An XML-IR-DB Sandwich: Is it Better With an Algebra in Between?

Vojkan Mihajlović Djoerd Hiemstra Henk Ernst Blok Peter M. G. Apers
CTIT, University of Twente
P.O. Box 217, 7500AE Enschede, The Netherlands
{v.mihajlovic, d.hiemstra, h.e.blok, p.m.g.apers}@utwente.nl

ABSTRACT

In this paper we address the problem of immediate translation of XPath+IR queries to relational database expressions and exert the benefits of using an intermediate algebra. Adding an intermediate algebra on the logical level of a database enables a level of abstraction from both query languages for IR in XML documents and the underlying relational storage. This paper proposes a region algebra that can be extended to support ranking operators in an elegant way while staying algebraic. Furthermore, region algebra operator properties provide a firm ground for query rewriting and optimization.

1. INTRODUCTION

Despite the numerous existing systems dealing with XML querying, the problem of expressing as well as executing Information Retrieval-like (IR-like) queries over XML databases is still an open issue [1]. An IR-like query (an example is given in Figure 2 in Section 2) does not specify hard conditions on XML elements, but queries the collection for elements 'about' a certain topic. For instance, an XML element that is relevant to a query for elements about "relational databases" might not contain the phrase "relational databases", or even both words "relational" and "databases". IR-like queries should result in a ranked list of XML elements, in decreasing order of some score value that the system assigns to each element. The score value has to reflect the probability (or degree) of relevance of the element to the IR-like query.

A promising approach to executing XPath and XQuery is the use of relational database technology [9, 17], which is easily extended to IR-like querying of XML [6, 12]. What would be the most effective way to support IR-like querying in XPath and XQuery using relational database technology? The semantics of XPath and XQuery give rules for navigation through XML structure, but not the rules that specify how score values for XML elements should propagate and relate to each other. Similarly, the semantics of relational algebra introduce rules for manipulating relational tables that describe XML data, but again the rules for score computation and propagation cannot be derived from the relations present in the relational database.

We follow a three level database approach for developing an XML-IR system, consisting of conceptual, logical, and physical level. The benefits of the usage of a three level database management system is that we are able to provide data independence between the relational representation on the physical level, the proposed algebra on logical level, and the query language used on the conceptual level, and provide a certain level of abstraction from the information retrieval model used for ranked retrieval.

The introduction of an intermediate level allows for the usage of algebraic properties for query rewriting and optimization. The optimization should be achieved not only for the regular XPath/XQuery queries but for the IR-like queries as well. In this paper we study the usefulness of algebraic properties for query optimization and for developing and understanding IR-like extensions. The algebra we propose is based on so-called region algebras [2, 4, 11, 13]. Region algebras are sufficiently simple to study algebraic properties in depth, and they are sufficiently powerful to express IR-like queries as those proposed in the NEXI query language used for the evaluation of XML retrieval in INEX [16]. NEXI stands for "narrowed extended XPath". It only uses the descendant axis step from XPath, and it extends XPath with a special about-function that provides IR-like search. The region algebra can be easily extended to support other XPath axis steps with additional parent information [14]. The basic idea behind the algebra is to support as much as possible for the full text search requirements [3] and it is driven by the wish to integrate XML databases and information retrieval as discussed in [1].

Unlike many approaches for ranked retrieval in XML, the algebra we define assumes that the ranking is a part of the algebra and not a side effect of performing some operations on regions (like in [13]) or a separate IR module (like in many IR approaches for XML retrieval). Therefore, we follow the approach taken by Fuhr et al. [7, 8], although we base our algebra on containment model rather than path model, and do not make any restrictions on the definition of retrieval model. By defining the algebra in such a way we have the opportunity to utilize the optimization methods not just for basic region algebra operators, but for the ranking region

WIRD'04, the first Workshop on the Integration of Information Retrieval and Databases (WIRD'04), Sheffield, United Kingdom

```
<article lang=''en'' date=''10/02/04''>
   <title>Region algebra</title>
   <bdy>
      <sec>
         <p>Structured documents ...</p>
         <p>Text search ...</p>
      </sec>
      ...
   </bdy>
   ...
</article>
```

Figure 1: Example XML document.

algebra operators as well. This allows for the introduction of more powerful optimization techniques concerned with speeding-up the execution of operations that compute score values for ranked retrieval.

The paper is organized as follows. In Section 2 we explain how relational technology is used to process NEXI queries. We give the translation of NEXI queries into relational algebra and discuss why we need an intermediate level. Section 3 introduces our region algebra and discuss region algebra operator properties. In Section 4, we illustrate how operators for ranked retrieval follow the properties of basic region algebra operators and discuss the opportunities for query optimization in our region algebra, extended for ranked retrieval. We conclude the paper with a discussion and our plans for future research.

2. XML AND RELATIONAL DATABASES

In this section we explain the formation of the XML data set and discuss some issues on the indexing of XML documents. The relational storage of such documents is also discussed, along with the relational algebra expressions for two NEXI query examples.

2.1 Representing XML in Relational Databases

Most of the database approaches to XML choose to index XML documents before storing them into relational tables. The rationale for this is the structural organization of XML documents and the benefits that can be achieved when querying such indexed relational representation of XML documents. For an illustration we refer to [10] where the authors used the pre-post and stretched pre-post indexing scheme for the relational storage of XML documents. In our approach we used a variant of the stretched pre-post indexing[1] scheme that also indexes each word in XML text nodes. Note that the indexing also produces the initial data set for the data model that we define in Section 3.

The data set creation, i.e., the formation of the initial data set from (plain text) XML documents can be explained through the usage of a two step indexing process[2]. The indexing process is explained using an example XML document given in Figure 1. In the first step each token in the XML document (denoted with D) is indexed regarding its relative position with respect to its beginning and its type: $I_1 : D \rightarrow X$. As a result we obtain a set of elements: $x \in X$, uniquely identified by their position in the XML document. Each element has the form of $x = \{position, token, token\_type\}$ as shown in Table 1.

Table 1: Intermediate index structure (X) obtained after initial indexing (I_1) of XML document depicted in Figure 1.

position	token	token_type
0	<article>	start tag
1	lang	attribute name
2	''en''	attribute value
3	date	attribute name
4	''10/02/04''	attribute value
5	<title>	start tag
6	region	term
7	algebra	term
8	</title>	end tag
9	<bdy>	start tag
10	<sec>	start tag
11		start tag
12	structured	term
13	documents	term
...	...	...
54		end tag
...	...	...
576	</sec>	end tag
...	...	...
9876	</bdy>	end tag
...	...	...
10034	</article>	end tag

The second step produces regions that we can consider as the initial data set. These regions are produced by pairing corresponding tokens that represent opening and closing tags, attribute names and values, etc., and by removing mark-up delimiters from the tokens: $I_2 : X \rightarrow R$. This will result in a data set like the one presented in Table 2. Thus, the initial data set construction can be defined as a composition of two indexing procedures: $I = I_1 \circ I_2$. Although the indexing is a two step process it can be implemented as a single walk through an XML document using the SAX parser and stack structures (see [10]).

Table 2: Data model for XML document presented in Figure 1 obtained after the composition of initial indexing (I_1) and final indexing (I_2).

start	end	name	type
0	10034	article	node
1	2	lang	attr_name
2	2	en	attr_value
3	4	date	attr_name
4	4	10/02/04	attr_value
5	8	title	node
6	8	-	text
6	6	region	term
7	7	algebra	term
9	9876	bdy	node
10	576	sec	node
11	54	p	node
12	53	-	text
12	12	structured	term
13	13	documents	term
...	...	...	...

An indexed XML document, however, is not stored in one relational table since this table will be huge and in most

[1]Note that the term indexing differs from the concept of indexing as defined in the traditional database systems. It denotes the method used for creation of the initial data set.

[2]Although XML documents are actually graphs we will simplify the XML structure and treat these entities as if they were organized as a hierarchical (tree-like) structure.

cases (on most platforms) hard to process. In many relational approaches to XML different fragmentations of this basic table are used. The fragmentation can be horizontal, based only on type of XML nodes (like in [10] and [12]), vertical based on a name and/or type of XML elements, e.g., [6], or based on paths to XML nodes in an XML tree structure (like in [15]). For illustrative purpose we use horizontal fragmentation of XML data as presented in [12]. Consequently, different relational tables are defined for the XML element nodes and attribute nodes and the word table is defined for the words in the XML text nodes. This is depicted in Table 3. In further discussion we will not consider the attribute table since it is not of the interest for the issues that we are discussing in this paper.

Table 3: Relational data model for storing XML document presented in Figure 1.

Node table $\mathcal{N}$

start	end	name	type
0	10034	article	node
5	8	title	node
6	8	-	text
9	9876	bdy	node
10	576	sec	node
11	54	p	node
12	53	-	text
...	...	...	...

Word table $\mathcal{W}$

start	name
6	region
7	algebra
12	structured
13	documents
...	...

Attribute table $\mathcal{A}$

start	owner	name	type
1	0	lang	name
2	0	en	value
3	0	date	name
4	0	10/02/04	value
...	...	...	...

2.2 From XML Queries to Relational Algebra

The two example queries given in Figure 2 will be used as our leading examples in the following sections. As query language we use NEXI query language which has officially been adopted for INEX 2004[3]. Its detailed description can be found in [16]. For now we consider that the *about* condition inside queries is strict (corresponds to a Boolean search, i.e., *about* behaves the same as XPath *contains* expression). Later on in this paper we elaborate more on the use of the *about* clause for ranking.

For the chosen storage model, composed of $\mathcal{N}$ and $\mathcal{W}$ (and $\mathcal{A}$), we can directly transform any NEXI expression into relational algebra expression. For NEXI example query 1 depicted in Figure 2 a possible relational algebra expressions could be specified as given in Figure 3. We disregard the type attribute in expressions for brevity.

Note that there is a frequent usage of a group of expressions consisting of join and projection operations that simulate the XPath descendant/ancestor step. This group of expressions actually represents the bottleneck for XPath query processing, since its naive execution is extremely slow. A number of techniques have been proposed to speed up the execution of XPath descendant and ancestor steps, such as multi-predicate merge join [17], staircase join [10], containment join [12], etc. Using such abstract join operators, denoted with $\bowtie_{\sqsupset}$ (for expression types R_7, R_8 and R_{10} in Figure 3) and $\bowtie_{\sqsubset}$ (for expression type R_6 in Figure 3), the query plan for NEXI query example 2 can be expressed as shown in Figure 4.

[3] http://inex.is.informatik.uni-duisburg.de:2004/.

Figure 3: Relational query plan for example query 1 given in Figure 2.

$$
\begin{aligned}
R_1 &= \sigma_{name=\text{“}article\text{“}}(\mathcal{N}) \\
R_2 &= \sigma_{name=\text{“}bdy\text{“}}(\mathcal{N}) \\
R_3 &= \sigma_{name=\text{“}sec\text{“}}(\mathcal{N}) \\
R_4 &= \sigma_{name=\text{“}structured\text{“}}(\mathcal{W}) \\
R_5 &= \sigma_{name=\text{“}documents\text{“}}(\mathcal{W}) \\
R_6 &= \pi_{start_2,end_2,name_2}(R_2 \bowtie_{start_2>start_1,end_2<end_1} R_1) \\
R_7 &= \pi_{start_3,end_3,name_3}(R_3 \bowtie_{start_3<start_4,end_3>end_4} R_4) \\
R_8 &= \pi_{start_3,end_3,name_3}(R_3 \bowtie_{start_3<start_5,end_3>end_5} R_5) \\
R_9 &= R_7 \cap R_8 \\
R_{10} &= \pi_{start_6,end_6,name_6}(R_6 \bowtie_{start_6<start_9,end_6>end_9} R_9)
\end{aligned}
$$

Figure 4: Relational query plan for example query 2 given in Figure 2.

$$
\begin{aligned}
R_1 &= \sigma_{name=\text{“}article\text{“}}(\mathcal{N}) \\
R_2 &= \sigma_{name=\text{“}bdy\text{“}}(\mathcal{N}) \\
R_3 &= \sigma_{name=\text{“}sec\text{“}}(\mathcal{N}) \\
R_4 &= \sigma_{name=\text{“}p\text{“}}(\mathcal{N}) \\
R_5 &= \sigma_{name=\text{“}region\text{“}}(\mathcal{W}) \\
R_6 &= \sigma_{name=\text{“}algebra\text{“}}(\mathcal{W}) \\
R_7 &= \sigma_{name=\text{“}XML\text{“}}(\mathcal{W}) \\
R_8 &= \sigma_{name=\text{“}information\text{“}}(\mathcal{W}) \\
R_9 &= \sigma_{name=\text{“}retrieval\text{“}}(\mathcal{W}) \\
R_{10} &= R_2 \bowtie_{\sqsubset} R_1 \\
R_{11} &= ((R_{10} \bowtie_{\sqsupset} R_5) \cap (R_{10} \bowtie_{\sqsupset} R_6)) \bowtie_{\sqsupset} (R_3 \bowtie_{\sqsupset} R_7) \\
R_{12} &= R_4 \bowtie_{\sqsubset} R_{11} \\
R_{13} &= (R_{12} \bowtie_{\sqsupset} R_8) \cap (R_{12} \bowtie_{\sqsupset} R_9)
\end{aligned}
$$

2.3 Do We Need an Algebra in Between?

There might be a number of reasons to define an algebra. First of all, as we saw in the previous section, to be able to express XPath+IR (NEXI) queries in relational databases we need new operators for efficient execution of XPath+IR subexpressions, such as descendant and ancestor steps, containment conditions, etc. The exact technique how we implement the subexpression is defined on the physical level, and it does not have to be unique, i.e., we can have multiple variants of relational expression for the same XPath+IR subexpressions. The execution times for distinct implementations differ regarding the relational storage of the XML data, parameters of the relational tables, and index structure used for the acceleration of relational expression execution in relational databases.

Another important issue concerning immediate translation of XPath+IR expressions into relational algebra is that the algebraic expressions are highly dependent on the relational model chosen for the storage of XML data. If we change the relational storage model, the relational algebra expressions for each query have to be rewritten according to the

Figure 2: NEXI queries.

Example NEXI query 1:

```
//article//bdy[about(.//sec, structured) and about(.//sec, documents)]
```

Example NEXI query 2:

```
//article//bdy[about(., region) and about(., algebra)][about(.//sec, XML)]//p[about(., information) and about(., retrieval)]
```

relational model. This is especially the case for XML, since usually huge relational tables, which have more than a million entries, are typically broken into a number of smaller ones using one of the fragmentation methods mentioned in section 2.1. Having a logical level with the algebra defined in it would provide the right level of abstraction considering different XPath+IR queries formed on the conceptual level, and the relational storage structure chosen on the physical level. In such a way we provide the needed *data independence* on logical level. Furthermore, the reasoning that can be done on the logical level can be useful for query rewriting and *optimization*. Using knowledge about the size of the operands and the cost for the execution of different operators on the physical level we are able to generate different logical query plans achieving faster execution times and lower usage of main memory when executing on physical level.

A final but important reason for defining an algebra is to enable the expression of IR-like queries (*about* in NEXI), i.e., score computation and ranking of XML elements. Therefore, the algebra should provide a specific level of *IR understanding* that is based on the retrieval model used for score computation. The operators used for score computation should adhere to certain operator properties which can be used for query optimization based on the definition of score operators.

The exact way of how we use region algebra operator properties on the logical level, and how we extend the region algebra to support ranked retrieval is explained in the next two sections.

3. REGION ALGEBRA

For defining the intermediate logical level we have chosen the region algebra approach, because it is already well established in the area of structured document retrieval [2, 4, 11, 13], and because of the useful properties of region algebra operators as we discuss in the remainder of the paper.

With the specification of the region algebra data model we provide a uniform platform for defining region algebra operators. We discuss the basic XML region algebra data model which can be defined using four region attributes, based on the indexed data set described in the previous section (for more details see [12]).

Definition 1. *The basic region algebra data model is defined on the domain R which represents a set of region tuples. A region tuple r ($r \in R$), $r = (s, e, n, t)$, is defined by these four attributes: region start attribute - s, region end attribute - e, region name attribute - n, and region type attribute - t. The 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 node names, content words, attribute names, attribute values, etc. To be able to distinguish different node types in XML the type information is needed.

Table 4: Basic score region algebra operators.

Operator	Operator definition
$\sigma_{n=name}(R)$	$\{r \mid r \in R \wedge n = name\}$
$R_1 \sqsupset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \exists r_2 \in R_2 \wedge s_1 < s_2 \wedge e_1 > e_2\}$
$R_1 \sqsubset R_2$	$\{r_1 \mid r_1 \in R_1 \wedge \exists r_2 \in R_2 \wedge s_1 > s_2 \wedge e_1 < e_2\}$
$R_1 \sqcap R_2$	$\{r \mid r \in R_1 \wedge r \in R_2\}$
$R_1 \sqcup R_2$	$\{r \mid r \in R_1 \vee r \in R_2\}$

Next, we define the basic region algebra operators. The definition of region algebra operators is based on the operators specified in the previous region algebra approaches, extended to support a specific XML structure. Table 4 defines the following five basic region algebra operators: selection (σ), containing ($\sqsupset$), contained by ($\sqsubset$), 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)[4].

As can be noticed in Table 4, instead of an interval operator used in [2, 4, 11, 13] (usually denoted with $I(token)$ in region algebra approaches) which is actually not a real region algebra operator but rather specifies the indexing function applied to a specified token that returns the region set of the occurrences of *token* in a document, we introduced a selection operator (σ). The selection operator is a unary region algebra operator that operates on a region set and produces a region set as a result. It is defined to enable the selection of regions formed during the initial data set creation (explained in section 2.1), based on name (and type) region attributes.

Following the query examples given in Figure 2, we give the same query execution plans defined using the region algebra operators instead of the relational ones. The region algebra query plans for query examples 1 and 2 are given in Figure 5 and Figure 6. We use C to denote the initial data set of regions. We can note a great resemblance between the previous relational query plans and region algebra query plans. This exerts the simplicity of transforming region algebra expressions into relational expression. However, the change in the relational storage will result in the change of query plan on the physical level, while the query plan on the logical level will remain the same.

We can state here that in order to model XML properly, we

[4]In Table 4 we do not use the type information as it is not of great importance for this paper. Thus, the exact definition of selection operator should be $\sigma_{n=name,t=type}(R) = \{r \mid r \in R \wedge n = name \wedge t = type\}$, if type attribute would be used.

Figure 5: Region algebra query plan for example query 1 given in Figure 2.

$$\begin{aligned}
\text{ARTICLE} &= \sigma_{n=\text{“}article\text{“}}(C)\\
\text{BDY} &= \sigma_{n=\text{“}bdy\text{“}}(C)\\
\text{SEC} &= \sigma_{n=\text{“}sec\text{“}}(C)\\
\text{STRUCTURED} &= \sigma_{n=\text{“}structured\text{“}}(C)\\
\text{DOCUMENTS} &= \sigma_{n=\text{“}documents\text{“}}(C)
\end{aligned}$$

$$\begin{aligned}
R_1 &= (\text{SEC} \sqsupset \text{STRUCTURED}) \sqcap (\text{SEC} \sqsupset \text{DOCUMENTS})\\
R_2 &= (\text{BDY} \sqsubset \text{ARTICLE}) \sqsupset R_1
\end{aligned}$$

Figure 6: Region algebra query plan for example query 2 given in Figure 2.

$$\begin{aligned}
\text{ARTICLE} &= \sigma_{n=\text{“}article\text{“}}(C)\\
\text{BDY} &= \sigma_{n=\text{“}bdy\text{“}}(C)\\
\text{SEC} &= \sigma_{n=\text{“}sec\text{“}}(C)\\
\text{P} &= \sigma_{n=\text{“}p\text{“}}(C)\\
\text{REGION} &= \sigma_{n=\text{“}region\text{“}}(C)\\
\text{ALGEBRA} &= \sigma_{n=\text{“}algebra\text{“}}(C)\\
\text{XML} &= \sigma_{n=\text{“}XML\text{“}}(C)\\
\text{INFORMATION} &= \sigma_{n=\text{“}information\text{“}}(C)\\
\text{RETRIEVAL} &= \sigma_{n=\text{“}retrieval\text{“}}(C)
\end{aligned}$$

$$\begin{aligned}
R_1 &= \text{BDY} \sqsubset \text{ARTICLE}\\
R_2 &= ((R_1 \sqsupset \text{REGION}) \sqcap (R_1 \sqsupset \text{ALGEBRA})) \sqsupset (\text{SEC} \sqsupset \text{XML})\\
R_3 &= (\text{P} \sqsubset R_2)\\
R_4 &= (R_3 \sqsupset \text{INFORMATION}) \sqcap (R_3 \sqsupset \text{RETRIEVAL})
\end{aligned}$$

could enrich the definition of a region with the additional information of XML references, parent or level information, etc. For details on some extensions on region algebra approaches we refer to papers [12] and [14].

3.1 Region Algebra Operator Properties

In this section we discuss properties of algebraic operators and their use for query rewriting and optimization. Some of the properties are illustrated using the examples given in Figure 5 and Figure 6. Many properties are mentioned in papers about region algebra by Clarke et al. [4], and Jaakkola and Kilpelainen [11], but none of the papers discuss their usage. Our study on region algebra shows that there are only few operators that have the basic operator properties such as: identity, inverse, commutativity, associativity, and distributivity. However, there is a number of region algebra specific properties which can be considered as a special case of distributivity and associativity properties.

In general, we can distinguish two classes of binary region algebra operators. The first class consists of containment operators: $\sqsupset$, $\sqsubset$, while the second class consists of the standard set operators: $\sqcap$ and $\sqcup$. Only set operators have the identity element, which is the initial data set C for operator $\sqcap$ (property (1)), and the empty set $\emptyset$ for operator $\sqcup$ (property (2)). There is no inverse element for any of the operators. The set operators are commutative (properties (3) and (4)) and associative (properties (5) and (6)). Considering the distributivity property, only some combinations of operators follow it. The operators $\sqsupset$ and $\sqsubset$ distribute over the operator $\sqcup$ (properties (7) and (8)), while the operator $\sqcap$ distributes over the operator $\sqcup$ and vice versa (properties (9) and (10)).

Identity

$$R \sqcap C = C \sqcap R = R \tag{1}$$

$$R \sqcup \emptyset = \emptyset \sqcup R = R \tag{2}$$

Commutativity

$$R_1 \sqcap R_2 = R_2 \sqcap R_1 \tag{3}$$

$$R_1 \sqcup R_2 = R_2 \sqcup R_1 \tag{4}$$

Associativity

$$(R_1 \sqcap R_2) \sqcap R_3 = R_1 \sqcap (R_2 \sqcap R_3) \tag{5}$$

$$(R_1 \sqcup R_2) \sqcup R_3 = R_1 \sqcup (R_2 \sqcup R_3) \tag{6}$$

Distributivity

$$R_1 \sqsupset (R_2 \sqcup R_3) = (R_1 \sqsupset R_2) \sqcup (R_1 \sqsupset R_3) \tag{7}$$

$$R_1 \sqsubset (R_2 \sqcup R_3) = (R_1 \sqsubset R_2) \sqcup (R_1 \sqsubset R_3) \tag{8}$$

$$R_1 \sqcap (R_2 \sqcup R_3) = (R_1 \sqcap R_2) \sqcup (R_1 \sqcap R_3) \tag{9}$$

$$R_1 \sqcup (R_2 \sqcap R_3) = (R_1 \sqcup R_2) \sqcap (R_1 \sqcup R_3) \tag{10}$$

Special cases of associativity and distributivity

There are several interesting properties of the region algebra operators which can be useful for query rewriting and optimization on the logical level of a database. Here we mention the special case of containment operator associativity (property (11)), containment operator normalization (property (12)), and special case of set operator distributivity (property (13)). The first two properties are mentioned in papers [4] and [11]. We know of no publication on region algebras that mentions the third property.

For operators $op1 \in \{\sqsupset, \sqsubset\}$ and $op2 \in \{\sqsupset, \sqsubset\}$ properties (11) and (12) hold.

$$(R_1\ op1\ R_2)\ op2\ R_3 = (R_1\ op2\ R_3)\ op1\ R_2 \tag{11}$$

$$(R_1\ op1\ R_2)\ op2\ R_3 = (R_1\ op1\ R_2) \sqcap (R_1\ op2\ R_3) \tag{12}$$

For operators $op1 \in \{\sqcap, \sqcup\}$ and $op2 \in \{\sqsupset, \sqsubset\}$ property (13) is true.

$$(R_1\ op1\ R_2)\ op2\ R_3 = (R_1\ op2\ R_3)\ op1\ (R_2\ op2\ R_3) \tag{13}$$

To illustrate properties (11) and (12) we use the region algebra expression specified for the example query 1, given in Figure 5:

$$(\text{BDY} \sqsubset \text{ARTICLE}) \sqsupset ((\text{SEC} \sqsupset \text{STRUCTURED}) \sqcap (\text{SEC} \sqsupset \text{DOCUMENTS}))$$

The expression may be read as follows: “Retrieve bdy-elements *contained-by* article-elements *containing* the *intersection* of sec-elements *containing* the term 'structured' and sec-elements *containing* the term 'documents'.” Using the property (11) we can rewrite this expression into:

$$(\text{BDY} \sqsupset ((\text{SEC} \sqsupset \text{STRUCTURED}) \sqcap (\text{SEC} \sqsupset \text{DOCUMENTS}))) \sqsubset \text{ARTICLE}$$

Furthermore, using the property (12) this expression can be rewritten into:

$$(\text{BDY} \sqsupset ((\text{SEC} \sqsupset \text{STRUCTURED}) \sqsupset \text{DOCUMENTS})) \sqsubset \text{ARTICLE}$$

or using again the property (11) to:

$$(\text{BDY} \sqsupset ((\text{SEC} \sqsupset \text{DOCUMENTS}) \sqsupset \text{STRUCTURED})) \sqsubset \text{ARTICLE}$$

Using properties (11) and (12) we are able to choose the most appropriate query plan assuming that we have the information on which subexpressions are more selective. This reasoning can be applied for choosing which subexpressions will be more selective for $\text{SEC} \sqsupset \text{DOCUMENTS}$ or $\text{SEC} \sqsupset$

STRUCTURED, or similarly for BDY $\sqsupset$ ((SEC $\sqsupset$ DOCUMENTS) $\sqsupset$ STRUCTURED) or BDY $\sqsubset$ ARTICLE expressions. For example, since usually all regions from the BDY region set are contained in the ARTICLE region set, BDY $\sqsubset$ ARTICLE expression should be pushed up in the query plan as it is not a selective expression. Also the formulations of the query with the $\sqcap$ operator can be useful for parallel execution of two containment subqueries, if there exist an opportunity for such execution.

Property (13) is explained on a part of the example query 2, denoted with R_2 in Figure 6. Using the expression $R_1 =$ BDY $\sqsubset$ ARTICLE the part of the query example 2 can be expressed in region algebra as follows:

$$((R_1 \sqsupset \text{REGION}) \sqcap (R_1 \sqsupset \text{ALGEBRA})) \sqsupset (\text{SEC} \sqsupset \text{XML})$$

Using property (13) for operators $\sqcap$ and $\sqsupset$, this expression can be rewritten into:

$$((R_1 \sqsupset \text{REGION}) \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqcap ((R_1 \sqsupset \text{ALGEBRA}) \sqsupset (\text{SEC} \sqsupset \text{XML}))$$

We would obtain a similar region algebra expression for the `or` expression of the example NEXI query 1 in Figure 2 instead of the `and` expression, where operator $\sqcap$ will be replaced with the operator $\sqcup$. This will provide the opportunity for e.g., parallelization.

Furthermore, using the property (11) the next expression could be obtained from the previous one:

$$((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset \text{REGION}) \sqcap ((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset \text{ALGEBRA})$$

and after the usage of property (12) the final expression is:

$$((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset \text{REGION}) \sqsupset \text{ALGEBRA}$$

Therefore, instead of six operands and five operators we have a reduction to five operands and four operators, where the selection of regions R_1 that contain sections that contain term XML is pushed down to the first subexpression (assuming it is highly selective).

A similar expression can be obtained for the `or` combination in the *about*, where the distributivity property (8) could be applied as the last step:

$$((R_1 \sqsupset (\text{SEC} \sqsupset \text{XML})) \sqsupset (\text{REGION} \sqcup \text{ALGEBRA})$$

4. UNDERSTANDING IR

In this section some issues about the impact of introducing relevance ranking (i.e., score computation) in region algebra are discussed.

4.1 Relevance Ranking in Region Algebra

Relevance ranking cannot be explicitely expressed in the native relational algebra. To store the score information additional attribute for each entry in relational tables must be introduced. It stores the ranking score values for particular XML regions during the query execution. Furthermore, a number of operators have to be defined in the relational algebra which combination should express the score computation, i.e. instead of a join operator in Figure 4 we would use a combination of relational score operators. However, the introduction of score operators in the region algebra is easier and more elegant than in the relational algebra since the score computation is done on the right level of abstraction (logical level) and without considering the issues of how these operators are implemented on the physical level, i.e., in the relational algebra.

We use the same example query expressions given in Figure 2, except that we treat the *about* clause as a vague constraint instead of the strict interpretation in previous sections. Thus, paths and terms in the about clause do not have to be strictly matched, and the vague match is defined by the retrieval model. To be able to express IR-like search in XML databases the region algebra can be extended to support ranked retrieval. For that purpose the basic region algebra data model is extended with an additional attribute called *score* (denoted with p to resemble the probabilistic value).

To enable the score computation and the region ranking based on computed scores new region algebra operators are introduced. For each binary region algebra operator defined in Table 4 the probabilistic counterpart is defined. To distinguish between basic region algebra operators and score region algebra operators we use the index $_p$ for score operators. The score operators are depicted in Table 5. Note that the operators $\sqsupset_p$ and $\sqsubset_p$ produce all regions from the first operand (R_1) as a result, except that the score value (p_3) is changed according to the operator definition. The definitions of other two operators ($\sqcap_p$ and $\sqcup_p$) are similar to the definitions of basic ones except that they include score manipulation.

In the definition of score operators we introduced two complex scoring functions: $f_{\sqsupset}$ and $f_{\sqsubset}$, as well as two abstract operators: $\otimes$ and $\oplus$, which define the retrieval model. By using such definition of operators we leave their exact implementation for the physical level (for more details on this issue see [12]). However, for the operator $\oplus$ we assume that there exist a default value for score (denoted with d), and in case when the region r_1 is not present in the region set R_2 the score is computed as $p3 = p1 \oplus d$ and in case when the region r_2 is not present in the region set R_1 the score is computed as $p3 = d \oplus p_2$.

The functions $f_{\sqsupset}(r, R)$ and $f_{\sqsubset}(r, R)$, applied to a region r_1 and region set R_2, result in the numeric value that takes into account the score values of region $r_2 \in R_2$ and the probabilistic value that reflects the structural relation between the region r_1 and the region set R_2. For containing operator usually many regions from the region set R_2 are contained in the region r_1 (e.g., sections inside the article element). Although for contained by operator there is a small chance that the region r_1 is contained by a set of regions present in R_2, it can happen that e.g., there are nested XML elements with the same name (e.g., section inside other sections), and therefore, one region can be contained in multiple regions with the same name.

Following the previous discussion we can define complex functions as follows:

$$f_{\sqsupset}(r, R) = p * \sum_{\bar{r} \in R \sqsubset R'} (g_{\sqsupset}(\bar{r}, r) * \bar{p})$$

$$f_{\sqsubset}(r, R) = p * \sum_{\bar{r} \in R' \sqsupset R} (g_{\sqsubset}(\bar{r}, r) * \bar{p})$$

We assume that R' is the region set containing a single region r, and $g_{\sqsupset}(\bar{r}, r)$ and $g_{\sqsubset}(\bar{r}, r)$ are abstract functions

Table 5: Region algebra operators for score manipulation.

Operator	Operator definition
$R_1 \sqsupset_p R_2$	$\{r \mid r_1 \in R_1 \wedge (s,e,n) := (s_1,e_1,n_1) \wedge p := p_1 * f_{\sqsupset}(r_1, R_2)\}$
$R_1 \sqsubset_p R_2$	$\{r \mid r_1 \in R_1 \wedge (s,e,n) := (s_1,e_1,n_1) \wedge p := p_1 * f_{\sqsubset}(r_1, R_2)\}$
$R_1 \sqcap_p R_2$	$\{r \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge (s_1,e_1,n_1) = (s_2,e_2,n_2) \wedge (s,e,n) := (s_1,e_1,n_1) \wedge p := p_1 \otimes p_2\}$
$R_1 \sqcup_p R_2$	$\{r \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge ((s,e,n) := (s_1,e_1,n_1) \vee (s,e,n) := (s_2,e_2,n_2)) \wedge p := p_1 \oplus p_2\}$

used to define the score propagation based on the structural relation between the region r and regions in the region set R. In the straightforward implementation functions $g_{\sqsupset}(\bar{r}, r)$ and $g_{\sqsubset}(\bar{r}, r)$ can be constant functions equal to e.g., 1. If we base the retrieval model on the term frequency, former function can be defined as $g_{\sqsupset}(\bar{r}, r) = \frac{size(\bar{r})}{size(r)}$. Similarly latter function can be defined as $g_{\sqsubset}(\bar{r}, r) = \frac{size(\bar{r})}{\sum_{\bar{r}} size(\bar{r})}$. Since the exact retrieval model is not the main issue in this paper we will not elaborate more on it.

The abstract operator $\otimes$ specifies how scores are combined in an `and` expression, while the operator $\oplus$ defines the score combination in an `or` expression inside the NEXI predicate. In this paper we take the simple approach where $\otimes$ is a product of two score values, while $\oplus$ is the sum of scores, as it shows good behavior for retrieval (see [12]).

To illustrate the elegance of expressing score computation in region algebra we show how we can express NEXI query 1 in score region algebra:

(BDY $\sqsupset_p$ ((SEC $\sqsupset_p$ STRUCTURED)$\sqcap_p$(SEC $\sqsupset_p$ DOCUMENTS))) $\sqsubset$ ARTICLE

which very much resembles the original query plan for example query 1 given in Figure 5.

4.2 Properties of Score Operators

Considering the properties of score operators we can exert that some of the properties follow ones defined for the region algebra without scores, some of them hold only if some conditions are satisfied (conditional properties which depend on the underlying retrieval model), and some of them are no longer valid.

Operator $\sqcap_p$ defines the Boolean-like AND combination of scores obtained for two regions with the same region bounds (i.e., s and e values). It preserves the identity and inverse element properties from the $\sqcap$ operator (property (1)), but only in case the default score value for all regions in the initial region set is the value which is the identity element for abstract operator $\otimes$, i.e., 1 for the multiplication.

$$R \sqcap_p C = C \sqcap_p R = R, \text{ i.e., } p * 1 = 1 * p = p, \forall r \in R \qquad (14)$$

Furthermore, the operator $\sqcap_p$ is commutative or associative (properties (3) and (5)) if the operator $\otimes$ is commutative or associative, respectively, which is the case for multiplication.

An extension of the set union operator is given by the $\sqcup_p$ operator. It defines the Boolean-like OR combination of scores for two regions. Similarly to $\sqcap_p$ operator, operator $\sqcup_p$ preserves the identity and inverse element properties from the $\sqcup$ operator (property (2)) but only in case the default value (d) taken for the $\sqcup_p$ operator is the value which is the identity element for the abstract operator $\oplus$, i.e., 0 for the summation in our case.

$$R \sqcup_p \emptyset = \emptyset \sqcup_p R = R, \text{ i.e., } p + 0 = 0 + p = p, \forall r \in R \qquad (15)$$

As in the $\sqcap_p$ operator case, commutativity and associativity properties depend on the definition of $\oplus$ operator. In other words, operator $\sqcup_p$ is commutative or associative (properties (4) and (6)) if the operator $\oplus$ is commutative or associative, respectively, which is true for the summation.

Following the reasoning above and the fact that each region can equally likely be the right answer to a user query, we will consider that the default value for region score in the initial data set C is 1, from now on, and that the default value for score d of a region not present in the region set for operator $\sqcup_p$ is 0.

Based on the definition of operators using the region frequency it can be proven that the operators $\sqsupset_p$ and $\sqsubset_p$ do not distribute over the operator $\sqcup_p$ in general case (properties (7) and (8)). However, the operator $\sqcap_p$ distributes over the operator $\sqcup_p$, since $*$ distributes over $+$ (property (9)). Vice versa is not the case (property (10)).

$$R_1 \sqcap_p (R_2 \sqcup_p R_3) = (R_1 \sqcap_p R_2) \sqcup_p (R_1 \sqcap_p R_3) \qquad (16)$$

There are some additional conditional properties of score operators which can be of interest for the optimization. If we assume that functions $f_{\sqsupset}(r, R)$ and $f_{\sqsubset}(r, R)$ are not dependant on the score value of a region r (i.e., $f_{\sqsupset}(r, R) = f_{\sqsupset}(s, t, n, R)$ and $f_{\sqsubset}(r, R) = f_{\sqsubset}(s, t, n, R)$ property (11) holds for $op1_p \in \{\sqsupset_p, \sqsubset_p\}$ and $op2_p \in \{\sqsupset_p, \sqsubset_p\}$.

$$(R_1\ op1_p\ R_2)\ op2_p\ R_3 = (R_1\ op2_p\ R_3)\ op1_p\ R_2 \qquad (17)$$

In other words the score for each region in the result region set, denoted with p, is computed as:

$$p = (p_1 * f(r_1, R_2)) * f(r_1, R_3) = (p_1 * f(r_1, R_3)) * f(r_1, R_2)$$

We use $f(r, R)$ to denote one of the functions $f_{\sqsupset}(r, R)$ or $f_{\sqsubset}(r, R)$.

Furthermore, if the score value for all regions in the first operand R_1 is equal to 1 (default value for all regions), and we assume that the regions in each operand, R_2 and R_3, have the same score value, denoted with p_2 and p_3, property (12) holds.

$$(R_1\ op1_p\ R_2)\ op2_p\ R_3 = (R_1\ op1_p\ R_2) \sqcap_p (R_1\ op2_p\ R_3) \qquad (18)$$

i.e., for every region in the result set we obtain score p:

$$\begin{aligned} p &= (1 * \sum_{\bar{r}} (g(\bar{r}, r_1)) * p_2) * \sum_{\bar{r}} (g(\bar{r}, r_1)) * p_3 \\ &= (1 * \sum_{\bar{r}} (g(\bar{r}, r_1)) * p_2) * (1 * \sum_{\bar{r}} (g(\bar{r}, r_1)) * p_3 \end{aligned}$$

where $g(\bar{r}, r)$ is used either for $g_{\sqsupset}(\bar{r}, r)$ or for $g_{\sqsubset}(\bar{r}, r)$ and $\bar{r} \in R \sqsupset R'$ or $\bar{r} \in R' \sqsubset R$ based on the type of operators $op1_p$ and $op2_p$.

If we consider the expression R_4 in the NEXI query 2 we can come up with two query plans shown below.

((P $\sqsubset_p$ R_2) $\sqsupset_p$ INFORMATION) $\sqcap_p$ ((P $\sqsubset_p$ R_2) $\sqsupset_p$ RETRIEVAL)

and ((P $\sqsupset_p$ INFORMATION) $\sqcap_p$ (P $\sqsupset_p$ RETRIEVAL)) $\sqsubset_p$ R_2

Although they are almost the same we could not apply property (18) to the first query plan since the scores of regions in $\text{P} \sqsubset_p R_2$ are not equal to 1 in general case. For the second query plan the score value for all regions in P is 1 and the property can be applied. Thus, at the end we can come up with the query plan as shown below:

$$((\text{P} \sqsupset_p \text{INFORMATION}) \sqsupset_p \text{RETRIEVAL}) \sqsubset_p R_2$$

A version of property (13) for score operators does not hold for $\sqcap_p$ score operator, but holds for $\sqcup_p$. For example next equation is not true.

$$(R_1 \sqcap_p R_2) \sqsupset_p R_3 = (R_1 \sqsupset_p R_3) \sqcap_p (R_2 \sqsupset_p R_3)$$

It will be true only if $f_{\sqsupset}(r_{1,2}, R_3) = 1$ which is not true in general case:
$(p_1 * p_2) * f_{\sqsupset}(r_{1,2}, R_3) \neq (p_1 * f_{\sqsupset}(r_{1,2}, R_3)) * (p_2 * f_{\sqsupset}(r_{1,2}, R_3))$
However, next equation is true for $op_p = \{\sqsupset_p, \sqsubset_p\}$.

$$(R_1 \sqcup_p R_2)\ op_p\ R_3 = (R_1\ op_p\ R_3) \sqcup_p (R_2\ op_p\ R_3) \qquad (19)$$

i.e.,
$(p_1 + p_2) * f_{op}(r_{1|2}, R_3) = (p_1 * f_{op}(r_{1|2}, R_3)) + (p_2 * f_{op}(r_{1|2}, R_3))$

5. CONCLUSIONS AND FUTURE WORK

In this paper we address the problem of translating and executing IR-like queries over XML documents stored in relational databases. We exert the usefulness of intermediate logical level, for which we chose region algebra. The region algebra provides a number of properties which can be used for *query optimization* on the logical level of a database. Furthermore, the region algebra can support score operators used for ranked retrieval as an integral part of the algebra, and not as a sideffect. The expressiveness considering ranked retrieval in our region algebra is far more sophisticated than in other region algebra approaches that support ranked retrieval, like [2] and [13].

An important property of the region algebra is that expressing query plans using the operators given in Table 4 and Table 5 preserves *data independence* between the conceptual, the logical, and the physical level of a database. Similarly, these operators partially enable the separation between the structural query processing and the underlying probabilistic model used for ranked retrieval: a design property termed content independence in [5].

We are planning to further investigate the usefulness of region algebra operator properties and to experimentally evaluate the benefits of intermediate logical level. Further study on the influence of the definition of score operators (score functions and abstract operators) on score operator properties is needed. Our future research is also concerned with the consequences of modifying or changing the retrieval model used, e.g., by adding background statistics (i.e., collection frequency, document frequency) or by adapting the model for phrase search, etc. Moreover, we will work on the theoretical foundations as a support of retrieval models used for handling scores in region algebra.

6. REFERENCES

[1] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A Full-Text Search Extension to XQuery. In *Proceedings of the 13th conference on World Wide Web*, pages 583–594, 2004.

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

[3] S. Buxton and M. Rys. XQuery and XPath Full-Text Requirements. Technical report, W3C, 2003.

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

[6] D. Florescu and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *Proceedings of the 9th International World Wide Web Conference*, pages 67–76, 2000.

[7] N. Fuhr and K. Großjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 172–180, 2001.

[8] N. Fuhr and K. Großjohann. XIRQL: An XML Query Language Based on Information Retrieval Concepts. *ACM TOIS*, 22(2):313–356, 2004.

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

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

[11] J. Jaakkola and P. Kilpelainen. Using sgrep for Querying Structured Text Files. Technical Report C-1996-83, Department of Computer Science, University of Helsinki, 1996.

[12] J. List, V. Mihajlović, A. de Vries, G. Ramirez, and D. Hiemstra. The TIJAH XML-IR System at INEX 2003. In *Proceedings of the 2nd Initiative on the Evaluation of XML Retrieval (INEX 2003)*, ERCIM Workshop Proceedings, 2004.

[13] 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.

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

[15] G. Ramirez and A. P. de Vries. Combinining Indexing Schemes to Accelerate Querying XML on Content and Structure. In *Proceedings of the first Twente Data Management Workshop (TDM'04),* to apear, 2004.

[16] 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.

[17] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 425–436, 2001.