

An Algebraic Query Model for Effective and Efficient Retrieval of XML Fragments

Sujeet Pradhan

Kurashiki University of Science and the Arts

Nishinoura 2640, Tsurajima-cho

Kurashiki, Japan

sujeet@cs.kusa.ac.jp

ABSTRACT

Finding a suitable fragment of interest in a non-schematic XML document with a simple keyword search is a complex task. To deal with this problem, this paper proposes a theoretical framework with a focus on an algebraic query model having a novel query semantics. Based on this semantics, XML fragments that look meaningful to a keyword-based query are effectively retrieved by the operations defined in the model. In contrast to earlier work, our model supports filters for restricting the size of a query result, which otherwise may contain a large number of potentially irrelevant fragments. We introduce a class of filters having a special property that enables significant reduction in query processing cost. Many practically useful filters fall in this class and hence, the proposed model can be efficiently applied to real-world XML documents. Several other issues regarding algebraic manipulation of the operations defined in our query model are also formally discussed.

1. INTRODUCTION AND MOTIVATION

While it is widely-accepted that keyword search is the most friendly interface for querying XML documents, there is considerably less agreement on how to answer such keyword-based queries. Given a set of keywords as a query, there is no consensus on what portion or portions of an XML document should be retrieved as *the* answer.

An XML document is commonly modelled as a rooted tree (either ordered or unordered). The problem of finding a portion of interest in an XML document is thus equivalent to the problem of identifying an appropriate subtree of the tree which represents the document in consideration. While several studies have been done in the recent past regarding this issue in the literature[4][5][12][15][20], the primary focus has been on so-called *data-centric* XML documents such as bibliographic data. Data-centric XML documents are highly schematic and their element tag names are generally “semantically meaningful”. As a result, one can exploit both the schema and the tag names to identify meaningful XML

fragments to some precision. For example in [4] and [5], document components are indexed according to their inter-relationships which are determined by analysing semantic meanings of tag names such as `< book >`, `< author >` in a document.

In contrast, a *document-centric* XML document such as the one shown in Figure 1, hardly has any fixed schema, usually has long textual contents, and typically has tag names such as `< section >`, `< subsection >`, `< par >` etc. which only describe structural relationship, but offer little help in determining any semantic relationship among document components.

It is often argued that given a set of keywords as a query against an XML tree, the smallest subtree containing all the keywords is enough to answer this query[5][7][12][15][20]. While this argument seems logical enough in the realm of data-centric XML documents, it is not guaranteed to be effective to compute even a simple and intuitive answer to a query against general document-centric XML documents. As an illustration, consider a query `{XQuery, optimization}` against an XML document shown in Figure 1. According to the conventional query semantics, the smallest subtree containing both the keywords `XQuery` and `optimization` (the paragraph represented by node `n17` in Figure 1) would be the answer to this query. However, a general user may find the fragment represented by the nodes `n16`, `n17`, and `n18` more intuitive and more appropriate since it is self-contained and more informative than `n17` alone. Our first challenge is how to retrieve such a fragment as *one single answer unit* effectively by merely exploiting the structural relationship among the components of the underlying data.

Obviously the problem of retrieval unit in a document-centric XML document is more complex than in a data-centric one mainly because of the facts that 1) document-centric XML documents are non-schematic; 2) the tag names are not guaranteed to carry any literal semantics that would assist in determining the retrieval unit and 3) there is no prior knowledge of how keywords would be split across the nodes of a desired XML subtree (refer to Figure 2). As a result, we need a more effective query mechanism, which goes beyond the smallest subtree semantics, for identifying potential fragments of interest, particularly in document-centric XML documents.

In this paper, we intend to compute fragments of interest, which in most cases, may be larger than the fragments that would have been retrieved according to the smallest subtree semantics. One question that naturally arises then is: How large a fragment is large enough? There is no con-

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '06, September 12-15, 2006, Seoul, Korea.

Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09.

```

<proceedings title="VLDB Conference"> n0
  <paper title="A New Query Language for XML data"> n1
    <section heading="Introduction"> n2
      <par> ... XML data is ...</par> n3
    </section>
    ...
    <section heading="Query Processing"> n14
      <par>.. logical ... .. </par> n15
      <subsection heading=" Optimization Issues"> n16
        <par> ... successful implementation of XQuery ... query optimization ... </par> n17
        <par> ... some of these techniques in XQuery are borrowed from ... </par> n18
      </subsection>
    </section>
    ...
  </paper>
  ...
  <paper title="Efficient Stream Data Management"> n79
    <section heading="Introduction and Motivation"> n80
      <par> ... optimization is an integral part of any ... </par> n81
    </section>
    ...
  </paper>
  ...
</proceedings>

```

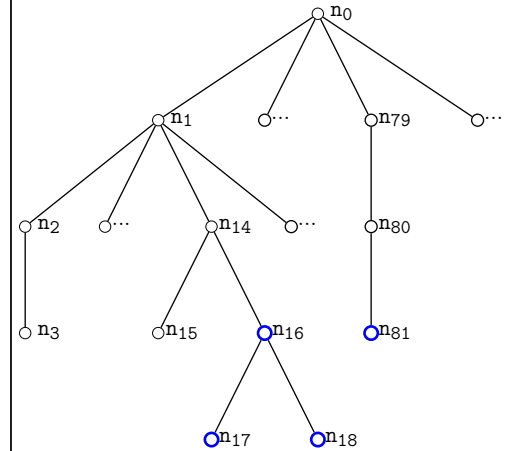


Figure 1: An Example XML Document and its Tree Representation

create answer to this question as the old adage goes, “One size does not fit all”. Although no one fixed size (or shape) of a fragment meets all queries, this paper attempts to find relevant-looking fragments; a relevant-looking fragment in this paper means a subtree containing all the query keywords, and, in addition, this subtree would have no extraneous nodes. Having said that, however, we do not wish to generate each and every computable fragment, because this may return an answer set with several potentially irrelevant fragments. For example, consider the fragment represented by the nodes **n0,n1,n14,n16,n17,n18,n79,n80,n81** in Figure 1. Even though this fragment consists of the nodes containing all the query keywords in our example query $\{XQuery, optimization\}$, it may still be considered irrelevant because a substantial portion of its contents are unrelated to each other. Not only will this overwhelm the user with a huge number of often irrelevant fragments, but it will also degrade the performance of the overall system. Therefore, our next challenge is how to avoid computation of fragments that are potentially irrelevant to a user.

This paper describes the formal framework of an algebraic query model designed to address the two challenges stated above for an effective and efficient XML keyword search. This query model is supported by *selection* and *join* operations; the two primary operations in traditional database systems. Keyword queries are transformed into algebraic expressions and XML fragments of interest are effectively computed by joining several document components whose contents would contain the specified keywords.

We then shift our focus on restricting the generation of a large number of irrelevant fragments in an answer set. For that purpose, we provide various filters, which in fact are selection predicates. Our main interest is however, not only in the restriction of the size of the query result, but also in investigating whether such filters would contribute to significant reduction of query processing cost. We explore the algebraic properties of several filters and show that not all filters are capable of contributing to query optimization. In

addition, this paper proves that *selection* with a specific class of filters, what we call *anti-monotonic* filters, can indeed be pushed down in the query evaluation tree ahead of *join* operations. As a result, by incorporating practically useful filters with anti-monotonic property in our query mechanism, we can expect a substantial performance gain in our query processing. We also discuss several other algebraic manipulation of the operations of our model, which could enable us to achieve better processing efficiency under certain conditions.

Our contributions in this paper can be summarized as:

- To find appropriate fragments of interest with a simple keyword search in a non-schematic XML document, we provide a novel query semantics — a semantics that is different from the smallest subtree semantics commonly adopted for schematic XML documents.
- Our algebra has a strong theoretical foundation with several promises for query optimization, which can be exploited by a query processor for performance gain.
- To prevent an overwhelming size of an answer set, we propose database style filtering instead, which would complement existing XML retrieval systems that apply IR-style ranking techniques.
- Although no experiments have been conducted yet to verify the viability of our model, we provide formal proofs and convincing examples to justify our claims.

The rest of the paper is organized as follows. In Section 2, query model is formalized by defining all the algebraic operations required to compute answers to a query. Optimization techniques are discussed in Section 3. In Section 4, we give an illustrative example to explain different query evaluation strategies that the model offers. Section 5 sheds some light on a few important issues that are not fully investigated in this paper. Related work is provided in Section 6, and finally we conclude by highlighting our contributions.



Figure 2: A few possible variations of the way two keywords k_1 and k_2 are split across the nodes of the target subtrees of interest

2. QUERY MODEL

In this section, we formally describe our query model. Preliminary ideas regarding this model are also given in our earlier work[14].

2.1 Basic Definitions

DEFINITION 1 (DOCUMENT). An XML document, is a rooted ordered tree $\mathcal{D} = (\mathbf{N}, \mathbf{E})$ with a set of nodes \mathbf{N} and a set of edges $\mathbf{E} \subseteq \mathbf{N} \times \mathbf{N}$. There exists a distinguished root node from which the rest of the nodes can be reached by traversing the edges in \mathbf{E} . Each node except the root has a unique parent node.

Each node \mathbf{n} of the document tree is associated with a logical component such as `< section >` or `< par >` of the document. As in [7][9], we do not distinguish between tag/attribute names and text contents. There is a function `keywords(n)` that returns the representative keywords of the corresponding component in \mathbf{n} . The nodes are arranged in such a way that the depth-first pre-order traversal of the tree would preserve the topology of the document. We write `nodes(D)` for all the nodes \mathbf{N} .

DEFINITION 2 (DOCUMENT FRAGMENT). Let \mathcal{D} be an XML document. Then $\mathbf{f} \subseteq \mathcal{D}$ is a document fragment, or simply a fragment, iff `nodes(f) ⊆ nodes(D)` and the subgraph induced by `nodes(f)` in \mathcal{D} is a rooted tree. In other words, the induced subgraph is connected.

A fragment can thus be denoted by a subset of nodes in a document tree — the tree induced by which is also a rooted ordered tree. A fragment may consist of only a single node or all the nodes which constitute the whole document tree. In Figure 1, the set of nodes $\langle \mathbf{n16}, \mathbf{n17}, \mathbf{n18} \rangle^1$ is a fragment of the sample document tree. Hereafter, unless stated otherwise, the first node of a fragment represents the root of the tree induced by it. For clarity, we refer to a single-node fragment simply as a node.

2.2 Algebra

To formally define the operational semantics of a query, we first need to define operations on fragments and sets of fragments. The operations can be basically classified as (1) *selection* and (2) *join* operations.

DEFINITION 3 (SELECTION). Supposing \mathbf{F} be a set of fragments of a given document, and \mathbf{P} be a predicate which maps a document fragment into true or false, a selection from \mathbf{F} by the predicate \mathbf{P} , denoted by $\sigma_{\mathbf{P}}$, is defined as a subset \mathbf{F}' of \mathbf{F} such that \mathbf{F}' includes all and only fragments satisfying \mathbf{P} . Formally, $\sigma_{\mathbf{P}}(\mathbf{F}) = \{\mathbf{f} \mid \mathbf{f} \in \mathbf{F}, \mathbf{P}(\mathbf{f}) = \text{true}\}$.

¹For clarity, we use ‘(’ and ‘)’ instead of conventional ‘{’ and ‘}’ to enclose the nodes of a fragment

Hereafter, the predicate \mathbf{P} is also called a filter of the *selection* $\sigma_{\mathbf{P}}$.

The simplest filter is for the keyword selection of the type ‘keyword = k ’ which selects only those document fragments having the word ‘ k ’. Several other filters will be introduced in Section 3.3 below.

Next, we define various *join* operations on document fragments.

DEFINITION 4 (FRAGMENT JOIN). Let $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}$ be fragments of the document tree \mathcal{D} . Then, fragment join between \mathbf{f}_1 and \mathbf{f}_2 denoted by $\mathbf{f}_1 \bowtie \mathbf{f}_2$ is \mathbf{f} iff

1. $\mathbf{f}_1 \subseteq \mathbf{f}$,
2. $\mathbf{f}_2 \subseteq \mathbf{f}$ and
3. $\nexists \mathbf{f}'$ such that $\mathbf{f}' \subseteq \mathbf{f} \wedge \mathbf{f}_1 \subseteq \mathbf{f}' \wedge \mathbf{f}_2 \subseteq \mathbf{f}'$

Intuitively, the fragment join operation takes two fragments \mathbf{f}_1 and \mathbf{f}_2 of \mathcal{D} as its input and finds the minimal fragment \mathbf{f} in \mathcal{D} such that the resulting fragment would contain both the input fragments \mathbf{f}_1 and \mathbf{f}_2 , and there exists no other smaller fragment \mathbf{f}' contained by \mathbf{f} in \mathcal{D} , which would also contain the input fragments \mathbf{f}_1 and \mathbf{f}_2 . Figure 3 (b) shows the operation between two fragments $\langle \mathbf{n4}, \mathbf{n5} \rangle$ and $\langle \mathbf{n7}, \mathbf{n9} \rangle$ (refer to Figure 3 (a)) which finds its minimal subgraph fragment $\langle \mathbf{n3}, \mathbf{n4}, \mathbf{n5}, \mathbf{n6}, \mathbf{n7}, \mathbf{n9} \rangle$ (fragment inside dashed line in Figure 3 (b)). By its definition, the fragment join operation between arbitrary fragments $\mathbf{f}_1, \mathbf{f}_2$ and \mathbf{f}_3 has the following algebraic properties.

Idempotency $\mathbf{f}_1 \bowtie \mathbf{f}_1 = \mathbf{f}_1$

Commutativity $\mathbf{f}_1 \bowtie \mathbf{f}_2 = \mathbf{f}_2 \bowtie \mathbf{f}_1$

Associativity $(\mathbf{f}_1 \bowtie \mathbf{f}_2) \bowtie \mathbf{f}_3 = \mathbf{f}_1 \bowtie (\mathbf{f}_2 \bowtie \mathbf{f}_3)$

Absorption $\mathbf{f}_1 \bowtie (\mathbf{f}_2 \subseteq \mathbf{f}_1) = \mathbf{f}_1$

These properties not only enable an easy implementation of the operations but also lay foundation for optimizing query evaluation by enabling algebraic manipulation of operations defined further below.

Next, we extend this operation to a set of fragments. called *pairwise fragment join*, which is the set-variant of fragment join.

DEFINITION 5 (PAIRWISE FRAGMENT JOIN). Let \mathbf{F}_1 and \mathbf{F}_2 be two sets of fragments in a document \mathcal{D} , pairwise fragment join of \mathbf{F}_1 and \mathbf{F}_2 , denoted by $\mathbf{F}_1 \bowtie \mathbf{F}_2$, is defined as a set of fragments yielded by taking fragment join of every combination of an element in \mathbf{F}_1 and an element in \mathbf{F}_2 in a pairwise manner. Formally,

$$\mathbf{F}_1 \bowtie \mathbf{F}_2 = \{\mathbf{f}_1 \bowtie \mathbf{f}_2 \mid \mathbf{f}_1 \in \mathbf{F}_1, \mathbf{f}_2 \in \mathbf{F}_2\}.$$

Figure 3 (a),(c) illustrates an example of *pairwise fragment join* operation. For given $\mathbf{F}_1 = \{\mathbf{f}_{11}, \mathbf{f}_{12}\}$ and $\mathbf{F}_2 = \{\mathbf{f}_{21}, \mathbf{f}_{22}\}$, $\mathbf{F}_1 \bowtie \mathbf{F}_2$ produces a set of fragments $\{\mathbf{f}_{11} \bowtie \mathbf{f}_{21}, \mathbf{f}_{11} \bowtie \mathbf{f}_{22}, \mathbf{f}_{12} \bowtie \mathbf{f}_{21}, \mathbf{f}_{12} \bowtie \mathbf{f}_{22}\}$.

For arbitrary fragment sets $\mathbf{F}_1, \mathbf{F}_2$, and \mathbf{F}_3 , *pairwise fragment join* has the following algebraic properties.

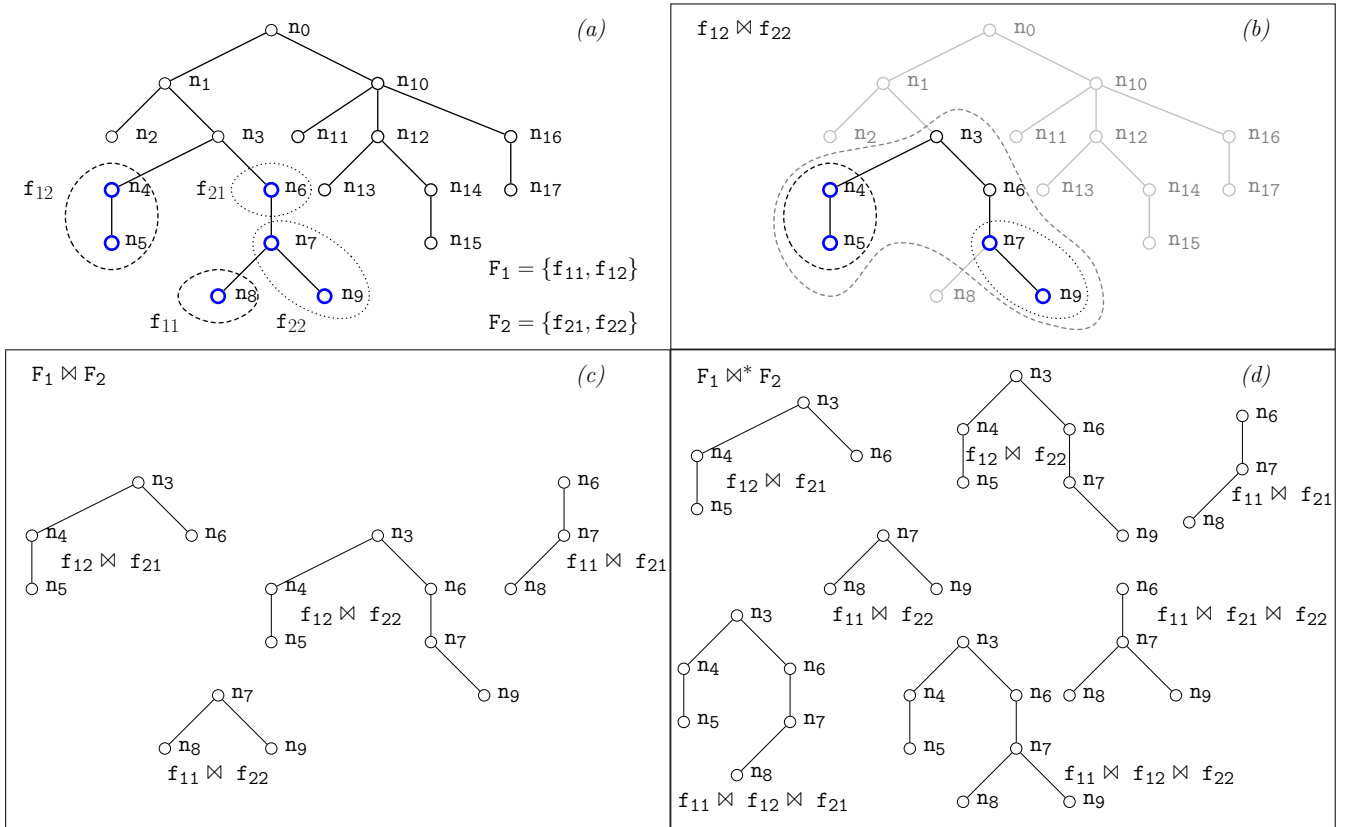


Figure 3: (a) A Document Tree (b) Fragment Join (c) Pairwise Fragment Join and (d) Powerset Fragment Join Operations

Commutativity $F_1 \bowtie F_2 = F_2 \bowtie F_1$

Associativity $(F_1 \bowtie F_2) \bowtie F_3 = F_1 \bowtie (F_2 \bowtie F_3)$

Monotonicity $F_1 \bowtie F_1 \supseteq F_1$

Distributive Law $F_1 \bowtie (F_2 \cup F_3) = (F_1 \bowtie F_2) \cup (F_1 \bowtie F_3)$

The *pairwise fragment join* operation does not satisfy the idempotency property as we can easily prove by showing counter examples for it.

We now define *powerset fragment join* — another variant of the *fragment join* operation.

DEFINITION 6 (POWERSET FRAGMENT JOIN). Let F_1 and F_2 be two sets of fragments in a document \mathcal{D} , powerset fragment join between F_1 and F_2 , denoted by $F_1 \bowtie^* F_2$, is defined as a set of fragments produced by applying fragment join operation to an arbitrary number (but not 0) of elements in F_1 and F_2 . Formally,

$$F_1 \bowtie^* F_2 = \{ \bowtie (F'_1 \cup F'_2) \mid F'_1 \subseteq F_1, F'_2 \subseteq F_2, F'_1 \neq \phi, F'_2 \neq \phi \}$$

where $\bowtie \{f_1, f_2, \dots, f_n\} = f_1 \bowtie \dots \bowtie f_n$.

Figure 3 (a),(d) illustrates an example of *powerset fragment join* operation. It should be noted here that for the same two sets of fragments $F_1 = \{f_{11}, f_{12}\}$ and $F_2 = \{f_{21}, f_{22}\}$ in Figure 3 (a), *powerset fragment join* produces more fragments than *pairwise fragment join* (refer to Figure 3 (c)). It should also be noted that some of the fragments are produced more than once due to the algebraic properties of *fragment join* and *pairwise fragment join*.

2.3 Query Evaluation

DEFINITION 7 (QUERY). A query can be denoted by $Q_P\{k_1, k_2, \dots, k_m\}$ where k_j is called a query term for all $j = 1, 2, \dots, m$ and P is a selection predicate.

We write $k \in \text{keywords}(n)$ to denote that query term k appears in the textual contents (element contents in XML terminology) associated with the node n .

DEFINITION 8 (QUERY ANSWER). Given a query $Q_P\{k_1, k_2, \dots, k_m\}$, answer A to this query is a set of document fragments defined to be

$$\{ f \mid (\forall k \in Q) \exists n \in f : n \text{ is a leaf node of } f \wedge k \in \text{keywords}(n) \wedge P(f) = \text{true} \}.$$

Note that as in [7], we also adopt conjunctive query semantics. Intuitively, an answer to a query is a document fragment consisting of several structurally-related logical components and each keyword in the query must appear in at least one component that constitutes the fragment. In addition, the fragment must satisfy the selection predicate(s) specified in the query.

A query represented by $\{k_1, k_2\}$ and a selection predicate P against a document \mathcal{D} can be evaluated by the following formula.

$$Q_P\{k_1, k_2\} = \sigma_P(F_1 \bowtie^* F_2)$$

where $F_1 = \sigma_{\text{keyword}=k_1}(F)$, $F_2 = \sigma_{\text{keyword}=k_2}(F)$ and $F = \text{nodes}(\mathcal{D})$.

