An Aggressive Aggregation of XML Documents for Summary Data Generation

Jong P. YOON
The Center for Advanced Computer Studies
University of Louisiana, Lafayette, LA 70504-4330

Larry KERSCHBERG
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030-4444

ABSTRACT

Aggregate functions are critically important and widely used to build summary data in WWW. Aggregation queries, that are used to summarize source data, may often result in incorrect answers due to the irregularity of XML data: an XML-element appears with irregular content structure and contains non-atomic or empty content although it follows a DTD or XML Schema. To cope with this problem, we propose an aggressive aggregation method for summarizing XML data. The contribution of this paper includes sound and complete collection of information from irregular XML data, and construction of summary data for XML documents in WWW. The method proposed in this paper can also be used for many other Web-based applications involving semistructured documents for electronic commerce and for OLAP data cubes.

Keywords: XML Summary Data, XML Aggregate Functions, XML Concept Hierarchy

1 INTRODUCTION

Extensible Markup Language (XML) is a new standard for representing and exchanging data in a variety of applications, each with its own special needs. Web information is increasingly represented in XML. As such XML web information increases, the generation of summary data about massive XML documents is needed.

Typical summary data generation methods include a technique of On-Line Analytical Processing (OLAP). OLAP in relational databases allows users to view aggregate XML data along a set of dimensions and hierarchies. Summary data can be organized in a data cube, which is simply a multidimensional hierarchy of aggregate values.

However, aggregation queries, or simply "aggregates," that are used to summarize source data, may often result in incorrect answers. That is, generating summary data for XML documents in World Wide Web (WWW) is unsatisfactory. It is partly because XML-element appears irregularly and has various content types. XML data is tagged by an XML-element. XML-elements are properly nested. The content of XML-elements may be characters, elements, a mix of characters and elements, or empty. An XML-element can be classified according to how its content is bound to the element:

- *Text-content* element. This element contains character data only or text only content.
- *Element-content* element. This element contains sub-element only content.
- *Mixed-content* element. This element contains both characters and sub-elements together.
- *Empty-content* element. This element does not contain any value of above.

For example, in Figure 1, `<p_name>` in line `(2)` is a text-content element, while `<part>` in line `(5)` is an element-content element. Note that XML-elements can contain whitespaces which can be parsed by an XML parser. `<product>` in line `(1)` is a mixed-content element, and `<cost>` in line `(28)` is an empty-content element. In addition to these four types of XML-elements together, with one more element type called the "*virtual-content element*" which will be described in the later section, comprise a new method of performing aggregate functions for XML data. In addition, we investigated that there needs one more type called "virtual-content element" which will be defined in Section 3.

```
 (0) <prodlist>
 (1)  <product> Equipment
 (2)    <p_name> Computer </p_name>
 (3)    <p_id> cp4 </p_id>
 (4)    <price> 800 </price>
 (5)    <part> <c_name>Monitor </c_name>
 (6)           <cost> 100 </cost></part>
 (7)    <part> <c_name> Body </c_name>
 (8)           <cost> 300 </cost>
               <part> <c_name>Processor </c_name>
                      <cost> 70 </cost></part>
               <part><c_name>Board</c_name></part></part>
 (9)    <part> <c_name>Keyboard </c_name>
(10)           <cost> 50 </cost></part></product>
(11)  <product> Equipment
(12)    <p_name> Computer </p_name>
(13)    <p_id> pc5
(14)    <price> 500 </price>
(15)    <part> <!-- the value "body" omitted. -->
(16)      <part> <c_name>Processor </c_name>
(17)             <cost> 70 </cost></part>
(18)      <part> <c_name>Board </c_name></part></part>
(19)    <part>
(20)        <c_name>Monitor </c_name></part>
(21)    <part>
(22)      <c_name>Keyboard </c_name><cost>80</cost></part>
(23)  </product>
(24)  <product> Equipment
(25)    <p_name> PC </p_name>
(26)    <p_id>pc6<price>700</price><part>Monitor</part>
(27)      <part>Body</part><part><c_name>Keyboard</c_name>
(28)      <cost /> </part>
(29) </product></prodlist>
```

Figure 1: Example of XML Data (in XDB2)

1.1 Motivating Examples

Consider an XML data as shown in Figure 1.

EXAMPLE 1.1 Consider that a query is posed to compute the number of computer "body" from WWW. Suppose that XML data in Figure 1 is available on the WWW. Computer "body" cannot be matched in line (15) because the element `part` is an element-content element, meaning that it does not have the value "body" bound to itself. Instead, it contains sub-elements only. Although there may exist unassembled bodies, there is no computers without having a body. In this case, one may want to count the sub-parts as a computer body if all the necessary sub-parts are available separately from the body.

EXAMPLE 1.2 What if one wants to count the micro "processors" again. In the same XML data, `part` in line (7) is a text-content element, meaning that there is no sub-elements. However, assume that the body is specified and so implicitly are its components. If one wants to count a processor for such a computer, we need to take that into account.

To cope with the problems illustrated in the motivating examples above, we propose a novel method of aggregating XML data to build summary data. We use DTD (Document Type Definition) or XML Schema that contains information about XML-elements of XML document instances. In addition, the concept tree will be constructed to specify the synonyms and component hierarchies among those terms that are useful to generate summary data.

1.2 Related Work

OLAP is a technology that supports applications requiring multidimensional analysis of data [5], which may be not only relational but also semistructured. OLAP allows users to have aggregate-results from both relational and semistructured data (e.g., number of parts for assembly) along a set of dimensions (e.g., inventory, production order and shipment, time) and dimension hierarchies (e.g., day, month, and year) [9].

Summary or fact tables are important in conventional data warehousing [16]. OLAP data cubes are generated by using the group-by operator in SQL and the size of those data cubes can be estimated [9]. Operations are developed [4], and interactive issues of aggregation[12], and incompleteness issues [11] have been investigated.

With the recent emergence of XML [6], a proposed standard for exchanging information on the Web [13], and the remarkable similarity of XML to typical models for semistructured data, support for query languages for semistructured data – and the performance of such queries over large semistructured databases – is of increasing importance.

Views are actively generated in the application of electronic commerce [2]. View generation in general can be used to build summary data [8] although these XML based views were not exploited in that way.

Queries or views initially expressed to build XML summary data need to be rewritten to produce sound and complete results. Literature on query rewriting falls into three general approaches. (1) The first approach is syntactic query rewriting [7, 15]. User-provided queries can be rewritten using structure hierarchy information if syntactically exact matching is not the only one available in a database schema. (2) The second approach is semantic query rewriting [17, 18]. User-provided queries can be rewritten using semantic information such as view, integrity constraints, and rules. The above two approaches are handled internally by a system. In contrast, (3) the third approach is user-interactive query rewriting [19]. User-provided queries are rewritten directly by users or at least by using user inputs.

2 PRELIMINARIES

2.1 XML Document and DTD

XML provides a simple and general markup facility as shown in the document of Figure 1. Consider a DTD for XML documents Figure 1 that is essentially a context-free grammar with several restrictions. For example, one may define the well-formed DTD declaration to constrain the XML documents as follows:

```
<!ELEMENT prodlist (product)*>
<!ELEMENT product (#PCDATA|(p_name,p_id,price?,part*))*>
<!ELEMENT part (#PCDATA|(c_name,cost?)|part*)*>
```

By defining an element as #PCDATA with other subelements, the element can be a mixed-content element. Notice that all elements are of #PCDATA type unless further specified.

Of the recently proposed standardization specifications, XML Schema is considered in this paper. XML Schema [1] expresses shared vocabularies, and the structure, content and semantics of XML documents. One XML Schema example for the element part in the DTD is as follows:

```
<xsd:element name="part" type="xsd:partType" />
<xsd:complexType name="partType" mixed="true"
                 minOccurs="0" maxOccurs="2" />
```

The element part is recursively defined as a mixed-content element in both DTD and XML Schema. In the XML Schema above, a part consists of at most two sub-parts. This information is very useful in that although a part information may be missing, if three sub-parts are matched, so are the super parts "by definition" (of XML Schema). In the later section, we will explain this in more detail.

2.2 Querying XML Documents

Research on semistructured data has addressed query-language design [3, 7, 10], and query processing and optimization [14]. In this section, we use the XML-QL language [10], as an example language to query XML documents and further to expand a user query to generate summary data.

The query examples below are specified with respect to the examples in Section 1, respectively. XML-QL queries consists of a WHERE clause, specifying what to select, and a CONSTRUCT clause, specifying what to return.

EXAMPLE 2.1: (XML-QL for EXAMPLE 1.1)

Count all (computer) bodies from the XML documents in www.a.b.e/XDB2.

```
WHERE <product> <part> $t </>
      </> IN "www.a.b.e/XDB2",
             (CONTAINS 'body', $t)
CONSTRUCT <result> <#bodies> COUNT ($t) </>
          </>
```

EXAMPLE 2.2: (XML-QL for EXAMPLE 1.2)

Count all (micro-) processors from the XML documents in www.a.b.e/XDB2.

```
WHERE <product> <part><part> $t </></>
      </> IN "www.a.b.e/XDB2",
             (CONTAINS 'processor', $t)
CONSTRUCT <result> <#processors> COUNT($t) </>
          </>
```

The expression <product> ... </product> in the WHERE clause is called a *condition pattern*, where the one in the CONSTRUCT clause is called a *display pattern*. Notice that the predicate CONTAINS is a user-defined function that takes a word and data, and returns true if the word is contained in the data. COUNT is an aggregate function. All aggregate functions (SUM, MAX, MIN, AVG, etc) can be considered. In the later section, the condition pattern will be rewritten to retrieve sound and complete aggregations from XML data.

2.3 XML Concept Tree

The terms or values used in XML documents are called *concepts* if they are essential for query matching. That is, the content of XML-elements can refer to the concepts. Such concepts are also represented in XML. The concept tree represents ⟨Hierarchy-type, Term⟩ pairs. There are two types of hierarchies: synonyms and components. For example, assume that the term "Computer" has its components such as "Monitor," "Keyboard," and "Body," and the term "Body" has, in turn, its components like "Processor," and "Board." Also, one may assume that the terms "Computer" and "PC" are synonymous, then the XML concept tree is constructed as in Figure 2.

```
 (0) <concept>
 (1)  <synonym>
 (2)   <term> Computer </term>
 (3)   <term> PC </term> </synonym>
 (4)  <synonym>
 (5)   <term>Body</term><term>Unit</term></synonym>
 (6)  <component>Computer <!-- Component Hierarchy -->
 (7)   <term> Monitor </term>
 (8)   <term> Keyboard </term>
 (9)   <term>Body<term>Processor</term>
(10)     <term> Board </term> </term> </component>
(11)  </concept>
```

Figure 2: Concept Tree Example in XML

3 XML AGGREGATE FUNCTIONS

In this section, we describe a novel method of aggregating XML data. Summary data require one or more aggregate functions like SUM, MAX, MIN, AVG, COUNT, etc. When aggregating XML data, broadly speaking, there may be the two approaches with respect to the following assumptions:

- Naive Approach (Optimistic Assumption): Assume that the structure of XML data models adequately the real-world enterprise. With this optimistic assumption, whatever results returned from the XML data are acceptable.

- Aggressive Approach (Pessimistic Assumption): Assume that the structure of XML data is usually irregular and initially partially unknown. With this pessimistic assumption, results returned from the XML data may not be satisfactory. One may want to expand the result to some extent. This aggressive approach makes use of schematic or semantic information. In this paper, we exploit two types of information: DTD (or XML Schema) of the schematic information, and a concept tree of the semantic information. Of course, for this, we may use other types of information, e.g., rules extracted by using data mining techniques. The aggregation operation in this approach is called *expanded aggregation*.

We will take one aggregate function COUNT as an example. The "counting" method we propose in this section can be modified to any other aggregate function without difficulty. We first construct a concept tree that represents synonyms and component hierarchies among terms used in XML data.

3.1 Expanded Aggregation Using XML Schema

We first assume for simplicity that COUNT($p_1 =' A'$) denotes α for an XML-QL query, WHERE $p_1 =' A'$ CONSTRUCT <result> <#p_1> COUNT(p_1) </></>. The aggressive approach is by nature applied to element-content elements and mixed-content elements when XML schemas are used to compute expanded aggregation of XML data. Notice that text-content elements or empty-content elements can be retrieved simply by the naive approach.

Consider the XML Schema as follows.

```
<xsd:element name="$e" type="xsd:$t" />
<xsd:complexType name="$t" minOccurs="$n$"
                             maxOccurs="$m$" />
```

The XML-element $e is defined as an element-content element in the schema, and it consists at least n and at most m of the type $t. Consider an aggregation query, COUNT($p_1 =' A'$). Suppose that p_1 is either an element- or mixed-content element, and it does no value bound to itself. Then, we need to expand the given query for the aggregation of element-/mixed-content element.

1. Aggressive Approach for Super Elements.
   (a) Verify which element value contains the requested value.
   (b) If an element-content element does not contain the same requested value, then that element implicitly has the same requested value.
2. Aggressive Approach for Sub-Elements.
   (a) First, count the number of sub-elements of the same type. If the outcome of the counting is out of the range $[n, m]$, then no answer is provided.
   (b) If the outcome of the counting is the maximum occurrence m, then we know that the XML document describes all required sub-elements for p_1, the element p_1 exists implicitly. Therefore, the total count increases by one.
   (c) Otherwise, the expansion aggregation is dependent on the application domain, which can be represented in a concept tree that will be used as well in the following subsection.

Examples

Recall the XML schema that we considered in Section 2.1:

```
<xsd:element name="part" type="xsd:partType" />
<xsd:complexType name="partType" mixed="true"
                 minOccurs="0" maxOccurs="2" />
```

A full expansion of the queries in EXAMPLE 2.1 and 2.2 can be as follows:

EXAMPLE 3.1: (XML-QL for EXAMPLE 2.1)

```
WHERE (<product> <part> $t </>
      </> IN "www.a.b.e/XDB2",
            (CONTAINS $t, 'body')) or
      (<product> <part> $s <part> $x </></>
      </> IN "www.a.b.e/XDB2.xml",
            COUNT($x)=2 )
CONSTRUCT <result> <#bodies> COUNT($t) + COUNT($s) </>
          </>
```

The requested "body" can be matched and counted by a traditional method. If a product has no body content for the part element, as shown in the second part of the WHERE part, sub-elements is counted. If the outcome of the counting reaches the maximum requirement, then it can be assumed to exist. For example, in lines (15) - (18) of Figure 1, there is no value "body" but two values of the sub-elements. Finally, those two outcomes are added.

EXAMPLE 3.2: (XML-QL for EXAMPLE 2.2)

```
WHERE <product> <part> $y <part> $t </></>
      </> IN "www.a.b.e/XDB2",
           (CONTAINS $t, 'processor') or
     ((NOT (CONTAINS $t, 'processor')) and (EXISTS ($y)))
CONSTRUCT <result> <#processors> COUNT($t) + COUNT($y) </>
          </>
```

Suppose that there are XML documents which do not have the value `processor` for the super element `part`. For example, in line `(27)` of Figure 1, "body" has implicitly its component value. Because the super element is specified, its component elements `processor` should exist implicitly.

3.2 Expanded Aggregation Using Concept Tree

We again consider an aggregate function, α. As illustrated in the motivating examples, we need to take a few types of XML-elements into account. We do not consider mixed-content elements and empty-content elements because such elements have values explicitly bound to themselves and so they can be clearly evaluated by the conventional method. Instead, we consider in this paper text-content element types and element-content element types in addition to the new type which will be defined below, the *Virtual-content* element.

Definition 3.1: An XML-element may have one or more sub-elements if the value (term) bound to that element has one or more component terms according to a concept tree. A *virtual-content* element is defined as a sub-element such that it does not exist and its parent element is a text-content element.

Although a current XML-element is a text-content element, we may assume that that element is regarded as a mixed-content element if it contains virtual-content elements. The virtual-content element is also one of the element types we need to consider because it may be specified as the value to be aggregated. One example has been illustrated in EXAMPLE 1.2.

A concept tree represents synonyms and component hierarchies among the terms used in XML data. This subsection describes a method of making use of a concept tree. Consider an aggregation function α, is required for summary data. For each element type, we describe the naive and aggressive approaches. In the aggressive approach, there are two ways of expanding aggregation queries: (1) by using synonyms, and (2) by using component hierarchies.

- Aggregation of Text-content Element Types.

  1. Naive Approach. A conventional aggregation (counting in this case) method can be applied. For example, suppose that a query is expressed to build XML summary data: $\mathsf{COUNT}(<\texttt{p\_name}> =' Computer')$. In Figure 1, if the element `<p_name>` as in lines `(2)`, `(12)`, and `(25)` is a text-content type, a simple method of aggregation can be performed.
  2. Aggressive Approach using Synonyms.
     One may still want to expand the aggregation using synonyms. For a synonym B of the given term A, the expanded aggregation is: $\mathsf{COUNT}(\mathrm{p}_1 =' A')+ \mathsf{COUNT}(\mathrm{p}_2 =' B')$. Notice that "Computer" is synonymous with "PC." The expanded query can then be $\mathsf{COUNT}(<\texttt{p\_name}> =' Computer')+ \mathsf{COUNT}(<\texttt{p\_name}> =' PC')$.
  3. Aggressive Approach using Component Hierarchies. It does not apply.

- Aggregation of Element-content Element can be modified easily from the above

- Aggregation of Virtual-content Element.

  1. Naive Approach. None is retrieved to be aggregated.
  2. Aggressive Approach using Synonym. The expanded query is as follows if a synonym B is found in the concept tree: $\mathsf{COUNT}(p_1 =' B')$.
  3. Aggressive Approach using Component Hierarchies. Check the parent element to see if A is a component of the content of the parent element. The expanded query is simply 1.

Depth/Degree of Aggregation Expansion

The degree or the depth of the aggregation query expansion becomes an important issue. There are two degrees of the expansion in addition to non-expansion:

- Partial Expansion: Depending upon user preference or input, the expansion process terminates. If one time expansion is allowed, then the aggregation query using a synonym B can be $\mathsf{COUNT}(\mathrm{p}_1 =' A')+ \mathsf{COUNT}(\mathrm{p}_1 =' B')$, and the query using a component hierarchy can be n, where n is the number of those elements which are of element content type and also have all the terms in the sub-elements.

- Full Expansion: All possible synonyms and all component hierarchies are taken into account for query expansion.

  1. Expansion using Synonyms. For given term A, if there are the n number of terms, B's, that are synonymous, the counting aggregation query can be $\mathsf{COUNT}(\mathrm{p}_1 =' A') + \Sigma_{i=1}^{n} \mathsf{COUNT}_i(\mathrm{p}_1 =' B_i')$.

2. Expansion using Component Hierarchies. If an element is an element-content element, then components are examined. However, if not all components are available, and if there exists an element-content element, we need to check its sub-elements with component hierarchies. The aggregation query in full expansion is $\mathsf{COUNT}(p_1 =' B'_1 \wedge p_1 =' B'_2 \wedge ... \wedge p_1 =' B'_{k-1})$ if $(p_1 =' C'_1 \wedge p_1 =' C'_2 \wedge ... \wedge p_1 =' C'_{l-1})$ holds and if in turn $(p_1 =' D'_1 \wedge p_1 =' D'_2 \wedge ... \wedge p_1 =' D'_{m-1})$ holds and so on, where the term $A, B, C, D, ...$ has the k, l, m, n number of components, respectively.

Examples

Using the concept tree in Figure 2, a full expansion of the queries in EXAMPLE 2.1 and 2.2 can be as follows:

EXAMPLE 3.3: (XML-QL for EXAMPLE 2.1)

```
WHERE <product> <part> $t <part> $x </></>
      </> IN "www.a.b.e/XDB2",
          (CONTAINS $t, 'body') or (CONTAINS $t, 'unit')
or ((CONTAINS $x, 'processor') and (CONTAINS $x, 'board'))
CONSTRUCT <result> <#bodies> COUNT($t) + COUNT($s) </>
          </>
```

In Figure 2, lines (4) - (6) represent that "body" and "unit" are synonymous. The term "body" consists of "processor" and "board."

EXAMPLE 3.4: (XML-QL for EXAMPLE 2.2)

```
WHERE <product> <part> $y <part> $t </></>
      </> IN "www.a.b.e/XDB2",
              (CONTAINS $t, 'processor') or
       ((NOT (CONTAINS $t, 'processor')) and
       ((CONTAINS $y, 'body') or (CONTAINS $y, 'unit')))
CONSTRUCT <result> <#processors> COUNT($t)+COUNT($y)</>
          </>
```

4 CONCLUSION

This paper has described the "aggressive" approach to generate summary data for XML documents in WWW. The summary data contains aggregation information of XML documents. In this paper, we considered only a counting method by using XML Schema and the concept hierarchy. We constructed a concept tree in XML which can be easily integrated with user-provided queries. The proposed approach can be easily applied to other aggregation operations such as SUM, MAX, MIN, AVG. The contribution of this paper includes a new method of generating complete summary data about XML documents. Our extended work includes approximate aggregation of XML documents in WWW.

References

[1] XML Schema specification. Technical report, www.c3.org/TR/xmlschema/, 2000.

[2] S. Abiteboul, B. Amann, S. Cluet, A. Eyal, L. Mignet, and T. Milo. Active views for electronic commerce. In *Proc. Intl. Conf. on Very Large Data Bases*, 1999.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal of Digital Libraries*, 1(1):68–88, 1997.

[4] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Intl. Conf. on Data Engineering*, 1997.

[5] D. Barbara. Special issue on supporting on-line analytical processing. *IEEE Data Engineering*, 20:2–44, 1997.

[6] T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (XML) 1.0. *World Wide Web Consortium Recommendation. Available at* http://www.w3.org/TR/REC-xml, 2 1998.

[7] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 505–516, 1996.

[8] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *Intl. Conf. on Data Engineering*, 2000.

[9] P. Deshpande, J. Naughton, K. Ramasamy, A. Shukla, K. Tufte, and Y. Zhao. Cubing algorithms, storage estimation, and storage and processing alternatives for OLAP. *IEEE Data Engineering*, 20:3–11, 1997.

[10] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier, and D. Suciu. Querying XML data. *IEEE Data Engineering*, 22:10–18, 1999.

[11] C. Dyreson. Using an incomplete data cube as a summary data sieve. *IEEE Data Engineering*, (20):19–26, 1997.

[12] V. Harinarayan. Issues in interactive aggregation. *IEEE Data Engineering*, (20):12–18, 1997.

[13] R. Light and T. Bray. *Presenting XML*. Sams, Indianapolis, Indiana, 1997.

[14] J. McHugh and J. Widom. Query optimization for XML. In *Proc. Intl. Conf. on Very Large Data Bases*, pages 93–100, 1999.

[15] Ami Motro. FLEX: A tolerant and cooperative user interface to database. *IEEE Transactions on Knowledge and Data Engineering*, 2, 1990.

[16] I. Mumick, D. Quases, and B. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 100–111. 1997.

[17] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 455–466, 1999.

[18] J. Yoon, A. Hafez, and V. Raghavan. Query rewriting for multimedia XML data retrieval. In *the 6th Int'l Workshop on Multimedia Information Systems*, Chicago, IL, 2000.

[19] J. Yoon and S. Kim. A three-level user interface to multimedia digital libraries with relaxation and restriction,. In *IEEE Int'l Conference on Advance Digital Libraries*, pages 206–215, Santa Barbara, CA, 1998.