# Combining indexing schemes to accelerate querying XML on content and structure

Georgina Ramírez
CWI
P.O. Box 94079
1090GB Amsterdam
The Netherlands
georgina.ramirez@cwi.nl

Arjen P. de Vries
CWI
P.O. Box 94079
1090GB Amsterdam
The Netherlands
arjen@acm.org

## ABSTRACT

This paper presents the advantages of combining multiple document representation schemes for query processing of XML queries on content and structure. We show how extending the *Text Region* approach [2] with the main features of the *Binary Relation* approach developed in [8] leads to a considerable speed-up in the processing of the XPath location steps. We detail how, by using the combined scheme, we reduce the number of *structural joins* used to process the XPath steps, while simultaneously limiting the amount of *memory usage*. We discuss optimisation strategies enabled by the new 'combined representation scheme'. Experiments comparing the efficiency of alternative query processing strategies on a subset of the queries used at INEX 2003 (the Initiative for the Evaluation of XML Retrieval [4]) demonstrate a favourable performance for the combined indexing scheme.

## 1. INTRODUCTION

Different approaches have been developed for the storage, processing and retrieval of XML documents. This paper investigates the suitability of such approaches in the design of document-centric XML retrieval systems, where queries specify constraints on both the XML structure (a data retrieval problem) and the document content (an information retrieval problem). Examples of such queries have been defined for the 'content-and-structure' task (CAS) at the Initiative for the Evaluation of XML Retrieval (INEX) [4]. CAS queries define the structural constraints in the queries using (a subset of) XPath, and the content part of the query with a special-purpose about-clause, which expresses that the elements should be ranked by the expected relevance of their textual content to the user's information need. The experiments on this paper are centered on the *Strict* CAS task, meaning that the structural constraints of the query must be exactly satisfied by the retrieved components. INEX's query language is defined precisely in [6].

The *Text Region* approach for processing such content-and-structure queries [2] views the XML document as an ordered sequence of tokens (including the document text itself), representing the documents using a preorder-postorder tree encoding scheme, as an extension of [9, 5]. The main advantages of this approach are the efficiency of processing descendant axis steps (using structural joins on the representation of the XML tree in the 'pre-post plane'), and the low cost of reconstructing the XML document when returning an answer to the user.

The *Binary Relation* approach described in [8] views the XML document as a rooted (syntax) tree and stores the document in binary associations representing the edges of the syntax tree (i.e., parent-child and node-attribute relations). In addition, it stores a *path summary* table containing all unique paths in the tree, used as an index to restrict access to only those tables that could contain answers. The main advantages of this approach are the efficent processing of consecutive child axis steps, and the use of smaller tables to do the processing. The main drawbacks are the inefficient way of processing descendant steps, the high cost of reconstructing XML answers, and, the impossibility to directly access the text nodes when, for instance, term frequency statistics should be determined for ranked retrieval.

This paper shows how a redundant representation that combines properties of both schemes reduces the number of structural joins needed in the query plan and limits the memory used in the processing of the XPath steps, leading to an improved runtime performance. Also, we point out how several optimization techniques can be applied to further improve the performance on this combined scheme.

The paper is organized as follows. Section 2 details the document model and indexing schemes of the two approaches used in this paper. Section 3 proposes a new document representation by combining these schemes. Section 4 presents a procedure for processing queries on the new indexing scheme, and several optimization aspects are discussed. Section 5 presents experiments on the INEX collection and shows the performance obtained. The paper ends with conclusions and ideas for future work.

TDM'04, the first Twente Data Management Workshop on XML Databases and Information Retrieval, Enschede, The Netherlands

## 2. DOCUMENT INDEXING SCHEMES

When indexing XML documents using a relational database management system, we should take the following two aspects into account. First, from an *effectiveness* point of view, one should decide which type of information to store (data and metadata), depending upon the kind of operations to be performed on the data; e.g., order of the elements, level of the nodes in the tree, parent/child relationship, etc. Second, from an *efficiency* point of view, one has to choose how to store the information, in order to execute these operations in the fastest possible way; e.g., the level of fragmentation, the definition of indices, replication.

This paper focuses on decisions to be taken in the efficiency viewpoint, under the assumption that the choices to be made from the effectiveness viewpoint have been determined fully by the retrieval model. The retrieval model influences our paper in so-far, that satisfying queries with about-clauses may require term frequency statistics in context of any node in the XML tree (usually identified by tagname). Given this requirement, we are interested in finding a balance between the system resources consumed by the indexing scheme and the response times for user requests. More specifically, the paper investigates design choices that replicate data to improve the runtime efficiency of the resulting XML retrieval system.

### 2.1 Text Region approach

The *Text Region* approach used in this paper is the one described in [2]. It extends the well-known preorder-postorder tree encoding scheme of [9, 5] for the ranked retrieval on textual content (the about-clauses). The XML document is viewed as a sequence of tokens. Where tokens are opening and closing tags as well as the pre-processed text content. The pre-processing consists of word form normalizations, stemming, and stopword removal. Each component is a text region, i.e., a contiguous subsequence of the document. XML text regions are represented by triplets $\{ t_i, s_i, e_i \}$, where:

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

The storage scheme (the physical representation) consists of two large tables:

- The *node-index* $\mathcal{N}$ stores the set of all XML region tuples;
- the *word-index* $\mathcal{W}$ stores all the index terms.

Notice that the index terms are stored in a different table, even though they are also considered as text regions. The reason is that the regions corresponding to the index terms consist of a single token each, so the start position always equals the end position. By separating the word-index from the node-index, we may store these word positions only once, reducing the memory requirements of the text region representation scheme considerably.

To illustrate the storage scheme and the resulting query plans for processing queries on content and structure, consider the example document given in Figure 1. Figure 2 shows the document as a sequence of tokens with start and end positions assigned. The document information is then stored in the node-index and word-index as shown in Figure 3. XPath location steps are executed through structural joins over the node-index.

**Figure 1: XML document example**

```
<bibliography>
  <article>
    <author>Ben Bit</author>
    <title>How to Hack</title>
  </article>
  <article>
    <author>Ben Byte</author>
    <author>Ken Key</author>
    <title>Hacking and RSI</title>
  </article>
</bibliography>
```

**Figure 2: The XML example document as a sequence of tokens. Start and end positions assignment. Note that the document from Figure 1 has already been pre-processed, stop-words have been removed and terms have been stemmed**

$< bibliography >^0 < article >^1 < author >^2 ben^3\ bit^4$
$< /author >^5 < title >^6 how^7\ hack^8 < /title >^9 < /article >^{10}$
$< article >^{11} < author >^{12} ben^{13}\ byte^{14} < /author >^{15}$
$< author >^{16} ken^{17}\ key^{18} < /author >^{19} < title >^{20}$
$hacking^{21}\ rsi^{22} < /title >^{23} < /article >^{24} < /bibliography >^{25}$

**Figure 3: *Text Region* storage scheme for the example document**

| $\mathcal{N}$ |
|---|
| $< bibliography, 0, 25 >$ |
| $< article, 1, 10 >$ |
| $< author, 2, 5 >$ |
| $< title, 6, 9 >$ |
| $< article, 11, 24 >$ |
| $< author, 12, 15 >$ |
| $< author, 16, 19 >$ |
| $< title, 20, 23 >$ |

| $\mathcal{W}$ |
|---|
| $<$ *"ben"*$, 3 >$ |
| $<$ *"bit"*$, 4 >$ |
| $<$ *"how"*$, 7 >$ |
| $<$ *"hack"*$, 8 >$ |
| $<$ *"ben"*$, 13 >$ |
| $<$ *"byte"*$, 14 >$ |
| $<$ *"ken"*$, 17 >$ |
| $<$ *"key"*$, 18 >$ |
| $<$ *"hacking", 21* $>$ |
| $<$ *"rsi"*$, 22 >$ |

Consider for example the following query, requesting a ranked list of articles written by an author named "Ben" where the title is about hacking:

- /bibliography/article[contains(.//author, "Ben") AND about(.//title, "Hack")]

The physical query plan for this example query is shown in Figure 4. XPath name tests correspond to selections on the node-index, and location steps translate into structural

joins. For example, $R5$ results from a structural join performed to correlate all the pairs article-author, from the article nodes and the author nodes previously selected, that satisfy the descendant condition. In other words, this table contains all the pairs of starting positions of articles and authors where author is descendant of article. The processing of the about-clause is not detailed in this paper (refer to [2] for more information). The implementation of the AND operator (last line) combines the ranked results for each of the articles.

**Figure 4: *Text Region* approach physical query plan for the example query. The relations $Ri$ represent the intermediate results**

$R1 := select(NodeIndex, name = \text{'bibliography'})$
$R2 := select(NodeIndex, name = \text{'article'})$
$R3 := R1 \bowtie_{\supset} R2$
$R4 := select(NodeIndex, name = \text{'author'})$
$R5 := R3 \bowtie_{\supset} R4$
$R6 := select(NodeIndex, name = \text{'title'})$
$R7 := R3 \bowtie_{\supset} R6$
$R8 := contains(R5, \text{"ben"})$
$R9 := about(R7, \text{"hack"})$
$R10 := R9 \cap R8$

## 2.2 Binary Relation approach

The *Binary Relation* approach views the XML document as a rooted syntax tree. The information stored represents the edges of the tree as parent-child and node-attribute relations. Schmidt defines the storage scheme as follows [8]: Given an XML document $d$, the 'Monet transform' (the mapping of the document into a tabular representation in the MonetDB) is a quadruple $M_t(d) = (r, \mathcal{E}, \mathcal{A}, \mathcal{T})$, where:

- $\mathcal{E}$ is the set of binary relations that contain *all* associations between nodes (parent-child relations);
- $\mathcal{A}$ is the set of binary relations that contain *all* associations between nodes and their attribute values, including character data (node-attribute relations);
- $\mathcal{T}$ is the set of binary relations that contain *all* associations between nodes and their position with respect to their siblings (node-postion relations);
- $r$ is the root of the document.

To improve the efficiency during query processing, this approach creates a so-called *path-summary* that contains the table name corresponding to each unique path in the document. The binary relations for each such path are stored separately. So, the path-summary provides direct access to the corresponding edges in the document syntax tree that share the same path. Notice that the binary relations can be viewed as the 'join indices' used in relational query processing (introduced by Valduriez in [10]).

As an example, consider again the example document (Figure 1). Figure 5 shows the tree view of the XML example document and the oid assignments. The *Binary Relation* approach stores the information of $\mathcal{E}$ and the *path-summary* as depicted in Figure 6. Query processing first retrieves the

**Figure 7: *Binary Relation* approach physical query plan for the example query. The relations $Ri$ represent the intermediate results and $Ti$ represent the names of the binary relations shown in Figure ??**

$path := /bibliography/article$
$R1 := \emptyset$
*foreach* $Ti \in MatchingPaths(path\ .\ //author)$ *do*
  $R2 := select(PathSummary, name = Ti)$
  $R2 := contains(R2, \text{"Ben"})$
  $R1 := R1 \cup JoinUp(R2, Ti, \text{"}article''\text{)}$
$R3 := \emptyset$
*foreach* $Ti \in MatchingPaths(path\ .\ //title)$ *do*
  $R4 := select(PathSummary, name = Ti)$
  $R4 := about(R4, \text{"Hack"})$
  $R3 := R3 \cup JoinUp(R4, Ti, \text{"}article''\text{)}$
$R5 := R1 \cap R3$

filter elements from the path-summary, selects the nodes according to the predicates and then proceeds to *'join up'* against the other relations until it reaches the target element. The physical query plan for the example query of Subsection 2.1 is given in Figure 7. Here, $MatchingPaths$ is a shorthand notation for identifying the unique paths in the document (i.e., table names in the path-summary) that satisfy the XPath expression. The $JoinUp$ procedure starts with a join between two relations, e.g., $R2 \bowtie T3$ in the example, and proceeds to compute the transitive closure over the parent relationship; i.e., the first $JoinUp$ corresponds to expression $R2 \bowtie T3 \bowtie T2$.

In the example, only one path matches each of the descendant steps in the query example. Because the query is selective, query processing accesses only a small proportion of the collection, and is very efficient. Indeed, Florescu and Kossmann found a similar mapping to be most efficient for XML query processing in their experiments [3]. In cases where multiple paths match however, each of these must be 'joined up' individually, and the results unioned. This procedure may become expensive on large collections with a 'lenient' DTD, when many paths need to be followed to determine the query results.

# 3. THE COMBINED INDEXING SCHEME

This paper proposes to combine the ideas from both approaches discussed before. This *combined scheme* views the document as both a syntax tree for which we store the edge information, and, a set of regions, for which we store the triplets defined in Subsection 2.1. While the *type* of information we store is just a direct combination of both approaches, the *way* the information is stored in the resulting data representation differs from just storing all the defined tables.

We define the combined indexing scheme formally as follows: Given an XML document instance, the indexing scheme consists of a 5-tuple: $Mt'(d) = (\mathcal{N}, \mathcal{W}, \mathcal{E}, \mathcal{A}, \mathcal{P})$, where

- $\mathcal{N}$ is the set of all XML region tuples;
- $\mathcal{W}$ is the set of all index terms in the document;

**Figure 5: *Binary Relation* tree view of the example document and oid assignments**

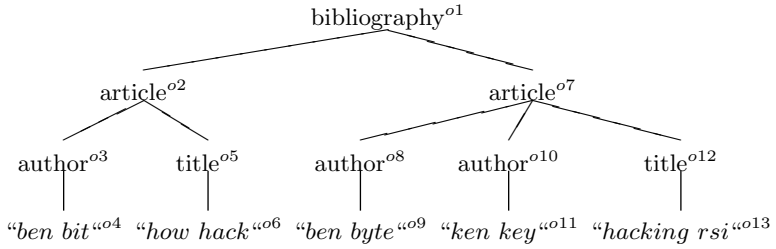


**Figure 6: *Binary Relation* storage scheme for the example document**

| Path Summary |
|---|
| $< T1$, */bibliography/article* $>$, |
| $< T2$, */bibliography/article/author* $>$, |
| $< T3$, */bibliography/article/author/cdata* $>$, |
| $< T4$, */bibliography/article/author/cdata/string* $>$, |
| $< T5$, */bibliography/article/title* $>$, |
| $< T6$, */bibliography/article/title/cdata* $>$, |
| $< T7$, */bibliography/article/title/cdata/string* $>$ |

| Relation | Content |
|---|---|
| $T1$ | $\{< o_1, o_2 >, < o_1, o_7 >\}$ |
| $T2$ | $\{< o_2, o_3 >, < o_7, o_8 >, < o_7, o_{10} >\}$ |
| $T3$ | $\{< o_3, o_4 >, < o_8, o_9 >, < o_{10}, o_{11} >\}$ |
| $T4$ | $\{< o_4$, *"ben bit"* $>, < o_9$, *"ben byte"* $>$, $< o_{11}$, *"ken key"* $>\}$ |
| $T5$ | $\{< o_2, o_5 >, < o_7, o_{12} >\}$ |
| $T6$ | $\{< o_5, o_6 >, < o_{12}, o_{13} >\}$ |
| $T7$ | $\{< o_6$, *"how hack"* $>, < o_{13}$, *"hacking rsi"* $>\}$ |

- $\mathcal{E}$ is the set of binary relations that contain *all* associations between nodes;
- $\mathcal{A}$ is the set of binary relations that contain *all* associations between nodes and their attribute values;
- $\mathcal{P}$ is the set of all unique paths linked to the binary relations.

Notice that the set of binary relations $\mathcal{E}$ differs from the one defined in the *Binary Relation* approach. First, we do not store the *node-text* relations. Word-index $\mathcal{W}$ is needed for computing the ranking (specified by the probabilistic retrieval model) and storing this data redundantly would incur a huge cost without benefits for query processing. Similarly, we can also remove the set of relations $\mathcal{T}$, as this information can be extracted directly from the start and end positions from the regions in the node-index $\mathcal{N}$. We have not stored the root element at this time, because we indexed one collection only; so, it is already stored in the node-index table with a start position equal to zero.

The proposed combined indexing scheme trades storage cost for increased query processing performance. This seems a reasonable trade-off, as disk capacity is not a main problem nowadays, especially when compared to memory size. We consciously duplicate information in order to reduce the amount of irrelevant data accessed during query processing, by limiting the size of the tables that are loaded to memory. As an additional advantage of replicated information, we may speed up the processing of different types of user requests by taking optimization decisions at run-time, dependent on the properties of the specific user request being executed.

# 4. QUERY PROCESSING

This Section presents a procedure for processing content-and-structure queries on the proposed *combined indexing scheme*, and discusses its advantages regarding memory usage and the number of structural join operations needed.

The retrieval model considered is described in detail in [7], which assumes the text region based approach for document representation. Queries are classified into three different patterns and split into one or more subqueries. After processing each of the subqueries, their scores are combined for a final ranking. Table 1 reviews the patterns and summarises the processing involved. We refer the reader to [7] for more detailed information on the *Patterns* approach.

Extending this pattern approach for the combined indexing scheme, processing content-and-structure queries follows a 4-step procedure:

1. Split the query into different subqueries following the *Pattern* approach detailed in [7] and summarized in Figure 1.
2. Split each of the XPath location steps into *canonical paths*.
3. Process the subpaths and subqueries according to the algorithm described in Figure 8.
4. Combine the results of the subqueries for a final score according to the *Pattern* approach.

This paper focuses on steps 2 and 3 in this process. We show how we can make use of the *combined scheme* to accelerate the processing of the XPath steps (xp and axp). Step 2 follows a straightforward algorithm that splits the location paths that occur in the query into series of *canonical location paths*. A *canonical path* is a path that either contains child axis steps only, or, when starting with a descendant axis step, is followed by child axis steps only.

**Table 1: Pattern approach. Splitting queries and combining their results. Note that *xp*, *xp2*, *axp*, *axp1* and *axp2* are location steps, and *'t/p'* denotes any set of terms or phrases to search for. Patterns 2 and 3 are split into two or more instances of pattern 1 and queried with *all* the keywords (t12/p12). After the processing of the pattern 1 instances, their results (Pn) are combined as indicated in the last column. $P()$ is the probability/score of a node and $size()$ is the text length of the region determined by a node**

| Pattern | Definition | Subqueries | Combining results |
|---|---|---|---|
| $P_1$ | xp[about(axp, 't/p')] | xp[about(axp, 't/p')] | if multiple axp in xp node then $P(xp_{node}) = \frac{\sum_i size(axp_i) * P(axp_i)}{\sum_i (size(axp_i))}$ |
| $P_2$ | xp[about(axp1, 't1/p1') AND/OR about(axp2, 't2/p2')] | xp[about(axp1, 't12/p12')] xp[about(axp2, 't12/p12')] | AND: product Pn(1) * Pn(2) OR: average (Pn(1) + Pn(2)) / 2 |
| $P_3$ | xp[about(axp1, 't1/p1')]//xp2[about(axp2, 't2/p2')] | xp[about(axp1, 't12/p12')] xp//xp2[about(axp2, 't12/p12')] | Propagation scores: Pn(xp) * Pn(xp2) |

As an example, consider the following path used to query the INEX data:

/books/journal//article/bdy//sec/p/it

This path is split into the following three canonical paths:

- /books/journal
- //article/bdy
- //sec/p/it

Because the document representation stores its data redundantly, the resulting subpaths can be processed in a number of ways. The goal is to make the best use of the multiple representations available. Figure 8 presents our algorithm for generating an efficient query plan for the processing of the XPath location steps.

**Figure 8: Algorithm to process the XPath location steps**

```
NP := 3; NC := 3;
for each of the subpaths p do
  if StartsChild(p) then ConcatenateToPrevious(p);
  else
    if NumberPathsMatching(p) <= NP then
      FlattenStep(p);
      ConcatenateToPreviousPath();
    else
      if NumberConsecChildren(p) >= NC then
        SelectEndingPaths(p);
        UseStructuralJoinToJoinFigures(p);
      else ContinueWithStructuralJoin();
      endif;
    endif;
  endif;
endfor;
```

The general idea underlying the algorithm is twofold. On the one hand, it decides (based on constant NP) whether to execute the descendant axis step with a structural join, or, alternatively, to *flatten* it. Flattening the descendant step means to find all matching paths in the path-summary table, and unioning the obtained relations. On the other hand, it decides (based on constant NC[1]) how to proceed after a structural join is used for processing a descendant step. A canonical path starting with a descendant step can be followed by one or more child axis steps. The 'standard' solution is to apply an additional structural join for each remaining child step. The alternative solution retrieves from the path-summary all paths that end in the same sequence of child steps as the canonical path. The resulting tables are unioned, and then a single structural join correlates this unioned table with the intermediate results of the previously processed canonical path.

Consider for example the XPath expression introduced before with its three canonical subpaths, the result from step 2 in the query processing procedure. The algorithm then proceeds as follows. The path-summary table gives only one path ending with 'article'. So, the descendant step of the second canonical path is flattened and concatenated to the first one, obtaining the concatenated XPath expression /books/journal/article/bdy.

When processing the third canonical path (//sec/p/it), the algorithm chooses a structural join to identify the 'sec' elements, as more than NP=3 unique matching paths exist in the path-summary. Because the number of remaining child axis steps (two) is smaller than NC=3, we process these steps using structural joins. So, the final query plan selects from the path-summary the relation that corresponds to the concatenated subpaths (/books/journal/article/bdy), performs the structural join to identify the contained 'sec' elements, and then executes two more structural joins to process child axis steps '/p' and '/it'.

Finally, for comparison to the query plans of the other two document indexing approaches, let us return to the example introduced in Subsection 2.1. The physical query plan using the *combined scheme* is shown in Figure 9.

Generation of the query plan using our algorithm allows for a variety of further optimisation techniques. First, when processing a descendants query that leads to many differ-

[1]The best values for constants NP and NC are still an open issue for further research. We decided to first set them to three because, given the INEX topics in section 5 and the algorithm in Figure 8, we would be able to see some differences in the processing. Note that, for this set of queries, to set NC to any number different than 1 would produce the same effect.

**Figure 9: Physical query plan for the example query using the *combined scheme*. The relations $Ri$ represent the intermediate results**

$path := /bibliography/article$
$R1 := select(PathSummary, name = path\ .\ /author)$
$R2 := contains(R1, \text{“Ben”})$
$R3 := select(PathSummary, name = path\ .\ /title)$
$R4 := about(R3, \text{“Hack”})$
$R5 := R4 \cap R2$

ent canonical paths, each involving many child axis steps, then performing a single structural join over the node-index might be a more efficient alternative. In some cases however, flattening a descendant step over the path-summary could access much less data than executing the location step using the node-index. These optimisation decisions require statistics about the data, possibly precomputed, such as the number of elements of each kind, the number of matching paths, and table sizes. The next Section investigates the impact of such optimisation opportunities experimentally. A query optimiser should also consider how the static typing information from a DTD or XML Schema may improve the generation of query plans. This topic is however postponed to future research.

# 5. EXPERIMENTS

We have investigated experimentally the run-time performance and cost of the new approach on a subset[2] of the INEX 2003 topics (Table 2). The Text Region's physical representation of the INEX collection as node-index and word-index accumulates to 877MB (this excludes a small number of auxiliary tables needed for processing the about-clauses). The combined storage scheme increases the memory usage by 76MB (i.e., the parent-child relations and the path-summary). The speed-up observed clearly outweights the relatively small additional cost for the redundancy in the storage scheme.

Three runs have been performed, characterised in Table 3, to observe the behaviour and performance of our *combined* approach in comparison to the processing on the Text Region approach. We have not performed runs on the 'pure' Binary Relation indexing scheme, because this document representation is not suitable for handling the about-clauses that rank elements by their content; the query plans resulting from the INEX topics using this mapping are very inefficient (and do not give any insights). Table 5 shows the wall-clock times for each of the queries in each of the runs.

The different runs were performed on an AMD Opteron 1.4Ghz machine with 2GB of main memory. MonetDB [1] was used as the database kernel.

Using our algorithm with the combined indexing scheme performs usually better than the equivalent query plan using the *Text Region* approach used at INEX 2003. It is interesting however that using the combined scheme does not necessarily imply a performance gain. Comparing the results between runs TR and C1 shows for example that a strategy that always flattens the first descendant step leads to performance differences between the various topics. Take for instance topics 78, 80, and 84. The 'flattening always' strategy works well for topic 80, because only a single path in the collection leads to 'article' nodes. In the case of topic 78 however, following this strategy results in a run-time efficiency that is worse than the times obtained with the Text Regions approach. Here, the number of paths in the INEX collection that lead to 'vt' elements is 11. Looking at topic 84, we can clearly see that the use of a structural join performs much better than flattening the descendant step. In this case, 637 different paths in the collection end in 'p' elements. Summarising, we conclude from these experiments that it is important to decide upon the best query processing strategy given the query at hand, providing support for following the 'database approach' using a layered design of XML retrieval systems, as proposed in e.g. [2] and [7].

To analyse in more detail the performance gain obtained for most topics, let us look closer into the execution of INEX topic 69:

/article/bdy/sec[about(.//st,'"information retrieval"')]

Processing this query with the Text Region approach first correlates the 'article' nodeset with the 'bdy' nodeset, and correlates the result with the 'sec' nodeset. Each of these structural correlations is performed by a structural join. We proceed with another structural join, to retrieve the 'st' nodes that are descendants of the nodes in the result so far. To summarize, the query plan consists of three structural joins. The physical query plan and the sizes of the corresponding intermediate result relations are detailed in Figure 10.

The combined storage scheme allows to perform some of the correlations between nodesets with 'normal' relational joins, using the parent/child relationships. Processing the same INEX topic with the previously described algorithm on the new combined scheme accesses the path summary to obtain the result table for XPath expression /article/bdy/sec. We correlate the result to the descendant 'st' nodes with a structural join. So, the query plan consists of a selection on the path-summary followed by one structural join.

Notice that, obviously, the observed run-time performance improvement is not an immediate consequence of the *number* of structural join operators in the query plan. Instead, it results from the reduced sizes of tables accessed during processing, in combination with the number of intermediate results and their sizes. The physical query plan and the sizes of the relations produced are detailed in Figure 10.

# 6. CONCLUSIONS AND FUTURE WORK

We presented the possible advantages for query processing when the XML storage scheme represents a document collection redundantly, corresponding to a trade-off often made in database management systems between storage requirements and run-time efficiency. Specifically, we augment our previously developed Text Region indexing scheme (used in our participation in INEX) with join indices on path expressions, as applied previously to XML data collections in

[2]For clarity and comparison, we selected only the topics with different structural constraints that could be "affected" by the new algorithm and therefore, could give some insights into the performance

**Table 2: Queries used on the experiments. Subset of the INEX03 topics.**

| Topic number | Query title |
|---|---|
| 61 | //article[about(., 'clustering +distributed') a nd about(.//sec, 'java')] |
| 64 | //article[about(.,'hollerith')]//sec[about(.,'DEHOMAG')] |
| 69 | /article/bdy/sec[about(.//st,"'information retrieval"')] |
| 70 | /article[about(./fm/abs, "'information retrieval" "digital libraries"')] |
| 72 | //article[about(./fm/au/aff,'United States of America')]/bdy//*[about(.,'weather forecasting systems')] |
| 73 | //article[about(.//st,'+comparison') and about (.//bib,"'machine learning"')] |
| 78 | //vt[about(.,"'Information Retrieval" student')] |
| 79 | //article[about(.,'XML') AND about(.,'database')] |
| 80 | //article/bdy/sec[about(.,"'clock synchronization" "distributed systems"')] |
| 82 | //article[about(.,'handwriting recognition') AND about(./fm/au,'kim')] |
| 83 | /article/fm/abs[about(.,"'data mining" "frequent itemset"')] |
| 84 | //p[about(.,'overview "distributed query processing" join')] |

**Table 5: Run times. Text Region approach vs. Combined scheme. Time in seconds. Average result for 10 runs, after removing the best and worst results (for each topic in each of the runs separately).**

| Run | t61 | t64 | t69 | t70 | t72 | t73 | t78 | t79 | t80 | t82 | t83 | t84 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TR | 3 | 10.88 | 13 | 9.88 | 16.5 | 11.13 | 5.5 | 6.13 | 14.63 | 17 | 10.25 | 4.5 |
| C1 | 3 | 7.88 | 5 | 6.13 | 13.38 | 11.5 | 6 | 3.75 | 3 | 6.63 | 1 | 9.5 |
| C2 | 3 | 7.75 | 4.88 | 6.38 | 13.75 | 12.13 | 6.75 | 3 | 3.38 | 3.63 | 1 | 6 |

**Figure 10: Physical query plan and tables' sizes (in #tuples/1000) using the Text Region approach (left) and the Combined scheme (right). The relations $Ri$ represent the intermediate results**

| Operation | \|Input\| | \|Result\| |
|---|---|---|
| $R1 := select(\mathcal{N}, name = \text{'article'})$ | 8240 | 12 |
| $R2 := select(\mathcal{N}, name = \text{'bdy'})$ | 8240 | 12 |
| $R3 := R1 \bowtie_{\supset} R2$ | 12, 12 | 12 |
| $R4 := select(\mathcal{N}, name = \text{'sec'})$ | 8240 | 70 |
| $R5 := R3 \bowtie_{\supset} R4$ | 12, 70 | 65 |
| $R6 := select(\mathcal{N}, name = \text{'st'})$ | 8240 | 146 |
| $R7 := R5 \bowtie_{\supset} R6$ | 65, 146 | 140 |
| $R8 := about(R7, \text{"qw"})$ | 140 | 140 |
| $R9 := avg(R8)$ | 140 | 58 |

| Operation | \|Input\| | \|Result\| |
|---|---|---|
| $R1 := select(PathSummary,$ $name = /article/bdy/sec)$ | 10 | 65 |
| $R2 := select(\mathcal{N}, name = \text{'st'})$ | 8240 | 146 |
| $R3 := R1 \bowtie_{\supset} R2$ | 65, 146 | 140 |
| $R4 := about(R3, \text{"qw"})$ | 140 | 140 |
| $R5 := avg(R4)$ | 140 | 58 |

**Table 3: Experimental runs**

| Run | Description |
|---|---|
| TR | Text Region approach. Use of Structural Joins to process XPath steps. |
| C1 | Combined Scheme. *Flattening* only the first descendant step when the path starts with a descendant axis step. |
| C2 | Combined Scheme. Using algorithm of Figure 8 with $NP = 3$ and $NC = 3$. |

**Table 4: Run times statistics. Text region approach vs. Combined scheme. Time in seconds**

| Run | Total | Average | Max | Min |
|---|---|---|---|---|
| TR | 122.4 | 10.2 | 16.5 | 4.5 |
| C1 | 76.77 | 6.4 | 13.38 | 1 |
| C2 | 71.65 | 5.97 | 13.75 | 1 |

Schmidt's PhD thesis [8].

Our experiments with INEX topics demonstrate how a minor increase in storage requirements can lead to a considerable acceleration of query processing. The number of structural joins required to produce the results of the content-and-structure queries has been reduced, and the memory requirements during query processing diminished.

The most interesting contribution of this paper is that it demonstrates how document-centric XML query processing should benefit from a layered system design similar to the architecture of relational database management systems, where the query is declarative, and the query optimiser determines the actual query execution plan. We have shown how the most appropriate query processing strategy can only be chosen at run-time, based on statistics about the data collection indexed.

Our main plans for future research in this area consist of a deeper investigation of optimisation techniques enabled by the redundant representation of XML documents. Also, no-

tice that the collection we used in this paper exhibits a fairly regular structure. While the combined scheme could be similarly efficient for selective queries in a heterogeneous collection, the path-summary would grow significantly, increasing the cost of identifying the relations to be used. Another future direction is therefore to investigate whether this cost can be reduced by maintaining only a selection of the join indices, rather than simply defining them for all possible location paths occurring in the document.

## 7. ACKNOWLEDGEMENTS

The authors would like to thank Johan List for his valuable comments and help.

## 8. REFERENCES

[1] P. A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[2] 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)*, pages 179–192. Springer-Verlag, Berlin, New York, etc., August 2003.

[3] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDBMS. *IEEE Data Engineering Bulletin*, 22:27–34, 1999.

[4] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas. INEX: INitiative for the Evaluation of XML retrieval. In *Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval*, 2002. `http://www.is.informatik.uni-duisburg.de/bib/xml/Fuhr_etal_02a.html`.

[5] T. Grust, M. van Keulen, and J. Teubner. Accelerating XPath Evaluation in any RDBMS. *ACM Transactions on Database Systems (TODS)*, 2003.

[6] INEX'03 Guidelines for Topic Development. In Norbert Fuhr, Saadia Malik, and Mounia Lalmas, editors, *INEX 2003 Workshop Proceedings*, pages 192–199, 2003. `http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf`.

[7] Johan List, Vojkan Mihajlovic, Arjen P. de Vries, Georgina Ramirez, and Djoerd Hiemstra. The TIJAH XML-IR system at INEX 2003. In Norbert Fuhr, Saadia Malik, and Mounia Lalmas, editors, *INEX 2003 Workshop Proceedings*, pages 102–109, 2003. `http://inex.is.informatik.uni-duisburg.de:2003/proceedings.pdf`.

[8] Albrecht Schmidt. *Processing XML in Database Systems*. PhD thesis, University of Amsterdam, 2002.

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

[10] Patrick Valduriez. Join indices. *ACM Transactions on Database Systems (TODS)*, 12(2):218–246, 1987.