Flexible Retrieval Based on the Vector Space Model

Carolyn J. Crouch, Aniruddha Mahajan, and Archana Bellamkonda

Department of Computer Science,
University of Minnesota Duluth,
Duluth, MN 55812,
(218) 726-7607
ccrouch@d.umn.edu

Abstract. This paper describes the current state of our system for structured retrieval. The system itself is based on an extension of the vector space model initially proposed by Fox [5]. The basic functions are performed using the Smart experimental retrieval system [11]. The major advance achieved this year is the inclusion of a flexible capability, which allows the system to retrieve at a desired level of granularity (i.e., at the element level). The quality of the resultant statistics is largely dependent on issues (in particular, ranking) which have yet to be resolved.

1 Introduction

Our original goal when we began our work with INEX in 2002 was to assess the utility of Salton's vector space model [12] in its extended form for XML retrieval. Familiarity with Smart [11] and faith in its capabilities led us to believe that it might prove useful in this environment. Early results [2, 3] led us to believe that such a system could be utilized for XML retrieval if particular problems (e.g., flexible retrieval, ranking issues) could be solved. During 2002, much effort was spent on the translation of documents and topics from XML to internal Smart format and then back again into INEX reporting format. In 2003, we produced an operational system, but it did not include a flexible component (that is, it could only retrieve at the document level). During 2004, query formulation (for both CO and CAS queries) was completely automated. CAS queries received special attention to insure that all conditions of the query were met [1]. Our major improvement was the design and implementation of a flexible capability which allows the system to retrieve elements at various degrees of granularity [10]. Investigations with respect to relevance feedback in a structured environment were initiated.

We now have a system which, we believe, has the potential to function well in the XML environment. Significant issues, which stand to impact results markedly, remain open to investigation. These include, in particular, ranking and length normalization.

2 Background

One of the basic models in information retrieval is the vector space model, wherein documents and queries are represented as weighted term vectors. The weight as-

N. Fuhr et al. (Eds.): INEX 2004, LNCS 3493, pp. 292 – 302, 2005.

signed to a term is indicative of the contribution of that term to the meaning of the document. Very commonly, *tf-idf* weights [13] or some variation thereof [14] are used. The similarity between vectors (e.g., document and query) is represented by the mathematical similarity of the corresponding term vectors.

In 1983, Fox [5] proposed an extension of the vector space model—the so-called extended vector space model—to allow for the incorporation of objective identifiers with content identifiers in the representation of a document. An extended vector can include different classes of information about a document, such as author name, date of publication, etc., along with content terms. In this model, a document vector consists of a set of subvectors, where each subvector represents a different class of information. Our current representation of an XML document/query consists of 18 subvectors (*abs, ack, article_au_fnm, article_au_snm, atl, au_aff, bdy, bibl_atl, bibl_au_fnm, bibl_au_snm,, bibl_ti*, *ed_aff, ed_intro, kwd, pub_yr, reviewer_name, ti, vt*) as defined in INEX guidelines. These subvectors represent the properties of the document or article. Of the 18, eight are subjective, that is, contain content-bearing terms: *abs, ack, atl, bdy, bibl_atl, bibl_ti, ed_intro, kwd* (abstract, acknowledgements, article title, body, title of article cited in the bibliography, title of publication containing this article [i.e., journal title], editorial introduction, and keywords, respectively). Similarity between extended vectors in this case is calculated as a linear combination of the similarities of the corresponding subjective subvectors. (The objective subvectors serve here only as filters on the result set returned by CAS queries. That is, when a ranked set of elements is retrieved in response to a query, the objective subvectors are used as filters to guarantee that only elements meeting the specified criteria are returned to the user.)

Use of the extended vector model for document retrieval normally raises at least two issues: the construction of the extended search request [4, 6] and the selection of the coefficients for combining subvector similarities. For XML retrieval, of course, the query is posed in a form that is easily translated into an extended vector. The second problem—the weighting of the subvectors themselves—requires some experimentation. Experiments performed with the 2003 INEX topic set identified the following subjective subvectors as being particularly useful for retrieval: *abs, atl, bdy, bibl_atl, kwd.* We found our best results were obtained with subvector weights of 1, 1, 2, 2, and 1, respectively. (The three remaining subjective subvectors received 0 weights.) More investigation is required in this area with respect to the 2004 topics.

Another issue of interest here is the weighting of terms within subjective subvectors. Experiments indicated that the best results were achieved for the 2003 topics with the respect to both article and paragraph indexings when *Lnu.ltu* term weighting [15] was used. Our 2004 results are based on *Lnu.ltu* term weighting, as defined below:

$$\frac{(1 + \log(tf)) \,/\, (1 + \log(\text{average } tf))}{(1 - \text{slope}) * \text{pivot} + \text{slope} * (\#\text{ unique terms})}$$

where *tf* represents term frequency, slope is an empirically determined constant, and pivot is the average number of unique terms per document, calculated across the entire collection.

3 System Description

Our system handles the processing of XML text as follows:

3.1 Parsing

The documents are parsed using a simple XML parser available on the web.

3.2 Translation to Extended Vector Format

The documents and queries are translated into Smart format and indexed by Smart as extended vectors. We selected the paragraph as our basic indexing unit in the early stages. Thus a typical vector in this system (based on a paragraph indexing) consists of a set of subjective and objective subvectors with a paragraph in the *bdy* subvector. (Other indexing units were later added to include section titles, tables, figure captions, abstracts, and lists.) *Lnu-ltu* term weighting is applied to all subjective subvectors.

3.3 Initial Retrieval

Retrieval takes place by running the queries against the indexed collection using Smart. The queries used in the initial retrieval are simple (rather than extended) vector queries. That is, each query consists of search terms distributed only in the *bdy* subvector. The result is a list of *elements* (paragraphs) ordered by decreasing similarity to the query. Consider all the elements in this list with a non-zero correlation with the query. Each such element represents a terminal node (e.g., paragraph) in the body of a document with some relationship to the query.

3.4 Flexible Retrieval

A basic requirement of INEX is that the retrieval method must return to the user components of documents or elements (i.e., abstract, paragraphs, sub-sections, sections, bodies, articles, figure titles, section titles, and introductory paragraphs) rather than just the document itself. The object is to return the most relevant element(s) in response to a query. Thus a good flexible system should return a mixture of document components (elements) to the user. These elements should be returned in rank order. The method to determine rank should incorporate both exhaustivity (relevance) and specificity (coverage).

Our flexible retrieval module (which we call Flex), is designed as follows. It takes as input a list of elements (e.g., paragraphs), rank-ordered by similarity to the query as determined by Smart in the initial retrieval stage. Each such element represents a leaf of a tree; each tree represents an article in the document collection. (The query at this stage, used to determine correlation with a paragraph, is a simple subvector query; that is, it consists only of search terms distributed in the *bdy* subvector.)

Consider Figure 1, which represents the tree structure of a typical XML article. The root of the tree is the article itself, whereas the leaf nodes are the paragraphs. Flexible retrieval should return relevant document components (e.g., <sec>, <ss1>, <ss2>, <p>, <bdy>, <article>) as shown in Figure 1. In order to determine which components or elements of a tree to return, the system must first build the tree

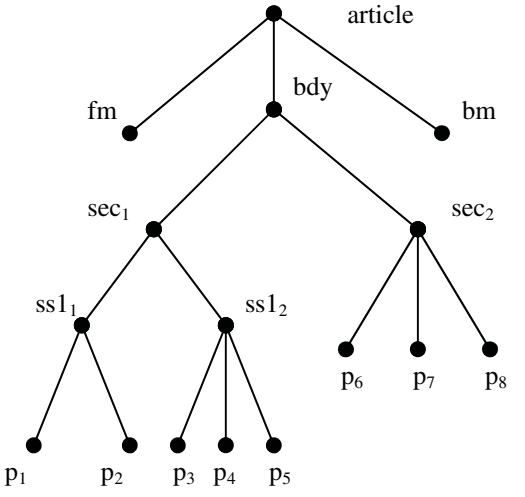

Fig. 1. Tree Structure of a Typical XML Document

structure and then populate the tree by assigning a value to each non-terminal node in the tree. (The value in this case represents a function of exhaustivity and specificity.)

Flex takes a bottom-up approach. All leaf elements having non-zero correlations with the query have already been assigned similarity values by Smart. Consider the set of all such leaf elements which belong to the same tree. For a particular query, trees are constructed for all articles with leaves in this set. To construct the trees (and deal with the issue of specificity [coverage]), at each level of the tree, the number of siblings of a node must be known. The process is straight-forward. Suppose for example in Figure 1 that p_1, p_2, and p_7 were retrieved as leaf elements of the same tree. Flex would then build the tree represented in Figure 2. Any node on a path between a retrieved terminal node and the root node (*article*) is a valid candidate (element) for retrieval.

Building the tree is simple; populating it is not. We have weights (similarity values) associated with all terminal nodes. We consider each such value representative of that node's exhaustivity (e-value) with respect to the query. A question that arises at this point is how to produce a corresponding value representing specificity (s-value) for this node. Our current approach assigns an s-value equal to the e-value for that node. Since all the elements in question here (i.e., the terminal nodes) are relatively small in terms of the number of word types contained therein, this appears to be a reasonable initial approach.

Now that all the terminal nodes of the document tree have associated e-values and s-values, populating the tree (i.e., assigning e- and s-values to the rest of the nodes) begins. For every leaf node, we find its parent and determine the e- and s-values for that parent. The values of a parent are dependent on those of its children (i.e., relevance is propagated up the tree, whereas coverage may diminish as a function of the

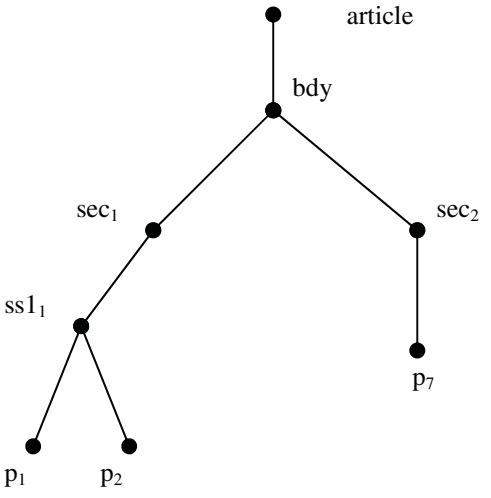

Fig. 2. Tree of Relevant Elements

total number of children [relevant and non-relevant] of a node). The process continues until all the nodes on a path from a leaf to the root (the *article* node) have been assigned e- and s-values. .

After all trees have been populated (and we have e- and s-values associated with each node of each tree), one major problem remains. How can we produce a rank-ordered list of elements (document components) from these populated trees? We need a method which ranks the document components on the basis of e- and s-value. We considered a number of approaches. Our current method uses a simple function of e- and s-value to produce a single value for ranking. (The description given here presents the logical view of Flex; see [10] for a detailed view of Flex implementation.)

3.5 Rank Adjustment

Once a rank-ordered list of elements is produced by Flex, we can undertake the final step in the retrieval process. Initial retrieval produces a set of terminal node elements (paragraphs) having some relationship with the query. Flex produces the expanded set of elements (subsections, sections bodies) associated with those terminal nodes. Taken together, this is the set of potentially relevant elements associated with the query.

Up to this point, only the *bdy* subvector has been utilized in the retrieval process. We can now use the remaining subjective subvectors to adjust the ranking of the element set. By correlating the extended vector representation of the query with the extended vector representation of each element in the returned set, another and potentially more accurate ranking is produced. This step, rank adjustment, is not yet implemented in our current system.

Once a rank-ordered list is produced, the elements are reported for INEX evaluation.

4 Experiments

In the following sections, we describe the experiments performed with respect to the processing of the CO and CAS topics, respectively. In all cases, we use only the topic title as search words in query construction. Term weighting is *Lnu.ltu* in all cases. All indexing is done at the paragraph (basic indexing unit) level unless otherwise specified. No rank adjustment is performed.

4.1 Using CO Topics

Our initial experiments using flexible retrieval were performed using the 2003 topic set. We used several simple methods for calculating e- and s-values and propagating these values up the tree. Our 2004 results are based on one of those methods, wherein the e-value of a node is calculated as the sum of the e-values of its children whereas its s-value is the average of the s-values of its children. Final ranking of a node is based on the product of its e-value and s-value. (The calculation of e- and s-values, their propagation, and the ranking of results will be a focus of attention in the coming year.)

Results indicated that flexible retrieval produced an improvement over document retrieval for the 2003 topic set. The approach was subsequently applied to the 2004 topic set as seen in Table 1. Results achieved by flexible retrieval (labeled Flex) are compared with the results retrieved by the same *Lnu.ltu* weighted query set against various *Lnu.ltu* weighted indexings of the documents (at the article, section, subsection, and paragraph levels, respectively). These results improve to 0.06 (average precision-recall) when the input to Flex is filtered so that trees are built only when a terminal node (paragraph) occurs in one of the top 500 documents retrieved by the query in an article-based indexing. The last entry in Table 1 utilizes an all-element index (an indexing of document elements at all levels of granularity—the union of the article, section, subsection, and paragraph indices). See Figure 3 for more detail.

Table 1. CO Processing (2004 Topic Set)

Indexing	Avg Recall-Precision	Generalized Recall
Article	0.02	4.04
Sec	0.02	28.89
Subsec	0.03	26.57
Para	0.03	32.94
Flex (on para)	0.05	34.94
All Elem	0.06	38.31

It is worth noting that filtering the input to Flex, which with respect to a specific query reduces the number of trees built and ensures that each such tree represents a document in the top (in this case, 500) documents retrieved by that query, produces a average recall-precision of 0.06—the same value produced by a search of the all-element index.

4.2 Using CAS Topics

We process CAS topics in much the same fashion as CO topics, with some important exceptions. During pre-processing of the CAS queries, the subjective and objective portions of the query and the element to be returned (e.g., abstract, section, paragraph) are identified.

Depending on its syntax, a CAS query can be divided into parts, which can be divided into subparts depending on the number of search fields. Further subdivision, depending on the presence of plus or minus signs (representing terms that should or should not be present) preceding a search term, is also possible. CAS preprocessing splits the query into the required number of parts, each of which is processed as a separate Smart query. For example, suppose the query specifies that a search term is to be searched for in field 1. If the field to be searched is an objective subvector, the search term is distributed in that subvector. If the search field specifies a specific subjective subvector, the search term is distributed in that subvector, otherwise the search takes place in the paragraph subvector. The result in this last case is a set of elements (terminal nodes) returned by the Smart search which is used as input to Flex. Flex produces a ranked set of elements (terminal, intermediate, and/or root nodes) as the final output of this small search. After all subsearches associated with the query are complete, the final result set is produced (i.e., the original query is resolved). The element specified for return in the original query is then returned from each element in the result set. See [1] for more details.

CAS processing using flexible retrieval based on a paragraph indexing was applied using the 2004 CAS topic set. The first entry in Table 2 reports the result. In this experiment, the structural constraints specified in the query are strictly maintained (i.e., only the element specified by the query is returned). That is, Flex is run on the paragraph indexing, objective constraint(s) applied, and the element specified is returned from that result set. The second entry in this table shows the result when no structural constraints are maintained (that is, all relevant elements, regardless of type, are returned from the result set). The next two entries duplicate experiments 1 and 2, but instead of running Flex on the paragraph index retrieve on the all-element index. The last

Table 2. CAS Processing (2004 Topic Set)

Indexing	Avg Recall-Precision	Generalized Recall
Flex (on para) constraints maintained	0.04	27.99
Flex (on para) constraints ignored	0.04	9.93
All Elem constraints maintained	0.02	18.33
All Elem constraints ignored	0.02	13.40
All Elem CAS as CO	0.05	36.30

experiment reports on results obtained by relaxing the conditions specified in the query. It treats a CAS query essentially as if it were a CO query. Instead of breaking the query into subqueries and processing each part separately, it combines the search terms and searches the all-element index with the combined terms. See Figure 4 for more detail.

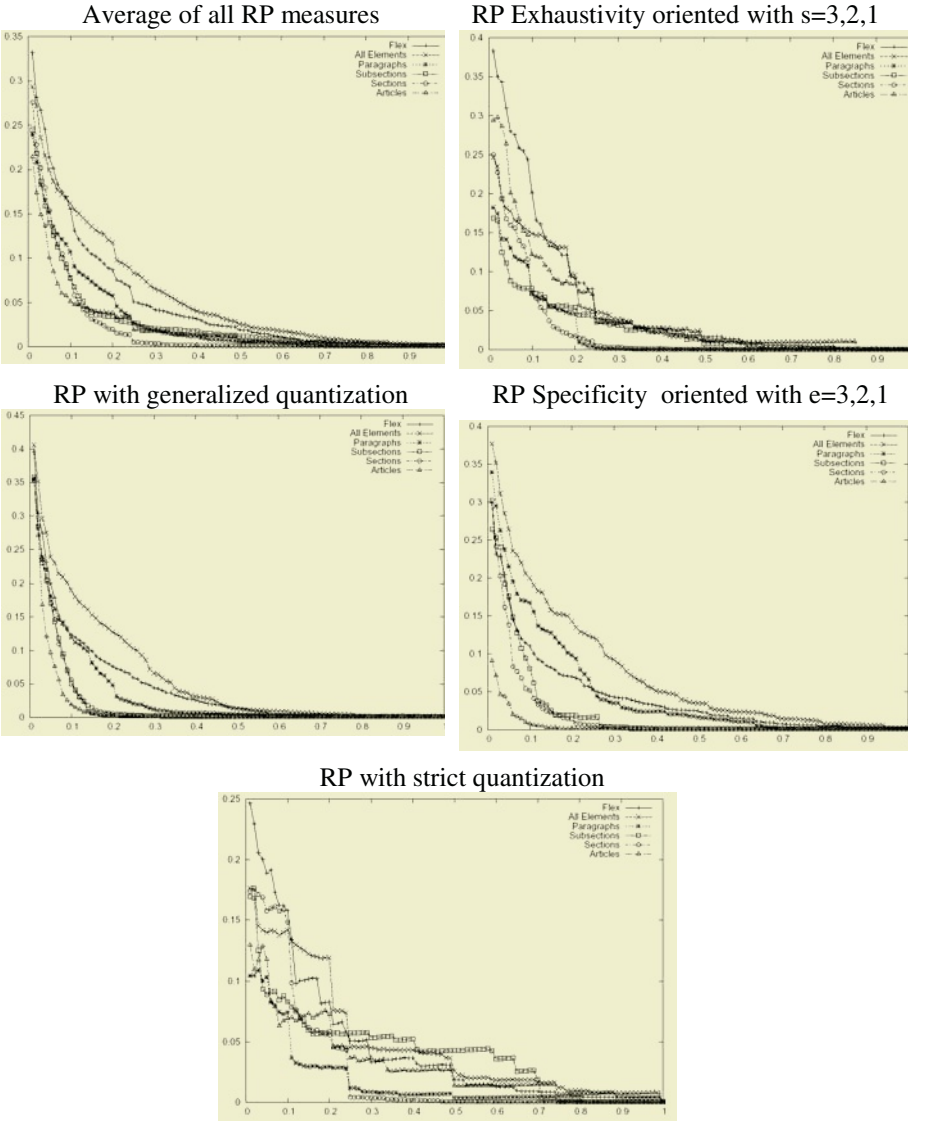

Fig. 3. Comparison of CO Results

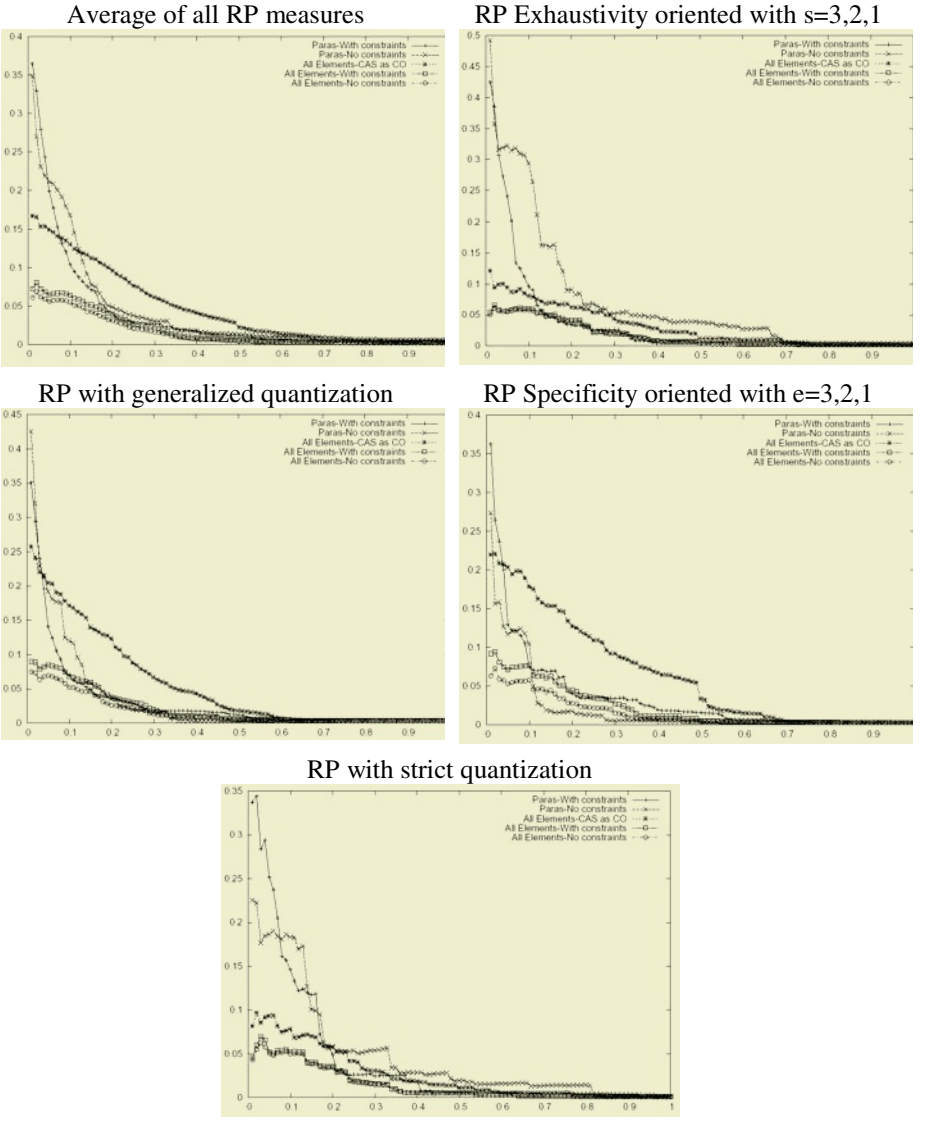

Fig. 4. Comparison of VCAS Results

4.3 Results

During 2004, our work centered on two important aspects of the INEX ad hoc task: retrieving at the element level (i.e., flexible retrieval) and insuring that the result returned from a CAS search met the search requirements (thus producing improvements under SCAS but not necessarily under VCAS). We note that our 2004 results are not competitive (as opposed to 2003 when we were only able to retrieve at the article level). This year, with flexible retrieval based on a paragraph indexing, our current

ranking scheme favors the return of smaller elements rather than larger ones. Yet as Kamps, *et.al*. [8] clearly show, in 2003 the probability of relevance increased with element length. Our method of propagating values and ranking elements needs to take this into consideration. It appears that using an all-element index improves results. However, the very large index required, the redundancy within it, and the inefficiencies of storage and search time which result leave us to conclude that flexible retrieval, which produces a result dynamically based on a single index, is preferred if it can produce even a comparable result.

4.4 Relevance Feedback in INEX

The importance and value of relevance feedback in conventional information retrieval have long been recognized. Incorporating relevance feedback techniques within the domain of structured retrieval is an interesting challenge.

Working within the constraints of our system, the first question which arises is how to translate the exhaustivity and specificity values associated with each INEX element into an appropriate measure of relevance in our system. Conventional retrieval views relevance assessment as binary. In INEX, we have a range of values for exhaustivity and a corresponding set of values for specificity. There are many possibilities to consider when mapping these values to relevance.

As an initial approach, we decided simply to recognize as relevant those paragraphs with e-values of 3. In other words, we recognize as relevant to the query all elements that are highly exhaustive (disregarding other combinations). We were interested in determining the feasibility of this approach, which depends strongly on having enough of these elements available in the top ranks of retrieved elements. We completed a run using Rocchio's algorithm ($\alpha = \beta = 1$, $\gamma = 0.5$) on a paragraph index with the relevance assessments of the top 20 paragraphs used when constructing the feedback query. The query set consisted of the CO queries in their simple form with search terms distributed in the *bdy* subvector only. The result of the feedback iteration is a set of paragraphs rank-ordered by correlation with the query. The feedback iteration produces an average recall-precision of 0.03 compared to 0.02 in the base case. Flex is then run on this set to produce the associated elements, yielding an aggregate score of 0.04. We look forward to producing more experiments and results next year.

5 Conclusions

Our system, as a result of work done in the past year, is now returning results at the element level, i.e., retrieving at the desired level of granularity. The incorporation of a flexible retrieval facility is required before meaningful INEX experiments can take place. However, there are a number of problems still to be solved with this system, including in particular the propagation of e- and s-values upwards through the document hierarchy and the ranking of elements based on those values. Various approaches have been suggested [7, 9]. Much of our work in the coming year will focus on this issue. A good working result is important, since regardless of how well it does everything else, in the end all results depend on the system's ability to retrieve good

elements. We believe that the initial retrieval through Smart produces valid terminal nodes with meaningful e-values. How well we build on this foundation will determine the final utility of the system.

A second area of interest is the extension of relevance feedback techniques to structured retrieval. There are many interesting questions to be addressed in this area in the coming year.

References

[1] Bellamkonda, A. Automation of Content-and-Structure query processing. Master's Thesis, Dept. of Computer Science, University of Minnesota Duluth (2004). http://www.d.umn.edu/cs/thesis/bellamkonda.pdf

[2] Crouch, C., Apte, S., and Bapat, H. Using the extended vector model for XML retrieval. In *Proc of the First Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, (Schloss Dagstuhl, 2002), 99-104.

[3] Crouch, C., Apte, S. and Bapat, H. An approach to structured retrieval based on the extended vector model. In *Proc of the Second Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, (Schloss Dagstuhl, 2003), 87-93.

[4] Crouch, C., Crouch, D. and Nareddy, K. The automatic generation of extended queries. In *Proc. of the 13th Annual International ACM SIGIR Conference*, (Brussels, 1990), 369-383.

[5] Fox, E. A. Extending the Boolean and vector space models of information retrieval with p-norm queries and multiple concept types. Ph.D. Dissertation, Department of Computer Science, Cornell University (1983).

[6] Fox, E., Nunn, G. and Lee, W. Coefficients for combining concept classes in a collection. *In Proc. of the 11th Annual International ACM SIGIR Conference*, (Grenoble, 1988), 291-307.

[7] Fuhr, N. , and Grossjohann, K. XIRQL: A query language for information retrieval in XML documents. In *Proc of the 24th Annual International ACM SIGIR Conference*, (New Orleans, 2001), 172-180.

[8] Kamps, J., de Rijke, M., and Sigurbjornsson, B. Length normalization in XML retrieval. *In Proc of the 27th Annual International ACM SIGIR Conference* (Sheffield, England, 2004), 80-87.

[9] Liu, S., Zou, Q., and Chu, W. Configurable indexing and ranking for XML information retrieval. *In Proc of the 27th Annual International ACM SIGIR Conference* (Sheffield, England, 2004), 88-95.

[10] Mahajan, Aniruddha. Flexible retrieval in a structured environment. Master's Thesis, Dept. of Computer Science, University of Minnesota Duluth (2004). http://www.d.umn.edu/ cs/thesis/mahajan.pdf

[11] Salton, G. *Automatic information organization and retrieval*. Addison-Wesley, Reading PA (1968).

[12] Salton, G., Wong, A., and Yang, C. S. A vector space model for automatic indexing. *Comm. ACM* 18, 11 (1975), 613-620.

[13] Salton, G. and Buckley, C. Term weighting approaches in automatic text retrieval. In *IP&M* 24, 5 (1988), 513-523.

[14] Singhal, A. AT&T at TREC-6. In *The Sixth Text REtrieval Conf (TREC-6)*, NIST SP 500-240 (1998), 215-225.

[15] Singhal, A., Buckley, C., and Mitra, M. Pivoted document length normalization. In *Proc. of the 19th Annual International ACM SIGIR Conference*, (Zurich,1996), 21-19.