A Survey in Indexing and Searching XML Documents in 337 one-liners

Maarten M. Fokkinga

Version of March 31, 2004, 9:00

Introduction This is my personal "summary in 337 one-liners" of *A Survey in Indexing and Searching XML Documents* by Luk *et al.* (2002) [1]. I focus on technical aspects, omitting all system names and references.

In my opinion, one *cannot learn* any technique from the survey: it only *mentions* various techniques but *does not explain* any. Alas, my 337 one-liners are even less informative.

The survey itself can be used as a test whether you already knew all the things that it mentions, and the classification that it gives. Further, the survey is useful as a rather complete collection of references to the literature (and this aspect is completely omitted in this summary).

I am impressed by the apparent completeness of the survey, but I consider most of it badly written (unclear, incomprehensible and in bad English).

Part I: Indexing

Flat-File Indexing

An XML document can be viewed as, or coerced to, a flat file;
then indexing is similar to indexing for conventional IR systems.

- Simplest method: discard tag entirely.
  Advantage: simplicity; disadvantage: loss of tag information.
- Another method: discard ⟨ and ⟩ symbols.
- Another method: treat tags ⟨*xxx*⟩, including the brackets, as normal words.
  Issue: distinguish between frequency counts for tags and content?

Disadvantage: tree structure lost (and not accounted for in proximity notion).

Semistructured Indexing

Choices: indexing all structure, only predefined structure, or only specific structure types (e.g., tree, segment).

- Field-based indexing.
  Represent search term for ⟨*title*⟩*Engineering*⟨/*title*⟩ by *title*:*Engineering*.
  Similarly for a fixed set of so-called fields (*title*, *author*, *date*, . . .).
  Issue: how to find fields?
  (Derived from meta-data, inserted as xml, automatically extracted.)
  Issue: indexing on fields?
  (Simple IR/DB techniques for non-overlapping structureless fields.)

- Segment-based indexing.
  A document is viewed as a set of overlapping *regions*.
  No model [in the survey] fully supports overlapping & nesting.
  Issue: what region to return (top-level only?), and what type of region?
- Tree-based indexing.
  Often: each node has an *Id* such that the parents *Id* can be computed from a child's *Id*, and so no extra structure info apart from the *Id*s has to be stored.
  - ⊛ *AllNodes*-method: store for each word *w* all relevant pairs (*DocId*, *EltId*).
  - ⊛ *LeafNodes*-method: same but only for leaf elements (nodes).
    - ○ Novelty: dynamic generation of XPath to results during searching;
      ranking based on aggregate of all weights of matched terms. [???]
    - ○ Implementation (BUS arch.): DB engine maintains all query postings;
      so XML doc contents may be changed without a need for re-indexing. [???]
  - ⊛ *Proximity*-method: indexing based on proximity;
    because many XQL queries can be answered based on proximal nodes.

Structured Indexing

Here, *structured* = 'exploiting a *schema*' (XML or DB schema).
Advantage of *structural* index: efficiency of retrieval, precision of result.
Note:

- Storing XML doc into DBs: easy querying, efficient retrieval.
- Storing DB into XML docs: easy info exchange, presentation, and heterogeneity.

Nowadays, there is a convergence in integrating IR, XML, and DBs.
There are four types of structured indexing:

- IR/DB indexing.
  - ⊛ The survey gives a survey of well-known DB indexing:
    Indexed Sequential Access Method (ISAM), primary/secondary index,
    clustering, multiple level indexing, B^+ tree: $O(\mathit{log}\, n)$, hashing: $O(1)$, ...
  - ⊛ "XML×DB" asks for indexing and retrieving of XML docs having structure of Rel DBs:
    Some systems do so...
    - ○ Handling exact querying:
      use index based on attributes.
    - ○ Handling inexact/approximate querying:
      use inverted files...,
      or use signature-based indexes (in particular in IR systems). [???]
    - ○ Extending an IR system into a search engine for DBs:
      Each DB record is considered/made into a separate XML doc: e.g.,
      $(\ldots, \ldots, \ldots\ldots) \mapsto \langle \mathit{stud} \rangle\ \langle \mathit{id} \rangle \ldots \langle / \mathit{id} \rangle\ \langle \mathit{name} \rangle \ldots \langle / \mathit{name} \rangle\ \ \ldots\ldots\ \ \langle / \mathit{stud} \rangle$.
      Then $\langle \mathit{name} \rangle$ *Jo Jans* $\langle / \mathit{name} \rangle$ is indexed as *name*:*Jo* and *name*:*Jans*.
      Deal with proximity by positional info, or another method.
      SQL queries can now be posted as IR queries.
      Care is needed for joins!
      Wildcards and boolean connectives turn up in these IR queries.

    This has the following (dis)advantages:
    + Ranking of DB queries can be supported; 25% better precision.

        + No relational model is needed to build the RDB [???]
          [**MMF: Nonsense! Even when 'reln model' is replaced by 'db schema'.**]
          Arbitrary joins can be computed, even over heterogeneous sources (since everything can be indexed and searched, in the IR approach).
        − There is some overhead in storing records as xml docs:
          the amount of xml markup is 30-200% of the original ascii size.
          Therefor, some xml compression is needed.

- Path-based indexing.
  - ⊛ In (hierarchical) structures, querying involves *navigating*.
    Following oid-chains in *forward* direction is efficient.
    For the reverse direction (= *join* in RDBs!) store reverse links.
    Still, accessing intermediate objects is costly.
    Solution: build *path dictionary index* (based on data dictionary):
    - ○ It mimics the structure in an economic way, with shortcuts.
    - ○ Augmented with attribute indices that map highly-selective attributes to path info (instead of to oids).
  - ⊛ Path-based indexing: one index for content and one for structure.
    Each path is assigned an identifier *xId*, also used as doc identifier.
    Paths are indexed too: e.g., *title* maps to all path *xId*s containing *title*.
    Now, '*path*[*term*]' is the query: What *xId*s contain *term* and match *path*?
    Possibly, add (costly) positional index to paths (and not to content), so that structural relations can be asked for. Alternatively, use pattern matching afterwards to decide satisfaction of the required structural relation.
- Position-based indexing.
  - ⊛ Spatial indexing = using R-tree or *kb*-tree.
  - ⊛ Regard an xml doc as having 2-dimensional ("spatial") pretty print layout.
  - ⊛ Advantage: Same spatial indexing scheme for rendered and textual doc.
  - ⊛ Also: spatial structure closely corresponds to logical structure.
  - ⊛ Issue: doc updating effects the index; so, use *relative* region coordinates.
- Multi-dimensional Indexing.
  - ⊛ Historical notes:
    - ○ Traditional indexing on several attributes gives *independent* indices, one of which may be chosen as the *primary* one.
    - ○ For multi-dimensional spatial data, where space is divided into (non)-overlapping *region*s, R-*tree*s have been invented (generalizing B-trees).
    - ○ In OLAP, data cubes are a representation of multi-dimensional data.
      R-trees are useful for processing of, amongst others, dimension reduction, proximity search, similarity matching, roll-up, roll-down.
      Signature indexing (in the form of bitmap indices) arose recently.
  - ⊛ Application: xml docs may be considered as multi-dimensional data.
    - ○ So, *be inspired by the previous historical notes.*
    - ○ Refinement: index not on tuples but on *chunks* (modeled as xml docs).
      Each chunk can be processed and cached independently.
      Query analysis can yield great precision in the chunks (overcoming Internet bottleneck).
  - ⊛ Application: IR systems may load flat files into multi-dimensional DBs.
    - ○ Use standard attributes *time*, *location*, ... and also IR attributes *term*, *frequency*, ... as dimension.
    - ○ Now, multi-dim sql-like queries can be issued, involving IR criteria.

Part II: Searching

Full-text search

- Note: `grep` utilities facilitate Regular Expression (RE) search (including positional conditions), but cannot deal with structural information.
- Improvement: construct and exploit the *current path* on the fly.
- Improvement: `sgrep`, a generalization for *structures* (dealing with regions).
- Improvement: extended RE, supporting hierarchic and nested text fragments. (Hosoya: 'RE *with type inference*', for matching multi-nested xml fragments.)

XML Assisted Search

- XML can support IR:
  - ⊛ Encode ontologies within xml documents.
  - ⊛ Encode bilingual glossary (including collocation, etc).
  - ⊛ Electronic catalogues in e-commerce already use XML.
  - ⊛ One xml doc as updatable blackboard in distributed, collaborative search.
  - ⊛ One xml doc to gather all answers from distributed IR.
- XML Extender: a repository for xml docs and dtd's, indexed on elements and attributes, itself searchable via, e.g., SQL.
- Xyleme: a DB repository of xml docs; queries expressed on various *abstract* views. The engine translates the queries to queries on the collection of *concrete* docs.

Multi-stage Search

- Typically:
  1: find elements/identifiers from inverted index;
  2: find relevant structural information from the found elements/identifiers;
  3: optionally: filter (e.g., delete element with unqualified structural information);
  4: optionally: determine score aggregation from structural information found;
  5: combine sub-results into one final result.
- Alternatively:
  1: find relevant DTDs, use them to filter irrelevant documents.
- Alternatively:
  1: do IR-search for best *category*;
  2: do SQL query against the DB for that category (using the DB schema).

Search Results

- Ranking of search results.
  - ⊛ Issue: does ranking improve recall-precision when xml tags are considered?
  - ⊛ Conventional IR ranking possibly not appropriate because frequency of term occurrences might differ greatly between flat texts and xml documents.
  - ⊛ Inverse Document Frequency IDF might be inappropriate since results units are xml elements rather than xml documents.
    Inverse Element Frequency IEF might be more appropriate.
  - ⊛ Ranking performance drops when documents are fragmented.
    Solution(?): let user judge relevance of fragments — alas, user gets lost.
    Solution(?): aggregate scores over all segments — alas, structure is lost.
    Solution(?): aggregate scores of all nodes with weighting for structure.
    Solution(?): . . . and weight for datatypes of various sub-nodes.
  - ⊛ Proximity is problematic in (tree)structured xml documents.
    Solution(?): distance = number of edges to be traversed.
    Generalization to graph structure is possible (and realized).

  - Similarity between xml documents:
    Defined as the number of edit operations to make them equal.
    Computationally hard, but practically feasible (and realized).
  - More problems because of difference between xml doc and its xml layout!
- Granularity of search results.
  Possible units of search result:
  - Entire xml document.
    - How to rank docs with various degrees of structure?
      (Highly structured versus rich-text documents.)
    - How to aggregate score of various components?
      (May depend on retrieval model.)
  - Xml element plus its content.
    - What kind of element/fragment to return?
      Just the matching fragments — too fine-grained (loss of context).
      Least upper bound — might not work (as has been shown).
      Ehhh, uhhhm, . . . index nodes, down-weighting scores. . . [???]
      The "authorative" element (cf. link analysis for Web search engines).
  - Synthesis of various result units.
    - Topic distillation and link analysis . . . . . . [I don't understand, MMF] [???]
- Layout of search results.
  - Accordion summary (= M$ Explorer directory presentation).
  - Extra mark-up, e.g., in case of multi-lingual documents.
  - Extra *forms* for query and answer; useful for standardized reporting.

Part III: Retrieval Models

Relational Model

A Data Model characterizes the structure of data dealt with in DBs.
The marriage XML $\times$ DB asks for adaptation of Relational Model.

- Basic Relational Model.
  Conversion between DB and XML exists; from XML to DB needs care:
  use meta-DB to achieve interoperability.
  Querying of XML-filled DB goes via XML query languages (with *forms*).
  Issue: updates of the XML-filled DB possible via the query language?
  For mobile Web use of XML, performance needs attention (middelware!).
  Query Languages:
  (XML-QL, XML-GL, DataX, TSIMMIS, LoreL, . . . etc. . . .)
  (UnQL: SQL-like language with pattern matching on trees; like path expr's.)
  (StruQL: more expressive than UnQL; query and construction phases.)
  XQuery seems to be most elaborated, with its powerful FLWOR expression.
  Issue: extend the ER model to make it suitable for XML
  (since often the XML tree structure hides the conceptual organization).
- Relational Model with Uncertainty.
  Uncertainty and imprecision are unavoidable in real life.
  Three-valued logic and *NULL* is the classical approach for missing info.
  Uncertainty is handled by statistics/probability, or logic (see below).
- Probabilistic Relational Model.
  Attributes or tuples have probabilities, giving the degree of uncertainty.
  A-priori values are sometimes problematic; only estimated from the data itself.
  Application: ranking hypotheses from medical expert systems.
  Issue: all operations on data or tuples need to propagate the probability values.
  Extension to XML documents is easy.
- Logical Relational Model.
  - 3-valued and n-valued logic is too simple to model uncertainty.
    Fuzzy-logic is better, and has been used (e.g., in rule-based expert systems).
    (Theories for fuzzy functional dependency seem practical!)
  - The powerful and successful Datalog and LDL can incorporate probability
    (and have been implemented efficiently under the independence assumption).
  - IR's handling of uncertainty seems quite well extensible to the xml context.

Object-Oriented Model

- The XML hierarchy + Xlink, Xpointer is suitable to model OO structures.
  Alas, the large amount of small xml docs incurs storage overhead (for the tags).
  Conversely, retrieval from xml docs is a kind of object view.
- XML Schema extends DTD mainly by having inheritance,
  facilitating easy translation between XML and OODB.
  Alternatives: XDR (from M$) and SOX (joint industry), both less expressive.
- Issue: . . . 'superimposed coding signature' index . . . false drops . . . [???]
  Issue: naming, since xml docs arise from heterogeneous sources;
  (namespaces help, but do not relate names between different schemas).

Extended Vector Space Model, EVSM

- Basic VSM for XML: element names dealt with as normal terms;
  e.g., query '$\langle title\rangle$*My Story*$\langle /title\rangle$' becomes vector ($\langle title\rangle$, *My*, *story*).
  Alas: structural aspects get lost (but it is easy for the user).
  Improvement: structure and content in 2 vectors, e.g., (($\langle title\rangle$), (*My*, *story*)).
- EVSM: query take form $\{\, path[term, weight], path[term, weight], \ldots \}$.
  Distance between terms: number of edges to traverse (influences ranking).
- Generalization: handle score aggregation in hierarchy;
  apply ESVM recursively and take intelligent aggregation of weighted subscores.

Weighted Boolean Model

- Queries take the form: Boolean combination of $path[term]$.
  Use $path[term_1 \wedge term_2]$ to abbreviate $path[term_1] \wedge path[term_2]$ (also for $\vee$, ...).
- Extension with weights: see Probabilistic Inference Model below.

Probabilistic Model

Traditional probabilistic model: retrieval is a Baysian decision problem. [???]
Inverse Document Frequency IDF is related to probabilities of term relevance.

- Probabilistic vector model.
  Generalizes traditional model, based on maximum entropy idea.
  Ranking $score(d, \vec{x})$ of doc d for a query consisting of $path[term]$-items $\vec{x}$ is:
  monotone in $P(rel \mid \vec{x}) \,/\, P(irrel \mid \vec{x})$, where
  $P(rel \mid \vec{x})$ = "the prob. that a doc, containing same x's as d, is relevant".
  To estimate $P(rel \mid \vec{x})$, term independence is wrongly *assumed* for simplicity and:
  - either: use extended estimation of the Combination Match Model,
  - or: use Inverse Element Frequency IEF,
  - or: gradually improve the estimate by user judgments,
    (issue: what element to return for user judgments?)
    (issue: how to save user from too many judgments?)
  - or: user Dempster-Shafer to find evidence for $P(rel \mid \vec{x})$:
    each query term occurrence in a doc is 'evidence' that the doc is relevant
    (possibly weighted by a user given belief of relevance);
    together aggregated (in the D-S way) into a joint total relevance.

  It is not yet know which theory (e.g., D-S) is best, under what IR context.
- Probabilistic inference model.
  Extends the traditional and vector model by more expressive logical combinators.
  - E.g., base ranking on $P(d \Rightarrow q)$ ("the prob. that doc d implies the query q").

  Extension to XML:
  - use $P(docFragment \mid path[term])$ instead of $P(doc \mid term)$.
    - Consequence: redo estimation of relevance score.
    - Consequence: some laws invalid (e.g., $P(d \mid t) \neq \Sigma_p P(d \mid p[t])$) [typo?]

  Extension of the propositional logic part to predicate logic:
  - Evaluate a query by transforming it into facts+rules in pDatalog.
  - This gives powerful expressiveness, but large scale efficiency remains unclear.

  Footnote:
  For structureless docs, the Boolean, fuzzy set, and vector model are special cases of the probabilistic inference model, but this does not hold for the extensions that *do* deal with XML structure.

Part IV: Some Open Issues

Heterogeneous Data

Important issues for a dynamic and heterogeneous environment (Internet) are:

- How to deal with user-defined document structures?
  Diversity of doc structures and tag names affect search performance.
- How to deal with multi-lingual xml documents?
  Translation of words (in tags) is an *n*-*n* relation; which translation to choose?

Ranking

- What estimation of term weights and retrieval model gives best performance?
  Recall: Combination Match Model has been extended to retrieve xml docs.
  Recall: EVSM has been extended to certain types of retrieval of xml docs.
- What effect should smallness of fragments have on significance of term counts?
  Recall: traditionally the significance is inversely proportional to *df*.
  In xml context: significance might be proportional to *element frequency.*
- How to rank when combining DB search with IR search?
  - ⊛ Note: DB search result (record) is much smaller in size than IR result (doc).
  - ⊛ Search results of DB and IR presented in two columns (practically relevant).

Evaluation of techniques and systems

- Problem: there is no large xml data repository available.
  - ⊛ Cause: xml was only recently born.
  - ⊛ Cause: published data is converted to html (for browser compatibility).
  - ⊛ Cause: published data is stripped of xml mark-up for competitive reasons.

  Nevertheless, we $\frac{hope}{expect}$ that xml may be adopted for IR and public data exchange.
  We need a survey of organizations using xml, to get a picture of xml utilization.
- Evaluation is technically difficult.
  - ⊛ Cause: diversity of types of documents (ranging from DB to full-text).
  - ⊛ Cause: lack of end-users that form realistic complex queries.
  - ⊛ Cause: for complex results, the unit for measuring performance is unclear.
  - ⊛ Suggestion: define user model for evaluation (as is done for html searching).

Retrieval Model

- It is unclear which existing retrieval model is best (for indexing and searching).
  - ⊛ Cause: suitability depends on search context (simple end-user vs. expert).
- Alternative: design *new* retrieval model for XML documents.
  - ⊛ There are already examples, but based on languages for the users' info need.
  - ⊛ Needed are:
    - ◦ new data models — possibilities are not surveyed here.
    - ◦ new models for uncertainty in retrieving relevant documents —
      (besides prob. inference model, also fuzzy set model, etc, are applicable).

Indexing

- There is a trade-off between retrieval efficiency and query expressiveness.
  Proximal nodes represent good trade-off.
  Since BUS arch. (*line* 32) has good storage efficiency, this arch. *might* help.
- Issue: efficiency of updates (relevant since e.g. patent collections forever change).
  - ⊛ The BUS solution has large storage costs (over 100% of doc collection store).
  - ⊛ Possible solution(?): index mechanism that allows for *inserts of new docs.*

Searching

- Searching for *patterns* in xml documents is difficult.
  Matching *paths* (given in a pattern) is important for retrieval speed,
  not only for path-based indexing methods, but also for others methods.
- To be explored:
  - ⊛ Merging IR search and DB search.
  - ⊛ Relevance feedback:
    - ○ Feedback on both document type and document content.
    - ○ Used to avoid complicated query language for naive end-users.
      When the required doc types have been identified from feedback,
      the system should merge them into an intended sophisticated query.

Document Management

Of course, efficient and efficient management of xml documents is needed.

- E.g., mark-up dealing with corrections may affect search engines.
- E.g., doc management may require new doc types and new retrieval functions.
  E.g., multi-lingual doc management may require multi-lingual IR functions,
  just for finding a doc, or even for assisting further processing of found docs.
- For these, special search engine interfaces could be designed.

References

[1] Robert W.P. Luk, H. V. Leong, Tharam S. Dillon, Alvin T.S. Chan, W. Bruce Croft, and James Allan. A survey in indexing and searching xml documents. *J. Am. Soc. Inf. Sci. Technol.*, 53(6):415–437, 2002.

The survey's description of Xyleme (literally! — page 424, line 6 *ff*):

1 The Xyleme project is an [. . .] attempt to build a "data warehouse" to collect all the useful XML documents for efficient querying, indexing, and searching.

2 [. . .] Xyleme is based on retrieval from XML repository, expressed as queries on different *views*.

3 Views are defined based on abstract summary of real documents, the form of concept trees.

4 Queries can then be made against a path on the view tree, which are then translated into queries against actual collection of XML document hierarchy.

5 Finally, the collection should be stored within DBs for efficient retrieval.

6 As such, Xyleme can be considered as an integrated IR and DB system.

My understanding (one-liners 132–133):

1 Xyleme: a DB repository of xml documents; queries are expressed on various *abstract* views.

2 The engine translates the queries to queries on the collection of *concrete* documents.