure they provide. Combining *effective* and *efficient* search over such heterogeneous collections within a single search engine will remain a major challenge, especially when the structure of documents, like in XML with potentially diverse schemata, hierarchical embeddings, and semantic annotations, should be exploited by queries and be taken into account for result ranking.

The TopX engine aims to solve this issue, addressing and seamlessly integrating recent trends of integrating DB and IR [CRW05]. It is a comprehensive framework for *unified* indexing and querying large collections of unstructured, semistructured, and structured data, comprising a full-fledged solution for ranked retrieval on desktops or intranets with annotated text or semistructured data, and, ultimately, the Web. It comes with a flexible, yet powerful and self-throttling query relaxation and/or expansion technique as an adequate means for coping with the inevitable diversity when merging various data sources that provides a controlled influence on the result ranking. While the current implementation focuses on IR-style search, the proposed methods and results can be carried over to various application areas like multimedia similarity search on high-dimensional feature vectors of images, music, or video, or preference queries over structured data such as product catalogs or customer support data in a very straightforward way.

TopX seamlessly integrates efficient query evaluation and versatile scoring models for ranked result output residing at the very synapse of database (DB) engineering and information retrieval (IR). As for the DB point-of-view, we aim at providing an efficient algorithmic basis for scalable, top-k-style processing of large amounts of data. Our focus lies on adaptive, disk-oriented cost models for accessing large, disk-resident index structures, with highly developed solutions for storing and efficiently querying large document collections (possibly in the order of Terabytes). As for the IR point-of-view, TopX provides a whole bunch of state-of-the-art, effective scoring approaches for Web IR, multi-attribute structured data, and ranked XML retrieval including XML full-text search. It supports an efficient, self-throttling query expansion mechanism that helps to increase effectiveness for difficult queries in terms of both recall as well as precision at the top ranks.

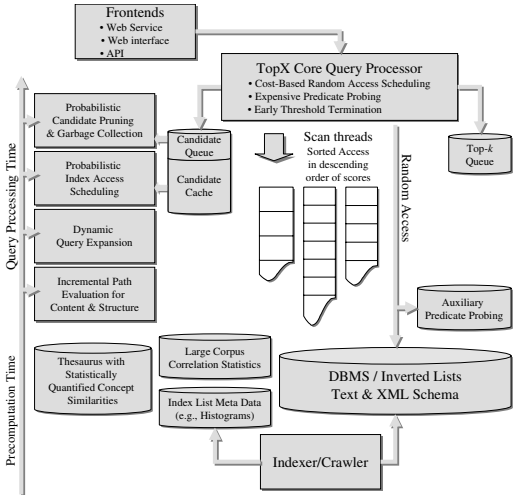

Figure 1: TopX Architecture

2 TopX System Overview

Figure 1 depicts the TopX main components. TopX comes with different, general-purpose Web and file crawlers for indexing text and XML data in a generalized schema on top of a relational back-end (which is described in the following for *Oracle 10g* but could easily be generalized to arbitrary DBMS or even a completely customized index using inverted files). The TopX core query processor is in charge of the bookkeeping of intermediate results and coordinates the sequential and random index list accesses in a multi-threaded architecture. It provides the algorithmic basis for exact and efficient top-k query evaluations with early threshold termination, with the option of gradually plugging in specialized components for probabilistic candidate pruning and cost-oriented index access scheduling that help to substantially accelerate query executions, as well as dynamic query expansion for IR-style vague search. Different levels of probabilistic extensions can stepwisely be incorporated for the probabilistic components, including basic selectivity estimators, more sophisticated index list histograms, or parameterized score estimators with convolutions for aggregated scores, index list correlations, and structural selectivity estimators for basic XML patterns such as twigs and path structures.

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

3 Querying Text Data

3.1 Index Structures

As for plain text retrieval, queries consist of a set of (possibly weighted) keywords or terms. The result of such a query is a ranked list of documents. TopX precomputes, for each term, an index list by which one can access the document identifiers in descending order of the "local" score with regard to this term, for example, the TF·IDF- or BM25-

based score [GF05]. These lists are stored on disk in a relational Oracle database in a table `TextFeatures`*(term, docid, score)*, where $docid$ is a numerical id of documents. Efficient sequential or sorted access (SA) and random access (RA) on top of the DBMS is supported by two B$^+$-trees on the attributes concatenated in the order $(term, score, docid)$ for SA and in the order $(docid, term, score)$ for RA. Additionally, optional index list meta data such as score histograms and term correlation statistics are precomputed and stored in the database and may be leveraged to accelerate query executions.

3.2 Core Query Engine

In order to find the top-k matches for multidimensional queries, scoring, and ranking them, TopX adopts and substantially extends variants of the best-known, general-purpose algorithm for evaluating top-k queries, Fagins family of threshold algorithms (TA)[FLN01]. These algorithms leverage the observation that sequential disk I/O (i.e., sorted index scans) largely benefit from asynchronous prefetching and a high locality in the hardware's and processor's cache hierarchy; so sorted access has much lower amortized cost than random disk access that is inevitably requiring additional index structures and key lookups for individual object identifiers.

TopX scans all relevant index lists in an interleaved manner; for efficiency, sorted accesses are performed in batches of fixed size b. In each scan step, when the engine sees the score for a document in one list, it is hash-joined with the partial scores for the same document previously seen in other index lists and aggregated into a global score. The algorithm maintains the worst score among the current top-k results and the best possible score for all other candidates and documents not yet encountered, by adding to the worst score the current top scores of all lists where the candidate has not yet been seen. The best score serves as a threshold for pruning a candidate when its best score does not exceed the worst score of the currently kth ranked result; the index scans are stopped when no candidate exceeds this threshold.

Efficient Candidate Queuing: While this focus on inexpensive sequential scans often results in a good query performance, it leaves uncertainty about the final scores of candidates and therefore implies some form of bookkeeping or queuing not only for the intermediate top-k results, but for all candidates that may still qualify for the final top-k. We investigated various queuing options for efficient candidate management in real-world, large collection setups; it turned out that usually, the most effective approach is the combination of a queue of bounded length with a hash-based cache. Albeit a heuristic, the queue bound q may be chosen in the order of the batch size b which is typically a safe choice. Then testing the top candidate in the queue with the k^{th} ranked top-k item allows for a lightweight, any-time threshold test for algorithm termination [TWS04].

Index Access Scheduling: In addition to the sequential disk accesses for sorted index scans, the query engine also has the option of performing random accesses to directly resolve score uncertainty of documents. Altogether, this entails scheduling for the 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 highly specialized probabilistic cost models, thus leading to individual index access scheduling decisions for result candidates which can substantially improve the performance of the retrieval engine with no loss in result quality [BMT$^+$06b].

Probabilistic Candidate Pruning: Since the user's goal 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 to identify one or a few relevant and novel pieces

of information, it is intriguing to allow IR-style, approximate variants of the threshold family of algorithms to reduce runtime costs. TopX provides an optional *approximate* top-k algorithm based on probabilistic arguments [TWS04]. When scanning index lists, various forms of convolutions over underlying score distributions and derived bounds are employed to predict whenever it is safe, with high probability, to drop candidates and to prune the index scans for early algorithm termination. These probabilistic candidate pruning techniques can provide up to two orders of magnitude performance gains with a controllable loss in result quality and a very good quality/runtime ratio.

Multi-threaded Query Processing: The query processor is organized as a triple-tier, multi-threaded hierarchy consisting of a single *main thread* that iteratively maintains the data structure for candidate bookkeeping and optionally updates probabilistic predictors for candidate pruning and adaptive scheduling decisions after a batch of b index accesses; *scan threads* that continuously read and join input tuples from the inverted list buffers for a batch of sorted accesses, and *buffer threads* that continuously refill a small buffer cache and control the actual disk I/O for each inverted list. This three-level architecture builds on the observation that candidate pruning and scheduling decisions are computationally expensive and should be done only iteratively, because joining and evaluating score bounds for candidate may incur high CPU load, while the actual sequential index accesses are not critical in terms of CPU load. Synchronization (i.e., object locking) for shared data structures only takes place when a candidate is pulled from the cache and the queue is updated, or when (occasionally) a new candidate is promoted into the top-k queue.

Top-k Shifts: The TopX query processor supports incrementally increasing the number of query results, k, without the need for restarting the whole query, thus addressing the typical behavior of a human surfer who may first look and digest the first page of 10 results, then eventually browses through the next page, and so on. In this interactive setting, in-memory candidate pruning is disabled; whenever k is increased, the currently best k candidates are identified and the scan threads continue scanning until the new top-k' candidates are found.

4 Querying XML Data

In XML IR, retrievable units are no longer whole documents but individual XML elements. Query languages for XML IR like W3C's XQuery 1.0 and XPath 2.0 Full-Text or NEXI [TS04] combine content conditions on elements with structural conditions like tag names and paths. A typical example for such a *content-and-structure* (CAS) query is the NEXI query `//article//section[about(.,``XML database'') and about(.//figure,``architecture'')]` that searches for article sections about XML databases (which is a content condition) that include a figure about the architecture (i.e., that has this term somewhere in its content). As a special case, *content only* (CO) or wildcard queries like `//*[about(., ``XML'')]` do not restrict the tag of target elements – similarly to keyword queries in text retrieval – but continue to return XML elements instead of whole documents. To search a collection with a heterogeneous or complex schema, a user would probably start with a CO query and would then refine it to a CAS query, either manually or with system support.

4.1 Index Structures

Extending the per-term inverted lists from text IR, TopX indexes occurrences of terms in XML elements together with the corresponding tags, i.e., it maintains index lists for combined tag-term pairs [TSW05b]. Such a tag-term list consists of all elements with the

tag that have the search term in their content, together with the corresponding document's id, some navigational information, the local score, and the maximal local score of any element in the same document within that list. Local scores are computed using a variant of the BM25 scoring model that has been adapted for XML, by considering individual element frequencies and element sizes instead of the document-based counterparts in classic text IR. All elements within the same document are grouped together and form a coherent *element block* in the inverted lists, which are then sorted by descending maximal element score. As navigational information that supports all XPath axes, we store the pre, $post$ and $level$ numbers using the XPath accelerator technique [Gru02], in order to implement a combined inverted index for XML content and structure in a compact way.

Inside the Oracle database, the tag-term lists are stored in a single index-only table (IOT), using the schema `TagTermFeatures`*(docid, tag, term, score, maxscore, pre, post, level)*, that allows for efficient random index accesses to all instances of a tag-term pair within a document. For efficient sorted accesses, we build an additional B$^+$-index over the full range of attributes, but in the order $(tag, term, maxscore, docid, score, pre, post, level)$; a sorted scan then corresponds to an index range scan for a given $(tag, term)$ key. As an additional enhancement, we use Oracle's *index key compression* option to automatically truncate redundant index key prefixes and skip dispensable key prefix replications at the inner index nodes of the B$^+$-tree structures.

4.2 XML Extensions to the Query Engine

For each tag-term pair occurring in the query, TopX scans the corresponding list in descending max-score order, reading complete blocks instead of single elements. Such a sequential block scan prefetches all tag-term pairs for the same document-id in one shot and keeps them in memory for further processing, which we refer to as *sorted block-scans*.

To answer a content-and-structure query, we need to find an embedding of the query structure, which may form a directed acyclic graph (DAG), into the document tree. This embedding is not necessarily unique, as there may be multiple valid embeddings per document, because several of its elements may match each of the tag-term conditions. Score bounds are then computed for each document, derived from the score of the best embedding found so far. TopX by default returns documents as entry points to answer a query, with the option to show for each document either only the best embeddings or all embeddings ranked by aggregated element scores. Internally, a hybrid processing technique combines document- and element-specific query processing in a single engine: in document mode the k^{th} document's score is used for pruning, in element mode the $k'^{\,th}$ best embedding (obtained from the $k' < k$ elements) determines the threshold which allows an even more aggressive pruning.

In non-conjunctive (aka. "andish") retrieval, a result document (or subtree) should still satisfy most structural constraints, but we may tolerate that some tag names or path conditions are not matched. This is useful when queries are posed without much information about the underlying schema (which is typical for document-centric XML like articles etc.), so the structural constraints merely provide a hint on how the actual text contents should be connected. In conjunctive query mode on the other hand, i.e., when *all* content and structure conditions have to be matched, if `C` in the above example is not a valid descendant of `A`, we may already safely prune the candidate document from the priority queue. TopX provides different approaches to judiciously schedule random accesses for testing the structural query conditions only for the most promising candidates according to their content-related scores [TSW05b] obtained through inexpensive sequential disk I/O.

5 Querying Structured Data

TopX treats data with a well-defined structure (e.g., data from a relational database, but also data-centric XML) as a special case of semistructured data, conceptually mapping non-XML data to a virtual XML document. As an example, for relational data, each tuple of a table corresponds to a document, and its attribute-value pairs are mapped to child elements of this document's root. Retrieval then considers only the XML-ified data, where queries are conjunctions of elementary conditions of the form `A=B` (A is an attribute name, B is a value) that are automatically mapped to the corresponding XML queries.

6 Dynamic Query Expansion

TopX can utilize automatic query expansion to enhance result quality for difficult queries where good recall and/or precision at the top ranks is a problem. For difficult text queries like the ones in the TREC Robust track [TRE], e.g., queries for "transportation tunnel disasters" or "ship losses", query expansion needs to integrate external knowledge. State-of-the-art approaches use one or a combination of the following sources to generate additional query terms: thesauri such as WordNet with concept relationships and some form of similarity measures, explicit user feedback or pseudo relevance feedback, query associations derived from query logs, document summaries such as Google top-10 snippets, or other sources of term correlations. In all cases, the additional expansion terms are chosen based on similarity, correlation, or relative entropy measures.

6.1 Ontology Service

The TopX Ontology Service provides unified access to multiple thesauri or ontology sources such as WordNet or OpenCyc in a compact API. Internally, the ontology service maintains a concept graph where the nodes denote the meaning of a set of terms and the edges are form the semantic relationships between concepts (like hypernyms, meronyms, antonyms, etc.) as provided by the different sources. The similarity of two related nodes is estimated by the correlation of the corresponding terms in a large corpus, e.g., using statistical measures such as Dice coefficients. The Ontology service further facilitates methods to disambiguate a term in a given document context and for finding related concepts that are similar to it.

6.2 Static vs. Dynamic Expansion

Static query expansion approaches typically choose a fixed set of expansion terms from an external knowledge base based on a predefined similarity threshold. However, such a static expansion technique faces three major problems [BZ04]: (1) the threshold for selecting expansion terms needs to be carefully hand-tuned for each query, (2) an inappropriate choice of the threshold may result in either not improving recall (if the threshold is set too conservatively) or in topic dilution (if the query is expanded too aggressively), and (3) the expansion may often result in queries with more than 50 or 100 terms, which in turn leads to very high computational costs in evaluating the expanded queries.

TopX addresses these issues and provides a practically viable, novel solution [TSW05a]. Our key techniques for making query expansion efficient, scalable, and self-tuning are to avoid aggregating scores for multiple expansion terms of the same original query term and to avoid scanning the entire 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 for a document. Rather than that, we use a max-score aggregation function that counts only the *best match* of a document out of all possible expansions for the same original query concept, optionally weighted by the similarity of the expansion to the original concept. Furthermore and most importantly for efficiency, we open scans on the index lists for expansions as late as possible, namely, only when the best possible candidate document from a yet unseen part of the expanded index lists may still contribute to the top-k results.

The algorithm conceptually merges the index lists of the expansion terms with the list of the original query term in an incremental, on-demand manner during the runtime of the query in descending order of scores. For further speed-up, probabilistic score estimations can be used as well. The great advantage of this method is that we do not have to predetermine the depth of the scanning and merging process itself. The superordinate top-k operator just incrementally inquires the subordinate Incremental Merge operator for the next document on-the-fly, thus preserving the efficient sorted access paradigm.

7 Nested Top-k Operators

For more sophisticated query sub-conditions such as phrase expansions, local scores for individual conditions cannot be fetched from materialized index lists but need themselves to be computed dynamically. This poses a major problem to any top-k algorithm that wants to primarily use sorted accesses. A possible remedy would be that the global top-k operator "guesses" a value k' and asks the dynamic source to compute its top-k' results upfront, with k' being sufficiently large so that the global operator never needs any scores of items that are not in the local top-k'.

TopX treats such situations by running a nested top-k operator on the dynamic data source(s), which iteratively reports candidates to the caller (i.e., the global top-k operator), and efficiently synchronizes the candidate priority queues of caller and callee. The callee starts computing a top-∞ result in an incremental manner, using a TA-style method itself without a specified target k, hence top-∞. It gradually builds a candidate queue with upper and lower score bounds for each candidate. The caller periodically polls the nested top-k operator for its currently best intermediate results with their score bounds and integrates this information into its own bookkeeping. From this point, the caller's processing simply follows the standard top-k algorithm (but with score intervals instead of scores). Note that this architecture also allows for a special Boolean – but ranked – retrieval mode, e.g., enforcing conjunctions at the top-level top-k operator while allowing disjunctions at the leaf operators for high-dimensional expansion tasks.

8 Evaluation

We focus our experiments with textual data on the TREC Terabyte collection which is the largest benchmark corpus currently being available with relevance assessments, consisting of about 25 million documents with a size of roughly 425 GB, with 50 reference queries from the 2005 Terabyte Ad-Hoc task. For XML, we chose the new 6 GB INEX Wikipedia collection with about 660,000 XML-ified Wikipedia articles and the respective batch of the 125 INEX 2006 Ad-Hoc queries. On a mainstream server machine with a dual XEON-3000 CPU, 4GB of RAM, and a SCSI RAID-5, indexing these collections took between 280 minutes for INEX and 14 hours for Terabyte, including stemming, stopword removal, and computing the BM25-based scores. The materialization of the B^+-indexes required roughly the same amount of time as it included sorting a huge intermediate table.

As for efficiency, we consider abstract query execution costs defined as $cost = \#SA +$

$c_R/c_S \, \#RA$, i.e., a weighted sum of the number of tuples read through sorted and random accesses from our disk-resident index structures, as our primary metric analogously to [FLN01]. The cost ratio c_R/c_S between a single sorted vs. a single random access has been determined to optimize our runtime figures at a value of 150, which nicely reflects our setup using Oracle as backend and JDBC as connector, with a relatively low sequential throughput but good random access performance because of the caching capabilities of the DBMS. Wallclock runtimes were generally good but much more sustainable to these very caching effects, with average CPU runtimes being in the order of 0.3 seconds for Wikipedia and 1.2 for Terabyte, and wallclock runtimes being 3.4 and 6.2 seconds, respectively. All the reported cost figures are sums for the whole batch of benchmark queries, whereas the precision figures are macro-averaged.

8.1 Terabyte Runs – Efficiency vs. Effectiveness

Figure 2 compares the cost figures for TopX on the Terabyte setup with a DBMS-style merge join that first joins all documents in the query-relevant index lists by their id and then sorts the joined tuples for reporting the final top-k results (eventually using a partial sort). For $k = 10$, the non-approximate TopX run with the conservative pruning already outperforms the full-merge by a factor 5.5, while incurring query costs of about 9,323,012 compared to 54,698,963 for the full-merge. Furthermore, we are able to maintain this good performance over for a very broad range of k; only queries of considerably more than 1,000 requested results would let our algorithm degenerate over the full-merge approach.

The approximate TopX with a relatively low probabilistic pruning threshold of $\varepsilon = 0.1$ generally performs at about 10 percent lower execution costs than the exact TopX setup which confirms exactly to the pruning behavior we would expect and the probabilistic guarantees for the result quality we provide in [TWS04].

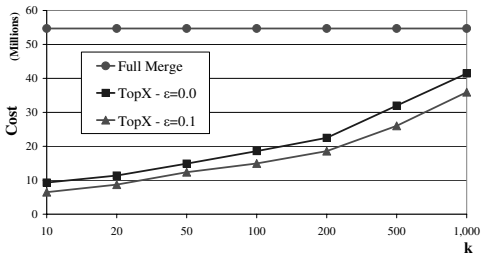

Figure 2: Query execution costs for Terabyte as functions of k.

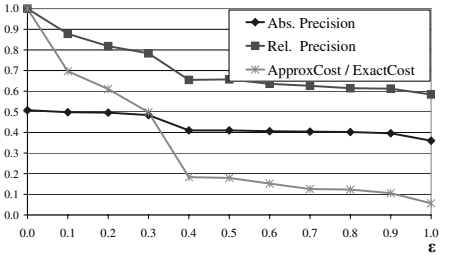

Figure 3: Relative vs. absolute precision for Terabyte as functions of ε, for $k = 10$.

Figure 3 investigates the probabilistic pruning behavior of TopX for the full range of $0 \leq \epsilon \leq 1$ for a fixed value of $k = 10$, with $\epsilon = 1.0$ (i.e., the extreme case) meaning that we immediately stop query processing after the first batch of b sorted accesses. Since Terabyte is shipped with official relevance judgments, we are able to study the result quality for both the *relative precision* (i.e., the overlap between the approximate and the exact top-k) and the *absolute precision* (i.e., the fraction of results officially marked as relevant by a human user for a particular topic). We see that the relative precision drops much faster than the absolute precision which means that, although different documents are returned at the top-k ranks, they are mostly equally relevant from a user perspective. Particularly remarkable is the fact that the ratio of the execution cost between the approximate and the exact top-k generally drops at a much faster rate than both the absolute and relative precision values. This observation holds for all collection and query setups we considered so far.

8.2 Wikipedia Runs – Efficiency vs. Effectiveness

As for Wikipedia, we provide a detailed comparison of the CO and CAS interpretations of the queries. Figure 4 shows that we generally observe similarly good performance trends as for Terabyte, with cost-savings of a factor of 7 for CAS and 2.5 for CO, and the performance advantage remains extremely good even for large values of k, because we never need to scan those long (i.e., lowly selective) element lists for the navigational query tags which would have to be performed by any non-top-k-style algorithm.

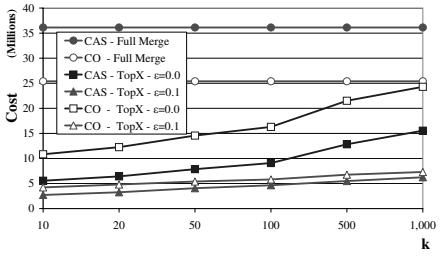

Figure 4: Query execution costs for Wikipedia as functions of k.

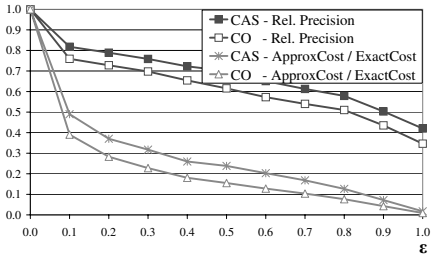

Figure 5: Relative precision for Wikipedia as functions of ε, for $k = 10$.

Figure 6 depicts a detailed comparison of the query costs being split into individual sorted (#SA) and random (#RA) disk I/Os for the CO and CAS flavors of the Wikipedia queries. It shows that we successfully limit the amount of RA to less than about 2 percent of the #SA according to our cost model. This ratio is maintained also in the case of structured data and -queries which is a unique property among current XML-top-k engines (see also [TSW05b] for a comparison to state-of-the-art competitors). Figure 7 shows an impressive runtime advantage for the dynamic query expansion approach compared to both full-merge and TopX when performing static expansions (measured for the CAS case). The reported numbers reflect large, automatic thesaurus expansions of the original Wikipedia queries based on WordNet, with up to $m = 292$ distinct query dimensions (keywords). Figure 6 finally demonstrates a similarly good pruning behavior of TopX for both the CO and CAS queries in Wikipedia, again showing a very good quality vs. runtime ratio for the probabilistic candidate pruning component.

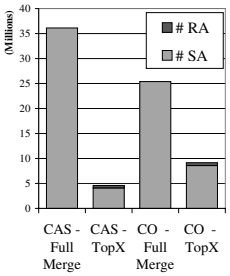

Figure 6: #SA and #RA for full-merge vs. TopX, for $k = 10$ and $\varepsilon = 0$.

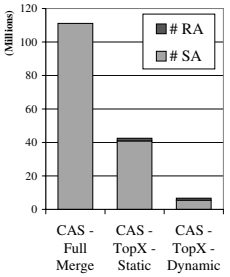

Figure 7: #SA and #RA for full-merge vs. TopX, for $k = 10$ and $\varepsilon = 0$.

8.2.1 Further Experiments

TopX has been extensively evaluated throughout two years of active participations in the two major benchmark series in IR, using the largest available collections of the Text RE-

trieval Conference (TREC) [TRE] on text IR and the Initiative for the Evaluation of XML Retrieval (INEX) [M$^+$06] focusing on XML IR. For INEX, the 2005 benchmark included a set of 40 keyword-only and 47 structural queries with relevance assessments that were evaluated on the INEX IEEE corpus. TopX performed very well for CAS queries, ranking among the top-5 of 25, with a peak position 1 for two of the five official evaluation methods. See [TSW06, BMT$^+$06a, TSW07] for more detailed discussions of the results on various TREC and INEX tasks.

9 Conclusions and Future Work

TopX is an efficient and effective search engine for unstructured, semistructured, and structured data. Our future work will concentrate on (1) including linkage information in the retrieval process, especially for XML, (2) extending our top-k algorithms to support non-monotonous scores like proximity, and (3) implementing an efficient inverted file structure. TopX has been the official host used for the INEX 2006 topic development phase, and its Web Service interface will be used by the INEX 2006 Interactive Track. During the topic development phase, more than 20,000 CO and CAS queries from roughly 70 different participants world-wide were conducted partly in parallel sessions over the new Wikipedia XML index.

*Bibliography

[BMT$^+$06a] Holger Bast, Debapriyo Majumdar, Martin Theobald, Ralf Schenkel, and Gerhard Weikum. IO-Top-k at the TREC Terabyte track 2006. In *TREC*, 2006.

[BMT$^+$06b] Holger Bast, Debapriyo Majumdar, Martin Theobald, Ralf Schenkel, and Gerhard Weikum. IO-Top-k: Index-access Optimized Top-k Query Processing. In *VLDB*, 2006.

[BZ04] Bodo Billerbeck and Justin Zobel. Techniques for Efficient Query Expansion. In *SPIRE*, 2004.

[CRW05] Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *CIDR*, 2005.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, 2001.

[GF05] D. A. Grossman and O. Frieder. *Information Retrieval*. Springer, 2005.

[Gru02] Torsten Grust. Accelerating XPath location steps. In *SIGMOD*, 2002.

[M$^+$06] Saadia Malik et al. Overview of INEX 2005. In *INEX*, 2006.

[TRE] Text REtrieval Conference. `http://trec.nist.gov/`.

[TS04] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). available from `http://www.cs.otago.ac.nz/postgrads/andrew/2004-4.pdf`, 2004.

[TSW05a] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. Efficient and Self-tuning Incremental Query Expansion for Top-k Query Processing. In *SIGIR*, 2005.

[TSW05b] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. An Efficient and Versatile Query Engine for TopX Search. In *VLDB*, 2005.

[TSW06] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. TopX & XXL at INEX 2005. In *INEX*, 2006.

[TSW07] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. TopX at the INEX Ad-Hoc and Relevance Feedback tasks 2006. In *INEX*, 2007.

[TWS04] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k Query Evaluation with Probabilistic Guarantees. In *VLDB*, 2004.