Identifying and Ranking Relevant Document Elements

Andrew Trotman and Richard A. O'Keefe
Department of Computer Science
University of Otago
Dunedin, New Zealand

andrew@cs.otago.ac.nz, ok@otago.ac.nz

ABSTRACT

A method of indexing and searching structured documents for element retrieval is discussed. Documents are indexed using a modified inverted file retrieval system. Modified postings include pointers into a collection-wide document structure tree (the corpus tree) describing the structure of every document in the collection.

Retrieval topics are converted into Boolean queries. Queries are used to identify relevant documents. Documents are then ranked using Okapi BM25 and finally relevant elements are identified using coverage. Search results are presented sorted first by document then coverage.

The design is presented in the context of the second annual INEX workshop.

1. INTRODUCTION

Otago first entered INEX [2] during its second year. There were three objectives: understand the participation process, gain access to this and last year's judgments, and create a baseline for comparing future experiments.

Participation involved design of six topics, generation and submission of search results, and online judging of three topics. Of these, generating the results was the most problematic as it required software changes.

The chosen retrieval engine was designed from the onset for retrieval of whole academic documents in XML [1]. A predecessor can be seen on BioMedNet and ChemWeb [4]. This engine, like that used in the IEEE digital library, returns relevance ranked lists of whole documents – the natural (citable) unit of information in an academic environment. From experience, information vendors are not interested in converting their documents from propriety DTDs into a common DTD or any other format – so software was needed to handle documents in heterogeneous formats.

Boolean searching, field restricting and relevance ranking were already supported, so modifications focused on identifying and ranking document elements. The modified retrieval engine can be thought of as working in three parts. Candidate documents are identified using a Boolean query. Candidates are then ranked using Okapi BM25 [7]. Finally, relevant non-overlapping elements are identified and presented as the result. Although it is easier to understand in three parts, in fact the most relevant elements of the most relevant documents are computed in a single pass of the indexes.

2. INDEXING

Much of the index design has already been described elsewhere [8]. Inverted file retrieval is used. There is one dictionary file and each dictionary term points to a single inverted list of postings.

An unstructured inverted list is usually represented $\{<d_1, f_1>, <d_2, f_2>, ..., <d_n, f_n>\}$ where d_n is a document ordinal number and f_n is the frequency of the given term in the given document. For structured retrieval, each $<d_n, f_n>$ pair is replaced by the triple $<d_n, p_n, f_n>$, where p_n is a position in the document. When phrase or proximity searching is required, this triple is replaced with the triple $<d_n, p_n, w_n>$ where w_n is the ordinal number of the term in the collection (starting from 0 at the start of the collection, incrementing by 1 for each term, not incrementing for tags, and not reset at the beginning of each record). On disk the postings are stored compressed.

The p_n value in each posting is a position in the corpus tree. The tagging structure for any one document represents a tree walk. Start at the root of the tree. When an open tag is encountered, the branch labelled with the tag name is followed downwards. When a close tag is encountered, the walk backtracks one branch. For a well-formed XML document, the walk will start and end at the root. This tree-walking property also holds for a collection of well-formed documents. The tree they collectively describe is called the corpus tree and can be built during single pass indexing. As each node is encountered for the first time, a branch is added to the tree and labelled with a unique ordinal identifier, p_n. Terms can lie either at the nodes or the leaves of this tree.

The corpus tree includes every single path in every single document, but is unlikely to match the structure of any one document. In Figure 1, three well-formed documents are given, as is the corpus tree for those documents. For clarity, the branches of the tree are labelled with which document they describe although this information is not computed and not stored.

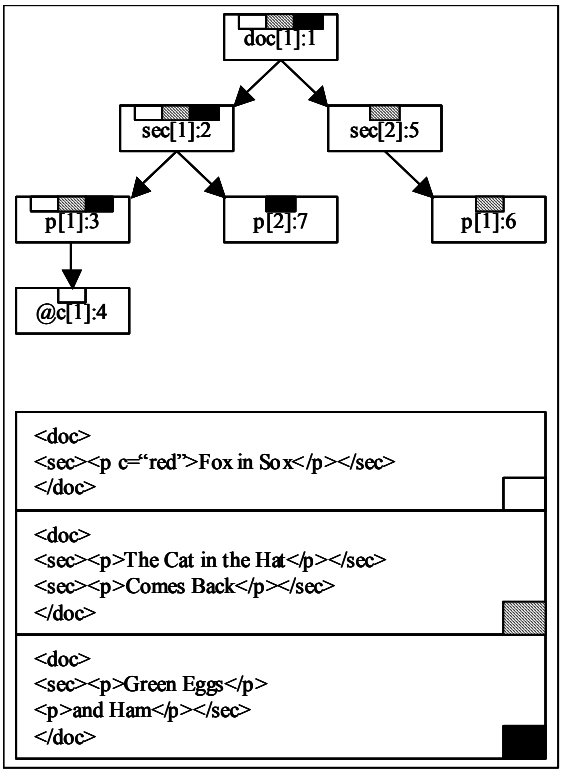

Figure 1: Three documents and the corpus tree including every path through every document, but not matching the structure of any one document. For the purpose of this figure each document is marked white, gray, or black and each node with which documents include that path. Each node is numbered with the instance of the tag (e.g. p[2]) and the node id, p_n (after the colon).

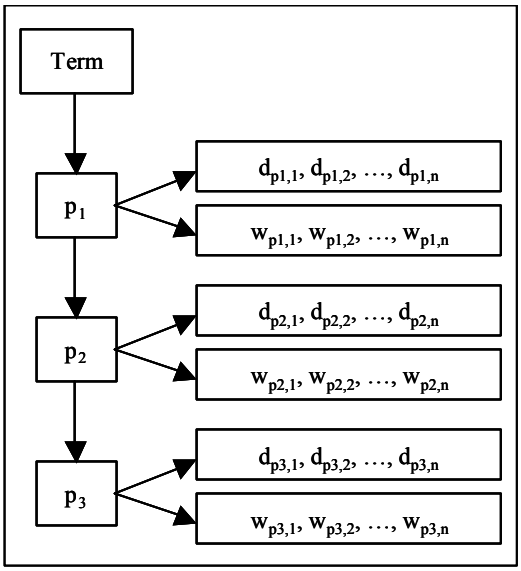

Figure 2: The in-memory postings structure allows quick access to only those postings relevant to the required document elements.

The inverted lists are built and processed using the structure represented in Figure 2. Postings for each term are ordered by increasing p_n. Each p_n points to the list of document ids (the *d*-sublist) and word ids (the *w*-sublist) found at that point in the tree. Each list is held in increasing order and compressed.

To search the collection for a given term, each *d*-sublist is examined in turn. By doing so, documents may not be examined in turn. This does not matter so long as all documents that would be examined are examined. Further, whole documents may not be examined in turn – this, too, does not matter as many ranking functions can be computed piecewise[1]. To field-restrict a term, a restricted set of sublists is examined. The *w*-sublists are used for proximity searching.

Storing and processing the postings in this way has computational advantages. For a field-restricted search, postings not pertaining to the restriction can be skipped. As postings are stored compressed, they need not even be decompressed. Word postings are used only for proximity searching. On disk the *w*-sublists are collected together and stored after all *d*-sublists. They are not even loaded from disk if not needed.

3. SEARCHING

As the retrieval engine starts up, the corpus tree is loaded and an additional structure is created from it, the field list. For each instance of each tag, the list of nodes at or below that node is collected. For each tag, the same is collected. These lists are then merged and sorted.

Table 1: The field list for the corpus tree given in Figure 1.

Field	Restriction
@c	{4}
@c[1]	{4}
doc	{1, 2, 3, 4, 5, 6, 7}
doc[1]	{1, 2, 3, 4, 5, 6, 7}
p	{3, 4, 6, 7}
p[1]	{3, 4, 6}
p[2]	{7}
sec	{2, 3, 4, 5, 6, 7}
sec[1]	{2, 3, 4, 7}
sec[2]	{5, 6}

The field list for the Figure 1 corpus tree is given in Table 1. From this, a search restricted to 'sec' requires postings at or below all 'sec' nodes of the corpus tree, or where p_n*={2, 3, 4, 5, 6, 7}*. To

[1] BM25 cannot, so the lists are merged then processed.

search in 'p[1]', the postings are needed where $p_n=\{3, 4, 6\}$. For a search restricted to 'p[1] in sec', these two lists are ANDed together (giving $p_n=\{3, 4, 6\}$), and the members of this list are checked against the corpus tree to ensure they satisfy 'p[1] in sec' and not 'sec in p[1]'.

Equivalence tag restrictions are also computed from the field list. The restrictions for each equivalent tag are ORed giving the equivalent restriction. If, for example, 'p[2]' and '@c' were equivalent in Table 1, the restriction would be $p_n=\{4, 7\}$.

Several extensions were added to support element and attribute retrieval:

- Attributes are now distinguished from tags by prefixing attributes with an @ symbol. This symbol was chosen because it makes for easy parsing of INEX queries, which use the same symbol.
- The attribute value is considered to be content lying not only within the attribute, but also the tag. For example, "<tag att="number"> term </tag>", is equivalent to "<tag> <@att> number </@att> term </tag>". In this way, a search for "number in tag" will succeed.
- Tags can now be identified not only by their name and path, but also by the tag instance. Where before it was only possible to restrict to paragraph for example, it is now possible to restrict to the second paragraph.

Trotman [8] suggests the corpus tree will be small for real data. In this extended model this no longer holds true. In the TREC [3] Wall Street Journal collection there are only 20 nodes, for INEX there are 198,041 nodes after 'noise' nodes are removed (4,789 with attributes and instances also removed).

Table 2: Tags ignored during indexing.

ariel	en	item-text	ss
art	entry	label	stanza
b	enum	large	sub
bi	f	li	super
bq	it	line	tbody
bu	item	math	tf
bui	item-bold	proof	tfoot
cen	item-both	rm	tgroup
colspec	item-bullet	rom	thead
couplet	item-diamond	row	theorem
dd	item-letpara	scp	tmath
ddhd	item-mdash	sgmlf	tt
dt	item-numpara	sgmlmath	u
dthd	item-roman	spanspec	ub

Many tags are used to mark elements too small to be relevant. An example of such a tag is 'ref', used to mark references in the text. This tag cannot be relevant to any topic as the contents are simply reference numbers. Some tags were used for visual appearance such as 'b' used to mark text in bold. Others were used as typesetting hints such as 'art' used to specify the size of an image. If any of these tags, or those in Table 2 were encountered during indexing, tagging was ignored (until the matching close tag), but the content still indexed. Tags in this group were hand selected even though automated systems for choosing such tags have been proposed [5].

4. QUERY FORMATION

The title of the topic is extracted and converted into a Boolean query. This query is used to determine which documents to retrieve. Ranking is computed from the postings for the search terms.

For content and structure (CAS) topics, the target element is computed and stored for later use. The complete path for each about-function is computed by concatenating the about-path to the context-element restricting it. All equivalent paths are then computed by permuting this path with the equivalence tags. This fully specified path now replaces the original about-path and the context-element is removed.

At this point, the topic has been transformed from INEX topic syntax into a query whereby each about-clause is Boolean separated and explicitly field restricted.

Create mandatory by ANDing each mandatory term (+)
Create optional by ORing each optional term
Create exclusion by ORing each exclusion term (-)

If all three sub-expressions are non-null, combine:
    mandatory AND (* OR optional) NOT exclusion

If two sub-expressions are non-null, combine using one of:
    mandatory AND (* OR optional)
    optional NOT exclusion
    mandatory NOT exclusion

If only one sub-expression is non-null, use one of:
    mandatory
    optional
    * NOT exclusion

Where '*' finds all documents

Figure 3: Algorithm to convert an about phrase into a Boolean expression.

Examining the about-string, optional, mandatory (+), and exclusion (-) terms are allowed. These terms are converted into a Boolean expression. Optional terms are collected and converted into a sub-expression by ORing ("a b c" → "a OR b OR

c"). Likewise, exclusion terms are also ORed. Mandatory terms are collected and ANDed ("+d +e +f" → "d AND e AND f"). These three sub-expressions are then combined to form a complete about-query. The whole algorithm is presented in figure 3.

Separate about clauses are already Boolean separated so these operators are preserved. Finally, all context-elements must be satisfied so these are ANDed together.

For content only (CO) topics, a Boolean expression is computed exactly as for one about-string using the algorithm presented in Figure 3.

5. RANKING

The retrieval engine is a Boolean ranking hybrid. Result sets are computed in two parts; a bit-string of documents satisfying a strict interpretation of the query, and a set of accumulators holding document weights.

5.1 Document Ranking

The Boolean expression constructed above is converted into a parse tree then evaluated. At each leaf, the posting are loaded and converted into a bit-string, one bit per document.

If a given leaf in the parse tree is not tag-restricted, each posting is examined in turn, and the bit at position d_n of each posting is set. Should the leaf be tag-restricted, only those postings for the given tags are examined (see Section 3) and converted.

The bit-strings are combined at the nodes of the parse tree using the operator there. At the root of the tree, the bit-string has set bits for all documents exactly satisfying the query and unset for those that do not.

The accumulator values are the sum of Okapi BM25 scores computed at each leaf of the parse tree. Scores are summed regardless of the operators in the parse tree.

For AND and OR nodes scores are summed because the influence at these nodes is the sum of influences of the children.

For NOT nodes, they are also summed. If a document is excluded from the result set, the accumulator value is irrelevant. If a document is not, it is either re-included through other terms (e.g. mammal OR (dog NOT cat)), or there is a double negative in the query (e.g. cat NOT (dog NOT cat)). In both cases, the document has successfully satisfied a query leaf so receives a positive weight.

5.2 Element Ranking and Selection

The Boolean ranking hybrid engine was extended to include element ranking. Although whole documents are valid as results for CO topics, SCAS topics specify a target element. This targeting establishes the retrieval unit. If the target element is 'sec', this tag must be returned. It essentially directs the retrieval engine to search and rank each given tag instance separately.

Wilkinson [9] suggests that ranking whole documents then extracting elements from these is a poor ranking strategy. The opposite may hold for this collection. A relevant element lies in the greater context of a relevant document. A relevant document will lie in a relevant journal, which, in turn, lies in a relevant collection. To this end, every paragraph of every section of every document is contextually placed so extracting elements from relevant documents may be a good approach.

The coverage of any one posting is computed as those nodes in the document tree at or above the posting. Each posting is already annotated with a pointer into the tree, p_n. To compute the coverage, the tree is traversed upwards from p_n to the root. Coverage is computed for each document with respect to each search term.

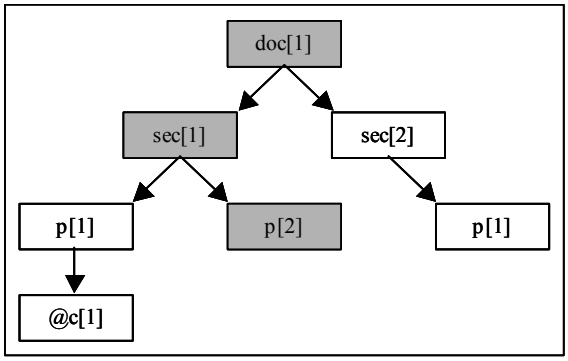

Figure 4: Coverage of a term occurring at p[2]. The coverage includes all those nodes at and above the occurrence node; those parts of the tree that "cover" the term.

In figure 4, the term "ham" occurs at p[2]. The coverage includes all nodes above that point in the document tree. In this example that is sec[1] and doc[1]; all nodes that "cover" the search term – those highlighted in grey.

For each document in the result set, the weighted coverage is computed as the covered branches of the document tree and how many search terms cover that branch. This is computed during a single pass of the indexes by storing the weighted coverage as part of each accumulator.

For the query "eggs and ham" against the documents in Figure 1, the weighted coverage is shown in Figure 5. doc[1] and sec[1] have a weight of 2, while p[1] and p[2] each have a weight of 1.

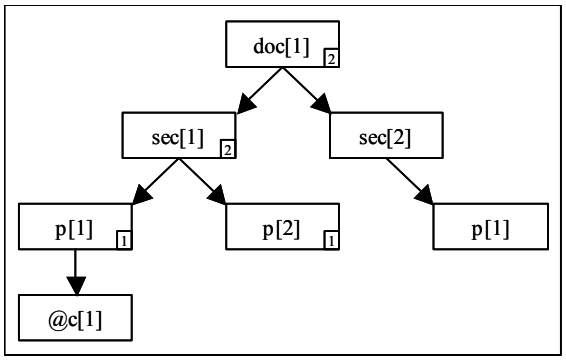

Figure 5: Weighted coverage of each node is the number of search terms that occur at or below that point in the tree. Weighted coverage is shown in the bottom right of those nodes with weights greater than zero.

In any given document, the document root must have the highest weighted coverage, but this can be equal to that of other nodes. For CO topics, all branches of the document tree with coverage less than the root are pruned. The remaining leaves are presented as the result set for that document (in Figure 5, the result is //doc[1]//sec[1]). In this way, the most information dense elements in the document are considered most relevant and no part of any document is returned more than once (overlapping is eliminated).

If a target element is specified in a SCAS topic, all non-target branches are pruned. From the remains, those branches with the highest weighted coverage are presented as the result set for that document.

5.3 Ranking summary

Recall is determined by evaluation of the Boolean expression, documents are then ranked using Okapi BM25, and elements are selected by weighted coverage. As all the metrics needed for ranking are available at search time, the search and rank process is computed in a single pass of the postings.

6. RESULTS

Evaluation results are presented in Table 3.

Table 3: INEX performance measures

Strict	Precision	Rank
CO	0.0243	42nd
SCAS	0.1799	24th
CO-ng-o	0.1359	5th
CO-ng-s	Unknown	Not top 10
Generalized	Precision	Rank
CO	0.0241	34th
SCAS	0.1214	28th
CO-ng-o	0.1542	1st
CO-ng-s	0.1405	5th

The retrieval engine performed badly using the INEX_EVAL measure. This is most likely because this measure treats each tag in a hierarchy as relevant but coverage eliminates overlapping tags – the measure is inappropriate for this retrieval technique.

Good results were shown when performance is measured using INEX_EVAL_NG. NG measures the ratio of relevant to irrelevant information returned. Coverage finds those parts of the document that contain most of the search terms. The correlation between information density and coverage is reflected in the result.

The results show the best performance when generalized quantization is used. This suggests the ordering of the results is not optimal for strict quantization – or the most relevant documents are not ranked before less relevant documents. This may be a consequence of sorting into document order before coverage order.

7. OTAGO AT INEX

The participation process involved the design and contribution of six topics. Of these, four were selected for inclusion in the final topic set. Otago was assigned three of these to assess. The assessment took three people one week each; this was one week per topic.

The retrieval engine described herein was used for designing the contributed topics. This was somewhat problematic as the topic parser was written at the same time the topics were being written, each with few examples.

From the final CAS topic set, 19 required corrections, corrections finally running to 12 rounds! This suggests the topic syntax is unnecessarily complex. See our further contribution [6] for a discussion on a possible language to use for future workshops.

The assessors were overburdened by the multitude of obviously irrelevant documents to assess. Examining some of these documents suggests many retrieval engines were aiming at high recall by retrieving any document containing any of the title terms. In particular, the word ‘java’ appeared in one topic; this was a somewhat popular research area over the years included in the IEEE collection. The assessment task could be reduced by carefully designing topics (and retrieval engines) to avoid this problem.

8. CONCLUSIONS

Element ranking was added to a Boolean ranking hybrid retrieval engine. Relevant documents were identified using Boolean searching. Documents were ranked using Okapi BM25. Finally coverage was used to rank elements within documents.

The results suggest coverage is a good method of identifying relevant and non-overlapping elements. Performance was best for generalized quantization, so ordering is not ideal. This may be a consequence of presenting results in document order.

9. ACKNOWLEDGEMENTS

In addition to the authors, Yerin Yoo contributed to the assessment task. Without her contribution we would not have been able to complete the task.

This work was supported by University of Otago Research Grant (UORG) funding.

REFERENCES

[1] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., & Cowan, J. (2003). Extensible markup language (XML) 1.1 W3C proposed recommendation. The World Wide Web Consortium. Available: http://www.w3.org/TR/2003/PR-xml11-20031105/ [2003.

[2] Fuhr, N., Gövert, N., Kazai, G., & Lalmas, M. (2002). INEX: Initiative for the evaluation of XML retrieval. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*.

[3] Harman, D. (1993). Overview of the first TREC conference. In *Proceedings of the 16th ACM SIGIR Conference on Information Retrieval*, (pp. 36-47).

[4] Hitchcock, S., Quek, F., Carr, L., Hall, W., Witbrock, A., & Tarr, I. (1988). Towards universal linking for electronic journals. *Serials Review,* 24(1), 21-33.

[5] Kazai, G., & Rölleke, T. (2002). A scalable architecture for XML retrieval. In *Proceedings of the 1st workshop of the initiative for the evaluation of XML retrieval (INEX)*, (pp. 49-56).

[6] O'Keefe, R. A., & Trotman, A. (2003). The simplest query language that could possibly work. In *Proceedings of the 2nd workshop of the initiative for the evaluation of XML retrieval (INEX)*.

[7] Robertson, S. E., Walker, S., Beaulieu, M. M., Gatford, M., & Payne, A. (1995). Okapi at TREC-4. In *Proceedings of the 4th Text REtrieval Conference (TREC-4)*, (pp. 73-96).

[8] Trotman, A. (2003). Searching structured documents. *Information Processing & Management,* (to appear) doi:10.1016/S0306-4573(03)00041-4, available on ScienceDirect since 6 June 2003.

[9] Wilkinson, R. (1994). Effective retrieval of structured documents. In *Proceedings of the 17th ACM SIGIR Conference on Information Retrieval*, (pp. 311-317).