TIJAH: Embracing IR Methods in XML Databases

Johan List[1] Vojkan Mihajlović[2] Georgina Ramírez[1]
Arjen P. de Vries[1] Djoerd Hiemstra[2]
Henk Ernst Blok[2]

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

[2]University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands
{v.mihajlovic, d.hiemstra,
h.e.blok}@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 *score region algebra* (SRA) 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 an analysis of experiments performed with the INEX test collection.

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 [2]. 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 possibility of using different data models and query languages at each of those levels [25]. 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 [10].

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 region algebra extension for supporting ranked retrieval.

The TIJAH system has been used for participation in the Initiative for the Evaluation of XML Retrieval (INEX). INEX offers an XML document collection of approximately 500MB, a set of topics, relevance assessments and a collection of evaluation measures. The document collection consists of IEEE scientific articles of various research fields from the years 1995 to 2002. In 2002 and 2003, the topics were created by the participating groups in the Initiative. The participants were also asked to perform the relevance assessments for their accepted final topics. Relevance assessments were done on two dimensions: *exhaustivity* and *specificity*. Exhaustivity describes the extent to which a retrieved component discusses the topic of request, while specificity describes the extent to which the retrieved component focuses on the topic of request. Both dimensions have a four-graded scale.

Three types of search tasks were defined for INEX 2003: Content-Only (CO), Strict Content-and Structure (SCAS) and Vague Content-and-Structure (VCAS). The CO task focused on retrieval in the XML collection without imposing any structural constraints. The SCAS and VCAS tasks were employed for queries that consider both structure and content. The SCAS task considered any structural constraints specified as strict, that is, the structural constraints specified in the query had to be fully satisfied. For the VCAS scenario, structural constraints could be relaxed.

Our paper is organized along the layers of the TIJAH system design. Section 2 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 introduces the score region algebra (SRA) and explains how the three query patterns are expressed at the logical level. Section 4 explains how the algebraic expressions are mapped into efficient relational algebra expressions over a physical representation of the document collection using the 'pre-post numbering scheme'. We conclude with a discussion of experiments performed with our approach for the three INEX search tasks.

2 Conceptual Level

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 the retrieval model.

2.1 Retrieval Model

The retrieval model used for the *about* function has been based on our previous work, applying the language modeling approach to information retrieval to the INEX retrieval tasks, see e.g. [17, 14]. The exhaustivity dimension of relevance seems to be captured rather intuitively in the language modeling approach. It estimates relevance using a foreground model and a background model (both probability distributions: respectively $P(T_i|D_j)$ denoting the probability of a term T_i given a particular document component[1] D_j, and $P(T_i)$ denoting the probability of the term T_i in general English), which are linearly combined to give a final score for probability of relevance $P(R)$:

$$P(R) \quad = \quad \prod_{i=1}^{n} (\lambda P(T_i|D_j) + (1-\lambda)P(T_i))$$

in which n is the query length. The linear combination of the probabilities is also known as smoothing, and is done to avoid the sparse data problem [13]. We want to avoid assigning a value of 0 to the entire product of probabilities, when a single term T_i does not occur in the document component D_j (the foreground probability value for term i will then be 0).

The foreground and background probabilities, $P(T_i|D_j)$ and $P(T_i)$ respectively, are based on specific collection statistics and are defined as maximum likelihood estimators. In the following

[1] A document component is an XML fragment, or, a subtree of an XML syntax tree.

equations, variable t ranges over the term domain, i.e., all terms in the collection, and d over the document domain, i.e., all documents in the collection. For the foreground probability, we use a well-known estimator based on the term frequency (measuring how many times a term T_i occurs in a document component D_j:

$$P(T_i|D_j) = \frac{tf_{i,j}}{\sum_t tf_{t,j}}$$

For estimating the background probability, common estimators are collection frequencies (measuring how many times a term occurs in the collection) or document frequencies (measuring in how many unique documents the term occurs). Document frequencies are actually document component frequencies in our case, measuring in how many unique document components a term occurs.

$$P_{cf}(T_i) = \frac{cf_i}{\sum_t cf_t},\ P_{df}(T_i) = \frac{df_i}{\sum_t df_t}$$

The language model scores can be adjusted by a document component prior. The 'standard' choice for a prior probability $P(D_j)$ has been based on the rationale that longer document components have a higher *a priori* probability of containing relevant information, simply due to their length:

$$P_{ls}(D_j) \ = \ \frac{\sum_t tf_{t,j}}{\sum_{t,d} tf_{t,d}}$$

Note that this 'standard prior' rewards long document components, which might not be very specific. Nevertheless, the prior produces good overall effectiveness in the experiments presented in Section 5. The specificity dimension of relevance might be better described by an alternative length prior, e.g., following a standard-log-normal distribution. We speculate here that the characteristics of the standard-log-normal distribution reflects behavior of a real user. Very short documents are likely to contain insufficient information, and consequently, are likely to not be satisfying retrieval units. On the other hand, very long document components require much more effort from the user in locating the relevant pieces of information, which could be considered unsatisfying as well. The characteristics of the log-normal distribution fit this user preference: its steep slope at the start discounts the likelihood that very short document components are relevant, while the long distribution tail reflects that we do not expect long document components to be very focused on the topic of request. The standard-log-normal length prior is defined as follows, where $|D_j|$ denotes the length of the document component D_j.

$$P_{logn}(D_j) \quad = \quad \frac{1}{|D_j|\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\log|D_j|-\mu}{\sigma})^2}$$

2.2 The CO Task

For INEX 2002 and 2003, we combined the estimators outlined in subsection 2.1 in the following two scoring functions for performing the CO retrieval task (RSV stands for Retrieval Status Value of the document component D_j):

$$\begin{aligned} RSV_1 &= \prod_{i=1}^{n} (\lambda P(T_i|D_j) + (1-\lambda) P_{cf}(T_i)) \\ RSV_2 &= P_{logn}(D_j)\ RSV_1 \end{aligned}$$

We used the scoring functions in our three originally submitted experimental scenarios. Scoring function RSV_1 was used for two retrieval scenarios:

- fixed retrieval, where the retrieval unit was restricted to article components;
- 'free retrieval', where the retrieval unit was not restricted to any specific component type.

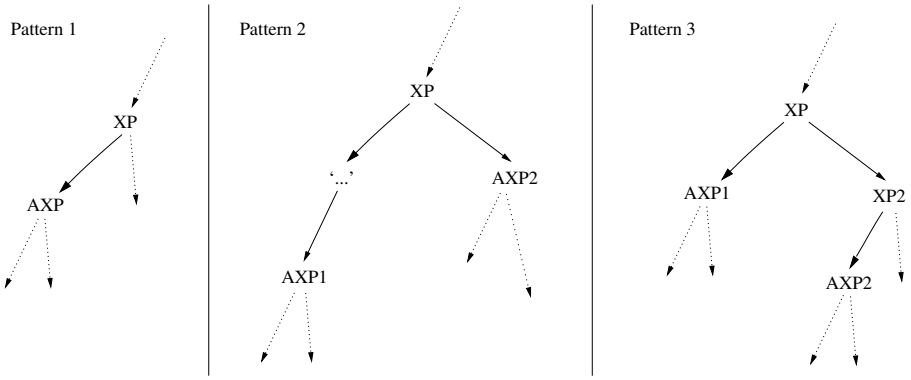

Figure 1: Visualisation of different query patterns.

For additional experiments, we created three other scoring functions:

$$
\begin{aligned}
RSV_3 &= P_{ls}(D_j) RSV_1 \\
RSV_4 &= \prod_{i=1}^{n} (\lambda P(T_i|D_j) + (1-\lambda) P_{df}(T_i)) \\
RSV_5 &= P_{ls}(D_j)\ RSV_4
\end{aligned}
$$

Finally, comparison of the evaluation results between our TIJAH system and the system described by Sigurbjörnsson, Kamps and de Rijke [23], lead to the discovery of a significant performance difference. Since both systems are based on the same language modeling technique, we decided to also experiment with their approach, using evidence from the surrounding article component; the score of each component is smoothed with the score of its ancestor article node. We refer to the article-weighted versions of RSV_2 and RSV_3 as RSV_6 and RSV_7 respectively.

2.3 The SCAS and VCAS Tasks

For both the SCAS and VCAS tasks, we introduced the concept of *patterns*. Since we lacked an automatic query processing mechanism at the time of evaluation, we processed the queries manually (but in a mechanic fashion). Inspection of the topic set highlights three repeating query patterns, shown in Table 1. Processing these 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.

Table 1: SCAS and VCAS pattern set. Note that *xp*, *xp2*, *axp*, *axp1* and *axp2* are location steps; *'t|p'* denotes a set of terms or phrases.

Pattern	**Pattern definition**
P_1	`xp[about(axp, 't|p')]`
P_2	`xp[about(axp1, 't1|p1') AND about(axp2, 't2|p2')]` `xp[about(axp1, 't1|p1') OR about(axp2, 't2|p2')]`
P_3	`xp[about(axp1, 't1|p1')]//xp2[about(axp2, 't2|p2')]`

The basic pattern for all XPath based queries is the single *location step*, as defined in [9], augmented with an *about* function call (pattern 1 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 XPath location steps may also apply

(Boolean) predicate filters, e.g., selecting nodes with a particular value range for year (*yr* element node in INEX collection).

Pattern 1 is considered to be the elementary pattern for processing. The two other (more complex) patterns 2 and 3 are essentially multiple interrelated instances of the basic pattern 1.

2.3.1 Pattern 1

The first query pattern consists of one location step (*xp*) used to identify the nodes to be retrieved, ranked by an *about* expression over a node-set reached by a second location step (*axp*). For pattern 1, no distinction is made between SCAS and VCAS scenarios. Scores collected for the nodes satisfying location step *axp* are aggregated to a single score for the target node (using function $f_{\blacktriangleright}$, defined at the logical layer, see Section 3.2).

2.3.2 Pattern 2

Pattern 2 distinguishes between the VCAS and the SCAS tasks, where we argue for the vague query scenario that absence of specified descendant steps does not render the requested (ancestor) target nodes irrelevant completely.

Pattern 2 for VCAS For pattern 2 in 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.

Consider the following expression:

```
//article[
    about(.//abs, 'information retrieval')
            AND about(.//sec, '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. Also, note that we use the terms and phrases of all separate *about* clauses together in each instance (experiments have shown that this gives better average performance on the INEX topics). So, the example above is split 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 are combined for a final ranking (see Section 3.2).

Pattern 2 for SCAS 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. Therefore, the distinction between SCAS and VCAS scenarios for pattern 2 is the requirement that all of the descendant nodes present in *axp1* and *axp2* are contained in the context of the *xp* target nodes.

2.3.3 Pattern 3

Similarly to pattern 2, the two CAS scenarios are treated differently for 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.

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, and process it as a vague scenario for pattern 2.

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 (similar to pattern 3 VCAS scenario);
- structural correlation (using descendant or ancestor axis tests) of the resulting node-sets of each location step.

The target elements for each sub pattern are ranked by their corresponding *about* predicate.

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')]
- //article//section[about(.//header, 't1|p1 t2|p2 t3|p3')]
- //article//section//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. Furthermore, the ranking results have to be combined (propagated) during this correlation, and assigned to the pattern that specifies the desired result node-set.

3 Logical Level

Patterns recognised at the conceptual level are translated into physical query plans via an intermediate step, the logical level. We have based the logical query algebra and data model on a region algebra extended to support ranked retrieval at the logical level. The region algebra concept was introduced by Salminen and Tompa [21] and Burkowski [3] for searching in structured documents and supporting search and ranked retrieval in text dominated databases, respectively. It has been extended to support overlap between regions by Clarke et al. [4], with operators for proximity search by Baeza-Yates and Navarro [1], and with direct inclusion (parent-child relation) and "both included" operators by Consens and Milo [5]. Jaakkola and Kilpelainen [15] introduced nesting properties in region algebra, while Miller [20] used the region algebra approach for processing of structured text. Finally, an attempt for adapting region algebra to ranked retrieval has been presented in [19, 18].

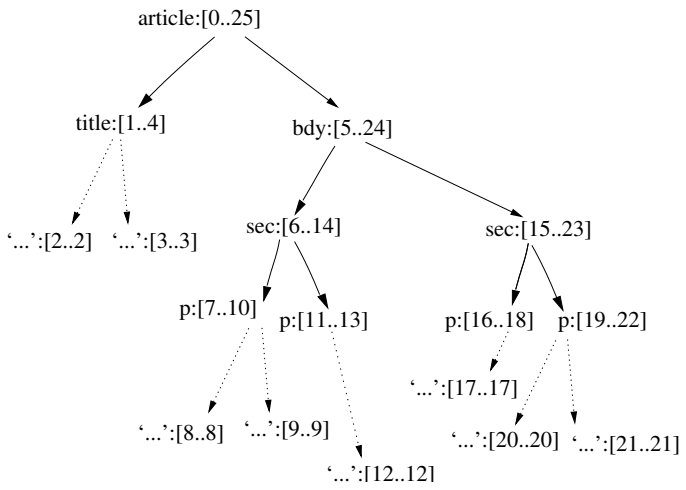

Figure 2: Example XML syntax tree with start and endpoint assignment.

The basic idea behind the region algebra approaches is the representation of text documents as a set of *extents*, where each extent is defined by its start and end positions. The application of the idea of text extents to XML documents is straightforward. Regarding 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 that entirely 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 the (closed) interval [5..24].

3.1 A Score Region Algebra and Data Model

Since we focus on XML documents and want to support as much as we can from the W3C XML query languages and IR query languages, we decided to use the rich information set of the XML data model [6] as a base for the definition of our score region algebra data model. Furthermore, given an expressive data model we would be able to distinguish between different information items in XML (see [6]) and to further extend the algebra with new operators and concepts for ranked retrieval. As we enriched the region definition with a score concept, we named the algebra score region algebra, or SRA in short.

The XML data model consists of element, text, comment, and processing instruction nodes, as well as attribute information and text node content. Although XML documents might be considered as graphs because of ID/IDREF typed attributes, we will simplify the XML structure and treat these entities as they are organized in a hierarchical (tree-like) structure. Regarding the complexity of the XML data model we could not directly apply any of the previous region algebra approaches for defining the logical data model.

In the specification of the SRA data model, we have to distinguish between the different node types in XML documents to provide a uniform platform for defining the region algebra operators. To be able to represent XML properly, we extended the definition of a region. The logical data model is based on *region sets*, where each region is defined as follows:

Definition 1 *The SRA data model is defined on the domain R which represents a set of region tuples. Region tuple r ($r \in R$), $r = (s, e, n, t, p)$, is defined by five attributes: region start attribute - s, region end attribute - e , region name attribute - n, region type attribute - t, and region score attribute - p. Region start and end attributes must satisfy ordering constraints ($e_i \geq s_i$).*

The semantics of region start and region end attributes are the same as in other region algebra approaches: they denote the bounds of a region. The region name attributes are used to denote

Table 2: Basic score region algebra operators.

Operator	Operator definition
$\sigma_{n=name,t=type}(R)$	$\{r \mid r \in R \wedge t = type \wedge n = name\}$
$R_1 \sqsupset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \exists r_2 \in R_2 \wedge s_1 < s_2 \wedge e_1 > e_2\}$
$R_1 \not\sqsupset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \nexists r_2 \in R_2 \wedge s_1 < s_2 \wedge e_1 > e_2\}$
$R_1 \sqsubset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \exists r_2 \in R_2 \wedge s_1 > s_2 \wedge e_1 < e_2\}$
$R_1 \not\sqsubset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \nexists r_2 \in R_2 \wedge s_1 > s_2 \wedge e_1 < e_2\}$
$R_1 \sqcap R_2$	$\{r_3 \mid r_3 \in R_1 \wedge r_3 \in R_2\}$
$R_1 \sqcup R_2$	$\{r_3 \mid r_3 \in R_1 \vee r_3 \in R_2\}$

node names, content words, attribute names, attribute values, etc. To distinguish between different name "roles" in XML we used the region type information item. We used *node* for the element node in XML, *text* for the text node, *term* for the term present in a text node, etc. Finally, the region score information item is used to specify the relevance score of a region with respect to a given query.

In SRA the result of the application of operators can not introduce new regions, that is, regions with different region bounds (i.e. s and e values) than originally specified in the database. Furthermore, the operators can not change the name and type (n and t) of regions identified in the database. However, algebra operators are allowed to change the value of the region score item (p). SRA supports ranked retrieval as a part of the algebra, and not as a side-effect, which distinguishes it from other proposals (specifically, [3] and [19]).

3.2 SRA Operators

Here we will define the score region algebra operators. SRA operators are based on the operators specified in the previous region algebra approaches, extended to support the SRA data model and enable ranked retrieval in XML documents.

Table 2 defines the following seven basic SRA algebra operators: selection (σ), containing ($\sqsupset$), not containing ($\not\sqsupset$), contained in ($\sqsubset$), not contained in ($\not\sqsubset$), region set intersection ($\sqcap$), and region set union ($\sqcup$). We use R_i ($i = 1, 2, ...$) to denote the region sets, their corresponding non-capitals to denote regions in these region sets (r_i), and corresponding indexed non-capitals to denote region attributes (s_i, e_i, n_i, t_i, p_i). The operators take one or two region sets as operands and produce a region set as result. We introduced a selection operator (σ) to enable the selection of regions based on a name and type region attributes. Leaving the selection criterion for one of the attributes unspecified corresponds to a wild-card, i.e., $\sigma_{t=node}$ will select all regions that have an XML element node type, regardless of their name attribute.

To enable the score computation and the region ranking based on computed scores new region algebra operators are defined, as depicted in Table 3. Four complex scoring functions are introduced in the additional SRA operators: ($f_{\sqsupset}$, $f_{\not\sqsupset}$, $f_{\blacktriangleright}$, and $f_{\blacktriangleleft}$), as well as two abstract operators ($\otimes$ and $\oplus$). The exact specification of these operators is given on the physical level (see Section 4.2). The functions $f_{\sqsupset}(r_1, R_2)$ and $f_{\not\sqsupset}(r_1, R_2)$, applied to a region r_1 and region set R_2, should result in the numeric value that specifies the probability that the region r_1 contains or does not contain a term, or a set of terms present in R_2. In other words they should define the retrieval model used.

The functions $f_{\blacktriangleright}(r_1, R_2)$ and $f_{\blacktriangleleft}(r_1, R_2)$ define how scores are propagated to the containing or contained regions, respectively. They specify whether the propagation is performed with or without normalization. Normalization can for example be based on the region size, or on the number of regions in a region set. The abstract operators $\otimes$ and $\oplus$ specify the combination of scores in AND and OR expressions respectively (e.g., taking the minimum or product score for AND, or the maximum or the average score for OR).

Expressing query plans using the operators given in Table 2 and Table 3 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

Table 3: Region algebra operators for score manipulation.

Operator	Operator definition
$R_1 \sqsupset_p R_2$	$\{r_3 \mid r_1 \in R_1 \wedge (s_3, e_3, n_3, t_3) = (s_1, e_1, n_1, t_1) \wedge p_3 = f_{\sqsupset}(r_1, R_2)\}$
$R_1 \not\sqsupset_p R_2$	$\{r_3 \mid r_1 \in R_1 \wedge (s_3, e_3, n_3, t_3) = (s_1, e_1, n_1, t_1) \wedge p_3 = f_{\not\sqsupset}(r_1, R_2)\}$
$R_1 \blacktriangleright R_2$	$\{r_3 \mid r_1 \in R_1 \wedge (s_3, e_3, n_3, t_3) = (s_1, e_1, n_1, t_1) \wedge p_3 = f_{\blacktriangleright}(r_1, R_2)\}$
$R_1 \blacktriangleleft R_2$	$\{r_3 \mid r_1 \in R_1 \wedge (s_3, e_3, n_3, t_3) = (s_1, e_1, n_1, t_1) \wedge p_3 = f_{\blacktriangleleft}(r_1, R_2)\}$
$R_1 \blacktriangle R_2$	$\{r_3 \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge (s_1, e_1, n_1, t_1) = (s_2, e_2, n_2, t_2) \wedge (s_3, e_3, n_3, t_3) = (s_1, e_1, n_1, t_1) \wedge p_3 = p_1 \otimes p_2\}$
$R_1 \blacktriangledown R_2$	$\{r_3 \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge (s_1, e_1, n_1, t_1) = (s_2, e_2, n_2, t_2) \wedge (s_3, e_3, n_3, t_3) = (s_1, e_1, n_1, t_1) \wedge p_3 = p_1 \oplus p_2\}$

used for ranked retrieval: a design property termed *content independence* in [8]. The instantiation of these probabilistic operators is implementation dependent and does not influence the global system architecture. This gives us the opportunity to change or to modify the existing model while keeping the system framework, creating the opportunity to compare different probabilistic models with minimal implementation effort.

3.3 Pattern Representation in SRA

To be able to express patterns identified on the conceptual level, an abstract operator $L(xp)$ is introduced. The operator is actually a translation operator which translates the XPath expression in a region algebra expression. In other words, operator $L(xp)$ denotes the sequential application of XPath location steps, i.e., axis- and node-tests (a combination of containment and selection operators). Optionally, the operator also processes predicate tests on node or attribute values specified in the XPath expression. The current SRA data model does not support the child axis step and it treats it as a descendant step. Notice however, that this simplified query model is consistent with the query language proposed in [24], which forms the basis for the query language to be used at INEX 2004.

To be able to express predicate steps, extra selection criteria are introduced in the select operator, enabling numeric value comparison on SRA name concepts. These criteria are denoted as $\sigma_{n \diamond value, t=type}(R)$, where $\diamond$ is one of the numeric value tests: $\diamond \in \{=, \neq, >, <, \geq, \leq\}$. For example, using the selection operator, XPath expression $xp[.//yr \geq 1999]$ can be expressed in the score region algebra as:

$$L(xp) \sqsupset (\sigma_{n=\text{“}yr'', t=node}(C) \sqsupset \sigma_{n \geq 1999, t=text}(C)),$$

where C represents the set of all regions in a database.

Table 4 gives the probabilistic region algebra expressions corresponding to the INEX query patterns identified before. Pattern 1 distinguishes between term (tm) and phrase expressions ($pe = [tm_1, tm_2, ..., tm_m]$), and between containing and not containing criteria. Patterns 2 and 3 are rewritten into several interrelated instances of pattern 1.

Notation $tp1$ is shorthand for $tm1|pe1$ or the combination of $tm1|pe1$ and $tm2|pe2$. The choice between these options is made at the conceptual level. Similarly, $tp2$ denotes either $tm2|pe2$ or a combination of $tm2|pe2$ and $tm1|pe1$.

4 Physical Level

The physical level of the TIJAH system relies on the MonetDB binary relational database kernel [2]. This Section details the implementation of the physical region algebra operators and the execution strategy for each of the patterns.

Table 4: Pattern definitions based on score region algebra operators.

Pattern	**Algebraic expression**
$P_1(xp, axp, tm)$	$L(xp) \blacktriangleright (L(axp) \sqsupset_p \sigma_{n=tm}(C))$
	$L(xp) \blacktriangleright (L(axp) \not\sqsupset_p \sigma_{n=tm}(C))$
$P_1(xp, axp, pe)$	$L(xp) \blacktriangleright ((L(axp) \sqsupset_p \sigma_{n=tm_1}(C))$
	$\blacktriangle (L(axp) \sqsupset_p \sigma_{n=tm_2}(C)) \blacktriangle ... \blacktriangle (L(axp) \sqsupset_p \sigma_{n=tm_n}(C)))$
	$L(xp) \blacktriangleright ((L(axp) \not\sqsupset_p \sigma_{n=tm_1}(C))$
	$\blacktriangle (L(axp) \not\sqsupset_p \sigma_{n=tm_2}(C)) \blacktriangle ... \blacktriangle (L(axp) \not\sqsupset_p \sigma_{n=tm_n}(C)))$
$P_2(xp, axp1, axp2, tp1, tp2)$	$P_1(xp, axp1, tp1) \blacktriangle P_1(xp, axp2, tp2)$
	$P_1(xp, axp1, tp1) \blacktriangledown P_1(xp, axp2, tp2)$
$P_3(xp1, xp2, axp1, axp2, tp1, tp2)$	$P_1(xp2, axp2, tp2) \blacktriangleleft P_1(xp1, axp1, tp1)$

Table 5: Text region operators at the physical level.

Operator	**Definition**
$r_1 \supset r_2$	$true \iff s_2 > s_1 \wedge e_2 < e_1$
$r_1 \subset r_2$	$true \iff s_1 > s_2 \wedge e_1 < e_2$
$R_1 \ltimes_{\supset} R_2$	$\{(r_1, s_2) \mid r_1 \leftarrow R_1,\ r_2 \leftarrow R_2,\ r_1 \supset r_2\}$
$R_1 \ltimes_{\subset} R_2$	$\{(r_1, s_2) \mid r_1 \leftarrow R_1,\ r_2 \leftarrow R_2,\ r_1 \subset r_2\}$

4.1 Physical Data Model

XML text regions are encoded at the physical level using a pre-order/post-order tree encoding scheme, following [11, 12]. Regions are stored as five-tuples $(s_i, e_i, n_i, t_i, p_i)$, where

- s_i and e_i represent the start and end positions of XML region i;
- n_i is the (XML) tag of region i;
- t_i is the type of region i;
- p_i is the score assigned to region i after executing the retrieval model (0 as default value).

The set of all XML region tuples is named the *node index* $\mathcal{N}$. Information about other types of XML entities (attributes, comments, etc.) is stored in separate tables; we apply horizontal fragmentation of the region table. Yet, since these tables are not needed for the retrieval experiments, these are not further considered in paper. Terms present in the XML documents are stored in a separate relation called the *word index* $\mathcal{W}$ as two-tuples (s_i, t_i). Terms are considered regions with a length of 1, but storing them separately reduces memory usage by not having to materialise the end positions for the word index. The physical layer has been extended with the physical text region operators shown in Table 5. Boolean predicate filters are always applied first. For further details on this indexing scheme, we refer to [7, 17].

4.2 The Retrieval Model

The complex scoring functions defined at the logical level, $f_{\sqsupset}(r, R)$ and $f_{\not\sqsupset}(r, R)$, implement the about function defined in Section 2. The context region in which we perform frequency counts is denoted by r, while the set R specifies the set of term regions. Since terms are considered regions themselves, a term selection over the word index $\mathcal{W}$ gives a region set as result, i.e., $R = \sigma_{n=term}(\mathcal{W})$.

We first introduce five auxiliary functions at the physical level, to compute the term frequency - $tf(r, R)$, the 'surrounding document' term frequency - $tf'(r, R)$, the collection frequency - $cf(R)$, the document frequency - $df(R)$, and the length prior - $lp(r)$. Variable λ represents the smoothing parameter for the inclusion of background statistics, α is the combination parameter for article weighting, and μ is the mean value for the log-normal prior.

These auxiliary functions are implemented using two physical operators: the size operator $size(r)$ returns the size of a selected region r, while the count operator $|R|$ returns the number of regions in a region set R.

Function $tf(r, R)$ computes the term frequency for terms in R. The term frequency of region set R (i.e., a set of term positions) in context region r is computed as:

$$tf(r, R) = \frac{|R \bowtie_{\subset} r|}{size(r)}$$

Function $cf(R)$ computes the collection frequency of R as follows:

$$cf(R) = \frac{|R|}{size(Root)},$$

where $Root$ represents the entire XML collection, i.e., the region that is not contained by any other region in the collection.

The document frequency function computes the document frequency for a *term* present in the region set R as follows:[2]

$$df(R) = \frac{|R \bowtie_{\subset} \mathcal{N}|}{|\mathcal{N}|}$$

Length prior $lp(r)$ of region r is defined by either the standard length prior (ls) or the log-normal length prior ($logn$).

Finally, function $tf'(r, R)$ computes the term frequency in a surrounding document:

$$tf'(r, R) = \frac{|R \bowtie_{\subset} (\sigma_{n=`article'}(\mathcal{N}) \bowtie_{\supset} r)|}{size(\sigma_{n=`article'}(\mathcal{N}) \bowtie_{\supset} r)}$$

The complex scoring functions defined at the logical level can now be implemented through the combination function g, which is defined in terms of these auxiliary functions and parameters:

$$f_{\sqsupset,\lambda,\alpha,\mu} = g_{\sqsupset,\lambda,\alpha,\mu}(tf(r, R), tf'(r, R), cf(R), df(R), lp(r))$$

$$f_{\not\sqsupset,\lambda,\alpha,\mu} = g_{\not\sqsupset,\lambda,\alpha,\mu}(tf(r, R), tf'(r, R), cf(R), df(R), lp(r))$$

We can now implement the retrieval models presented in Section 2.1 at the physical level, by instantiating combination function g with the right auxiliary functions. So, the instantiation of the combination function determines the actual retrieval model used. Note that other retrieval models may require the extension of the physical layer with different auxiliary functions, e.g., to support other frequency measures.

I changed logical into physical operators in this paragraph

Finally, the definitions of operators $f_{\blacktriangleright}(r_1, R_2)$ and $f_{\blacktriangleleft}(r_1, R_2)$ are based on region containment operators and operators *size* and *count*. Thus, $f_{\blacktriangleright}(r_1, R_2)$ is defined using the expression $R_2 \bowtie_{\subset} r_1$, while $f_{\blacktriangleleft}(r_1, R_2)$ is defined using $R_2 \bowtie_{\supset} r_1$. As an example, let the complex scoring function $f_{\blacktriangleright}(r_1, R_2)$ be defined as normalisation weighted by region size. Denoting the result of the application of the containment expression by $Y = R_2 \bowtie_{\subset} r_1$, the scoring function $f_{\blacktriangleright}(r_1, R_2)$ is defined as:

$$f_{\blacktriangleright}(r_1, R_2) = \frac{\sum_{y \in Y} size(y) p_2(y)}{\sum_{y \in Y} size(y)},$$

where $p_2(y)$ is the score value of region $y \in Y$.

[2]Notice however that in the article run, the document frequency is based on the article node set only ($\sigma_{n=`article'}(\mathcal{N})$).

4.3 Patterns for The SCAS and VCAS Tasks

4.3.1 Pattern 1

Processing pattern 1 (see Table 1) consists of two basic steps: relating node-sets *xp* and *axp* to each other, and processing the *about* operator. Node sets *xp* and *axp* must have a ancestor - descendant structural relationship. Processing is the same for the VCAS and SCAS tasks.

The pattern is processed as follows:

- 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 $\sqsupset_p$, $\not\sqsupset_p$, and ▲ operators on the logical level) using a combination of functions explained in the previous subsection;
- 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). The experiments in Section 5 apply varying aggregation functions (maximum, average, and weighted normalized sum) to compute the final score for the *xp* node in question. This step is the physical equivalent of the logical ► operator.

4.3.2 Pattern 2

VCAS As already stated in Section 2 pattern 2 is divided into two subpatterns. Both subpatterns are processed as pattern 1. The two resulting node-sets need to be combined for a final ranking. We have chosen two implementations for the combination function represented with the ▲ operator on the logical level: (1) taking the **minimum** of the (non-zero) scores, or (2) taking the **product** of the scores for corresponding regions. Similarly, for the ▼ operator we used (1) the **maximum** of scores, or (2) the **average** of scores for corresponding regions.

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 the implementation of abstract operators, ▲ (minimum or product) and ▼ (maximum or average), define together the difference between strict and vague scenarios.

4.3.3 Pattern 3

VCAS Pattern 3 has been transformed at the conceptual level into an instance of pattern 2. Therefore, it is processed like pattern 2, with the only difference that the target element lies deeper in the tree.

SCAS For the strict scenario however, pattern 3 is divided into a number of subpatterns that have the form of pattern 1 or pattern 2. As explained in Section 2 these subpatterns produce intermediate result node-sets that have to be structurally correlated to each other (including the score aggregation). We can choose to perform a top-down correlation sequence, or a bottom-up correlation sequence consisting of containment joins on these result node-set. In the current implementation, the patterns are always processed top-down. Yet, the choice between a top-down or bottom-up sequence is really an optimization decision, that should be made by the retrieval system at runtime. For example, if a collection contains many paragraph elements, not contained within article elements, the system should decide to limit the amount of unnecessary executed *about* predicates by choosing a top-down approach.

Table 6: CO experimentation runs; note that we used a length of 2516 as preferred component length for the $R_{comp-logn}$ run, and used a fixed value of 0.15 for the model smoothing parameter λ.

Run	Scoring	Retr. Nodeset	MAP
R^{CO}_{art}	RSV_1	$\sigma_{n='article'}(\mathcal{N})$	0.0392
R^{CO}_{comp}	RSV_1	$\mathcal{N}$	0.0387
$R^{CO}_{comp-logn}$	RSV_2	$\mathcal{N}$	0.0374

Again, we experiment with two choices for specializing logical operator $f_{blacktriangleleft}$: (1) use the score of the target element as the result score, and (2) propagate the score of the filter element by multiplication with the score of the target element.

5 Experiments

This section describes the experiments performed using the TIJAH XML-IR system and analyzes the experimental results. The experiments are performed on INEX collection (version 1.4), using the topic set of INEX 2003 (relevance assessments version 2.4). To find the best retrieval model and to test the effectiveness of different variants of our system, we performed a number of runs for both the content-only (CO) and the content-and-structure (CAS) topics. Note that for the CAS topics we only include the experiments done with the *strict* variant (SCAS). Relevance judgements and measures for the *vague* variant (VCAS) are still under development, so we cannot report upon the effectiveness of our system for this task.

5.1 CO Task Experiments

For the CO task, we designed three original experimentation runs, using the two scoring functions RSV_1 and RSV_2 described in Section 2.1. The first run (R_{art}) represents the baseline run of fixed 'flat-document' retrieval, where the retrieval unit was limited to the article level. The second run regarded all subtrees in the collection as separate documents (R_{comp}). The third run applies the log-normal distribution to model the specificity dimension ($R_{comp-logn}$). 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 6 summarizes experimental results, where MAP stands for *mean average precision*. The official CO-runs use the keywords present in the keyword-element of each topic. Before executing each topic, query stop words were removed using the SMART 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 terms shorter than two characters or longer than 25 characters.

Table 7 presents the results of several additional CO runs.[3] 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 create CO-runs where we replaced the log-normal element length prior (*logn* runs) with a standard element length prior (*ls* runs). Finally, we experimented with the article weighting (*aw* runs). From the average precision values in Table 7, the following observations are clear:

- large elements should not be discounted (under the current metrics of evaluation; difference between *logn* and *ls* runs);

[3]The differences between the R_{comp} and $R_{comp-logn}$ MAP scores in Tables 6 and 7 originate from the (different) ordering of elements with equal score.

Table 7: 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. We used the set $\mathcal{N}$ as the retrieval nodeset for all runs. For evaluation, the strict evaluation measure (for 2002) was used.

Run	**Scoring**	**T**	**TD**	**TK**
R^{CO}_{comp}	RSV_1	0.0341	0.0383	0.0447
$R^{CO}_{comp-logn}$	RSV_2	0.0351	0.0390	0.045
$R^{CO}_{comp-ls}$	RSV_3	0.0652	0.0766	0.0740
$R^{CO}_{comp-logn-aw}$	RSV_6	0.0697	0.0863	0.0905
$R^{CO}_{comp-ls-aw}$	RSV_7	0.1043	0.1224	0.1205

- combining element scores with their surrounding context 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 *T* and *TD*).

We hypothesize that the difference in effectiveness observed between the *logn* and *ls* runs, contradicting our intuition, should be sought in the current evaluation metrics, which reward exhaustivity over specificity, and also do not handle the problem of the over-populated recall base; refer to [16] for more details. Another explanation could be that the log-normal's mean value of 2516 words as desired component size is not appropriate for the 2003 relevance assessments.

5.2 SCAS Task Experiments

The main goal of the experiments performed for the *strict* CAS retrieval task was to evaluate the *pattern approach* described in this paper and to study the performance of different implementations of the complex functions ($f_{\sqsupset}$, $f_{\not\sqsupset}$, $f_{\blacktriangleright}$, and $f_{\blacktriangleleft}$) and the abstract operators ($\otimes$ and $\oplus$).

For each topic, all terms occurring in the separate *about* function calls present in the topic title component were first collected in a single term set together with all the keywords. Then, the *about* function was executed with this term set on the set of all component in $\mathcal{N}$. Processing concludes with resolving the structural constraints expressed in the query. For scoring the components, the scoring function RSV_1 was used with the λ parameter set to 0.15.

One of the advantages of the *pattern approach* taken for the CAS queries is that we can easily modify the implementation of the algebra operators that define the combination of different subqueries within a query. Therefore, we decided to study the following combination options:

- the aggregation mechanism used to score a target element containing multiple ranked subelements in pattern 1 (operator ▶);
- the processing of AND and OR operators in pattern 2 (operators ▲ and ▼);
- the combination of the different sub-queries for pattern 3 (operator ◀).

The first run, R^{SCAS}_{orig}, uses the *minimum* and the *maximum* to process the AND (▲) and OR (▼) operators respectively. The **average** mechanism is used to give a score to the target element when it has multiple scored descendants. The first *about* clause of Pattern 3 is treated as a filter, without propagating the scores. The second run uses the probabilistically more intuitive **product** and **avg** for AND and OR operators. The comparison of results obtained from these two runs is depicted in Table 8.

Table 9 summarizes the results obtained with various scenarios for handling multiple ranked descendants within the target element. To normalize the sum of multiple ranked descendants we

used a *weighted normalized sum* as defined at the end of Section 4.2. This weighting is supposed to punish both the short descendant elements and the long target elements. It is interesting to note that, whatever the processing of the AND and OR operators, the run using 'max' outperforms the 'avg'. Apparently, high scores from the language model indicate relevance more reliably than intermediate scores. An alternative (related) explanation is that users prefer (according to the evaluation metrics) a single, highly scored element over multiple less highly scored elements. Observe also that the *weighted normalized sum* is not working as well as we expected. Seemingly, the users do not agree in our way of penalizing document sizes. The users would prefer again documents that contain a highly score element even if the rest of the document is long and not so relevant, and they would not mind if the element containing the relevant information is very small as far as it is highly scored. On the other hand, we observed in additional runs (not in the table), that the *non-normalized sum* (not allowed in our probabilistic setting) gives the best results (MAP 0.3069). Further research will include to map the modelling of this behaviour into our model.

Finally, to study the combination of the different sub-queries for Pattern 3, we did a new run where the first *about* clauses are not just treated as a filter but their scores are propagated to be multiplied with the target element scores. We can see in Table 10 that to use the filter elements scores helps indeed to improve the performance of the system.

5.3 Lambda Estimation

The following experiments investigate the influence of smoothing parameter λ on the effectiveness results. Note that this estimation has not been performed to validate or invalidate a hypothesis; we just investigate *a posteriori* how sensitive the system results are for the right setting, given the collection, topicset and evaluation metric used in this study.

We performed two sets of runs. The first set consists of article retrieval with scoring functions RSV_1, RSV_3, RSV_4, and RSV_5. The second set of runs consists of component retrieval using the same four scoring functions. For each of the sets of runs, the λ parameter has been varied between the values 0 and 1, and the MAP values for each value of λ are computed. The value for λ has been incremented by 0.05 in each step. Evaluation of the runs was done with the evaluation programs offered by the INEX initiative. The results of both sets of runs are shown in Figures 3 and 4.

The MAP values for article retrieval in Figure 3 show behavior comparable with previously reported behavior on the Cranfield and TREC collections [13]. Also, as in the case of the Cranfield and TREC collections, the length prior modified runs perform better than the non-length prior modified ones (although the differences are not as large as those reported for the Cranfield and TREC collections). The performance seems rather stable across a large interval of values for the smoothing parameter. This phenomenon seems to be collection-independent, as the same was reported for the Cranfield and TREC collections.

The MAP values for component retrieval, shown in Figure 4, exhibit more interesting behavior. Of all runs, the run using document frequencies and a document component length prior performs best, achieving a MAP of 0.09768 for $\lambda = 0.60$. In the case of component retrieval, the length-modified runs score much better than the non-length prior modified runs. As with article retrieval, we also see an interval where performance does not change radically.

Table 8: SCAS experimentation runs. AND and OR processing: min-max vs. product-avg.

Run	**▲/▼**	**►**	**MAP**
R^{SCAS}_{orig}	min/max	avg	0.2681
$R^{SCAS}_{prod-avg}$	prod/avg	avg	0.2908

Table 9: SCAS experimentation runs. Multiple descendants processing.

Run	**▲/▼**	**►**	**MAP**
R^{SCAS}_{mm-avg}	min/max	avg	0.2681
R^{SCAS}_{mm-max}	min/max	max	0.2721
R^{SCAS}_{pa-avg}	prod/avg	avg	0.2908
R^{SCAS}_{pa-max}	prod/avg	max	0.2998
R^{SCAS}_{pa-ws}	prod/avg	weight norm sum	0.2854

Table 10: SCAS experimentation runs. Filter scores propagation in Pattern 3.

Run	**▲/▼**	**►**	**◄**	**MAP**
$R^{SCAS}_{pa-max-np}$	prod/avg	max	no propagation	0.2998
$R^{SCAS}_{pa-max-p}$	prod/avg	max	propagation	0.3143

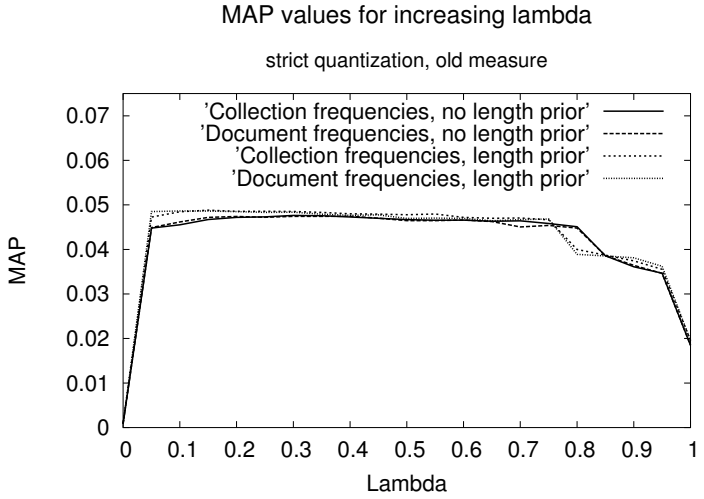

Figure 3: Lambda estimation results for article retrieval

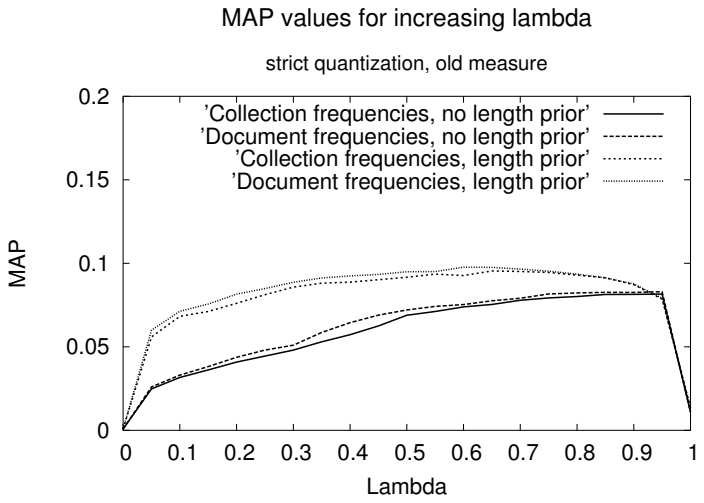

Figure 4: Lambda estimation results for component retrieval

We seek an explanation for the difference in performance between runs without length priors and those with a length prior in the fact that the language model without length prior will overestimate the relevance of (very) short document components that contain query terms, therefore outranking the larger (but usually preferable) components. Again, the influence of the artificially enlarged recall base could provide an additional reason.

5.4 Efficiency

Besides measuring the effectiveness of our retrieval system, we also measured the efficiency of indexing and querying the collection. Table 11 shows the average topic execution times of all (CO) 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 [22]. The time needed for the *logn* and *ls* 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 logarithms. Also, the *aw* runs take more execution time as a result of the extra containment joins needed to resolve the specified structural constraints. Each of the SCAS runs discussed in the previous subsection took approximately 50 seconds (wallclocktime) to execute.

Table 11: 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 official INEX 2003 runs are displayed in the first three rows (in boldface). The other timings are those of the additional, unofficial runs.

Run	K	TD	TK
R^{CO}_{art}	**6.75**	-	-
R^{CO}_{comp}	**44.08**	68.19	53.22
$R^{CO}_{comp-logn}$	**45.13**	69.58	54.47
$R^{CO}_{comp-ls}$	45.25	69.69	54.47
$R^{CO}_{comp-logn-aw}$	47.16	72.22	56.80
$R^{CO}_{comp-ls-aw}$	47.25	74.44	57

The time needed for indexing could be further reduced. 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.

5.5 Discussion

The main goal of the experiments has been to test the 'proof of concept' of our retrieval system, and especially the use of the database approach; using a layered design in a retrieval system. The main advantages of such a layered approach mentioned in Section 1 are *data abstraction* and *content abstraction*. We consider our proof of concept partially successful. Defining and implementing new experimental scenarios, including the context weighted version of the retrieval model, has proved straightforward.

However, after studying this paper, the skeptical reader of this paper might wonder about the goal and benefits of the logical layer, since the mappings from conceptual via logical to physical layers may seem rather one-on-one. And, the data models used on the logical and physical levels do not differ that much, so this mapping might as well be performed from conceptual to physical layer without intermediate step.

For our INEX 2002 and 2003 experiments, this has been somewhat true; but, of course, these experiments span only the retrieval of structured XML documents, from a rather homogeneous collection. The notion of a structured document also allows for other types of retrieval to be performed with the TIJAH system. We think that handling heterogeneous collections, even of different media types, would emphasize more the role of the logical layer in query processing. For example, the generally accepted structure of a video divides a video stream into scenes, which are composed of shots, and the shots themselves are composed of frames. Such video documents possess document structure (albeit simple), and, user annotations (MPEG-7) would introduce an even more meaningful structure for the set of videos.

6 Conclusions and Future Work

Our experience with the TIJAH system at the INEX evaluations can be considered a successful exercise in applying current and state of the art information retrieval technology to a database consisting of structured documents. We described our system architecture, aimed at simplifying the implementation of advanced retrieval models for the combination of structure and content in XML documents.

We expect to take further advantage of the obtained flexibility in our future research. For,

the current approach to retrieval has only used a limited proportion of the wealth of structural information present in XML documents. Also, 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. Longer term research plans include handling a wider variety of structured documents. More specifically, we will deploy the TIJAH system in our ongoing video retrieval research.

7 Acknowledgements

The authors are grateful to the Netherlands Organization for Scientific Research (NWO), for funding the research described in this paper (grant number 612.061.210).

References

[1] R. Baeza-Yates and G. Navarro. Integrating Contents and Structure in Text Retrieval. In *ACM SIGMOD Record*, volume 25, pages 67–79, 1996.

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

[3] 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.

[4] 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.

[5] 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.

[6] J. Cowan and R. Tobin. XML Information Set (Second Edition). Technical report, W3C, 2004.

[7] 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, 2003.

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

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

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

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

[12] 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, 2003.

[13] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Twente, The Netherlands, 2001.

[14] 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*, ERCIM Publications, 2002.

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

[16] G. Kazai, M. Lalmas, and A.P. de Vries. The overlap problem in content-oriented xml retrieval evaluation. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004. To appear.

[17] 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 Publications, 2003.

[18] K. Masuda, T. Ninomiya, Y. Miyao, T. Ohta, and J. Tsujii. A Robust Retrieval Engine for Proximal and Structural Search. In *Proceedings of HLT-NAACL 2003 Short papers*, pages 58–60, 2003.

[19] Katsuya Masuda. A Ranking Model of Proximal and Structural Text Retrieval Based on Region Algebra. In *Proceedings of the ACL-2003 Student Research Workshop*, pages 50–57, 2003.

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

[21] A. Salminen and F.W. Tompa. PAT Expressions: An Algebra for Text Search. In *Proceedings of the 2nd International Conference in Computational Lexicography, COMPLEX'92*, pages 309–332, 1992.

[22] 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.

[23] 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 Publications, 2004.

[24] A. Trotman and R. A. O'Keefe. The Simplest Query Language That Could Possibly Work. 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 Publications, 2004.

[25] 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.