Group-by and Aggregate Functions in XML Keyword Search

Thuy Ngoc Le[1], Zhifeng Bao[2], Tok Wang Ling[1], and Gillian Dobbie[3]

[1]National University of Singapore,
[2]University of Tasmania & HITLab Australia, [3]University of Auckland
{ltngoc,lingtw}@comp.nus.edu.sg;
zhifeng.bao@utas.edu.au, gill@cs.auckland.ac.nz

Abstract. In this paper, we study how to support group-by and aggregate functions in XML keyword search. It goes beyond the simple keyword query, and raises several challenges including: (1) how to address the keyword ambiguity problem when interpreting a keyword query; (2) how to identify duplicated objects and relationships in order to guarantee the correctness of the results of aggregation functions; and (3) how to compute a keyword query with group-by and aggregate functions. We propose an approach to address the above challenges. As a result, our approach enables users to explore the data as much as possible with simple keyword queries. The experimental results on real datasets demonstrate that our approach can support keyword queries with group-by and aggregate functions which are not addressed by the LCA-based approaches while achieving a similar response time to that of LCA-based approaches.

1 Introduction

Like keyword search in Information Retrieval, its counterpart over XML data has grown from finding the matching semantics and retrieving basic matching results in the last decade [3, 14, 6, 8, 18, 11, 17], and now it is enabling many more opportunities for users to explore the data while keeping the query in the form of keywords with additional requirements, such as visualization, aggregation, query suggestion, etc. In this paper, we study how to support group-by and aggregate functions beyond the simple XML keyword search, which to our best knowledge, no such effort has been done yet. In this way, it alleviates users from learning complex structured query languages and the schema of the data. For example, consider the XML document in Figure 1, in which there exist two many-to-many relationship types between `Lecturer` and `Course`, and between `Course` and `Student`. Suppose a user wishes to know *the number of students registered for course Cloud*, ideally she can just pose a keyword query like {`Cloud, count student`}. It is even better if keyword queries can express *group-by* functions. For example, *finding the number of registered students for each course* can be expressed as {`group-by course, count student`}.

This motivates us to propose an approach for XML search which can support keyword queries with *group-by* and aggregate functions including *max, min, sum, avg, count* (referred to as expressive keyword query) by just using a keyword based interface. As a result, our approach is able to provide a powerful and easy way to use a query

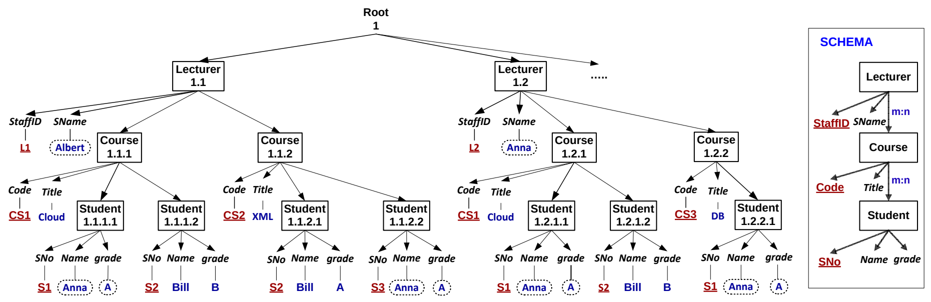


Figure 1: An XML database

interface that fulfills a need not addressed by existing systems. Group-by and aggregate functions are studied in XML structured queries such as [12, 2] and in keyword search over relational database (RDB) such as [10, 13]. However, to the best of our knowledge, there was no such work in XML keyword search.

Our approach has three challenges compared to the simple LCA-based approaches for XML keyword search (i.e., approaches based on the LCA (Lowest Common Ancestor) semantics) such as [3, 14, 6, 8, 18, 11, 17], which do not support group-by and aggregate functions. Firstly, query keywords are usually ambiguous with different interpretations. Thereby, a query usually has different interpretations. In simple XML keyword search, an answer can be found without considering which query interpretation it belongs to. On the contrary, in our approach, if all answers from different interpretations are mixed altogether, the results for group-by and aggregate functions will be incorrect. Secondly, an object and a relationship can be duplicated in an XML document because it can appear multiple times due to many-to-many relationships. Such duplication causes duplicated answers. Duplicated answers may overwhelm users but at least the answers are still correct for simple XML keyword search. In contrast, duplicated answers cause the wrong results for aggregate functions *count, sum, avg*. Thirdly, unlike simple XML keyword search where all query keywords are considered equally and answers can be returned independently, in our approach, query keywords are treated differently and the answers need to be returned in a way that the group-by and aggregate functions can be applied efficiently. Therefore, processing a keyword query with group-by and aggregate functions is another challenge.

To overcome these challenges, we exploit the ORA-semantics (Object-Relationship-Attribute-Semantics) introduced in our previous work [7, 5, 4]. The ORA-semantics includes the identification of nodes in XML data and schema. Once nodes in an XML document are defined with the ORA-semantics, we can identify interpretations of a keyword query. The ORA-semantics also helps determine many-to-many relationships to detect duplication.

Contributions. In brief, we propose an approach for XML keyword search which can support group-by and aggregate functions with the following contributions.

– Designing the syntax for an XML keyword query with group-by and aggregate functions (Section 2).

– Differentiating query interpretations due to keyword ambiguity in order not to mix together the results of all query interpretations (Section 3).
– Detecting duplication of objects and relationships to calculate aggregate functions correctly (Section 4).
– Processing XML keyword queries with group-by and aggregate functions including max, min, sum, avg, count efficiently (Section 5).
– Creating `XPower`, a system prototype for our approach. Experimental results on real datasets show that we can support most queries with group-by and aggregate functions which the existing LCA-based approaches cannot while achieving a similar response time to that of LCA-based approaches (Section 6).

2 Preliminary

2.1 Object-Relationship-Attribute (ORA)-semantics

We defined the ORA-semantics as the identifications of nodes in XML data and schema. In XML schema, an internal node can be classified as object class, explicit relationship type, composite attribute and grouping node; and a leaf node can be classified as object identifier (OID), object attribute and relationship attribute. In XML data, a node can be object node or non-object node. Readers can find more information about the ORA-semantics in our previous works [7, 5, 4].

For example, the ORA-semantics of the XML schema and data in Figure 1 includes:
– `Lecturer, Course` and `Student` are object classes.
– `StaffID, Code` and `SNo` are OIDs of the above object classes.
– `StaffID, SName`, etc are attributes of object class `Lecturer`.
– There are two many-to-many relationship types between object classes: between `Lecturer` and `Course`, and between `Course` and `Student`.
– `Grade` is an attribute of the relationship type between `Course` and `Student`.
– `Lecturer(1.1)` is an object node; `StaffID, L1, SName` and `Albert` are non-object nodes of `Lecturer(1.1)`.

Discovering ORA-semantics involves determining the identification of nodes in XML data and schema. If different matching nodes of a keyword have different identifications, then that keyword corresponds to different concepts of the ORA-semantics. In our previous work [7], the accuracy of discovery of the ORA-semantics in XML data and schema is high (e.g., greater than 99%, 93% and 95% for discovering object classes, OID and the overall process, respectively). Thus, for this paper, we assume the task of discovering the ORA-semantics has been done.

2.2 Expressive keyword query

This section describes the syntax of an expressive keyword query with group-by and aggregate functions including *max, min, count, sum, avg* supported by our approach. Intuitively, a group-by function is based on an object class or attribute. For example, `{group-by course}, {group-by grade}`. Thus, group-by must associate with an object class, or an attribute. On the other hand, aggregate functions *max, min, sum,* *avg* must associate with an attribute such as `max grade`, but not with an object class or a value because these functions are performed on the set of values of attributes. However, the aggregate function *count* can associate with all types of keyword: object class, attribute, and value because they all can be counted. For example, {`count course`}, {`count StaffID`}, {`count A`}. Based on these constraints, we define the syntax of an expressive keyword query in BackusNaur Form (BNF) as follows.

$\langle query \rangle ::= (\langle keyword \rangle[\text{``,''}])^*(\langle function \rangle[\text{``,''}])^*$
$\langle function \rangle ::= \langle group\_by\_fn \rangle \mid \langle aggregate\_fn \rangle$
$\langle group\_by\_fn \rangle ::= \text{``}group\_by\text{''}(\langle object\_class \rangle \mid \langle attribute \rangle)$
$\langle aggregate\_fn \rangle ::= \langle agg\_1 \rangle \mid \langle agg\_2 \rangle$
$\langle agg\_1 \rangle ::= (\text{``}max\text{''} \mid \text{``}min\text{''} \mid \text{``}sum\text{''} \mid \text{``}avg\text{''})(\langle attribute \rangle | \langle aggregate\_fn \rangle)$
$\langle agg\_2 \rangle ::= \text{``}count\text{''}(\langle keyword \rangle | \langle aggregate\_fn \rangle)$
$\langle keyword \rangle ::= \langle object\_class \rangle \mid \langle attribute \rangle \mid \langle value \rangle$

Since the parameters of aggregate function “count” are different from those of the other aggregate functions “max”, “min”, “sum”, “avg”, we use $\langle agg\_1 \rangle$ and $\langle agg\_2 \rangle$ to define them separately. With the above BNF, group-by and aggregate functions can be combined such as {`group-by lecturer, group-by course, max grade, min grade`} or nested such as {`group-by lecturer, max count student`}.

Terms related to an expressive keyword query. For ease of presentation, we use the following terms in this paper.

– *Group-by parameters* are query keywords following the term “group-by”.
– *Aggregate functions* are the terms “max”, “min”, “sum”, “avg” and “count”.
– *Aggregate function parameters* are keywords following the aggregate functions.
– *Content keywords* are all query keywords except reserved words (i.e., the term “group-by” and the aggregate functions). Content keywords can be values, attributes, or object classes in the query.
– *Free keywords* are content keywords not as group-by parameter and not as aggregate function parameters.

3 Query interpretation

The first challenge of our approach is that keywords are usually ambiguous with different interpretations as illustrated in Figure 2. Therefore, a query is also ambiguous with different interpretations, each of which corresponds to a way we choose the interpretation of keywords as described in the following concept.

Concept 1 (Query interpretation) *Given an expressive keyword query $Q = \{k_1, \ldots, k_n\}$, an interpretation of query Q is $\mathcal{I}_Q = \{i_1, \ldots, i_n\}$, where i_i is an interpretation of k_i.*

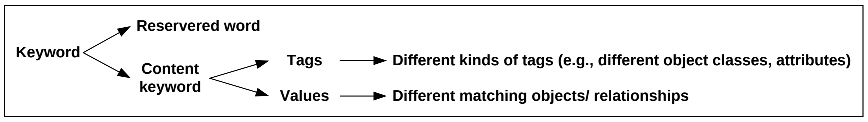

Reservered word
Keyword
Content keyword
Tags
Different kinds of tags (e.g., different object classes, attributes)
Values
Different matching objects/ relationships

Figure 2: Different possible interpretations of a keyword

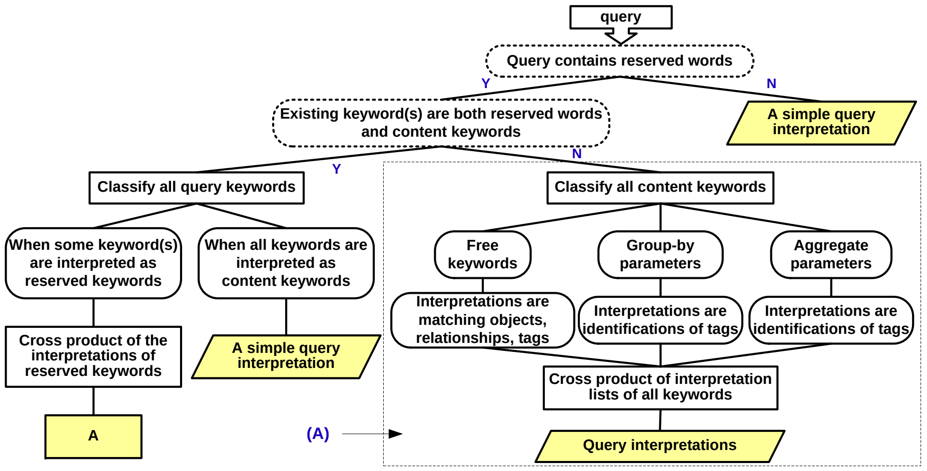

query
Query contains reserved words
Y
N
Existing keyword(s) are both reserved words and content keywords
A simple query interpretation
Y
N
Classify all query keywords
Classify all content keywords
When some keyword(s) are interpreted as reserved keywords
When all keywords are interpreted as content keywords
Free keywords
Group-by parameters
Aggregate parameters
Interpretations are matching objects, relationships, tags
Interpretations are identifications of tags
Interpretations are identifications of tags
Cross product of the interpretations of reserved keywords
A simple query interpretation
Cross product of interpretation lists of all keywords
A
(A)
Query interpretations

Figure 3: Generating query interpretations

3.1 Impact of query ambiguity on the correctness of the results

In simple XML keyword search approaches such as [3, 14, 6, 8, 18, 11, 17], which do not support group-by and aggregate functions, an ambiguous keyword corresponds to a set of matching nodes, whose identifications are not considered. In these approaches, an answer can be found without considering which query interpretation it belongs to. In contrast, not differentiating query interpretations affects the correctness of group-by and aggregate functions as illustrated below.

Example 1 *Consider query* {`Anna, count A`} *issued to the XML data in Figure 1, in which one lecturer and two students have the same name* `Anna`*. They are object* `<Lecturer:L2>` *(w.r.t. object node* `Lecturer (1.2)`*), object* `<Student:S1>` *(w.r.t. object nodes* `Student (1.1.1.1)`*,* `Student (1.2.1.1)`*, and* `Student (1.2.2.1)`*), and object* `<Student:S3>` *(w.r.t. object node* `Student (1.1.2.2)`*). Both keywords* `count` *and* `A` *have only one interpretation in the XML data. Specifically,* `count` *is an aggregate function and* `A` *is a value of attribute* `grade` *of the relationship type between* `student` *and* `course`*.*

Intuitively, the query has three interpretations: (1) finding the number of grade `A` *of students taking courses taught by* `Lecturer Anna`*, (2) finding the number of grade* `A` *of* `Student Anna` *whose SNo is* `S1`*, and (3) similar to the second interpretation but for student* `Student Anna` *whose SNo is* `S3`*. If the query interpretations are not considered, the numbers of grade* `A` *corresponding to three interpretations are mixed and counted altogether instead of being counted separately. This makes the results of aggregate function* `Count A` *incorrect.*

Therefore, to calculate group-by and aggregate functions correctly, we need to process each query interpretation separately. To speed up the processing, we have an optimized technique, in which we do not process each query interpretation at the beginning, instead we process them together with group-by functions (discussed in Section 5.2).

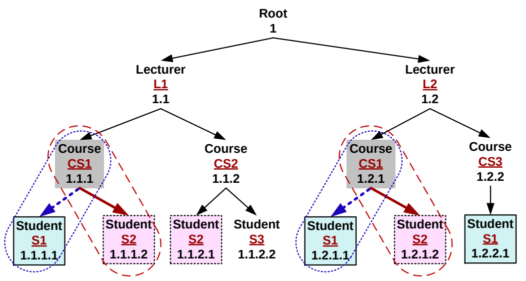
3.2 Generating query interpretations

Generating all interpretations of a query contains three tasks: (1) identifying all interpretations of each keyword; (2) once the interpretation lists of all keywords are available, query interpretations are generated by computing the cross product of these lists; (3) filtering out invalid query interpretations based on the syntax in Section 2.2. Instead of doing the three tasks separately, in the first task, we proactively identify the interpretations of a keyword such that they do not form invalid query interpretations.

Algorithm 1: Generating all valid interpretations of a query

```
Input: Query keywords $k_1, \ldots, k_n$
       The ORA-semantics
       The keyword-node lists and the node-object list
Output: List of valid interpretations $qInt$
Variable: List of query keywords as reserved words $L_{res}$
          A simple query interpretation $I_s^q$ without reserved words
//Task 1: identifying all interpretations of each keyword
for each query keyword $k$ do
    if $k$ is a reserved word then
        Add $k$ to $L_{res}$
//Query does not contain reserved words
if $L_{res}$ is empty then
    // It is a simple query without group-by or aggregate functions
    Add all keywords to $I_s^q$ // All are content keywords
    return $I_s^q$ // We do not care about interpretations for this case
//Query contains reserved words
Variable: The list of interpretations of a keyword $k$: $L_k^i$
          The list of free keywords $L_{fr}$
          The list of group-by parameters $L_g$
          The list of aggregate functions and parameters $L_{a,f}$
for each query keyword $k$ in $L_{res}$ do
    $L_k^i \leftarrow$ content interpretation and reserved word interpretation
    $k.tags \leftarrow$ retrieve all identifications of $k_i$ ($k_i$ as tags) from ORA-semantics
    $k.nodes \leftarrow$ retrieve all matching object nodes of $k_i$ ($k_i$ as values) from the keyword-node lists
$temp \leftarrow$ all cross product of the interpretation lists of keywords in $L_{res}$
for each interpretation $qI$ in $temp$ do
    // All keywords are content keywords
    if All interpretations in $qI$ are content interpretations then
        Add all query keywords to $I_s^q$
        Add $I_s^q$ to $qInt$ // a simple query interpretation
    else
        $L_{fr}$, $L_g$ and $L_{a,f} \leftarrow$ parse all keywords based on $L_{res}$
        for each keyword $k$ in $L_{fr}$ do
            $k.objects \leftarrow$ get objects of nodes for $k.nodes$ based on the node-object list
            Add all $k.tags$ and $k.objects$ to $L_k^i$
        for each keyword $k$ in $L_g$ do
            Add all $k.tags$ to $L_k^i$
        for each keyword $k$ in $L_{a,f}$ do
            if the aggregate function is "count" then
                $k.tags \leftarrow$ get tags of nodes for $k.nodes$ based on the ORA-semantics
            Add all $k.tags$ to $L_k^i$
//Task 2: generating all query interpretations
$qInt \leftarrow$ add all cross product of the interpretation lists $L_k^i$'s of all keywords
```

Identifying all interpretations of a keyword is not straightforward. Firstly, unlike a simple keyword query where interpretations of a keyword do not depend on those of the others, in our approach, to avoid invalid query interpretations, the interpretations of



Object	Duplication
<Course:CS1>	Course (1.1.1), Course(1.2.1)
<Student:S1>	Student (1.1.1.1), Student (1.2.1.1), Student (1.2.2.1)
<Student:S2>	Student (1.1.1.2), Student (1.2.2.1), Student (1.2.1.2)

Relationship	Duplication
{<Course:CS1>, <Student:S1>}	{Course (1.1.1), Student (1.1.1.1)}, {Course (1.2.1), Student (1.2.1.1)}
{<Course:CS1>, <Student:S2>}	{Course (1.1.1), Student (1.1.1.2)}, {Course (1.2.1), Student (1.2.1.2)}

Figure 4: Duplication of objects and relationships in the XML data in Figure 1

content keywords need to depend on those of the reserved words. Secondly, unlike simple keyword query where all keywords are treated equally, in our approach, different types of query keywords are treated differently and can provide different interpretations. Particularly, keywords as group-by and aggregate function parameters can only be interpreted as different tags, while keywords as free keywords can be interpreted as different matching objects[1], different matching relationships[2], or different tags as well.

Generating all valid query interpretations is illustrated in Figure 3 and is described in Algorithm 1. Since the reserved words may impact the interpretations of other keywords, we first need to identify whether a keyword is a reserved word, or a content keyword, or both. After that, the interpretations of content keywords depend on whether they are free keywords, group-by parameters or aggregate function parameters.

4 Duplication

Duplication of objects and relationships is another challenge of our approach. This section will discuss the impact of such duplication on the correctness of the results of aggregate functions and how to overcome it. For illustration, we use the XML data in Figure 1, in which duplicated objects and relationships are summarized in Figure 4, where we only show object nodes of the data for simplicity.

4.1 Duplicated objects and relationships

The duplication of objects is due to many-to-many $(m : n)$ or many-to-one $(m : 1)$ relationships because in such relationship, the child object is duplicated each time it occurs in the relationship. For example, in the XML data in Figure 1, since the relationship between lecturer and course is $m : n$, a course can be taught by many lecturers such as `<Course:CS1>` are taught by `<Lecturer:L1>` and `<Lecturer:L2>`. Thus, the child object (`<Course: CS1>`) is shown as two object nodes `Course (1.1.1)` and `Course(1.2.1)`.

$m : n$ and $m : 1$ relationships cause not only duplicated objects, but also *duplicated relationships*. For example, as discussed above, in the XML data in Figure 1, because of

[1] An object matches keyword k when any of its object node matches k.
[2] A relationship matches keyword k when any of its involved objects matches k.

the $m : n$ relationship between lecturer and course, the child object (`<Course:CS1>`) is shown as two object nodes `Course (1.1.1)` and `Course(1.2.1)`. Therefore, everything below these two object nodes is the same (duplicated), including the relationships between object (`<Course:CS1>`) and students such as the relationships between `<Course:CS1>` and `<Student:S1>` (the big dotted lines in Figure 4), and between `<Course:CS1>` and `<Student:S2>` (the big lines in Figure 4).

4.2 Impact of duplication on aggregate functions

We show the impact of duplicated objects and relationships in the following examples.

Example 2 (Impacts of duplicated objects) *To count the number of students taught by lecturer Albert, a user can issue a query* {`Albert, count student`} *against the XML data in Figure 1. Without considering duplicated objects, the number of students is four. However, object node* `Student (1.1.1.2)` *and object node* `Student (1.1.2.1)` *refer to the same object* `<Student:S2>`*. Hence, only three students are taught by lecturer* `Albert`*.*

Example 3 (Impacts of duplicated relationships) *Recall query* {`Anna, count A`} *discussed in Example 1. We use the second interpretation, i.e., finding the number of grade* `A` *of* `Student Anna` *whose SNo is* `S1`*, to illustrate the impacts of duplicated relationships. In the XML data in Figure 1, the relationship between* `<Student:S1>` *and* `<Course:CS1>` *is duplicated twice (the big dotted lines in Figure 4). This makes attribute* `grade` *of this relationship duplicated. Without considering duplicated relationships, the number of grade* `A` *is three. In contrast, by keeping only one instance for each relationship, the answer is only two.*

Therefore, to perform aggregate functions *sum, avg* and *count* correctly, we must detect duplicated objects and relationships and keep only one instance for each of them. Duplication does not impact on the correctness of aggregate functions *max* and *min*.

4.3 Detecting duplication

If there exists a $m : n$ or $m : 1$ relationship type between object classes A and B, then for all object classes (or relationship types) appearing as B or the descendants of B, the objects of those classes (or the relationships of those relationship types) may have duplication. Otherwise, with no $m : n$ or $m : 1$ relationship type, duplication does not happen. Therefore, to detect duplication, we first identify the possibility of duplication by checking $m : n$ and $m : 1$ relationship types. If there is no $m : n$ and no $m : 1$ relationship type, we can determine quickly the objects (or relationships) which are not duplicated. Identifying the possibility of duplication can be done with the ORA-semantics and is shown in Algorithm 2.

Once we determine that an object (or a relationship) is possibly duplicated, we determine whether two objects (of the same object class) are really duplicated by checking whether they have the same OID. A relationship is represented by a list of involved objects. Thus, two relationships (of the same relationship type) are duplicates if the two sets of objects involved by the two relationships are the same.

Algorithm 2: Detecting the possibility of duplication

```
Input: The aggregate function parameter $p$
       The ORA-semantics
1 $p.class$ ← get the object class of $p$ (based on the ORA-semantics)
2 $p.ancestor$ ← get all ancestor object classes and itself of $p.class$ (based on the ORA-semantics)
3 $p.RelType$ ← get all relationship types with $p.ancestor$ involved in (based on the ORA-semantics)
4 $p.card$ ← get the cardinality of each relationship type in $p.RelType$ (based on the ORA-semantics)
5 if existing $m : n$ or $m : 1$ relationship type in $p.RelType$ (determined in $p.card$) then
6     $p.possibility$ ← TRUE
```

Detecting real duplication is integrated with calculating aggregate functions and will be described in Section 5.2. For each aggregate function parameter, if it is possibly duplicated, before applying an aggregate function on an object or a relationship instance, we check whether duplication occurs.

5 Indexing and processing

Figure 5 describes the architecture of our approach, which consists of three indexes (presented in Section 5.1) and five processing components (discussed in Section 5.2). Like LCA-based approaches such as [3, 14, 6, 8, 18, 11, 17], our approach works on data-centric XML documents with no IDREFs and assumes updating does not frequently happen. Otherwise, adding or deleting one node can lead to change Dewey labels of all nodes in an XML document in those approaches.

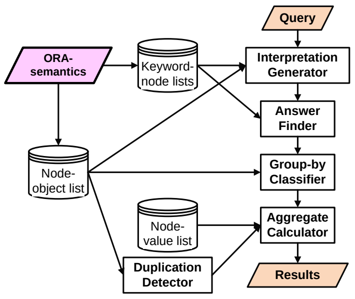


Figure 5: The architecture

5.1 Labeling and indexing

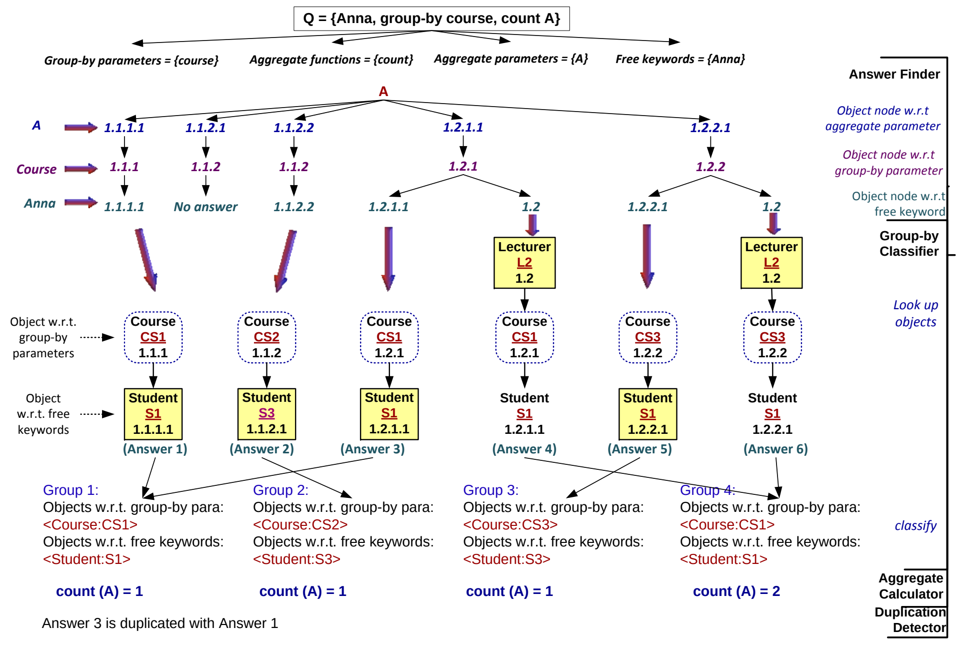
Labeling. Unlike conventional labeling schemes, where each node has a distinct label, we assign a *Dewey* label for only object nodes while non-object nodes use the same label with the object node they belong to as in Figure 1.

Indexing

A. Keyword-node lists. Each document keyword k has a list of matching object nodes.

B. Node-value list. We maintain a list of pairs of $\langle object\ node, values\rangle$ to retrieve values of an object node to operate group-by or aggregate functions.

C. Node-object list. We maintain a list of pairs of $\langle object\ node, object\rangle$ for two purposes. Firstly, it is used together with the keyword-node lists to find matching objects in order to generate query interpretations. Secondly, it is used to detect duplication.



Q = {Anna, group-by course, count A}
Group-by parameters = {course}
Aggregate functions = {count}
Aggregate parameters = {A}
Free keywords = {Anna}
Answer Finder
Object node w.r.t aggregate parameter
Object node w.r.t group-by parameter
Object node w.r.t free keyword
Group-by Classifier
Look up objects
classify
Aggregate Calculator
Duplication Detector
A
Course
Anna
1.1.1.1
1.1.2.1
1.1.2.2
1.2.1.1
1.2.2.1
1.1.1
1.1.2
1.2.1
1.2.2
No answer
1.2
Lecturer L2 1.2
Object w.r.t. group-by parameters
Object w.r.t. free keywords
Course CS1 1.1.1
Course CS2 1.1.2
Course CS1 1.2.1
Course CS3 1.2.2
Student S1 1.1.1.1
Student S3 1.1.2.1
Student S1 1.2.1.1
Student S1 1.2.2.1
(Answer 1)
(Answer 2)
(Answer 3)
(Answer 4)
(Answer 5)
(Answer 6)
Group 1: Objects w.r.t. group-by para: <Course:CS1> Objects w.r.t. free keywords: <Student:S1>
Group 2: Objects w.r.t. group-by para: <Course:CS2> Objects w.r.t. free keywords: <Student:S3>
Group 3: Objects w.r.t. group-by para: <Course:CS3> Objects w.r.t. free keywords: <Student:S3>
Group 4: Objects w.r.t. group-by para: <Course:CS1> Objects w.r.t. free keywords: <Student:S1>
count (A) = 1
count (A) = 1
count (A) = 1
count (A) = 2
Answer 3 is duplicated with Answer 1

Figure 6: Processing query Q = {`Anna, group-by course, count A`}

5.2 Processing

The processing of our approach can briefly be described as follows. As discussed in Section 3, we process each query interpretation separately in order not to mix together results of different query interpretations. *Interpretation Generator* is responsible for generating all valid interpretations of the input query. For each query interpretation, *Answer Finder* finds intermediate answers, each of which will be used as an operand of the aggregate function. *Group-by Classifier* classifies intermediate answers based on group-by parameters. *Aggregate Calculator* applies aggregate functions on intermediate answers. During the aggregate calculation, *Duplication Detector* detects *duplicated objects and relationships* in order to perform aggregate calculation correctly.

The above discussion is at conceptual level. At the lower level, we have an optimized technique for the process. We are aware that different query interpretations may produce the same sets of object nodes for group-by and aggregate function parameters. Therefore, instead of finding intermediate answers for each interpretation, we find all possible intermediate answers first, then classify them into suitable interpretations. Thereby, we can save costs of repeating the same thing. The optimized process is described in Algorithm 3[3].

[3] A query interpretation corresponds to a set of interpretations of free keywords, group-by parameters and aggregate function parameters. The input of Algorithm 3 is a query interpretation w.r.t. interpretations of group-by and aggregate function parameters, interpretations of free keywords will be handled after finding intermediate answers.

Interpretation Generator and *Duplication Detection* correspond to the discussion in Section 3 and Section 4 respectively. The following are details of the remaining three components. We use query {`Anna, group-by course, count A`} applied to the XML data in Figure 1 as a running example. Figure 6 shows the results produced by each component for the query.

Answer Finder. An intermediate answer contains a set of matching object nodes of content keywords which are aggregate function parameters, group-by parameters, and free keywords. We first find object nodes of aggregate function parameters (line 3 in Algorithm 3), then those of group-by parameters (line 6), and finally those of free keywords (line 11). This is because an intermediate answer corresponds to only one matching object node of an aggregate function parameter because it is used as only one operand in an aggregate function.

An object node of a *group-by parameter* must have an ancestor-descendant relationship with that of the aggregate function parameter because group-by functions are based on the relationship among objects. In XML, these relationships are represented through ancestor-descendant relationships (edges).

All nodes in an intermediate answer must be meaningfully related. For this purpose, we agree with the argument in [15] that the object class of the LCA (Lowest Common Ancestor) of nodes in an answer must belong to the LCA of the object classes of these nodes. We use this property to find object nodes of *free keywords* in an answer.

In the running example (in Figure 6), the aggregate function parameter `A` has five matching nodes. For each of them, we find the corresponding object nodes of the group-by parameter `course`. For each pair of object nodes of the aggregate function parameter and the group-by parameter, we find the corresponding object nodes of the free keyword `Anna`.

Scope. Like LCA-based approaches, our approach does not consider the cases where two objects are connected through several relationships which do not form an ancestor-descendant chain. This constraint is to find group-by parameter.

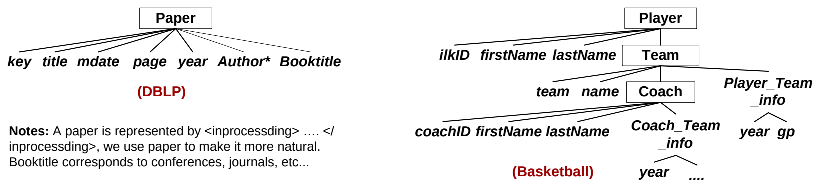
Group-by Classifier. We classify an intermediate answer into a group based on objects w.r.t. the group-by parameters and the free keywords. Thus, we first retrieve the corresponding objects from the set of nodes of an answer. Then, we compare that set of objects of an answer with that of groups (lines 14-19). In the running example, for the first answer, `<Course:CS1>` and `<Student:S1>` are objects corresponding to the group-by parameter and the free keyword respectively. So, it is classified into Group 1 which has the same set of objects.

Aggregate Calculator. Aggregate Calculator calculates aggregate functions on objects and relationships (or their attributes and values) of the aggregate function parameters in each group. During the calculation, we check the duplication of objects and relationships. We first check the possibility of duplication based on schema (line 25, referred to Algorithm 2). If this possibility is true, we will check whether the considered object or relationship is duplicated or not (line 26) before processing the aggregate functions (line 27). In the running example, two answers in Group 1 are duplicated. Thus, the result of `Count A` in this group is only one.

Algorithm 3: Processing an expressive keyword query

```
Input: Free keyword $k_1, \ldots, k_n$
The group-by parameters $g_1, \ldots, g_q$
The aggregate function parameters $a_1, \ldots, a_p$
The aggregate functions $f_1, \ldots, f_p$
The ORA-semantics
Indexes: the keyword-node lists, the node-value list, the node-object list
//Answer Finder
for each aggregate function parameter $a_i$ do
    for each matching object node $a\_node \in matchNode(a_i)$ do
        //find the corresponding object nodes matching group-by parameters
        for each group-by parameter $g_i$ do
            $L_{g_i\_node} \leftarrow$ the list of object nodes $u$ such that $u \in matchNode(g_i)$ and $u$ is an ancestor or
            a descendant of $a\_node$
        for each set of object nodes $\{g_1\_node, \ldots, g_q\_node\}$, $g_i\_node \in L_{g_i\_node}$ do
            $highest \leftarrow$ the highest object node among $g_i\_node$'s and $a\_node$
            //find the corresponding object nodes matching free keywords
            for each free keyword $k_i$ do
                $L_{k_i\_node} \leftarrow$ findObjectNodesFreeKeyword($k_i$, $highest$)
            //Group-by Classifier
            $L_{group} \leftarrow \{group|\ group \leftarrow \{g_1\_node, .., g_q\_node, k_1\_node, .., k_n\_node\}$, where
            $k_i\_node \in L_{k_i\_node}\}$
            for each $group \in L_{group}$ do
                $unit \leftarrow \{a\_node, group\}$
                if exist $group_i$ matches $group$ then
                    classify $unit$ into $group_i$ //Classify $unit$
                else
                    create new class $group_{i+1}$
                    Updated objects w.r.t. group-by parameters and free keywords for $group_{i+1}$
                    classify $unit$ into $group_{i+1}$ //Classify $unit$ into new class
                //Aggregate Calculator
                if $a_i$ is a relationship or a relationship attribute (based on the ORA-semantics) then
                    $rel(a\_node) \leftarrow$ get the relationship w.r.t. $a\_node$
                else
                    $obj(a\_node) \leftarrow$ get the object w.r.t. $a\_node$
                if $a_i.possibility$ is TRUE then
                    if DetectDuplication ($obj(a\_node)$/$rel(a\_node)$) is FALSE then
                        ApplyAggregateFunction ($f_i$, $a_i$, $a\_node$)
```

Complexity. Since the number of aggregate function parameters, group-by parameters and free keywords are few, the *For* loops in line 1, line 4 and line 9 do not have much affect on the complexity. Since all the lists of matching object nodes are sorted by the pre-order of labels of matching object nodes, the finding of matching object nodes of group-by parameters and free keywords can be obtained efficiently. Thereby, the complexity of finding all intermediate answers depends on the number of matching object nodes of aggregate function parameters. Thus, for Answer Finder, for each matching object of an aggregate function parameter, the costs are $\log(G)$ and $\log(K)$ for finding object nodes of a group-by parameter and of a free keyword in an answer respectively, where G and K are the length of their lists of matching object nodes respectively. In Group-by Classifier, the cost for classifying in the worst case is $\log(Gr)$ where Gr is the total number of groups (sorted). For Aggregate Calculator, the cost in the worst case is $O \times \log(O)$ where O is the maximum number of objects in a group.



Paper
key title mdate page year Author* Booktitle
(DBLP)
Notes: A paper is represented by <inprocessding> …. </inprocessding>, we use paper to make it more natural. Booktitle corresponds to conferences, journals, etc...
Player
ilkID firstName lastName
Team
team name
Coach
Player_Team _info
year gp
coachID firstName lastName
Coach_Team _info
year ....
(Basketball)

Figure 7: A part of schema of DBLP and Basketball used in experiments

Table 1: Queries for tested datasets

DBLP	User search intention	Query	XPower suports	Ambi-guity	Dupli-cation
QD1	Count the papers of Yi Chen	Yi Chen, count paper	Yes	Yes	No
QD2	Count the co-authors of Yi Chen	Yi Chen, count author	No	N.A	N.A
QD3	How many years Yi Chen have published papers and in how many conferences	Yi Chen, count booktitle, count year	Yes	Yes	Yes
QD4	Count the papers of Yi Chen in each conference for each year	Yi Chen, group-by year, group-by booktitle, count paper	Yes	Yes	Yes
QD5	How many conferences has Diamond published papers	Diamond, count booktitle	Yes	Yes	Yes
QD6	Find the latest year Diamond published paper in IEEE-TIT	Diamond, IEEE-TIT, max year	Yes	Yes	No
QD7	Count the papers of Brown published in each conference	Brown, group-by booktitle, count paper	Yes	Yes	No
QD8	Count the conferences Brown has published papers	Brown, count booktitle	Yes	Yes	Yes
Basketball					
QB1	How many players and coaches in team Celtics	Celtics, count player, count coach	Yes	No	Yes
QB2	How many teams Michael have worked for	Michael, count team	Yes	Yes	Yes
QB3	How many players Thomas have worked with	Thomas, count player	No	N.A	N.A
QB4	How many players Johnson have worked with	Johnson, count player	Partial	Yes	Yes
QB5	Find the latest year Edwards has worked for team Hawks	Edwards, Hawks, max year	Yes	Yes	No
QB6	When did player Edwards start to work	Edwards, min year	Yes	Yes	No
QB7	Count players in each team which Michael has worked for	Michael, group-by team, count player	Partial	Yes	No
QB8	How many players and coaches of each team	group-by team, count player, count coach	Yes	No	Yes

6 Experiment

We have implemented a framework, called `XPower` for evaluation on several aspects: enhancement, impacts of query interpretation and duplication, and efficiency. We compare `XPower` with `XKSearch` [14] because it is one of the most popular XML keyword search approaches. Like other XML keyword search approaches, `XKSearch` does not support group-by and aggregate functions, and does not consider the effects of duplication and query interpretation on search results. Thus, we compare with `XKSearch` on only efficiency. The experiments were performed on an Intel(R) Core(TM)i7 CPU 3.4GHz with 8GB of RAM. We used the subsets of two real datasets: Basketball (45MB)[4] and DBLP (570MB)[5,6]. A part of the schema of each dataset is given in Figure 7.

[4] http://www.databasebasketball.com/

[5] http://dblp.uni-trier.de/xml/

[6] In the updated DBLP dataset, authors of the same name can be distinguished. We make use this in differentiating query interpretations related to authors.

Table 2: Interpretations of keywords in tested queries

	DBLP			Basketball		
	Keyword	**Interpretations of keyword**		**Keyword**	**Interpretations of keyword**	
value	Yi Chen	6 authors	Match values only	Celtics	1 team	No keyword in multiple groups: value, tag, reserved word
	Diamon	many authors		Hawks	1 team	
		many titles		Edwards	4 players	
	Brown	many authors		Thomas	15 players	
		many titles		Michael	2 coaches and 13 players	
	IEEE-TIT	one conference		Johnson	5 coaches and 14 players	
tag	author	object class/ attribute	Match both tags and values of Title (in class Paper)	team	object class	
	paper	object class		coach	object class	
	year	attribute (in class paper)		player	object class	
	booktitle	object class/ attribute		year	relationship attribute	
reserved word	group-by	not in DBLP		group-by	not in document	
	count	in title of paper	both reserved words and values	count	not in document	
	max	in title and author		max, min	not in document	

6.1 Enhancement evaluation

Table 1 shows eight queries for each dataset used in the experiments and Table 2 provides interpretations of keywords in those queries. Table 1 also shows whether `XPower` was accurate on the tested queries. As can be seen, `XPower` can return answers for seven out of eight queries for each Basketball and DBLP dataset. This is because `XPower` handles group-by functions based on relationships, which are represented as edges in XML. However, in QD2, `Yi Chen` is an author, and it does not have any direct relationship with another author. In other words, there is no ancestor-descendant relationship between authors. Therefore, `XPower` cannot provide any answer for this query. This is similar to QB3 of Basketball. For QB4 and QB7 in Basketball, Michael and Johnson can be both players and coaches. If they are interpreted as players, `XPower` cannot provide an answer for the same reason as QB3. If they are interpreted as coaches, `XPower` can provide answers.

6.2 Impact of query interpretation due to keyword ambiguity

Table 3 shows three different results for each query in Basketball in three different scenarios: (1) `XPower` considering both query interpretation and duplication, (2) only considering query interpretation but not duplication, and (3) only considering duplication but not differentiating query interpretation. Because of space constraints, we only showed the results and explanations for Basketball although DBLP has similar results. We also describe whether duplication and keyword ambiguity impact on the results of each query in Table 1. As can be seen, keyword ambiguity impacts on the correctness of the results of all queries in DBLP, and five out of seven queries in Basketball (not considering queries with no answer). This verifies the importance of differentiating query interpretations. Otherwise, the results of all query interpretations are mixed together.

6.3 Impact of duplication

As we can see in Table 1, duplication impacts on the correctness of the results for four out of eight queries in both DBLP and Basketball. This is fewer than those affected by

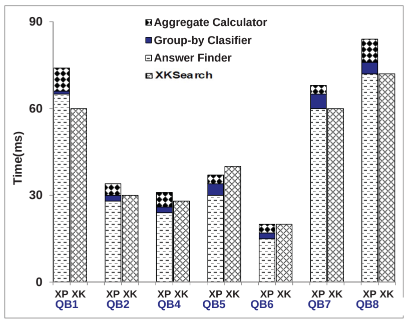
Table 3: Results of queries of Baketball dataset

	XPower results	**Results if not filter duplication**	**Reasons for duplication**	**Results if not differentiate queryInterpretation**	**Explain**
QB1	count player = **215** count coach = 13	count player = **2795** count coach = 13	Team Celtics has been coached by 13 coaches, thus its players are duplicated 13 times. Coaches are not duplicated.	same results with XPower	
QB2	**15 answers** for 15 persons (2 coaches, 13 players), each has a number of teams they have worked for. Sum of these numbers are **69.**	count team = **298**	Michael as players: a player can work with the same team (duplicated) under different coaches.	**1 answer:** count team = 69	mix the results of all interpre-tations
QB4	No answer for Johnson as players. Johnson as coaches: **5 answers** for 5 coaches, each has a number of players. Total number is **136.**	count player = **219**	A player can works for more than 1 team (duplicated) in different years under the same coach Johnson	**1 answer:** count player = 136	
QB5	**2 answers** for 2 players Edwards in team Hawks, with max year 1997, 2004 resp.	same results with XPower	duplication does not affect aggregate function max	**1 answer:** max year = 2004	
QB6	**4 answers** w.r.t. min year for 4 players Edwards: 1993, 1995, 1981, 1977 resp.	same results with XPower	duplication does not affect aggregate function min	**1 answer:** min year = 1977	
QB7	**6 answers:** Michael as players: No answer. Michael as coach 1: 3 teams (count players = 153, 256, 82 resp.) Michael as coach 2: 3 teams (count	same results with XPower	Although players are duplicated in documents, they are not duplicated under the pair of 1 coach and 1 team	**4 answers:** 4 teams (count player = 153, 236, 512, 164 resp.).	
QB8	Provide the number of players and those of coaches for each team	count player: diff count team: same	If a team is duplicated, all of its players are duplicated.	same results with XPower	

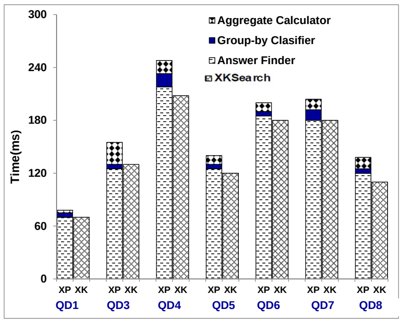
ambiguity but this number is still significant. This agrees our arguments about the importance of detecting duplication. Otherwise, the results of aggregate functions would not be correct. The number of queries affected by duplication is fewer than that of ambiguity because in Basketball, `coaches` are not duplicated, only `teams` and `players` can be duplicated. In DBLP, `papers` are not duplicated either. Therefore, there is no impact on the functions `count coach` in Basketball and `count paper` in DBLP. Moreover, duplication does not affect *max* and *min* functions as in QB5 and QB6.

6.4 Efficiency Evaluation

Figure 8 shows the response time of `XPower` (XP as abbreviation) and `XKSearch` (XK as abbreviation) for queries tested except the ones (QB3 and QD2) `XPower` does not provide any answer. Since `XKSearch` does not support group-by and aggregate functions, we dropped reserved words of tested queries when running `XKSearch`. Although `XPower` has the overhead of doing group-by and aggregate functions, the response time of queries are similar to those of `XKSearch`. This is because `XPower` does not find all SLCAs because many SLCAs do not correspond to any intermediate answer. For queries with complicated group-by and aggregate functions (e.g., QB1, QB7, QB8, QD3, QD4 and QD7), the overhead of processing those functions makes `XPower` run slightly slower than `XKSearch`. The response time of `XPower` is dominated by that of Answer Finder. Aggregate Calculator costs more than Group-by Classifier because it needs to detect duplication.



(a) Basketball



(b) DBLP

Figure 8: Efficiency comparison of `XPower` and `XKSearch` on Basketball and DBLP (Dropping reversed words of tested queries when running `XKSearch`)

7 Related work

XML keyword search. Most approaches for XML keyword search are based on the concept of LCA (Lowest Common Ancestor) first proposed in XRANK [3]. Later, many approaches extend the concept of LCA to filter less relevant answers. XKSearch [14] defines Smallest LCAs (SLCAs) to be the LCAs that do not contain other LCAs. Meaningful LCA (MLCA) [8] incorporates SLCA into XQuery. VLCA [6] and ELCA [18] introduces the concept of valuable/exclusive LCA to improve the effectiveness of SLCA. XReal [1] proposes an IR-style approach for ranking results. MaxMatch [9] investigates an axiomatic framework that includes the properties of monotonicity and consistency. MESSIAH [11] handles the cases of missing values in optional attributes. Although these works can improve effectiveness of the search, they do not support group-by and aggregate functions.
Group-by and aggregate functions. Group-by and aggregate functions are studied in XML structured queries such as [12, 2] and in keyword search over relational database (RDB) such as [10, 13]. However, there is no such work in XML keyword search. Arguably, XML can be shredded into RDB, and then we can apply the techniques of RDB for XML. However, [16] proves that the relational approaches are not as efficient as the native approaches (XML is used directly) in most cases.

8 Conclusion and future work

We proposed an approach to support queries with group-by and aggregate functions including sum, max, min, avg, count to query a data-centric XML document with a simple keyword interface. We processed query interpretations separately in order not to mix together the results of different query interpretations. To perform aggregate functions correctly, we detected duplication of objects and relationships. Otherwise, the results of aggregate functions may be wrong. Experimental results in real datasets showed the enhancement of our approach, the importance of detecting duplication and differentiating query interpretations on the correctness of aggregate functions. These results also showed the optimized techniques enable our approach to be almost as efficient as LCA-based approaches although it has some overhead. In the future, we will handle the problem by different techniques, including transferring XML keyword queries to structured queries such as XQuery queries. Moreover, we will solve the case where object nodes of group-by and aggregate function parameters do not have ancestor-descendant relationships.

References

1. Z. Bao, T. W. Ling, B. Chen, and J. Lu. Efficient XML keyword search with relevance oriented ranking. In *ICDE*, 2009.
2. C. Gokhale, N. Gupta, P. Kumar, L. V. S. Lakshmanan, R. Ng, and B. A. Prakash. Complex group-by queries for XML. In *ICDE*, 2007.
3. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.
4. T. N. Le, T. W. Ling, H. V. Jagadish, and J. Lu. Object semantics for XML keyword search. In *DASFAA*, 2014.
5. T. N. Le, H. Wu, T. W. Ling, L. Li, and J. Lu. From structure-based to semantics-based: Effective XML keyword search. In *ER*, 2013.
6. G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM*, 2007.
7. L. Li, T. N. Le, H. Wu, T. W. Ling, and S. Bressan. Discovering semantics from data-centric XML. In *DEXA*, 2013.
8. Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, 2004.
9. Z. Liu and Y. Chen. Reasoning and identifying relevant matches for XML keyword search. In *PVLDB*, 2008.
10. S. Tata and G. M. Lohman. SQAK: doing more with keywords. In *SIGMOD*, 2008.
11. B. Q. Truong, S. S. Bhowmick, C. E. Dyreson, and A. Sun. MESSIAH: missing element-conscious SLCA nodes search in XML data. In *SIGMOD*, 2013.
12. H. Wu, T. W. Ling, L. Xu, and Z. Bao. Performing grouping and aggregate functions in XML queries. In *WWW*, 2009.
13. P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD*, 2007.
14. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD*, 2005.
15. Y. Zeng, Z. Bao, H. V. Jagadish, T. W. Ling, and G. Li. Breaking out of the mismatch trap. *ICDE*, 2014.
16. C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On supporting containment queries in relational database management systems. *SIGMOD*, 2001.
17. J. Zhou, Z. Bao, W. Wang, T. W. Ling, Z. Chen, X. Lin, and J. Guo. Fast SLCA and ELCA computation for XML keyword queries based on set intersection. In *ICDE*, 2012.
18. R. Zhou, C. Liu, and J. Li. Fast ELCA computation for keyword queries on XML data. In *EDBT*, 2010.