The TIJAH XML-IR system at INEX 2003

Johan List[1] Vojkan Mihajlovic[2] Arjen P. de Vries[1] Georgina Ramírez[1]
Djoerd Hiemstra[2]

[1]CWI
P.O. Box 94079
1090GB Amsterdam
The Netherlands
{jalist, arjen, georgina}@cwi.nl

[2]CTIT
P.O. Box 217
7500 AE Enschede
The Netherlands
{vojkan,hiemstra}@cs.utwente.nl

ABSTRACT

This paper discusses our participation in INEX (the Initiative for the Evaluation of XML Retrieval) using the TIJAH XML-IR system. TIJAH's system design follows a 'standard' layered database architecture, carefully separating the conceptual, logical and physical levels. At the conceptual level, we classify the INEX XPath-based query expressions into three different query patterns. For each pattern, we present its mapping into a query execution strategy. The logical layer exploits *probabilistic region algebra* as the basis for query processing. We discuss the region operators used to select and manipulate XML document components. The logical algebra expressions are mapped into efficient relational algebra expressions over a physical representation of the XML document collection using the 'pre-post numbering scheme'. The paper concludes with a preliminary analysis of the evaluation results of the submitted runs.

1. INTRODUCTION

This paper describes our research for INEX 2003 (the Initiative for the Evaluation of XML Retrieval). We participated with the TIJAH XML-IR retrieval system, a research prototype built on top of the MonetDB database kernel [1]. Key feature of the TIJAH system is its layered design, following the basic system architecture of relational database management systems.

Traditional information retrieval systems represent a document as a 'bag-of-words'. Inverted file structures provide the basis for implementing a retrieval system for such 'flat' documents. In the case of structured documents however, we think designing the retrieval system following 'the database approach' is best to keep the more complex data representation manageable.

The main characteristic of the database approach is a strong separation between conceptual, logical and physical levels, and the usage of different data models and query languages at each of those levels [20]. In relational database systems, a significant benefit of this *data abstraction* (through the separation between the levels in database design) is to enable query optimization. A SQL query (a 'calculus expression') at the conceptual level is first translated into relational algebra. The algebraic version used at the logical level is then rewritten by the query optimizer into an efficient physical query plan. The physical algebra exploits techniques like hashing and sorting to improve efficiency [8].

For XML-IR systems, following this separation in layers gives another, additional advantage: by choosing the appropriate level of abstraction for the logical level, the development of probabilistic techniques handling structural information is simplified, and kept orthogonal to the rest of the system design. Section 3 details our approach, based on a probabilistic extension of text region algebras.

The paper is organized along the layers of the TIJAH system design. The following Section describes the query language used at the conceptual level, identifies three patterns in the INEX topic set, and explains how the language modeling approach to information retrieval is used for the *about* operator. Section 3 presents a probabilistic region algebra for expressing the three query patterns. Section 4 explains how the algebraic expressions are mapped into efficient relational algebra expressions over a physical representation of the XML document collection using the 'pre-post numbering scheme'. We conclude with a discussion of the experiments performed with our approach for the three INEX search tasks.

2. CONCEPTUAL LEVEL

For the conceptual level, we used the INEX query language as proposed by the INEX Initiative in 2002. The INEX query language extends XPath with a special *about* function, ranking XML elements by their estimated relevance to a textual query. As such, the invocation of the *about* function can be regarded as the instantiation of a retrieval model.

The retrieval model used for the *about* function is essentially the same as that used at INEX 2002 [12, 14]. We calculate the probability of complete relevance of a document component, assuming independence between the probability of relevance on exhaustivity and the probability of relevance on specificity.

The probability of relevance on exhaustivity, $P(R_E)$, is estimated using the language modeling approach to information retrieval [11]. Instead of document frequency, we have used collection frequencies for the background model. The probability of relevance on specificity, $P(R_S)$, is assumed to be directly related to the component length (following a log-normal distribution). Its steep slope at the start discounts the likelihood that very short document components are relevant. Its long tail reflects that we do not expect long document components to be focused on the topic of request either.

The language model as used by our system disregards structure within a document component, i.e., the model treats a document component as a 'flat-text' document. This model property, and an informal inspection of the INEX 2003 topic list, led us to use only a subset of possible location step axes within an *about* function call; we only used the *descendant-or-self::qname* location step axis. Allowing other axes, like *sibling::qname* or *following::qname* requires correct probabilistic modeling for estimating probabilities in the language model, which our model did not offer at the time of evaluation.

Table 1: SCAS and VCAS pattern set. Note that *xp*, *xp2*, *axp*, *axp1* and *axp2* are location steps, and *'t/p'* denotes any set of terms or phrases to search for.

Pattern	Pattern definition
P1	xp[about(axp, 't/p')]
P2	xp[about(axp1, 't1/p1') AND about(axp2, 't2/p2')]
	xp[about(axp1, 't1/p1') OR about(axp2, 't2/p2')]
P3	xp[about(axp1, 't1/p1')]/xp2[about(axp2, 't2/p2')]
	xp[about(axp1, 't1/p1')]//xp2[about(axp2, 't2/p2')]

Since we did not have an automatic query processing facility, we processed the queries manually but in a mechanic fashion. Processing the INEX query patterns takes place in two steps:

- classify the query into (a sequence of) three basic *query patterns* (shown in Table 1);
- create a query plan to process the queries. The query patterns are visualized in Figure 1.

The basic pattern for all XPath based queries is the single *location step*, as defined in [7], augmented with an *about* function call (pattern *P1* in Table 1). When referring to, for example *xp*, we refer to the node-set representing the location step *xp*; in other words, a path leading to a certain location (or node) in the XML syntax tree. The first query pattern consists of one location step to identify the nodes to be retrieved, ranked by an *about* expression over a node-set reached by a second location step. The two other (more complex) patterns *P2* and *P3* are essentially multiple interrelated instances of the basic pattern *P1*. The XPath location steps may also apply (Boolean) predicate filters, e.g. selecting nodes with a particular value range for `yr`.

3. LOGICAL LEVEL

The logical level is based on a probabilistic region algebra. Region algebra was introduced by Burkowski [2], Clarke et al. [3], and Tova and Milo [4]. The aim of the earliest text region algebra approaches has been to enable structured text search. Later, it has been applied to related tasks as well, including search on nested text regions [13], processing of structured text [17], and ranked retrieval from structured text documents [15].

The basic idea behind region algebra approaches is the representation of text documents as a set of 'extents', where each extent is defined by its starting and end position. The application of the idea of text extents to XML documents is straightforward. If we regard each XML document instance as a linearized string or a set of *tokens* (including the document text itself), each component can then be considered as a text region or a contiguous subset of the entire linearized string. Therefore, a text region a can be identified by its starting point s_a and ending point e_a within the entire linearized string. Figure 2 visualizes an example XML document (as a syntax tree) with the start point and end point numbering for the nodes or regions in the tree. As an example, the *bdy*-region corresponds to (closed) interval [5..24].

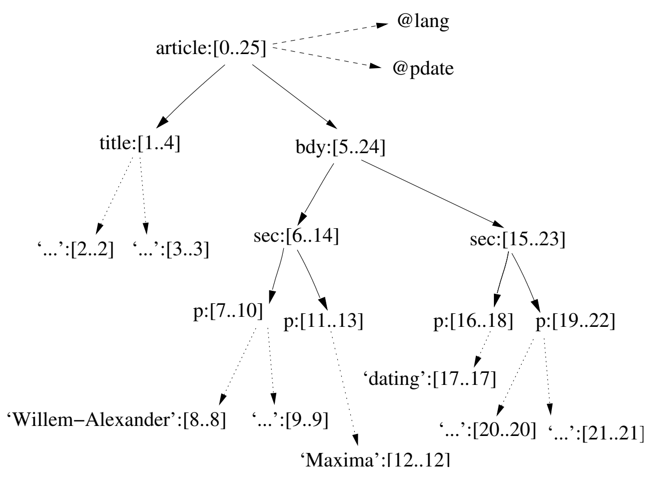

Figure 2: Example XML syntax tree with start and endpoint assignment.

Let us introduce the basic set of region operators. We use capital letters (A, B, C) to denote the region sets, and their corresponding non-capitals to denote regions in these region sets (a, b, c). The operators take region sets as input and give a result which is again a region set. The definition of region operators is given in Table 2. Interval operator $I(t)$ returns the region set representing the occurrences of term t as a content word in the XML document; note that it gives a result set in which $s_a = e_a$ for every region, assuming t is a single term and not a phrase. Location operator $L(xp)$ denotes the sequential application of XPath location steps, i.e., axis- and node-tests (a definition of axis- and node-tests can be found in [16]). Optionally, location step operator L also processes predicate tests on node or attribute values specified in the XPath expression.

Table 2: Region Algebra Operators.

Operator	Operator definition
$I(t)$	$\{a \mid s_a, e_a$ are pre and post index of term t$\}$
$L(xp)$	$C = XPath(xp)$
$A \rhd B$	$\{a \mid a \in A \wedge b \in B \wedge s_a \leq s_b \wedge e_a \geq e_b\}$
$A \lhd B$	$\{a \mid a \in A \wedge b \in B \wedge s_a \geq s_b \wedge e_a \leq e_b\}$
$A \bigtriangleup B$	$\{c \mid c \in A \wedge c \in B\}$
$A \bigtriangledown B$	$\{c \mid c \in A \vee c \in B\}$

Table 3 expresses the patterns identified in the previous Section using region algebra operators (ignoring ranking for

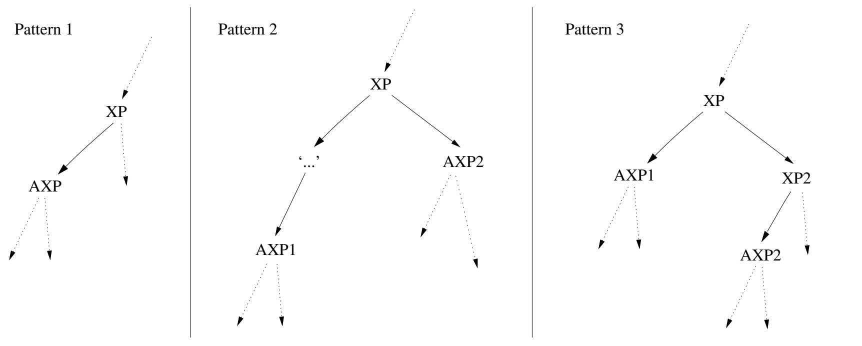

Figure 1: Example instances of the three defined patterns.

Table 3: Pattern definitions based on pure region algebra operators.

Pattern	**Algebraic expression**
$P_1(xp, axp)$	$L(xp) \rhd (L(axp) \rhd I(t))$ $L(xp) \rhd ((L(axp) \rhd I(t_1)) \bigtriangleup (L(axp) \rhd I(t_2)) \bigtriangleup ... \bigtriangleup (L(axp) \rhd I(t_n)))$
$P_2(xp, axp1, axp2)$	$P_1(xp, axp1) \bigtriangleup P_1(xp, axp2)$ $P_1(xp, axp1) \bigtriangledown P_1(xp, axp2)$
$P_3(xp1, xp2, axp1, axp2)$	$P_1(xp2, axp2) \lhd P_1(xp1, axp1)$

now). Pattern 1 distinguishes between term (t) and phrase expressions ($p = \{t_1, t_2, ..., t_n\}$). Patterns 2 and 3 are rewritten into several interrelated instances of pattern 1. Table 4 introduces a probabilistic extension of the pure region algebra operators. In order to introduce ranking, we extend the notion of region with its *relevance score*; i.e., every region a has an associated relevance score p_a. In cases where pure region algebra operators are used, the value of the introduced relevance score is equal to a predefined default value (e.g., $p_a = 1$) for each resulting region in a region set.

Table 5 gives the probabilistic region algebra expressions corresponding to the INEX query patterns identified before. The *tp1* is used to denote *'t1/p1'* or the combination of *'t1/p1'* and *'t2/p2'* (the choice between these options is made at the conceptual level). Similarly, *tp2* is either *'t2/p2'* or a combination of *'t2/p2'* and *'t1/p1'*.

Expressing query plans using the operators given in Table 4 preserves *data independence* between the logical and the physical level of a database. Similarly, these operators enable the separation between the structural query processing and the underlying probabilistic model used for ranked retrieval: a design property termed *content independence* in [6]. The instantiation of these probabilistic operators is implementation dependent and does not influence the global system architecture. This gives us the opportunity to change the probabilistic model used or to modify the existing model while keeping the system framework, creating the opportunity to compare different probabilistic models with minimal implementation effort.

4. PHYSICAL LEVEL

The physical level of the TIJAH system relies on the MonetDB binary relational database kernel [1]. This Section details implementation and execution strategy for each of the patterns.

The text extents used at the logical level are represented by XML text regions at the physical level, and encoded using a preorder/postorder tree encoding scheme, following [9, 10]. The XML text regions are stored as three-tuples $\{ s_i, e_i, t_i \}$, where:

- s_i and e_i represent the start and end positions of XML region i;
- t_i is the (XML) tag of each region.

The set of all XML region tuples is named the *node index* $\mathcal{N}$. Index terms present in the XML documents are stored in a separate relation called the *word index* $\mathcal{W}$. Index terms are considered text regions as well, but physically the term identifier is re-used as both start and end position to reduce memory usage. The physical layer has been extended with the text region operators shown in Table 6. Boolean predicate filters are always applied first. For further details on this indexing scheme, refer to [5, 14].

4.1 Pattern 1

Pattern 1 for VCAS Processing pattern 1 in Table 1 requires two basic steps: relating node-sets *xp* and *axp* to each other, and processing the *about* operator. Nodesets *xp* and *axp* must have a parent - descendant[1] structural relation-

[1]Parent - child relationships are considered a specific variant of parent - descendant relationships.

Table 6: Text region operators at the physical level.

Operator	**Definition**
$a \supset b$	$true \iff s_b > s_a \wedge e_b < e_a$
$a \subset b$	$true \iff s_a > s_b \wedge e_a < e_b$
$A \bowtie_{\supset} B$	$\{(s_a, s_b) \vert\ a \leftarrow A,\ b \leftarrow B,\ a \supset b\}$
$A \bowtie_{\subset} B$	$\{(s_a, s_b) \vert\ a \leftarrow A,\ b \leftarrow B,\ a \subset b\}$

Table 4: Probabilistic region algebra operators. Note that the "ranked containing" and "ranked and" operators are used to define the *about* function.

Operator	Operator description	Operator usage examples
$A \triangleright B$	ranked containing (based on LM)	$L(axp) \triangleright I(t)$
$A \trianglerighteq B$	average containing	$L(xp) \trianglerighteq (L(axp) \triangleright I(t))$
$A \Delta B$	ranked and (based on LM)	$L(xp) \trianglerighteq ((L(axp) \triangleright I(t_1))\Delta(L(axp) \triangleright I(t_2)))$
$A \trianglelefteq B$	average contained	$(L(xp1) \trianglerighteq (L(axp1) \triangleright I(t_1))) \trianglelefteq (L(xp2) \trianglerighteq (L(axp2) \triangleright I(t_2)))$
$A \bigtriangleup B$	complex and	$(L(xp) \trianglerighteq (L(axp1) \triangleright I(t_1))) \bigtriangleup (L(xp) \trianglerighteq (L(axp2) \triangleright I(t_2)))$
$A \bigtriangledown B$	complex or	$(L(xp) \trianglerighteq (L(axp1) \triangleright I(t_1))) \bigtriangledown (L(xp) \trianglerighteq (L(axp2) \triangleright I(t_2)))$

Table 5: Pattern definitions based on probabilistic region algebra operators.

Pattern	Algebraic expression
$P_1(xp, axp, t)$	$L(xp) \trianglerighteq (L(axp) \triangleright I(t))$
$P_1(xp, axp, p)$	$L(xp) \trianglerighteq ((L(axp) \triangleright I(t_1))\Delta(L(axp) \triangleright I(t_2))\Delta...\Delta(L(axp) \triangleright I(t_n)))$
$P_2(xp, axp1, axp2, tp1, tp2)$	$P_1(xp, axp1, tp1) \bigtriangleup P_1(xp, axp2, tp2)$
	$P_1(xp, axp1, tp1) \bigtriangledown P_1(xp, axp2, tp2)$
$P_3(xp1, xp2, axp1, axp2, tp1, tp2)$	$P_1(xp2, axp2, tp2) \trianglelefteq P_1(xp1, axp1, tp1)$

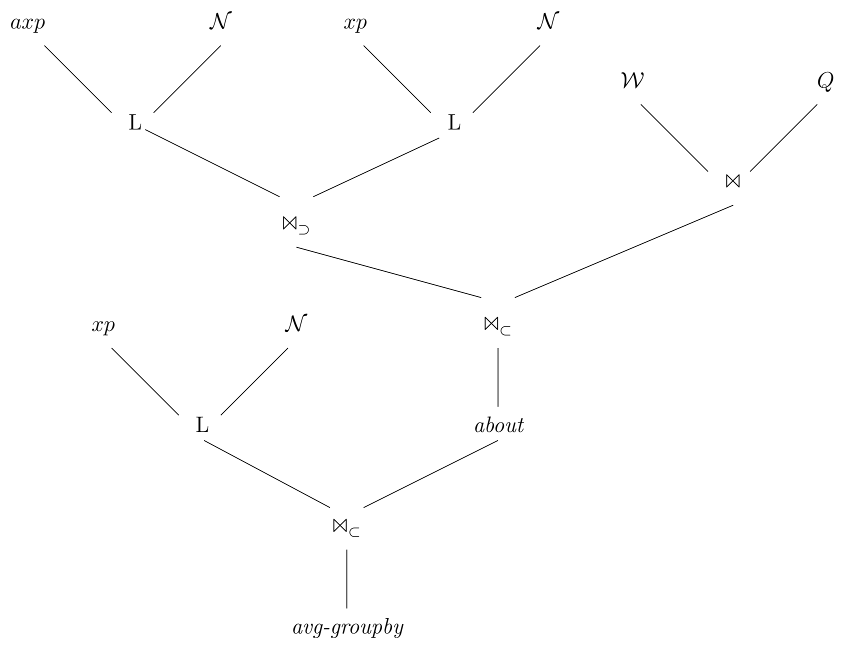

Figure 3: Physical query plan for pattern 1.

ship. So, the pattern is processed as follows (visualized in Figure 3):

- Determine the correct *axp* node-set for ranking. On the physical level, this is done by executing a containment join between the node-sets *xp* and *axp*: $axp \bowtie_{\subset} xp$. The result of this containment join is *cxp* or the set of those nodes of *axp* which are contained within nodes in *xp*;
- Perform the *about* operation on the nodes in *cxp* (the combination of $\triangleright$ and Δ operators on the logical level);
- Return the ranking for the *xp* node-set, based on the rankings of the nodes present in *cxp*. Note that it is possible that the ranking returns a ranking for multiple *axp* descendant nodes for a single *xp* node (for example, multiple sections within an article). In that case, we take the **average** as the final score for the *xp* node in question. This step is the physical equivalent of the logical $\triangleright$ (one descendant of the type of *axp*) or logical $\trianglerighteq$ (multiple descendants of the type of *axp*) operator.

Pattern 1 for SCAS The processing of pattern 1 for the SCAS run does not differ from the processing performed for the VCAS run. The containment join will automatically remove those *xp* nodes not containing one or more *axp* nodes. This ensures only the 'correct' *axp* nodes, those within a node from the *xp* node-set, will be ranked.

4.2 Pattern 2

Pattern 2 for VCAS For the processing of pattern 2 for the VCAS scenario, we assume that conjunctions and disjunctions specified in the query relate to the structure, and never to the query terms. In case node-sets *axp1* and *axp2* are equal, the pattern is rewritten to a pattern 1. If the node-sets *axp1* and *axp2* are not equal, it is possible these node-sets represent completely different parts of the (sub)tree below *xp*, as depicted in Figure 1. In path-based terms, if the (sub)tree starting at *xp* does not contain both paths *axp1* and *axp2*, that *xp* tree cannot be relevant for the strict scenario.

However, for a more vague query scenario, we argue that the absence of a descendant node does not render the requested (ancestor) target node irrelevant completely. Consider the following expression:

```
/article[
    about(./abstract, 'information retrieval')
            AND about(.//section, 'XML data')
    ]
```

If an article contains no abstract, but it does score on 'XML data' in one or more of the sections, the question is whether the article is completely irrelevant. For a vague retrieval scenario this might not be the case. Therefore, we decided to process these expression types as follows. We split up the expression into a series of pattern 1 expressions, and combine the results of the individual pattern 1 executions. The example above is split up into the following two pattern 1 expressions:

```
- /article[about(./abs, 'information retrieval XML data')]
- /article[about(.//sec, 'information retrieval XML data')]
```

Both subpatterns are processed as pattern 1. The two resulting node-sets need to be combined for a final ranking. An intuitive combination function for the $\triangle$ operator is taking the **minimum** of the (non-zero) descendant scores, and for the $\bigtriangledown$ operator the **maximum**. Note that, alternatively, a more formal probabilistic choice would be to use product and sum instead of minimum and maximum; whether this yields better results is an open question for further research.

Pattern 2 for SCAS For the SCAS scenario, all of the descendant nodes present in *axp1* and *axp2* need to be present in the context of an *xp* node. In path-based terms: if the path *xp* does not contain both a path *axp1* and a path *axp2*, the path *xp* cannot be relevant. We filter out those *xp* paths, not containing both the *axp1* and *axp2* paths. This additional filtering step and the choice of operator to implement the complex 'and' ($\triangle$) and 'or' ($\bigtriangledown$) operators define together the difference between strict and vague scenarios.

4.3 Pattern 3

Pattern 3 for VCAS Pattern 3 can be processed like pattern 2, except for the fact that the target element now lies deeper in the tree. We process this pattern by first splitting it up into multiple instances of pattern 1:

```
- xp[about(axp1, 't1/p1 t2/p2')]
- xp/xp2[about(axp2, 't1/p1 t2/p2')]
```

The pattern 1 execution already provides for aggregation of scores of a set of nodes of the same type, within a target element. The question remains however how to combine the scores of the nodes present in node-sets */xp/axp1* and */xp/xp2/axp2*. Like before, these node-sets can represent nodes in completely different parts of the (sub)tree.

Based on the observation that the user explicitly asks for the nodes present in the */xp/xp2* node-set, we decided to use the rankings of those nodes as the final rankings. The first *about* predicate reduces node-set *xp* to those nodes for which a path *axp1* exists. For the vague scenario however, we argue that absence or presence of *axp1* does not really influence target element relevance (similar to pattern 2 in subsection 4.2).

Summarizing, the first *about* predicate in the pattern mentioned at the start of this subsection is dropped, rewriting the resulting pattern to a pattern 1 instance:

```
/xp/xp2[about(axp2, 't1/p1 t2/p2')]
```

This results in the following execution strategy for pattern 3 under the VCAS scenario: remove all *about* predicates from all location steps, except for the *about* predicate on the target element.

Pattern 3 for SCAS The processing of pattern 3 for the SCAS scenario is stricter in the sense that we can not simply drop intermediate *about* predicates, as we did for the VCAS scenario. The general procedure consists of:

- splitting up the pattern into separate location steps;
- structural correlation of the resulting node-sets of each location step.

The target elements are ranked by their corresponding *about* predicate only; thus, ignoring the scores produced for the other *about* clauses in the query. Like in pattern 1, the target element can have multiple descendants; in that case, the descendants' scores are averaged to produce the target element scores.

As an example, consider the following expression:

```
    /article[about(./abstract, 't1/p1')]
            //section[about(./header, 't2/p2')]
            //p[about(., 't3/p3')]
```

We first split up the above expression into:

```
- /article[(about(./abstract, 't1/p1 t2/p2 t3/p3')]
- //section[about(./header, 't1/p1 t2/p2 t3/p3')]
- //p[about(., 't1/p1 t2/p2 t3/p3')]
```

All of the patterns above produce intermediate result node-sets that have to be structurally correlated to each other. We can choose to perform a top-down correlation sequence, or a bottom-up correlation sequence consisting of containment joins. The choice between a top-down or bottom-up sequence can be an optimization decision, made at runtime by the retrieval system. For example, if a collection contains many paragraph elements, not contained within article elements, the system might decide to limit the amount of unnecessary executed *about* predicates by choosing a top-down approach. In the current implementation, the patterns are always processed top-down.

5. EXPERIMENTS

For the content only (CO) topics, we designed three experimentation runs. The first run (R_{art}) represents the baseline run of 'flat-document' retrieval, i.e., retrieval of documents which possess no structure. After examination of the document collection, we decided to perform retrieval of article-components. The second run regarded all subtrees in the collection as separate documents (R_{comp}). For the third run we re-used the result sets of the second run and used a log-normal distribution to model the quantity dimension ($R_{comp-logn}$). To penalize the retrieval of extremely long document components, as well as extremely short document components, we set the mean at 2516. Experiments for INEX 2002 showed that 2516 words was the average document component length of relevant document components according to the strict evaluation function used in INEX 2002. Table 7 gives a summary of our experimentation runs.

For both the SCAS (strict content-and-structure) and VCAS (vague content-and-structure), we submitted one run each

Table 7: Original CO experimentation runs; note that we used a length of 2516 as preferred component length for the $R_{comp-logn}$ run. The experiments for INEX 2002 showed 2516 was the average document component length of relevant components, according to the strict evaluation function used in INEX 2002.

Run	Retr. Unit	Dimension(s)	MAP
R_{art}	$\{tr('article')\}$	*topicality*	0.0392
R_{comp}	$\{tr('*')\}$	*topicality*	0.0387
$R_{comp-logn}$	$\{tr('*')\}$	*top., quant.(2516)*	0.0374

(not mentioned in Table 7); the topics executed according to the conceptual, logical and physical SCAS and VCAS pattern rule-sets as detailed in the previous Sections. The mean average precision (MAP) value of the SCAS run is 0.2595.

The originally submitted CO-runs all used the keywords present in the keyword-element of each topic. Before executing each topics, query stop words were removed using the SMART query stop word list, and all remaining keywords were stemmed with the Porter stemmer. Stop word removal (using the SMART stop word list) and stemming was also performed on the indexed collection terms, as well as the removal of those terms shorter than 2 characters and longer than 25 characters.

We performed several additional CO runs of which the mean average precision values are summarized in Table 8.[2] First, we extracted, for each topic, the terms occurring in the title *about* clauses (T) and in the description (D) and keyword (K) component text. We then made combinations of the T, D and K keyword sets, and used the combinations in additional runs (TD and TK). Second, we also created CO-runs where we replaced the log-normal element length prior (*logn* runs) with a standard element length prior (*logs* runs):

$$lp(E) = \log(P(E)) = \log(\sum_t tf(t, E))$$

Finally, after observing a big difference in system performance with the approach by Sigurbjörnsson, Kamps and de Rijke [19], which is based on the same language modeling technique, we decided to reproduce their approach of combining surrounding document evidence with element evidence (*aw* runs).

From the average precision values in Table 8, the following observations are clear:

- large elements should not be discounted (under the current metrics of evaluation; difference between *logn* and *logs* runs);
- combining element scores with their surrounding con-

[2]The differences between the R_{comp} and $R_{comp-logn}$ MAP scores in Tables 7 and 8 originate from the (different) ordering of elements with equal score.

Table 8: Mean average precision values for the additional CO runs. The last three columns denote the topic part used for the run: T for title, TD for title and description terms, and TK for title and keyword terms. For evaluation, the strict evaluation measure (for 2003) was used.

Run	Task	K	TD	TK
R_{comp}	CO	0.0341	0.0383	0.0447
$R_{comp-logn}$	CO	0.0351	0.0390	0.045
$R_{comp-logs}$	CO	0.0652	0.0766	0.0740
$R_{comp-logn-aw}$	CO	0.0697	0.0863	0.0905
$R_{comp-logs-aw}$	CO	0.1043	0.1224	0.1205

text scores appears to improve performance significantly (*aw* runs);

- in spite of the noise in the description text, using the description terms improves retrieval results (comparing columns *K* and *TD*).

We plan to further investigate the cause of the performance difference between the *logn* and *logs* runs. One explanation could be that the log-normal's mean value of 2516 words, as desired component size, is not the correct value given the relevance assessments. Another explanation for this discrepancy between evaluation results and our intuition, expressed in the log-normal length prior, could be sought in the current evaluation metrics that reward exhaustivity over specificity.

Besides measuring the effectiveness of our retrieval system, we also measured the efficiency of indexing and querying the collection. Table 9 shows the average topic execution times of all created runs. For a given run, we averaged the topic execution times of the topics in that given run (with CO runs having 36 topics and the SCAS and VCAS runs having 30 topics). All measurements are wallclock timings, measured in seconds. The hardware used for the executions of the runs is an AMD Opteron machine, running at 1.4GHz and having 2GB of main memory. The indexing time is divided into two separate parts:

- the time needed for insertion of data T_{insert}, measured at 176 seconds;
- the time needed for post-processing $T_{postprocess}$, measured at 191 seconds. Post-processing consists of determining collection frequencies, component text lengths (component lengths disregarding markup) and indexing of topics.

Memory use of our system varied between 250MB and 1GB, where 1GB was reached when materializing large components, or large component sets (large with regard to the number of components in the result set) for executing the language model. Moreover, memory use was increased by behavior of the database kernel used: the kernel loads tables completely into memory when they are needed, even if not all parts of the table are used. This redundant memory use as a result of loading irrelevant data can be avoided by, for example, horizontal fragmentation of the tables as in [18]. The extra time needed for the *logn* and *logs* runs (when compared to the *comp* run) can be explained by extra join-operations against parts of the index, needed for retrieving the component text lengths and calculation of the logarithm values. Also, the *aw* runs take more execution time as a result of the extra containment joins needed to resolve the specified structural constraints.

Table 9: Average topic execution times for all runs, in seconds (wallclock time). Note that the first row is the original article run, performed with keywords only (the K column). The execution times of our originally submitted three runs are displayed in the first three rows and the third column (boldfaced). The other timings are the timings for the additional unofficial runs, and the last two rows show the execution times for our original SCAS and VCAS runs.

Run	Task	K	TD	TK
R_{art}	CO	**6.75**	-	-
R_{comp}	CO	**44.08**	68.19	53.22
$R_{comp-logn}$	CO	**45.13**	69.58	54.47
$R_{comp-logs}$	CO	45.25	69.69	54.47
$R_{comp-logn-aw}$	CO	47.16	72.22	56.80
$R_{comp-logs-aw}$	CO	47.25	74.44	57
R_{scas}	SCAS	-	-	35.37
R_{vcas}	VCAS	-	-	35.24

The time needed for indexing can be reduced further. First, for the sake of simplicity, the system indexes the full XPath (in string format) for each component in the collection. This full XPath indexing is redundant and can be replaced by a facility to resolve the component XPaths when presenting results to the user, or by a more compact index structure. Second, we are looking into possibilities for encoding other parts of the index into more compact structures, e.g., bitvectors.

6. CONCLUSIONS AND FUTURE WORK

Our participation in INEX can be summed up as an exercise in applying current and state of the art information retrieval technology to a structured document collection. We described a relatively straightforward approach to simplify the implementation of retrieval models that combine structural and content properties. We hope to take advantage of this flexibility to a larger extend in our future research, as the current approach to retrieval has only used a small proportion of all the structural information present in XML documents. Other research includes more extensive experimentation in the area of relevance feedback, and develop a different normalization mechanism to remove the bias of the language model on short components. Lastly, we aim to improve the efficiency of the system, both memory and CPU wise, by applying horizontal fragmentation and encoding of data into more compact structures.

7. REFERENCES

[1] P. Boncz. *Monet: a Next Generation Database Kernel for Query Intensive Applications.* PhD thesis, CWI, 2002.

[2] F.J. Burkowski. Retrieval Activities in a Database Consisting of Heterogeneous Collections of Structured Texts. In *Proceedings of the 15th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–125, 1992.

[3] C.L.A. Clarke, G.V. Cormack, and F.J. Burkowski. An Algebra for Structured Text Search and a Framework for its Implementation. *The Computer Journal*, 38(1):43–56, 1995.

[4] M. Consens and T. Milo. Algebras for Querying Text Regions. In *Proceedings of the ACM Conference on Principles of Distributed Systems*, pages 11–22, 1995.

[5] A. P. de Vries, J. A. List, and H. E. Blok. The Multi-Model DBMS Architecture and XML Information Retrieval. In H. M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors, *Intelligent Search on XML*, volume 2818 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI), Springer-Verlag*, pages 179–192. Springer-Verlag, Berlin, New York, etc., August 2003.

[6] A.P. de Vries. Content independence in multimedia databases. *Journal of the American Society for Information Science and Technology*, 52(11):954–960, September 2001.

[7] M.Fernández et al. XML Path Language (XPath 2.0). Technical report, W3C, 2003.

[8] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[9] T. Grust. Accelerating XPath Location Steps. In *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data*, pages 109–120, 2002.

[10] Torsten Grust and Maurice van Keulen. Tree Awareness for Relational DBMS Kernels: Staircase Join. In H. M. Blanken, T. Grabs, H.-J. Schek, R. Schenkel, and G. Weikum, editors, *Intelligent Search on XML*, volume 2818 of *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, pages 179–192. Springer-Verlag, Berlin, New York, etc., August 2003.

[11] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Twente, The Netherlands, 2000.

[12] D. Hiemstra. A database approach to content-based XML retrieval. In *Proceedings of the First Workshop of the Initiative for the Evaluation of XML Retrieval*, 2002.

[13] J. Jaakkola and P. Kilpelainen. Nested Text-Region Algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, 1999.

[14] J.A. List and A.P. de Vries. CWI at INEX 2002. In *Proceedings of the First Workshop of the INitiative for the Evaluation of XML Retrieval (INEX)*, ERCIM Workshop Proceedings, 2002.

[15] Katsuya Masuda. A Ranking Model of Proximal and Structural Text Retrieval Based on Region Algebra. In *Proceedings of the ACL-2003 Student Research Workshop*, 2003.

[16] V. Mihajlovic, D. Hiemstra, and P. Apers. On Region Algebras, XML Databases, and Information Retrieval. In *Proceedings of the 4th Dutch-Belgian Information Retrieval Workshop, to apear*, 2003.

[17] R.C. Miller. *Light-Weight Structured Text Processing*. PhD thesis, Computer Science Department, Carnegie-Mellon University, 2002.

[18] A.R. Schmidt, M.L. Kersten, M.A. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases (in conjunction with ACM SIGMOD)*, pages 47–52, 2000.

[19] B. Sigurbjörnsson, J. Kamps, and M. de Rijke. An element-based approach to XML retrieval. In N. Fuhr, M. Lalmas, and S. Malik, editors, *Proceedings of the Second Workshop of the INitiative for the Evaluation of XML retrieval (INEX)*, ERCIM Workshop Proceedings, 2004.

[20] D. Tsichritzis and A. Klug. The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. *Information systems*, 3:173–191, 1978.