# An Evaluation of the Configurable Indexing and Ranking System for Retrieving Information from XML Documents

Shaorong Liu, Qinghua Zou and Wesley Chu

UCLA Computer Science Department, Los Angeles, CA, USA 90095
{sliu, zou, wwc}@cs.ucla.edu

**Abstract.** Indexing XML documents and ranking query results are two key factors for efficient and effective XML information retrieval. Inappropriate indexing of tags can result in false negatives; and improper weighting of content may lead to a low precision/recall. To remedy these problems, we propose a configurable framework for XML information retrieval [13]. In this framework, users can specify appropriate index types and assign proper weights to different tags, which in turn adjusts the weight of a term based on its location in XML documents. In this paper, we have tested all 30 INEX 03 Structure and Content (CAS) topics on the INEX dataset with a set of ranking configurations. Our approach has significantly high precision when recall is low and achieves the highest average precision/recall (0.3309) compared to all 38 official INEX 03 submissions with the strict quantization of the evaluation metrics. Our experimental results reveal that properly tuning tag weights can significantly improve precision/recall.

**Keywords:** INEX, XML Information Retrieval, XML Indexing and XML Ranking

## 1 Introduction

As the World Wide Web (WWW) becomes a major means of disseminating and sharing information, there is an exponential increase in the amount of data in web-compliant formats such as HyperText Markup Language (HTML) and Extensible Markup Language (XML). XML is essentially a textual representation of the hierarchical (tree-like) data where a meaningful piece of data is bounded by matching starting and ending tags, such as <name> and </name>. Due to the simplicity of XML as compared to SGML and its relative expressiveness as compared to HTML, XML has become the most popular format for information representation and data exchange on the web.

To cope with the tree-like structures in the XML data model, much research has been done in the Information Retrieval (IR) community on providing flexible retrieval mechanisms to extract information from XML documents [e.g., 2, 4-9, 14, 15]. The INitative for the Evaluation of XML Retrieval (INEX) [18], for example, was proposed in April, 2002 and has prompted XML researchers worldwide to promote the evaluation of effective XML retrieval.

As opposed to traditional IR which focuses on text retrieval, XML information retrieval supports both Content Only (CO) queries and Content and Structure (CAS) queries. CAS queries enable users to specify queries more precisely than traditional CO queries. For example, users can retrieve relevant documents with a keyword or phrase within a specific context. CAS queries, however, introduce new challenges of indexing XML structures for efficient retrieval.

In addition, as oppose to traditional IR which has only one data type, i.e., plain text, XML supports various data types, such as plain text, numbers, date and time. Thus we need multiple content processing methods and indexing types for the heterogeneous content in XML documents to support various search predicates.

Further, not all tags in an XML document are semantically meaningful [1]. Improper indexing of non-semantic tags can result in false negatives. Thus, we need a user-configurable framework to differentiate semantically meaningful tags from tags used for presentation purposes only.

Finally, a query result from XML information retrieval may not always be an entire document. It can be any deeply nested XML element, i.e., dynamic document [14]. The traditional static document ranking method cannot support the varying granularity results, thus we need a new ranking method to deal with this problem.

Thus, we propose a configurable indexing and ranking framework for XML information retrieval. In this framework, XML documents are first "scanned" to collect statistics about tags and contents for each group of similar elements. These statistics are stored in spreadsheets and presented to users. Users can specify tag index types, content processing operations and index types for each group of similar elements based on the statistics. With the index configurations, the system transforms XML document trees into a compact indexing tree, Ctree, which provides both path summaries and detailed element-to-element relationships in XML document trees. Therefore, Ctree indexes can answer most structured queries very efficiently without accessing the original XML documents. In addition, to support the dynamic document concept in XML, we extend the classic Vector Space Model (VSM) [12] in traditional IR to the XML model and propose the concepts of "weighted term frequency," $tf_w$, and "inverted element frequency," *ief*. The extended VSM supports the ranking of XML query results of varying granularity.

To empirically evaluate the effectiveness of our proposed system, we have conducted a set of experiments on the INEX 03 dataset with CAS topics. Our experimental results show that our methodology is able to obtain significantly high precision at low recall and has the highest average precision/recall (0.3309) compared with all 38 official INEX 03 submissions using the strict quantization of the evaluation metrics.

The rest of the paper is organized as follows. In section 2, we introduce the XML data model and Ctree. Section 3 overviews our configurable indexing and ranking system for XML information retrieval. Section 4 describes our experiment setup and retrieval performance studies with varying ranking configurations. Section 5 discusses related work. Finally, we conclude and discuss our future work in section 6.

# 2 Background

## 2.1 XML data model

We model an XML document as an ordered, labeled tree where each element (attribute) is represented as a node and each element-to-subelement or element-to-attribute relationship is represented as an edge between the corresponding nodes. We assume that each node is a triple (*id*, *label*, <*value*>), where *id* uniquely identifies the node, *label* is the name of the corresponding element or attribute, and *value* is the corresponding element's text content or attribute's value. The value is optional because not every element has text content. Note that in the paper, we treat an attribute as a sub-element of an element and a reference IDREF as a special type of value.

For example, Figure 1 shows a sample XML document tree with 25 nodes numbered from 0 to 24. Each circle represents a node with the node id inside the circle and the label beside the circle. To distinguish text content from element (attribute) nodes, the value of a node is linked to the node by a dotted line. We now introduce the definitions for *label path* and *equivalent nodes* that will be used for describing Ctree in Section 2.2.

**Definition 1** (**Label Path**) A label path for a node *v* in an XML document tree is a list of dot-separated labels of the nodes on the path from the root node to *v*. For example, node 12 in Figure 1 can be reached from the root node through the path: node 0→10→11→12. Thus the label path for node 12 is *articles.article.fm.yr*.

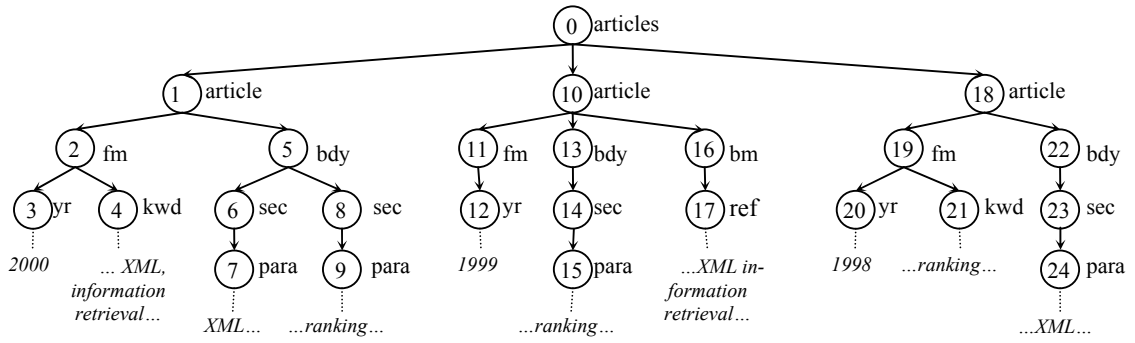



**Figure 1: An example of XML document tree.**

**Definition 2** (**Equivalent Nodes**) Two nodes in an XML document tree are equivalent if their label paths are the same. For example, nodes 3 and 12 in Figure 1 are equivalent since their label paths are the same.

## 2.2 Ctree

Indexing the structure of XML documents is very important in answering structured XML queries but so far has not received enough attention in the IR field. Many current indexing methods create indices only on the predefined nodes (e.g., [7]), such as leaf level nodes. Such approaches are simple and efficient, but sometimes may not be flexible enough to support queries with any structure constraint and to retrieve nodes at any level.

To overcome this problem, we transform an XML document tree *D* into a compact indexing tree, Ctree [16]. A Ctree is a two-level bidirectional tree: group level and element level. The group level provides path summaries for the XML document tree and contains edges from parent groups to their child groups. The element level provides detailed element-to-element relationships and has links pointing from child elements to their corresponding parent elements.

Similar to most path index approaches (e.g., DataGuide [10]), the first step in Ctree construction is to cluster equivalent nodes in *D* into groups. If the label path of group A is the longest prefix of that of group B, then there is an edge linking from group A to group B. For example, the path summary for the XML document tree (Figure 1) is shown in Figure 2a. Each group is represented as a dotted box with its *label* above the box. The numbers inside each dotted box are the node identifiers from Figure 1. For instance, nodes 3, 12, and 20 in Figure 1 share the same label path and are thus in the same group *yr* in Figure 2a.

As shown in past research, such path summaries greatly facilitate the evaluation of simple path expressions (i.e., path expressions with a single branch and without filters) by searching only relevant parts of the tree. For example, for a query ($Q_1$) */articles/article/bdy/sec*, the path summary in Figure 2a implies that all the nodes in group *sec* are the answers since their label paths match $Q_1$.

Such path summaries, however, are insufficient for answering non-simple path expressions due to their incompleteness. They do not preserve the hierarchical relationships among individual nodes in an XML document tree. For example, with the path summary in Figure 2a, we cannot determine the hierarchical relationships between node 19 in group *fm* and node 21 in group *kwd*. Such relationships, however, are important in answering non-simple path expressions. For example, for a query ($Q_2$): */articles/article/fm[.kwd]* (i.e., find an article's *fm* part that has a keyword part), the path summaries in Figure 2a indicate that nodes in group *fm* are candidate answers. We cannot determine, however, which nodes in group *fm* can answer $Q_2$ unless the hierarchical relationships between the individual nodes in group *fm* and those in group *kwd* are provided.

Therefore, the second step in Ctree construction is to order the nodes in a group into a list by their corresponding preorders in *D*. We shall call the nodes in a Ctree group as elements for differentiating them from the nodes on an XML document tree. The elements in a group list are accessible by their corresponding indexes. The index for an element *e* in a group

*g* is its relative order in *g*. Instead of storing the node identifiers in each group, Ctree stores the indexes of these elements' corresponding parent elements in the parent group of *g*. Since the root element in the root group has no parent, we set the value in the root group to -1.

Figure 2b shows the corresponding Ctree for the XML document tree in Figure 1, where each dotted box represents a group with its *id* and *label* above the box. Each box contains two columns of values: element indexes in the left column and their corresponding values in the right column. For simplicity, we use the notation *g:e* to represent an element in group *g* with index *e*. For example, nodes 9 and 8 in the path summaries (Figure 2a) correspond to the elements *g:e* = 7:1 and *g:e* = 6:1 in the Ctree (Figure 2b). From the Ctree, we can read that 6:1 is the parent of 7:1 since the value of 7:1 is 1 and the parent group of group 7 is group 6.

With the Ctree in Figure 2b, we can answer not only simple path expressions but also non-simple path expressions very efficiently without accessing XML document trees. For example, to answer $Q_2$, the values in group *kwd* imply that element *g:e* = 2:0 and *g:e* = 2:2 are the answers.

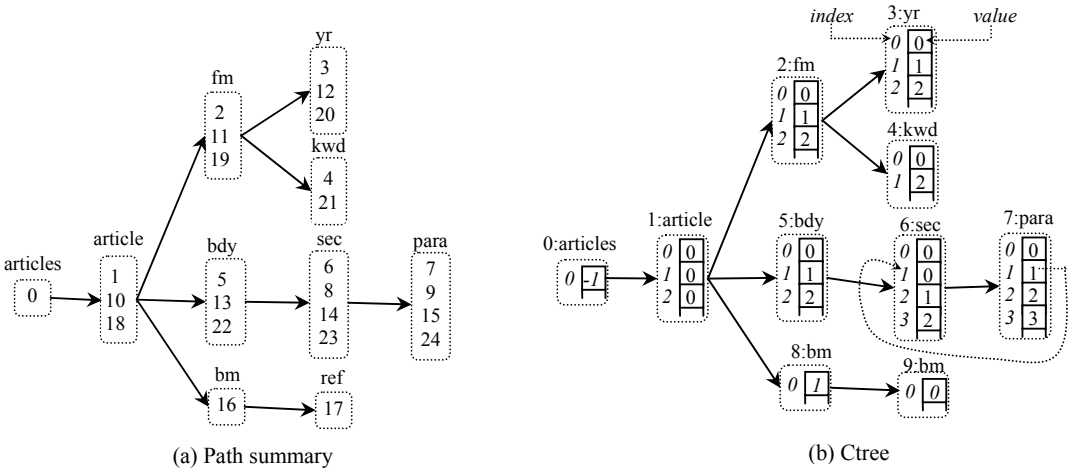



**Figure 2: The path summary and the Ctree for the XML document tree in Figure 1.**

## 3 A configurable XML information retrieval system

Figure 3 shows the architecture of the configurable XML information retrieval system, which performs two types of functions: document indexing and query evaluation.

Indices for a collection of XML documents can be built in the following three steps. First, XML documents are sent to a *Scan* module to collect statistics about the structure and content characteristics of the XML documents. These statistics are then stored in an Excel spreadsheet and presented to a user. Second, based on the statistics, a user configures index options for each group of equivalent nodes. Finally, based on the index configurations, the *Index Builder* correspondingly constructs a Ctree and builds content indexes. For complex datasets such as the INEX dataset, a user may be not familiar with the dataset characteristics even with the collected statistics. In this case, a user can either use the default index options or leverage on the index options provided by a domain expert.

The *Query Evaluation* module evaluates the incoming query based on the Ctree's structure and content indices, and then ranks the results based on users' ranking configurations to obtain a list of ranked results.

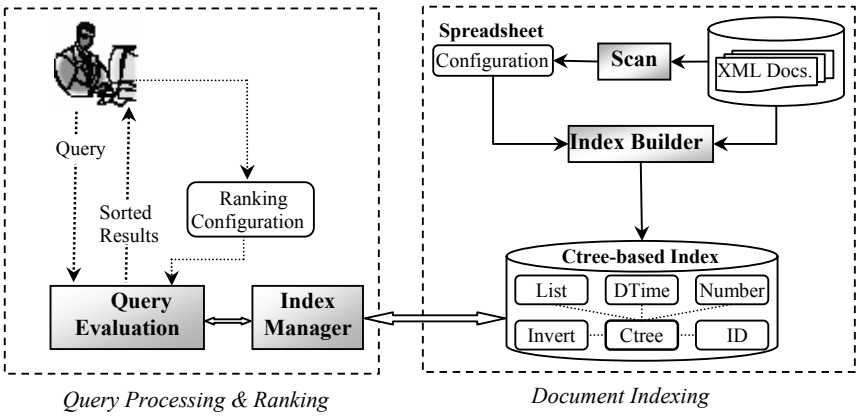



**Figure 3: Overview of the system architecture.**

## 3.1 Document indexing

### *3.1.1 Scan*

Many XML datasets, such as INEX dataset, do not have schemas with specific data types for each element and attribute. To facilitate index configurations for such datasets, we collect statistics from XML documents in a *Scan* module while parsing. Since equivalent nodes in XML document trees share similar characteristics, the *Scan* module collects the structure and content statistics for each group of equivalent nodes.

With the collected statistics, users can determine the index types for each group based on the statistics of this group. If an XML dataset contains too many groups of equivalent nodes, such as INEX dataset which contains 13262 distinct groups, then configuration can be based on each group of nodes with the same label. For instance, there are only 204 distinct node labels in the INEX dataset.

### *3.1.2 Configuration*

With the collected statistics, a user can specify a tag index type, a set of value preprocessing operations, and a value index type for each group of equivalent nodes or nodes with the same label, as described in the following:

- Tag Index Type: *Index* or *No Index*. The tag index type is for users to indicate whether to keep or ignore the tag during indexing.
- Value Preprocessing Operations: 1) *TokenType*: whether selecting digit, word or mixed tokens in the values for indexing; 2) *IsStopping*, whether removing stop words; 3) *IsStemming*: whether using stemming functions; and 4) *IsToLower*: whether transforming a text to its lower case.
- Value Index Type: 1) *No Index*; 2) *Invert*; 3) *Number*; 4) *DTime*; 5) *List* or 6) *ID*. *No Index* means the values for nodes in this group will be ignored during indexing and the latter five value index types will be explained in Section 3.1.3.

### *3.1.3 Value index types*

The heterogeneous nature of element contents and attribute values in XML documents requires multiple value index types. Thus we propose five value index types: *Invert*, *List*, *Number, DTime* and *ID* to support values of common data types, such as *xs:string* and *xs:decimal*, as defined in the XML schema [3] and some special data values such as values for IDREF attributes. Each value index type supports a common search function:

*List search (value, gid?)*

Given a value predicate and a group identifier, the *search* function returns a list of elements satisfying the value predicate in the group. If the group identifier is not specified, the *search* function returns a list of elements that satisfy the value predicate in any group.

The five value index types are:

- *Invert Type*: treats a value as a bag of tokens and maps a token to a list of elements.
- *List Type*: treats a value as a whole without further breaking it into tokens and maps a value to a list of elements.
- *Number Type*: maps a numeric value to a list of elements containing this number. Furthermore, a $B^+$-index is created on top of (number, element) pairs to support numerical range searches.
- *DTime Type*: maps a value of date or time type to a list of elements containing this value. Similarly, we create a $B^+$-index on (time, element) pairs to support range searches.
- *ID Type*: indexes IDREF attribute values in XML documents and maps a referring element to a referred element.

## 3.2 Query evaluation

### *3.2.1 Topic format and query modeling*

The topic format in INEX 03 [11] is based on a subset of XPath path expressions. Key constructs in an XPath path expression are the notations of *nodes*, *axes*, *filters* and *target nodes*. Target nodes refer to nodes that are returned as query results. There is usually only one target node in a path expression. To support content-oriented search, INEX 03 introduces the *about(path, string)* predicate to specify certain contexts (i.e., *path*) to be about a specific content (i.e., *string*). The *string* parameter in an *about* predicate may contain a set of terms separated by spaces, where a term is either a single word or a phrase in double quotes. Furthermore, term modifiers, such as "+" and "-", are introduced to facilitate users to specify preferences and rejections over certain terms. For example, suppose that a user is interested in articles about XML and information retrieval, published between 1999 and 2000, and with sections preferred to be *about* ranking, then the query can be formulated as $Q_3$: *//article[(./fm//yr = '1999' OR ./fm//yr = '2000') AND about(., 'XML "Information Retrieval"')]//sec[about(., '+ranking')]*.

Similar to the tree representation of XML documents, we also represent XML queries as trees: nodes in path expressions become the nodes in query trees. Axes are represented as edges between the corresponding nodes with a single arrow for a "/" axis and a double arrow for a "//" axis. Filters are represented as value predicates on the corresponding nodes. To distinguish the *about* predicates from other regular value predicates, *about* predicates are linked to their corresponding nodes with double dotted lines, while regular value predicates are linked to their corresponding nodes with single dotted lines. Finally, a target node is emphasized with a box. For example, Figure 4 is the tree-representation of $Q_3$.

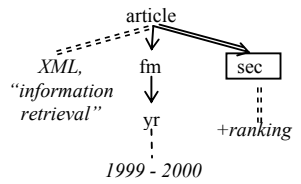



**Figure 4: The tree representation of $Q_3$.**

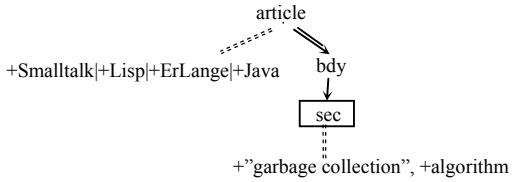



**Figure 5: The tree representation of INEX 03 query topic 68.**

To further differentiate the semantic relationships, such as "AND" and "OR," between different terms in an *about* predicate, we represent the string parameter *s* in an *about* predicate as a conjunctive normal form, where each conjunct is a disjunction (OR) of one or more terms. We use "," to represent an AND operator and "|" to represent an OR operator. For

example, Figure 5 shows the tree representation of INEX 03 query topic 68 (*//article[about(., '+Smalltalk') or about(., '+Lisp') or about(.,'+Erlang') or about(., '+Java')] //bdy//sec [about(., '+"garbage collection" +algorithm')]).*

### *3.2.2 Query processing*

After transforming an XML query into a tree representation, we can evaluate the query based on Ctree structure and value indexes in the following three steps, as shown in Figure 6.

**Input:** *T*, a Ctree with value index
$T_Q$, a query tree
**Output**: *R*, a ranked list of elements.
*1* An empty list →*R*
*2* Locate frames: Ctree groups→Query nodes
*3* For each frame *F*
*4* Evaluate value predicates;
*5* Combine results for each query node to a ranked list, *R'*;
*6* Merge *R'* to *R*;
*7* Return *R*

**Figure 6: The query processing algorithm based on Ctree index.**

First, the algorithm locates a set of frames. Each frame *F* is a set of groups in the Ctree such that each group in *F* matches a node in the query tree $T_Q$ and that these groups as a whole match the query's tree structure (Line 2). For example, there is one frame consisted of groups (1, 2, 3, 6) in the Ctree (Figure 2b) for $Q_3$, where (1, 2, 3, 6) matches the query nodes (*article*, *fm*, *yr*, *sec*) respectively.

Second, for each frame, the algorithm evaluates each value predicate on the elements in a Ctree group that is assigned to a query node with the corresponding value predicate (Line 4). Regular predicates are processed in a Boolean way. For example, for the predicate $1999 \le$ *//article/fm/yr* $\le 2000$ in $Q_3$, the elements in group 3 are either relevant or irrelevant. Irrelevant elements are pruned directly. An a*bout* predicate, however, is processed in a non-Boolean way and we will discuss its evaluation in detail in section 3.3.

Third, for each frame, the algorithm combines the evaluation results for each value predicate in the query based on the element-to-element relationships in the Ctree. The results from each frame are sorted into a ranked list, denoted as R' (Line 5).

Finally, the results from each frame are merged and ranked based on their retrieval status values (RSVs) (Line 6). A RSV indicates a result's relevancy to the query. Line 7 outputs a ranked result list.

## 3.3 Result ranking

In this section, we present how to calculate the RSVs for query results from a given frame. We first present how to calculate the RSVs for elements from a single *about* predicate. We then discuss how to combine the RSVs from multiple *about* predicates.

Before presenting our ranking scheme, we first define the following two user configurable parameters:

$\sigma(x)$ : weight for a label (or a tag name) *x*. For $\forall x\ \sigma(x) \ge 0$ and the default value is 1.

$\theta(m)$ : weight for a query term modifier *m* (i.e., "", "+" or "-"). θ("+") > θ("")>0 and θ("")>θ("-").

### *3.3.1 Computing RSV from a single "about" predicate*

Due to XML's hierarchical structure, the content of an element *e* is considered part of the content of any element *e'* where *e* can be within *e'* at any nesting level. This introduces the challenge of how to calculate the relevancy of a given term *t* within a certain element *e*, where *t* could appear in any element nested within *e*. The occurrences of a term *t* in different sub-elements of *e* are of different importance. For example, for the *about(article, 'XML "information retrieval"')* predicate

in $Q_3$, the term '*XML*' can be in a keyword, paragraph or reference part of an article. An "XML" in a keyword part is more important than an "XML" in a paragraph part, which in turn is more important than an "XML" in a reference part.

Let us now introduce the concept of "weighted term frequency" which assigns high weight to terms in important locations and low weights to terms in unimportant locations. For a given element *e*, we can identify the location of a specific sub-element *e'* containing *t* by the "relative" label path from *e* to *e'*. For example, in Figure 7, under the element *article*, the term "XML" appears once, twice and once in the relative paths *fm.kwd*, *bdy.sec.para*, and *bm.ref* respectively.

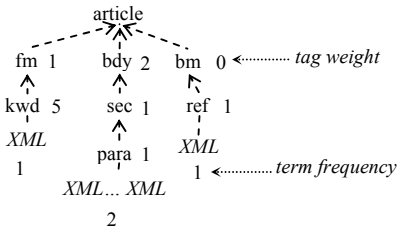



**Figure 7: An example of the term "XML" in element *article*.**

For a given element *e*, the importance (or weight) of a "relative" label path $l = x_1.x_2...x_s$ can be estimated by:

$$w(l) = \prod_{i=1}^{s} \sigma(x_i) \tag{1}$$

where *x* is a tag name and $\sigma(x)$ is the weight assigned by users to the tag name *x*.

Let *tf (g, e, t, $l_i$)* be the frequency of term *t* in a location $l_i$ under element *e* in group *g*, the weighted term frequency of a term *t* in an element *g:e*, denoted as $tf_w(g, e, t)$, is defined as:

$$tf_w(g, e, t) = \sum_{j=1}^{r} w(l_j) \cdot tf(g, e, l_j, t) \tag{2}$$

where *r* is the number of locations under element *e* in group *g* containing *t*.

To support the dynamic document concept introduced by XML, similar to the concept of "inverted document frequency," we propose the concept of "inverted element frequency," denoted as *ief(g, t),* which is defined as follows:

$$ief(g,t) = \log(\frac{|E_g|}{|E_g(t)|}) \tag{3}$$

where $|E_g|$ is the number of elements in group g and $|E_g(t)|$ is the number of elements containing *t* in group *g*.

With the introduction of weighted term frequency ($tf_w$) and inverted element frequency (*ief*), we are now ready to define the retrieval status value (RSV) of an element *g:e* to an *about* predicate $\alpha$ as follows:

$$RSV(g, e, \alpha) = Max(\sum_{t_k \in \alpha} \theta(m(t_k)) \cdot tf_w(g, e, t_k) \cdot ief(g, t_k), 0) \tag{4}$$

where $t_k$ is a term in $\alpha$ and $\theta(m(t_k))$ is the weight for the query term modifier associated with $t_k$. Since users may assign a negative value for $\theta$("-"), which can lead to negative RSVs, we compare the weighted sum of $tf_w$**ief* with 0, as shown in (4), to ensure that RSV(g, e, $\alpha$) is non-negative.

### *3.3.2 Combining RSVs from multiple " about" predicates*

Since a query Q may contain multiple *about* predicates, we need to merge the RSVs from each *about* predicate into the elements in the group that is assigned to Q's target node. Because RSVs are merged either from a descendant group to an ancestor group or from an ancestor group to a descendant group, there are two cases to consider:

1) Merging results from a descendant group to an ancestor group.

If the results are merged from a descendant group $g_D$ containing elements relevant to an *about* predicate $\alpha_1$ to an ancestor group $g_A$ containing elements relevant to another *about* predicates $\alpha_2$, the RSVs of elements in group $g_A$ are updated according to:

$$RSV(g_A, e_A, \alpha_1 \& \alpha_2) = RSV(g_A, e_A, \alpha_2) + \sum_{i=1}^{n} RSV(g_D, e_{Di}, \alpha_1) \quad (5)$$

where $e_A$ is an element in $g_A$, $e_{Di}$ is an element in group $g_D$ and n is the number of elements in group $g_D$ that are descendants of element $g_A{:}e_A$.

2) Merging results from an ancestor group to a descendant group.

If the results are merged from an ancestor group $g_A$ containing elements relevant to an *about* predicate $\alpha_2$ to a descendant group $g_D$ containing elements relevant to another *about* predicate $\alpha_1$, the RSVs of elements in group $g_D$ are updated according to:

$$RSV(g_D, e_D, \alpha_1 \& \alpha_2) = RSV(g_D, e_D, \alpha_1) + RSV(g_A, e_A, \alpha_2) \quad (6)$$

where $e_D$ is an element in group $g_D$, $e_A$ is an element in group $g_A$ that is an ancestor of element $e_D$.

# 4 Experimental studies

We use the INEX 03 dataset and CAS topics to evaluate the effectiveness of our configurable XML information retrieval system. For INEX 03 CAS topics, there are two tasks: strict CAS (SCAS) and vague CAS (VCAS). A query's structure must be strictly matched in SCAS, while it can be vaguely matched in VCAS. In this paper, we only focus on SCAS task. We implemented the configurable XML information retrieval system in C# [17] and ran the experiments on a 2.8 GHz PC with 1GB of RAM running Windows XP.

## 4.1 Index configurations

All the experiments use the same set of index configurations, which ignores non-semantic tags, such as tags for presentation-purpose only (e.g., <*bold*>, <*italic*> and <*scp*>), during indexing. For most elements being indexed and with non-numerical values, we remove stop words from their text contents and create inverted indexes on the remaining non-stop words. For elements with numerical values, such as year (<*yr*>) and page number (<*pp*>), we use the Number type to index these elements' text contents. For elements with text contents of limited variations, such as month (<*mo*>) and role (<*role*>), we index their text contents with the List type. Values of IDREF attributes are indexed with the ID type. The complete set of index configurations used for the experiments can be downloaded from our website [17].

## 4.2 System implementation

Before presenting our experimental studies, we first briefly introduce two aspects of our system's implementation:

First, our current system interprets the semantics of a rejection term modifier (i.e., "-") as a strict rejection. That is, a document component will not be judged as relevant to a query if it contains a term that is prefixed with "-" in the query. For example, for INEX 03 query topic 67: *//article//fm[(about(.//tig, '+software +architecture') or about(.//abs, '+software +architecture')) and about(., '-distributed -web')]*, any *fm* element containing "distributed" or "web" will not be returned as relevant even if it has a sub-element *tig* or *abs* highly relevant to software and architecture.

Second, our current system interprets an "AND" construct as a strict "AND". An element *g:e* is judged as relevant to an *about* predicate with a string parameter $s = s_1, s_2, \ldots s_n$ if and only if $\forall s_i$ ($1 \le i \le n$), *g:e* contains $s_i$. For example, for INEX 03 query topic 62 (*//article[about(.,'security +biometrics') AND about(.//sec,'"facial recognition"')]*), the semantic relation-

ship between "security" and "biometrics" is represented as an "AND" relationship. Therefore if an article contains only "security" or only "biometrics," it will not be judged as relevant even though it contains a section on "facial recognition."

In the following experiments, we first studied two sets of experiments: one evaluating our system's retrieval performance for several query topics under the same set of θ(m) but different σ(x); and another evaluating our system's retrieval performance for several query topics under the same set of σ(x) but different θ(m). We then studied our system's retrieval performance for all 30 CAS topics. The precision/recall curves for all experiment results are plotted with the INEX on-line tool [19].

### 4.3 Evaluation comparisons for different tag weight configurations

In this subsection, we first describe how we configure tag weights in the experiments and then present the experimental studies about the effects of different tag weight configurations on the retrieval performances.

Since our ranking scheme uses "relative label path" to assign the weight for a term, we configure the weight for a tag name according to other sibling tag names but not its parent or ancestors' tag names. More specifically, in our experiments, we start from the root node of the dataset's corresponding Ctree, i.e., *article* in the INEX dataset, select one of its child nodes, such as *bdy*, set its tag weight fixed, such as 1, and assign the tag weights for other sibling nodes accordingly. For example, in the INEX dataset, node *article* has three child nodes: *fm*, *bdy* and *bm* (Figure 8). Thus we first set σ (*bdy*) =1 and then configure the weights for *fm* and *bm* according to their relative importance as compared to *bdy*. After we finish configuring the tag weights for the nodes on the first level, we can configure the tag weights for the elements on the second level, third level and so on in a similar way. For example, in the INEX dataset, node *fm* is the parent of nodes *tig*, *abs* and *kwd*. Therefore, after we configure σ(*fm*), we select one of *fm*'s child nodes, such as *tig*, set σ(*tig*) = 1 and assign σ(*abs*) and σ(*kwd*) according to *abs* and *kwd*'s relative importance as compared to *tig*.

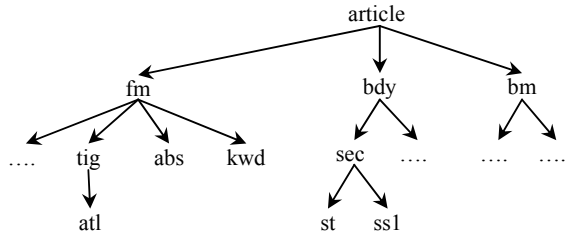



**Figure 8: The structure of the INEX XML dataset.**

In this set of experiments, we tested different tag weight configurations with the same set of term modifier weights (θ("+") = 1.8 and θ("") = 1) and studied several topics' corresponding evaluations. Due to space limitations, we only list three tag weight configurations in Table 1. Tags that are indexed but not listed below are with the default weight 1.

**Table 1: The list of tag weight configurations used for the first set of experiments with θ("+")=1.8 and θ("") =1.**

| | bm | fm | bdy | atl | abs | kwd | st |
|---|---|---|---|---|---|---|---|
| A | 1 | 3 | 1 | 3 | 1 | 2 | 3 |
| B | 0 | 5 | 1 | 5 | 1 | 3 | 5 |
| C | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

The main difference between tag weight configurations A and B is that the weight for tag *bm* is 0 in B, which prunes documents with only references or appendixes about topics of interest. For example, for INEX 03 topic 65 (*//article[./fm//yr > '1998' AND about(./, '"image retrieval"')]*), an article with only its *bm* part about "image retrieval" will be not judged as relevant under tag weight configuration B. Tag weight configuration C assigns a uniform weight (i.e., 1) to all tags.

We choose query topics, such as topic 65, but not topics such as topic 69 (*/article/bdy/sec[about(.//st, "information retrieval")]*) for evaluation studies of three tag weight configurations in Table 1. This is because in the INEX dataset, elements with tag names listed in Table 1 are sub-elements of *article*, but they are not sub-elements of *st*. Therefore, the evaluations for topics with an *about* predicate on element *article*, such as topic 65, should be different under the three configurations. The evaluations for topics, such as topic 69, however, should be the same under the three configurations.

The average precision/recall curves for query topic 65 (under both strict and generalized quantization) with tag weight configurations A, B and C are portrayed in Figure 9, 10 and 11 respectively. We notice that for query topic 65, our system has a relatively high precision under both strict and generalized quantization for all the three tag weight configurations. Furthermore, our system has a very high precision at low recall regions. For example, under the strict quantization and with the tag weight configuration A, the precision is 1 when recall is less than 0.318. With the tag weight configuration B, the precision stays at 1 when the recall is less than 0.545. This means that our system is able to return the most relevant answers in the top few answers. This is a very important feature since query results are usually presented to humans and they usually have only enough patience to browse the top few of the returned answers. By comparing figures 9, 10 and 11, we can see that properly adjusting tag weights can significantly improve the precision/recall. Similar results are observed for query topics, such as topic 79, 82, 87 and 88, which have an *about* predicate on element *article*.

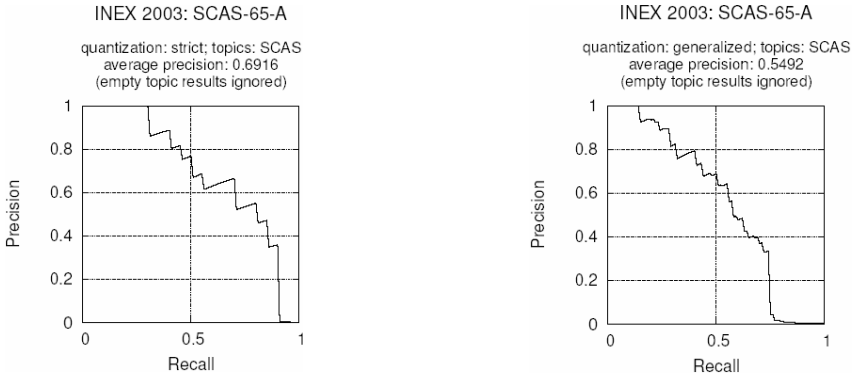



**Figure 9: The strict and generalized precision/recall curves for INEX 03 topic 65 with ranking configuration A.**

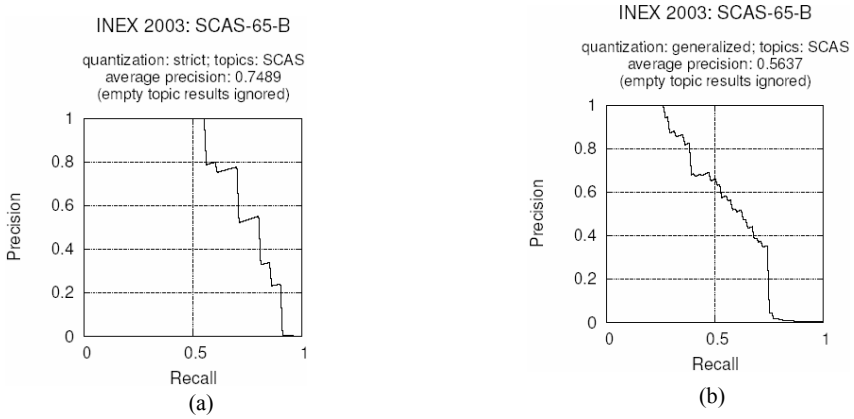



**Figure 10: The strict and generalized precision/recall curves for INEX 03 topic 65 with ranking configuration B.**

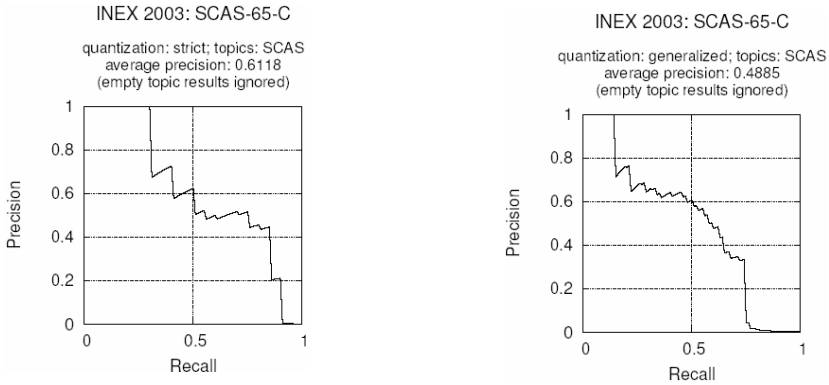



**Figure 11: The strict and generalized precision/recall curves for INEX 03 topic 65 with ranking configuration C.**

### 4.4 Evaluation comparisons for different term modifier weight configurations

In this set of experiments, we tested our system's retrieval performances for several query topics under the same $\sigma(x)$ (configuration B) but different $\theta(m)$, and compared their corresponding evaluations. Since our current implementation interprets a rejection term modifier "-" strictly, we only need to adjust the weights for a normal term modifier $\theta(\text{“”})$ and a preference term modifier $\theta(\text{“+”})$. In the following experiments, we keep $\theta(\text{“”})$ fixed ($\theta(\text{“”})=1$) and adjust $\theta(\text{“+”})$. Table 2 lists three $\theta(\text{“+”})$ weights used in the experiments with tag weight configuration B.

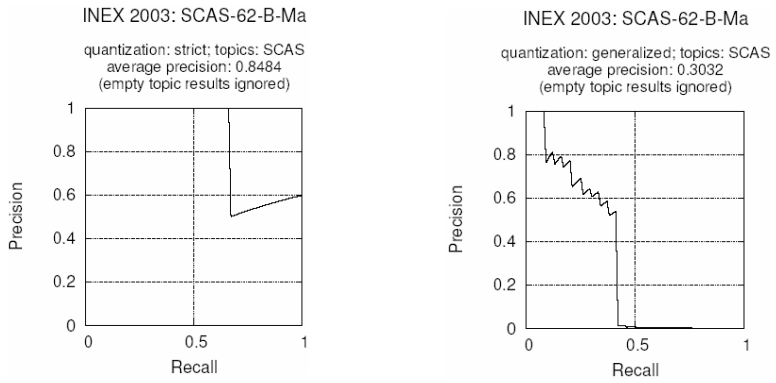
**Table 2: The list of θ(m) used for the second set of experiments with tag weight configuration B.**

| | θ(“+”) | θ(“”) |
|---|---|---|
| $M_a$ | 1.8 | 1 |
| $M_b$ | 3 | 1 |
| $M_c$ | 1 | 1 |

We choose query topics, such as topic 62 (*//article[about(., 'security, +biometrics') AND about(.//sec, '"facial recognition")]*) for this set of experiments because they contain *about* predicates with preference terms, i.e., terms prefixed with "+", and normal terms, i.e., terms without prefixing "+" or "-." Many other topics either have no *about* predicates with preference terms, such as 70 and 71, or have no *about* predicates with normal terms, such as topic 68. Adjusting θ("+") weight has no effect on the evaluations for topics without preference terms. It also has no effect on the evaluations for topics with all the terms prefixed with "+" since it only changes the absolute RSV of a result but not its relative ranking.

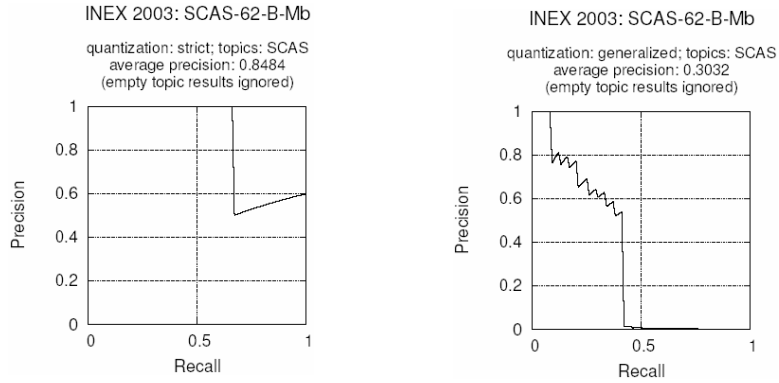
The average precision/recall curves for query topic 62 with three different θ("+") weights under both strict and generalized quantization are presented in Figures 12, 13 and 14 respectively. From Figures 12, 13 and 14, we note that the adjusting θ("+") weight has no effect on evaluations for topic 62. The reason are: first, in the INEX dataset, "security" is a much more popular term than "biometrics." In fact, under the tag weight configuration B, there are 2548 articles containing "security" and only 143 articles containing "biometrics." As a result, $ief(article, \text{“security”}) << ief(article, \text{“biometrics”})$, i.e., the weight for term "biometrics" is much higher than the weight for term "security" due to the inverted element frequency adjustment. Second, in the results for topic 62, the term "biometrics" occurs much more frequently than term "security," i.e., $tf_w(article, \text{“biometrics”}) >> tf_w(article, \text{“security”})$. For example, in article *co/2000/r2056*, a result for topic 62, term "security" occurs only four times in the article's *fm* and *bdy* parts, while term "biometric" occurs about 60 times. In such scenarios, the term "biometric" is a dominant factor in ranking due to the dataset characteristics. Therefore adjusting θ("+")

weight to a query's domain term has little effect on the query's evaluation. Similar phenomena are observed for the evaluations of topics 63 and 90 under three different θ("+") weights.



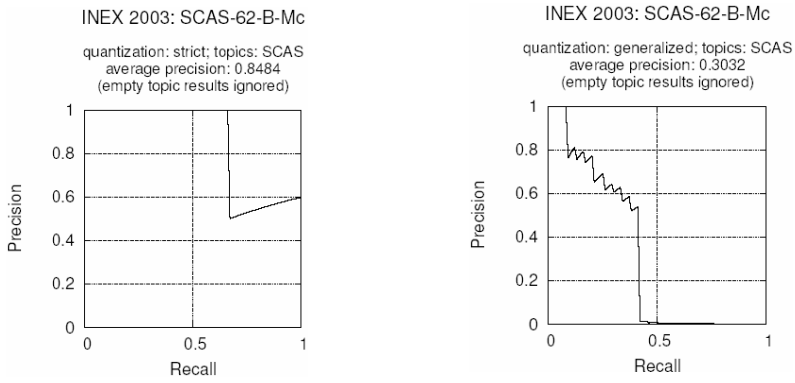

(a) (b)

**Figure 12: The strict and generalized precision/recall curves for INEX 03 topic 62 with θ("+") = 1.8.**




(a) (b)

**Figure 13: The strict and generalized precision/recall curves for INEX 03 topic 62 with θ("+") = 3.**




(a) (b)

**Figure 14: The strict and generalized precision/recall curves for INEX 03 topic 62 with θ("+") = 1.**

Even though adjusting θ("+") weight has little effect on INEX 03 query topics, we believe that adjusting θ("+") can improve the evaluation for certain topics in certain scenarios. For example, suppose that a user is interested in articles about

C++ or Java and with Java more preferred, and suppose that we have a dataset such that term "C++" and "Java" have similar *ief* and are almost equally frequent in each document. In such a scenario, adjusting θ("+") weight will definitely improve the evaluation of this topic.

### 4.5 Performance study for all the 30 INEX 03 CAS topics

In this set of experiments, we first test all the 30 CAS topics with tag weight configuration B and θ("+")=1.8. The average precision/recall curves (both strict and generalized quantization) for all the 30 CAS topics are shown in Figure 15. Our average precision/recall, 0.3309, is notably high and it is about 4% improvement over the official top 1 (0.3182) as compared to all the 38 official submissions. In addition, from Figure 15c, we can see that under the strict quantization, our system has the highest precision when recall is less than 0.5 as compared to all the official submissions, which implies that our system is able to return the most relevant answers earlier than other systems.

It is interesting to note that our system performs much better under the strict quantization than under the generalized quantization. This is because our system returns a small number of results for each topic due to the strict implementation of an "AND" operator, while most other systems return about 1500 results for each topic. Figure 16 shows the number of results for each CAS topic returned by our system as well as the number of relevant results in the relevant assessment under the generalized quantization. From Figure 16, we can see that the number of results returned by our system for most topics is less than 200 and that the result size for many topics are even less than 50. In addition, there are about 13 topics out of 30 CAS topics for which our system returns fewer results than the number of relevant results under the generalized quantization. Thus under the generalized quantization, because of the strict interpretation of an "AND" operator the average precision for our system for many topics is almost zero when recall is high.

One way to improve the average precision under the generalized quantization is to increase the result size by relaxing the strict interpretation of an "AND" operator. Therefore, we test another experiment with the tag weight configuration B and θ("+) = 1.8. In this experiment, we first interpret an "AND" operator strictly and get a list of ranked results, $R_1$, for each topic. Then we consider an "AND" operator the same as an "OR" operator and get a list of ranked results, $R_2$, for each topic. The average precision/recall (both strict and generalized) of the merged results for all the 30 CAS INEX 03 topics under ranking configuration B and θ("+")=1.8 are illustrated in Figure 17. By comparing Figure 17 with Figure 15, we can infer that increasing result size improves the evaluations under both the strict and generalized quantization. The improvement under the generalized quantization, 23.5%, is higher than that under the strict quantization, 12.1%. Similar phenomena are observed for the results under tag weight configuration A and θ("+) = 1.8, as illustrated in Table 3.

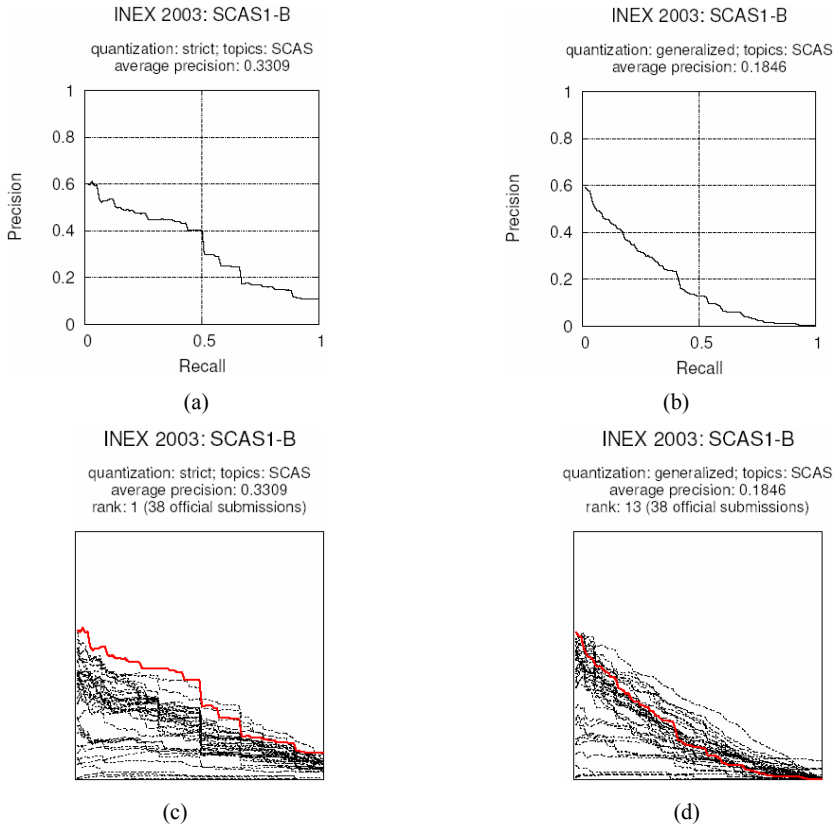



**Figure 15: Average precision/recall curve for 30 CAS INEX 03 topics with ranking configuration B and θ("+") = 1.8.**

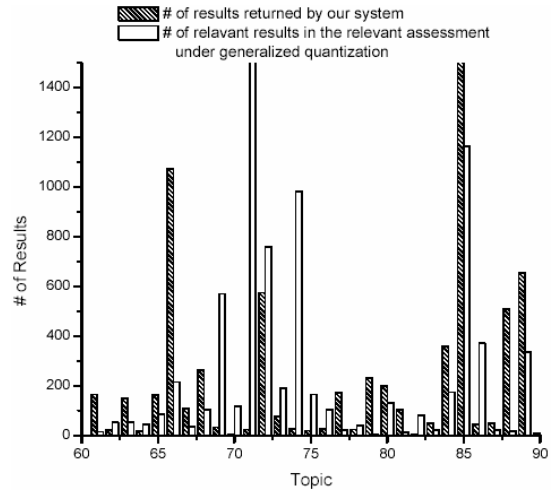



**Figure 16: # of results returned by our system vs. # of relevant results in the relevant assessment under the generalized quantization.**

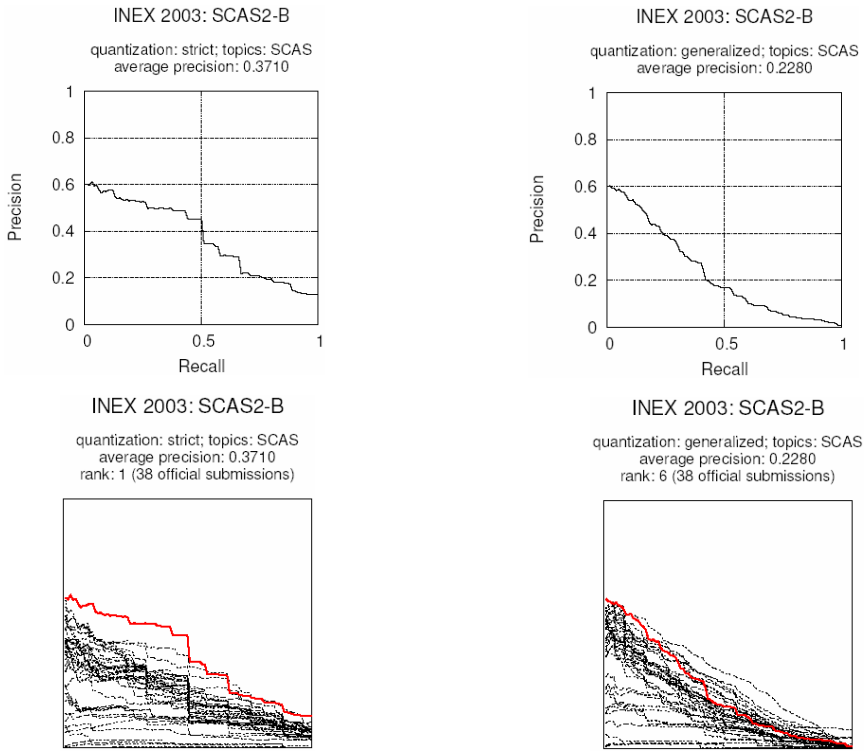



**Figure 17: Average precision/recall curve for 30 CAS INEX 03 topics with ranking configuration B and two runs.**

**Table 3: The average precision/recall for 30 SCAS topics with ranking configuration A and θ("+")=1.8.**

| Single Run – strict "AND" | | Two Runs – strict "AND" and then fuzzy "AND" | |
|---|---|---|---|
| Strict quantization | Generalized quantization | Strict quantization | Generalized Quantization |
| 0.3314 | 0.181 | 0.3685 | 0.2257 |

### 4.5 Discussions

Our experimental results reveal that with proper ranking configurations, our configurable framework can achieve high precision/recall. One way for further improvement is to consider the size of a document component in the ranking. It is well known that a document's size is an important ranking factor in traditional IR. Given two document components (or elements) with the same weighted term frequency, the shorter the document component, the more relevant it is to the query. For instance, for a query such as "*//article[about(.atl/, '"support vector machines"')]*)", i.e., finding articles with title about "support vector machines," our current ranking scheme will rank an article with a title "Support Vector Machines" and another article with a title "Using Support Vector Machines for 3D Object Recognition" as equally important. The first article, however, is more relevant to the query. Therefore it is desirable to include the size of a document component into the ranking scheme.

## 5 Related work

There have been a number of studies in the IR community investigating the problem of retrieving information from XML documents [e.g., 2, 4-9, 14, 15]. XML retrieval languages, XML indexing, and ranking are the three most important aspects and have been intensively investigated. We will focus the related work on these three aspects.

*XML retrieval language*

There are a number of XML retrieval languages proposed from the database community as well as IR community. XPath [21], XML-QL [20] and XQuery [22] are examples of XML query languages proposed by the database community. These languages focus on XML retrieval, but not ranking. XIRQL [7] is the first full-functional XML information retrieval language proposed in the IR community. It supports weighting and ranking, relevance-oriented searches, data types with vague predicates and semantic relativism. XIRQL is powerful but not simple enough for new users. To overcome this problem, [5] proposed the concept of XML fragments to represent queries, which is much simpler and allows users to specify term preferences and rejections. Based on the idea of [5] , INEX 03 topic development working group proposed an extended XPath query language with the inclusion of about predicates and term modifiers such as preference and rejection.

*XML indexing*

There exists a large body of work investigating XML indexing in the IR community. Inverted index is a well-established technique for indexing free text and it has been widely used for indexing XML text content. Our work also uses inverted index for indexing most XML text contents. Different from most other existing approaches, besides the inverted index, we also propose four additional value index types (e.g., List, Number, DTime and ID) to deal with the heterogeneity of XML text contents. Structure indexing is a new feature introduced by the XML model and plays an important role in efficient XML information retrieval. There is, however, not much work on indexing XML structures in the IR community. In [7], nodes of predefined categories are indexed and associated with term statistics. Users are allowed to search at the level of indexing nodes or nodes that are of hierarchical combination of indexing nodes. [9] generalized [7] to support the retrieval of nodes at varying granularity level by combining the statistics of those predefined nodes at query time. In [5], XML document collections are split into small "documents" based on the predefined elements. A vector of (term, context) pairs is extracted from each document. Indexes are created on these (term, context) pairs associated with statistics for ranking calculation. Our work differs from other existing work in that our structure indexes provide both structure summaries and detailed element-to-element relationships in XML document trees and can answer XPath queries efficiently.

*XML ranking*

Many ranking approaches have been proposed to support the dynamic document concept introduced by XML. Most of these approaches leverage on the mature models developed in traditional IR, such as Vector Space Model, Probabilistic Model and Bayesian Inference Model, and extend them to the XML model. For example, in [7], a novel "augmentation" technique is proposed to retrieve nodes that are hierarchical combinations of indexing nodes. During the "augmentation," the statistics and inverted lists of indexing nodes are propagated upward in the document tree with lower term weights. [5] extends the traditional VSM where each unit is a single term to the XML model with each pair (term, context) as a basic unit. The traditional cosine formula for measuring document and query similarity are updated correspondingly. Our work differs from [5] in that we focus on SCAS where the structure of a result strictly matches that of the query, while [5] allows the structure of a result to be fuzzily matched with that of the query. In addition, we propose the concept of "weighted term frequency" and "inverted element frequency" based on the classical VSM, and our ranking framework allows users to configure weights for tag names and query term modifiers.

## 6 Conclusion

In this paper, we proposed a configurable framework for indexing XML documents and ranking query results, which allows users to configure tag index types. It also enables users to select appropriate processing operations and index types for the

heterogeneous XML text contents. We used INEX 03 XML dataset and 30 CAS query topics, and tested a set of experiments to evaluate the performance of our proposed methodology. We first studied the effects of different tag weight configurations on retrieval performances and then compared the retrieval performances under the same tag weight configuration but different term modifier weights. The results from the first set of experiments indicate that properly adjusting tag weights can significantly improve the precision/recall. The second set of experiments reveal that adjusting term modifier weights has little effect for INEX 03 topics' evaluations. The third set of experiments evaluated all the 30 CAS topics under a set of ranking configurations. Our approach yields significantly high precision when recall is low and achieves the highest average precision/recall as compared with 38 official INEX 03 submissions with strict quantization of the evaluation metrics. We will include a document component's size into the current ranking scheme and extend current work to support CO topics and VCAS tasks in our future work.

## References


[1] Amer-Yahia S, Fernandez M, Srivastava D and Xu Y (2003) Phrase Matching in XML. In: VLDB 2003, pp. 177-188.

[2] Baeza-Yates R, Fuhr N and Maarek Y (2002) Second Edition of the XML and IR Workshop. In: SIGIR Forum, Volume 36 Number 2.

[3] Biron PV and Malhotra A (Eds) (2004) XML Schema Part 2: Datatypes. W3C Recommendation, May 2001. Http://www.w3.org/TR/xmlschema-2/ .

[4] Carmel D, Soffer A and Maarek Y (2000) XML and Information Retrieval Workshop. In: SIGIR Forum, Volume 34 Number 1.

[5] Carmel D, Maarek SY, Mandelbrod M, Mass Y and Soffer A (2003) Searching XML Documents via XML Fragments. In: Proceedings of SIGIR'03, Toronto, Canada, 2003, pp.151-158.

[6] Fuhr N, Gövert N, Kazai G and Lalmas M (2003) Initiative for the Evaluation of XML Retrieval (INEX). Proceedings of the First INEX Workshop. Dagstuhl, Germany, December 8-11, 2002. ERCIM Workshop Proceedings. ERCIM, Sophia Antipolis, France.

[7] Fuhr N and GrossJohann K (2001) XIRQL: A Query Language for Information Retrieval in XML Documents. In: Proceedings of SIGIR'2001, New Orleans, LA, 2001, pp.172-180.

[8] Fuhr N, Lalmas M and Malik S (2004). Initiative for the Evaluation of XML Retrieval (INEX). Proceedings of the Second INEX Workshop. Dagstuhl, Germany, December 15-17, 2003.

[9] Grabs T and Schek JH (2002) Generating Vector Spaces On-the-fly for Flexible XML Retrieval. In: [1].

[10] Goldman R and Widom J (1997) DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: the Proceedings of VLDB 1997, pp.436-445.

[11] Kazai G, Lalmas M and Malik S (2003) INEX 03 Topic Development Guide. In: [4], pp. 184-191.

[12] Salton D. and McGill JM (1983) Introduction to Modern Information Retrieval. McGraw-Hill, New York.

[13] Liu S, Zou Q and Chu WW (2004). Configurable Indexing and Ranking for XML Information Retrieval. In: the Proceedings of SIGIR 04, Sheffield, UK.

[14] Schlieder T and Meuss H (2002) Querying and Ranking XML Documents. In: Journal of American Society for Information Science and Technology, Volume 53, Issue 6 (April 2002), pp. 489-503.

[15] Theobald A and Weikum G (2000) Adding relevance to XML. In: WebDB (Informal Proceedings) 2000, pp.35-40.

[16] Zou Q, Liu S and Chu WW (2004) Ctree: A Compact Two-level Bidirectional Tree for Indexing XML Data. In: UCLA Computer Science Department *Technical Report*, #TR040010.

[17] Configurable XML Search Engine. http://fargo.cs.ucla.edu/inexdemo/inexsearch.aspx.

[18] INitiative for the evaluation of XML Retrieval http://qmir.dcs.qmul.ac.uk/INEX.

[19] INEX on-line evaluation tool, accessible from http://inex.is.informatik.uni-duisburg.de:2003/.

[20] XML-QL. http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

[21] XPATH. http://www.w3.org/TR/xpath.

[22] XQuery. http://www.w3.org/TR/xquery/.