

# USING SEMANTICS IN XML DATA MANAGEMENT

TOK WANG LING

*Department of Computer Science, National University of Singapore, Singapore*  
*E-mail: lingtw@comp.nus.edu.sg*

GILLIAN DOBBIE

*Department of Computer Science, University of Auckland, New Zealand*  
*E-mail: gill@cs.auckland.ac.nz*

XML is emerging as a de facto standard for information exchange over the Web, while businesses and enterprises generate and exchange large amounts of XML data daily. One of the major challenges is how to query this data efficiently. Queries typically can be represented as twig patterns. Some researchers have developed algorithms that reduce the intermediate results that are generated during query processing, while others have introduced labeling schemes that encode the position of elements, enabling queries to be answered by accessing the labels without traversing the original XML documents. In this paper we outline optimizations that are based on semantics of the data being queried, and introduce efficient algorithms for content and keyword searches in XML databases. If the semantics are known we can further optimize the query processing, but if the semantics are unknown we revert to the traditional query processing approaches.

*Keywords:* XML, query processing, query optimization, keyword search

## 1. Introduction

Semistructured data has become more prevalent with the increasing number of advanced applications on the Web. Many of these applications, such as electronic market places, produce and consume large volumes of data. XML (eXtended Markup Language) is emerging as the de facto standard for semistructured data on the Web. Although XML documents could have rather complex internal structures, they can generally be modeled as ordered trees.

In most XML query languages, the structures of XML queries are expressed as twig (i.e. a small tree) patterns, while the values of XML elements are used as part of the selection predicates. Efficiently matching

all twig patterns in an XML database is a major concern in XML query processing. Among them, holistic twig join approaches have been accepted as an efficient way to match twig patterns, reducing the size of the intermediate result.<sup>1</sup> Recently many algorithms have been proposed including TwigStack,<sup>2</sup> TJFast,<sup>3</sup> TwigStackList,<sup>4</sup> Tag+level,<sup>5</sup> prefix path stream (PPS),<sup>6</sup> OrderedTJ.<sup>7</sup> They are based on a form of labeling scheme that encodes each element in an XML database using its positional information. In order to answer a query twig pattern, these algorithms access the labels alone without traversing the original XML documents.

In this paper we outline an innovative way to process XML queries that takes into account the semantics of the data, and introduce an efficient algorithm for keyword searches in XML databases. The essence of the approach is that semantics of the data can be used in query optimization if the semantics of the data are known, otherwise a more traditional approach to query processing will be adopted. Some of the semantics can be represented in schema languages such as DTD<sup>8</sup> and XMLSchema<sup>9</sup> but there is other information that can be used in query processing that cannot be represented in these schema definition languages.

Typically, XML data is simply modeled as a tree structure without the important concepts object class, attribute of object class, relationship type defined among object classes, and attribute of relationship type. We have defined a data model called ORA-SS - Object-Relationship-Attribute Model for Semistructured Data, which includes the concepts in the Entity-Relationship data model together with constructs to capture the hierarchical structure of XML data. With the ORA-SS data model, many semantics of the XML database can be explicitly represented. Semantics that can be represented in the ORA-SS data model but cannot be specified by DTD and XMLSchema include:

- (1) Attribute vs. object class. Data can be represented in XML documents either as attributes or element. So, it is difficult to tell from the XML document whether a child element is in fact an attribute of its parent element or an object. DTD and XMLSchema cannot specify that a child element is an attribute of its parent element.
- (2) Multivalued attribute vs. object class. In XML document, multivalued attributes of an object class have to be represented as child elements. DTD and XMLSchema cannot specify that a child element is a multivalued attribute of its parent element.
- (3) Identifier (ID). DTD and XMLSchema cannot specify the identifier of an object class which appears as a child element and has a many to

many relationship with its parent element. ID and key of DTD and XMLSchema respectively, are not powerful enough to represent the identifier of such object classes.

- (4) IDREF or Foreign Key. As DTD and XMLSchema cannot represent identifiers of some object classes, so foreign key or ID reference of such object classes cannot be specified.
- (5) N-ary relationship type. DTD and XMLSchema can only specify child elements of a parent element, such a relationship is the parent-child relationship and it is a binary relationship type. Ternary relationship types and N-ary relationship types among object classes cannot be specified by DTD and XMLSchema.
- (6) Attribute of object class vs. attribute of relationship type. As DTD and XMLSchema do not have the concept of object classes and relationship types (they only represent the hierarchical structure of elements and attributes), there is no way to specify whether an attribute of a element is an attribute of the element (object class) or an attribute of some relationship type involving the element (object class) and its ancestors (object classes).
- (7) View of XML document. Since DTD and XMLSchema cannot specify identifiers and attributes of object classes and relationship types, DTD and XMLSchema do not contain semantics to define views of XML document which change the hierarchical order of object classes (elements) in the original XML document.

The above semantics (1 to 6) can be captured in the ORA-SS schema diagram, and because of these semantics, we can define a valid XML view which changes the hierarchical order of object classes using a swap operator.<sup>10</sup> With the semantics captured in the ORA-SS schema diagram of an XML database, twig pattern queries on the XML database can be optimized significantly. Using the work we have done with views, we can guarantee that if a query changes the hierarchical order of nodes, the semantics of the output is consistent with the semantics of the XML document. DTD and XMLSchema cannot be used to interpret the output part when the hierarchical ordering of nodes in the output is different than in the query part. With the semantics captured by the ORA-SS data model, we will be able to interpret XML queries correctly and improve the query evaluation performance using these semantics.

In the rest of the paper, we briefly review related work in XML query processing in Section 2, and use an example to introduce the key concepts of the ORA-SS data model in Section 3. In Section 4 we outline how the

semantics represented in the ORA-SS data model can improve the speed of query processing. In Section 6 we introduce an efficient algorithm for keyword searches in XML databases and we conclude in Section 7.

## 2. Related Work

XPath<sup>11</sup> is a language for finding information in an XML document by navigating through elements and attributes in an XML document. Its syntax is similar to directories in UNIX file systems. For example, find the names of the students taking the course “cs4221” can be expressed by the XPath expression:

```
/department/course[code="cs4221"]/student/stuName
```

An XPath query can be expressed graphically by a small tree called a twig pattern. The above XPath query is represented as the twig pattern query shown in Figure 1.

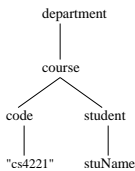


Fig. 1. Example Twig Pattern Query

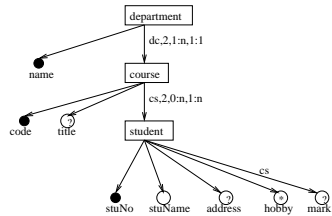


Fig. 2. ORA-SS Schema Diagram

Twig join processing is central to XML query evaluation. Extensive research efforts have been put into efficient twig pattern query processing with label based structural joins. Zhang et al.<sup>12</sup> first proposed multi-predicate merge join (MPMGJN) based on containment labeling of XML documents. The later work by Al-Khalifa et al.<sup>13</sup> proposed an improved stack-based structural join algorithm, called Stack-Tree-Desc/Anc. Both of these are binary structural joins and may produce large amounts of useless intermediate results. Bruno et al.<sup>2</sup> then proposed a holistic twig join algorithm, called TwigStack, to address and solve the problem of useless intermediate results. However, TwigStack is only optimal in terms of intermediate results for twig queries with only ancestor-descendent relationships. It has been proven that optimal evaluation of twig patterns with arbitrarily mixed ancestor-descendent and parent-child relationships is not feasible.<sup>14</sup> There are many subsequent works that optimize TwigStack in terms of I/O, or

extend TwigStack for different problems.<sup>4,5,7,15-17</sup> In particular, a list structure is introduced in TwigStackList<sup>18</sup> for a wider range of optimizations, and TSGeneric<sup>1</sup> is based on indexing each stream and skipping labels within one stream. These approaches describe how to execute a query more efficiently but do not address how the semantics of the data can be used in query optimization.

Chippimolchai et al.<sup>19</sup> developed a semantic query optimization framework in a deductive database setting. They outline an algorithm that transforms a query to an equivalent reduced form with the introduction of integrity constraints. Queries and integrity constraints are represented as clauses and the integrity constraints are derived from the real world. They cannot be derived from XMLSchema or DTDs.

### 3. The ORA-SS Data Model

The Object, Relationship, Attribute data model for Semistructured Data (ORA-SS)<sup>20</sup> has four basic concepts: object classes, relationship types, attributes and references, and consists of four diagrams: schema diagram, instance diagram, functional dependency diagram and inheritance diagram. In this paper we are concerned with the ORA-SS schema diagram.

An ORA-SS *schema* diagram represents an object class as a labeled rectangle. A relationship type between object classes is described by a label “name (object class list), n, p, c”, where *name* denotes the name of the relationship type, *object class list* is the list of objects participating in the relationship type, *n* is an integer indicating the degree of the relationship type, *p* and *c* are the participation constraints of the object classes in the relationship type, defined using the standard min:max notation. The edge between two object classes can have more than one such relationship type label to indicate the different relationship types the object classes participate in. Attributes of object classes or relationship types are denoted by labeled circles. Identifiers of object classes are denoted as filled circles. All attributes are assumed to be mandatory and single-valued, unless the circle contains a “?” indicating that it is single-valued and optional, or a “+” indicating that it is multivalued and required, or an “\*” indicating that it is optional and multivalued. Attributes of an object class can be distinguished from attributes of a relationship type. The former has no label on its incoming edge while the latter has the name of the relationship type to which it belongs on its incoming edge.

Figure 2 shows an ORA-SS schema diagram. The rectangles labeled *department*, *course*, and *student* are the object classes. Attributes *name*,

*code* and *stuNo* are the identifiers of the object class *department*, *course* and *student* respectively. Each *student* has a unique *stuNo*. The attributes *title*, *mark*, *address* and *hobby* are optional. Attribute *hobby* is multivalued, while *stuName* is required. There are two relationship types, called *dc* and *cs*. The former is a binary relationship type between object classes *department* and *course*, while the latter a binary relationship type between *course* and *student*. A *department* can have one or more (1:n) *courses*, and a *course* belongs to one and only one (1:1) *department*. A *course* can have zero or more (0:n) *students*, and a *student* can take 1 or more *courses*. The label *cs* on the edge between *student* and *mark* indicates that *mark* is a single valued attribute of the relationship type *cs*. That is, the attribute *mark* is an attribute of a *student* in a *course*. From these constraints, we can derive that  $\{course, student\} \rightarrow mark$ .

#### 4. Using Semantics in Query Processing

Here we outline how the semantics captured by ORA-SS schema can be used to optimize twig pattern query evaluation with three twig pattern queries. The queries refer to the schema shown in Figure 2.

*Query 1:* Find the *stuName* values of student elements having *stuNo* value equals to “s123”. The XPath expression is:

```
//student[@stuNo="s123"]/stuName
```

Using the ORA-SS schema in Figure 2, we know that *stuName* is a single valued attribute of the student object class and *stuNo* is the identifier of the student, so  $stuNo \rightarrow stuName$ . To process the query, we only need to find the first student element in the XML document with attribute *stuNo* equal to “s123”, and then find the value of its subelement *stuName*. However, if we use a DTD or XMLSchema of the XML data, we would not know that *stuNo* is the identifier of student or that *stuName* is a single valued attribute of student, so we would need to traverse the whole XML document.

Additionally Wu et al.<sup>21</sup> have proposed an algorithm that concentrates on searching for content or values with semantic information as compared to structure-focused query processing. We will discuss content search in more details in Section 5.

*Query 2:* Find the average marks of all the students.

To answer this query the processor needs to know that `stuNo` is the identifier of object class `student`, and `mark` is a single valued attribute of the relationship type between `course` and `student`. In fact, any person that writes this query in XQuery needs to use the same semantics to express the query:

```

for $sNo in distinct_values(//student/@stuNo)
let $mark_set := //course/student[@stuNo = $sNo]/mark
return
  <student stuNo = $sNo >
    <averagemark>{ avg($mark_set) }</averagemark>
  </student>

```

However, a DTD cannot express the semantics that `stuNo` is the identifier of `student` object, and also cannot express that `mark` is a single valued attribute of the relationship between `student` and `course`, that is  $\{course, student\} \rightarrow mark$ . Without this information, there is no way to know whether the XQuery query with an aggregation function (or twig pattern query) is written correctly or not.

*Query 3:* For each student, find all courses taken by the student with the marks the student achieved in the course. Consider for example the query

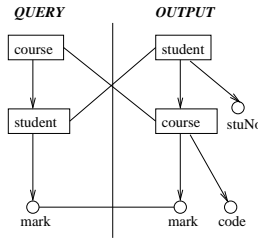


Fig. 3. Example Query Format

shown in Figure 3, where a rectangle represents an object class and a circle represents an attribute. On the left hand side the query is specified, and on the right hand side the output format is given. The lines between the query part and the output part show the correspondences between the objects in the query and those in the output. The query is asking for the marks of

students in courses and in the output courses must be nested within student rather than students within courses. This query can be written in XQuery as:

```

for $sNo in distinct_values(//student/@stuNo)
let $c := //course[student/@stuNo=$sNo]
return
  <student stuNo = $sNo >
    <course code = $c/@code >
      $c/student[@stuNo=$sNo]/mark
    </course>
  </student>

```

In order to write the query above correctly, users have to know that `stuNo` is the identifier of student, `code` is the identifier of course, `mark` is a single valued attribute of the relationship type between course and student, each course is offered by only one department, and each course only appears once in the XML document. This information can be captured in the ORA-SS schema diagram, while DTDs and XMLSchema cannot capture all the necessary semantics.

With the semantics captured by the ORA-SS data model, we will be able to interpret whether XML queries are correct and improve the query evaluation performance using these semantics. The graphical query language GLASS<sup>22,23</sup> can automatically generate the XQuery for the query represented in Figure 3 using the semantics stored in the ORA-SS schema diagram in Figure 2. There is no need for a user to write XQuery queries if the semantics of the data are stored in an ORA-SS schema diagram.

## 5. Content Search in XML

Processing a twig pattern query in XML document includes structural search and content search. Most existing algorithms do not differentiate content search from structural search. They treat content nodes the same as element nodes during query processing with structural joins. Due to the high variety of contents, to mix content search and structure search suffers from management problem of contents and low performance. Another disadvantage is to find the actual values asked by a query, they have to rely on the original document. Therefore, we propose a novel algorithm *Value Extraction with Relational Table (VERT)* to overcome these limitations.<sup>21</sup> The main technique of *VERT* is introducing relational tables

to store document contents instead of treating them as nodes and labeling them. Tables in our algorithm are created based on semantic information of documents. As more semantics captured, we can further optimize tables and queries to significantly enhance efficiency.

For example, consider the example XML tree with containment labels in Figure 4. Instead of storing the label streams for each XML tag and value contents, we can store the value contents together with the labels of their parent tags in relational tables as shown in Figure 5. With these relational tables, when a user issues a twig query as shown in Figure 6(a), the system can automatically rewrite it to the query shown in Figure 6(b), where the node *price*, the value node with value greater than 15, and their PC relationship are replaced by a node called  $price'_{>15}$ . Then, we can execute SQL in table  $R_{price}$  to get all labels of *price* elements with value greater than 15 to form the label stream for  $price'_{>15}$ ; and perform structural join based on label streams of *book*, *ISBN* and  $price'_{>15}$ . In this way, we save both the stream merge cost of all content values greater than 15 and the structural join between the merged label streams for content values and *price* element.

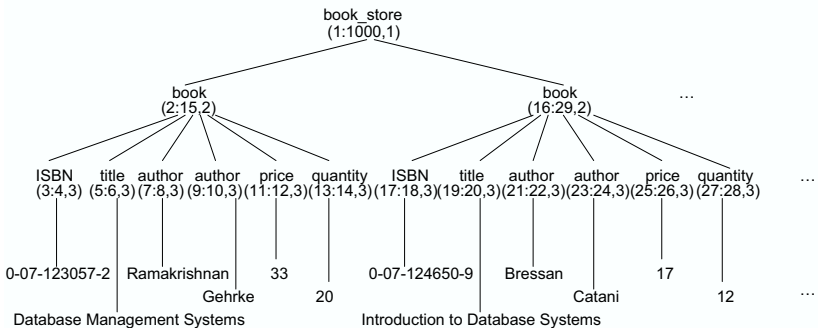


Fig. 4. Example labeled XML tree

Label	Content
(3:4,3)	0-07-123057-2
(17:18,3)	0-07-124650-9
...	...

Label	Content
(5:6,3)	Database Management Systems
(19:20,3)	Introduction to Database Systems
...	...

Label	Content
(11:12,3)	33
(25:26,3)	17
...	...

Fig. 5. Some example tables to store contents with their parent labels

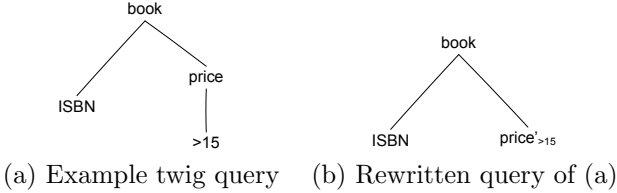


Fig. 6. Example twig patten queries and its rewritten query

Moreover, if we know that *price* is a property of *book* object class by exploiting the schema information, we can directly put the value contents of *price* with labels of *book* object class, instead of the labels of *price* element, as shown in Figure 7(a). In this way, when processing the twig query in Figure 6, we can also save the structural join between *book* object and its *price* property. Note that we can also store the labels of *book* objects with the contents of other properties, such as *title*, *author*, etc., which are not shown due to limited space.

Label	Content
(11:12,3)	33
(25:26,3)	17
...	...

Label	ISBN	title	price	quantity
(11:12,3)	0-07-123057-2	Database ...	33	20
(25:26,3)	0-07-124650-9	Introduction ...	17	12
...	...	...	...	...

(a) Table of *price* contents and *book* object labels      (b) Table of labels and pre-merged single-valued property contents of *book* object

Fig. 7. Example table to store contents with labels of object classes

Finally, if we further know that *ISBN*, *title*, *price*, etc. are single-valued properties of *book* object class according to semantics captured by ORASS, we can pre-merge the content values of these properties into a single relational table with the labels of *book* objects as shown in Figure 7(b). With the pre-merged table, to answer the twig query in Figure 6, we can simply perform an efficient selection on the pre-merged table without time consuming structural joins. Note that we should not merge multi-valued properties (e.g. *author*) into the table to avoid duplicate information.

Experimental evaluation shows that besides solving different content problems, *VERT* also has superiority in performance of twig pattern query processing comparing with existing algorithms.

## 6. Keyword Search with Semantics in XML

Keyword proximity search is a user-friendly way to query XML databases. Most previous efforts in this area focus on keyword proximity search in XML based on either a tree data model or a graph (or digraph) data model. Tree data models for XML are generally simple and efficient. However, they do not capture connections such as ID references in XML databases. In contrast, techniques based on a graph (or digraph) model capture connections, but are generally inefficient to compute. Moreover, most of the existing techniques do not exploit schema information which is usually present in XML databases. Without schema information, keyword proximity techniques may have difficulty in presenting results, and more importantly, they return many irrelevant results. For example, the LCA (Lowest Common Ancestor) semantics for keyword proximity search based on tree model may return the overwhelmingly large root of the whole XML database.

Therefore, we propose an interconnected object model to fully exploit the property of XML and the underlining schema information when the schema is present.<sup>24</sup> In our model, database administrators identify interested object classes for result display and the conceptual connections between interested objects. For example, the interested object trees in DBLP can be publications; and the conceptual connection between publications can be reference and citation relationships.

With interested object classes, the most intuitive result of keyword proximity search is a list of interested objects containing all keywords. We call these interested objects as ICA (Interested Common Ancestor) in contrast to the well-known LCA (Lowest Common Ancestor) semantics. Also, and more importantly, we propose IRA (Interested Related Ancestors) semantics to capture conceptual connections between interested objects and include more relevant results that do not contain all keywords. An IRA result is a pair of objects that together contain all keywords and are connected by conceptual connections. An object is an IRA object if it belongs to some IRA pair. For example, for query “XML query processing”, the paper with title “query processing” and citing or cited by “XML” papers are considered as IRA objects. Further, we propose RelevanceRank to rank IRA objects according to their relevance scores to the query. RelevanceRank is application dependent. For an intuitive example, in DBLP, for query “XML query processing”, a “query processing” paper that cites or is cited by many “XML” papers is ranked higher than another “query processing” paper that cites or is cited by only one “XML” papers. Other ranking metrics can also be incorporated with RelevanceRank. For example, for query

“John Smith”, we can use proximity rank to rank papers with author “John Smith” higher than papers with co-author “John” and “Smith”.

Experimental evaluation shows our approach is superior to most existing academic systems in terms of execution time and result quality. Our approach is also superior or comparable to commercial systems such as Google Scholar and Microsoft Libra in term of result quality.

## 7. Conclusion

One of the important areas in the management of semistructured data is providing algorithms that enable efficient querying of the data. Many researchers have investigated matching twig patterns, using clever matching algorithms and included labeling schemes which enable smart ways of determining the relationships between nodes in a tree, without traversing the tree.

In this paper, we outline some optimizations that can be introduced when semantics of the data are known. We introduce a data model, ORASS in which the necessary semantics can be represented, and describe the kinds of optimizations that can be done. We demonstrate how twig patterns can be optimized when semantics are included, how to process values in holistic twig join algorithms, and how conceptual connections between object classes can be used in keyword proximity searches.

In the future we will study how to use other semantics captured in ORASS schema diagrams to further optimize the evaluation of twig pattern queries, provide guidelines of where these optimizations are worthwhile, and show the improvement in processing speed through experimentation. The particular areas we will look at include how specific information in twig queries interact with optimization such as parent-child and ancestor-descendant relationships, negation, ordering of nodes, constant values, and output nodes.<sup>22</sup>

## References

1. H. Jiang, W. Wang, H. Lu and J. X. Yu, Holistic twig joins on indexed XML documents, in *VLDB*, 2003.
2. N. Bruno, N. Koudas and D. Srivastava, Holistic twig joins: optimal XML pattern matching, in *SIGMOD Conference*, 2002.
3. J. Lu, T. Chen and T. W. Ling, TJFast: effective processing of XML twig pattern matching, in *WWW (Special interest tracks and posters)*, 2005.
4. T. Yu, T. W. Ling and J. Lu, TwigStackList: A holistic twig join algorithm for twig query with not-predicates on XML data, in *DASFAA*, 2006.

5. T. Chen, J. Lu and T. W. Ling, On boosting holism in XML twig pattern matching using structural indexing techniques, in *SIGMOD Conference*, 2005.
6. T. Chen, T. W. Ling and C. Y. Chan, Prefix path streaming: A new clustering method for optimal holistic XML twig pattern matching, in *DEXA*, 2004.
7. J. Lu, T. W. Ling, C. Y. Chan and T. Chen, From region encoding to extended dewey: On efficient processing of XML twig pattern matching, in *VLDB*, 2005.
8. T. Bray, J. Paoli and C. M. Sperberg-McQueen, Extensible markup language (XML) 1.0. 2nd edition <http://www.w3.org/TR/REC-xml>, (Oct. 2000).
9. H. Thompson, D. Beech, M. Maloney and N. M. (Eds), XML Schema Part 1: Structures <http://www.w3.org/TR/xmlschema-1>, (May 2001).
10. Y. B. Chen, T. W. Ling and M.-L. Lee, Designing valid XML views, in *ER*, 2002.
11. J. Clark and S. DeRose, XMLpath language XPath version 1.0 <http://www.w3.org/TR/xpath>, (Nov. 1999).
12. C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo and G. M. Lohman, On supporting containment queries in relational database management systems, in *SIGMOD Conference*, 2001.
13. S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas and D. Srivastava, Structural joins: A primitive for efficient XML query pattern matching, in *ICDE*, 2002.
14. B. Choi, M. Mahoui and D. Wood, On the optimality of holistic algorithms for twig queries, in *DEXA*, 2003.
15. H. Jiang, H. Lu and W. Wang, Efficient processing of twig queries with or-predicates, in *SIGMOD Conference*, 2004.
16. J. Lu, T. W. Ling, T. Yu, C. Li and W. Ni, Efficient processing of ordered XML twig pattern, in *DEXA*, 2005.
17. B. Chen, T. W. Ling, T. Ozsu and Z. Zhu, *To appear in DASFAA 2007*.
18. J. Lu, T. Chen and T. W. Ling, Efficient processing of XML twig patterns with parent child edges: a look-ahead approach, in *CIKM*, 2004.
19. P. Chippimolchai, V. Wuwongse and C. Anutariya, Towards semantic query optimization for XML databases, in *ICDE Workshops*, 2005.
20. T. W. Ling, M. L. Lee and G. Dobbie, *Semistructured database design* (Springer, 2005).
21. H. Wu, T. W. Ling and B. Chen, VERT: an efficient algorithm for content search and content extraction in XML query processing. Submitted for publication.
22. W. Ni and T. W. Ling, GLASS: A graphical query language for semi-structured data, in *DASFAA*, 2003.
23. W. Ni and T. W. Ling, Translate graphical XML query language to SQLX, in *DASFAA*, 2005.
24. B. Chen, J. Lu and T. W. Ling, LCRA: effective semantics for XML keyword search. Submitted for publication.