The University of Sheffield

Department of Computer Science

A Labelling Technique Comparison for Indexing Large XML Database

A dissertation is presented By:

Samer Al-khazraji

for the MPhil Degree

Under Supervision of:

Dr. Siobhán North

30/07/2016

Abstract

The flexibility nature of XML documents has motivated researchers to use it for data transmission and storage in different domains. The hierarchical structure of XML documents is an attractive point to be researched for processing a user query based on labelling where each label describes the node structure in the tree. In this study, three categories of XML node labelling will be analysed to address the open problem of each category. A number of experiments are executed to compare performance of time execution and storage space required for labelling XML tree.

Acknowledgments

I would like to express my appreciation and thanks my supervisor Dr. Siobhán North for here guidance and support which develop my understanding to XML technology through XML labelling. I would like to thank the department staff and my colleagues in the VT group to give me a kind environment during my study. A big thanks to my parents, wife, daughter, and sons for their patience, support, and love.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

There is a large amount of exploitation of XML documents by different applications in different domains. the hierarchical structure of the documents makes it a challenge to access the information in the documents. A lot of researchers have investigated the improvement of a query process by simplifying the access of the information which is saved in XML tree using various labelling schemes. However, the tree structure of the XML tree is varies between wide and deep and designing a suitable labelling scheme for different tree structures is a challenge.

Another challenge that may face the researchers in developing an XML labelling scheme is that XML documents are dynamic and static labelling schemes are costly because the tree nodes need to be relabelled with each tree update. The relabelling process is costly from the time and storage perspectives because labels represent both the location and relationships of a node.

In this dissertation, three models of labelling schemes will be demonstrated: Interval labelling scheme, Prefix labelling scheme, and Multiplicative labelling scheme. The static technique for labelling trees will be explained to the issues that relate with it which motivated researchers to develop dynamic labelling schemes to avoid the relabelling process. While, dynamic labelling schemes also have drawbacks which relate with storage size and may lead to relabelling operations. This issue has attracted another group of researchers to design other labelling schemes that employ mathematical operations and represented by Multiplicative labelling scheme.

To study which labelling scheme is suitable for an XML document based on tree structure, a number of experiments were executed on variants XML tree structures. The results of these experiments identify the convenient labelling scheme for an XML tree.

1.2 Problem definition and Motivation

The employment of XML document for data storage has increased and the need for an efficient database management system which should be similar to the relational database management system is an urgent requirement (Almelibari, 2015). The index system in a relational database management system is designed for a table structure (Wu et al., 2004), however, an XML document has a hierarchical tree structure and it needs a different indexing system. Node labelling scheme has been employed as an indexing system in XML technology to assign a unique label to each node that represents the node relationships (Mlŷnková, 2008). Therefore, the structural query will be answered effectively and efficiency through labelling scheme without accessing the actual XML documents. In addition, processing user query based on the exact words will be time consumption because the application will process all words in the document that match the user query and also it needs large storage space to store the processing words (Wang and Liu, 2003; Yu et al., 2005).

In this study, three categories of labelling schemes will be explained: the Interval labelling scheme, the Preffx labelling scheme, and the Multiplicative labelling scheme. The earliest Interval labelling schemes were proposed for static XML documents and do not fulfill the requirements for the dynamic XML data. So, researchers developed dynamic models for interval labelling schemes, but sometimes they still need to relabel the document because of the overflow problem where the available space to label new nodes is not sufficient. Moreover, this category of labelling scheme does not represent all relationships between nodes such as sibling and it is costly in terms of both time and storage space because it generate labels exponentially as will be explained later in chapter 5.

Different groups of researchers adopted a scheme based on the Dewey indexing scheme use in libraries for labelling XML nodes (Sans and Laurent, 2008). This scheme is capable of representing different kinds of node relationships (Assefa and Ergenc, 2012). The labels in this section have two main parts separated by a delimiter: the prefix part which expresses ancestor path from the root and the second part which expresses the current position of the node in the tree (Assefa and Ergenc, 2012; Liu and Zhang, 2016; Tatarinov et al., 2002). The dynamic algorithms for this kind of labelling scheme is costly from storage space view point especially with deep trees (Haw and Lee, 2011; Xu et al., 2012).

Another group of researchers adopted atomic numbers to represent XML node labels and mathematical operations to define their relationships (Al-Shaikh et al., 2010; Haw and Lee, 2011; Wu et al., 2004). In this study, three algorithm were explained based on three mathematical principles: prime number, dominator and nominator, and graph vectors. The characteristics of this class of labelling schemes is that it has a complex procedure to find the nodes relationships and overflow problems frequently occur(Assefa and Ergenc, 2012).

The motivation for this dissertation is to find the most suitable labelling scheme for labelling a specific XML tree where, XML trees have different structures, some of them wide and another deep. So that, In the later technical development, three realistic XML databases were chosen for evaluation purposes: the dblp XML database is the widest tree with 3,332,130 elements and 6 levels, Treebank-e is the deepest tree as it has 24,376,66 elements and 36 levels, and the Nasa XML database which is in between the other databases from the depth and width perspective has 476,646 elements and 8 levels.

The experiments show that, the Prefix labelling scheme is good for XML trees which are wide but it is expensive from a storage space view point for deep XML trees such as Treebank-e. The Interval labelling scheme is good for deep XML trees from storage space view point but it is costly from time and storage space perspectives. The last experiments were done on labelling XML databases using an order vector-centric algorithm but the outcomes were discounted because labelling gave incorrect results.

1.3 Research Hypothesis and Methodology

The aim of the dissertation is to answer the following hypothesis, *There are many XML databases which have been designed based on business requirement with different structures from the width and depth view point. On the other hand, there are many labelling schemes have been developed to label these databases and the label size will increase with continuous use of the XML database. Moreover, the time required to generate the larger labels will increase as well. Therefore, it is necessary to find a suitable labelling scheme to label a specific XML database in fast and using minimal storage space.*

The dissertation exploited *System Development Methodology* (SDM) (Almelibari and North) which is heavily employed by software engineering researchers. The methodology is used to test the hypothesis based in four phases, identify the problem and design a solution, implement the solution for testing purpose, evaluate the result (e.g. an empirical evaluation), and contribute to knowledge based on the outcomes of the research.

1.4 Dissertation Outline

The dissertation consist of three parts; *part 1* consists of chapter 2 which includes a number of issues that relates with similarity comparison. *Part2* consists of two chapters, chapter 3 that includes the issues which relate to XML queries and chapter 4 which involves the problems of labelling an XML tree. *Part3* will be chapter 5 and 6 for the practical work and statistical analysis respectively.

Chapetr 2 : This chapter discuss the representation of XML documents as a tree or vector to compare the similarity between XML documents based on structure, semantics, and structure and semantics (hybrid).

Chapter 3 : In this chapter, the process of retrieving of an optimum answer to the user query based on the semantics LCA is discussed. A number of approaches were proposed to enhance XML query processing based on the *Lowest Common Ancestor* LCA semantics which is an ancestor node that contains all query keyword as a descendant nodes (Liu and Chen, 2007).

Chapter 4: The chapter illustrates algorithms of three categories of labelling schemes. The strength and drawbacks of each category will be explained which motivated researchers to propose new labelling schemes which may cover the drawbacks of the previous scheme and enhance the performance of XML query.

Chapter 5: The practical work is illustrated in this chapter. Two experiments were executed using static labelling schemes (Prefix and Interval) to measure the time and storage space required for labelling three XML databases. Another two experiments were executed using one of the Multiplicative labelling scheme to label Nasa databases dynamically.

The practical work and statistical analysis under LCA semantics will be illustrated. In chapter 4, a number of labelling schemes which have been proposed to enhance query processing will be explained. The last part will include chapter 5 which has the practical work and statistical results achieved from a number of experiments to analyse the performance of three categories of labelling schemes on three XML databases. A summary of each chapter will be dicussed in the following sections:

Chapter 6:The results of the experiments in the chapter 5 were analysed and evaluated with previous work.

All aspects which are mentioned in the previous section will be discussed in more detail in the next chapters.

2 XML Background

2.1 Preface

It is undoubtable that XML has become the standard for data transmission and representation in a wide range of domains (Algergawy et al., 2010, 2011; Bertino and Ferrari, 2001). This chapter will cover the meaning of XML and why it is important. The structure of XML documents is part of this discussion and the context of XML elements as well. This chapter will clarify both aspects: Well-Formed XML and Valid XML. The types of XML databases will be covered and the also the languages that are designed to deal with these data bases.

2.2 What is The XML Language

XML stands for Extensible Markup Language and also it is written as eXtensible Markup Language (Drol, 2008; Ethier, 2008; Piernik et al., 2015) and is recommended by W3C since 1998 (Harold and Means, 2004; Lee and Foo, 2008). (Drol, 2008) and (Lee and Foo, 2008) illustrated that XML embeds the information of the content in the same file using markup which will be described in the next section. (Harold and Means, 2004) defined XML as a common syntax used to mark up data with tags that are easy to understand by human. Moreover, it gives digital data a standard format which is flexible enough to be customised in different techniques in addition to web sites such as data integration, classification, clustering (Tekli et al., 2015) and query processing and query processing (Assefa and Ergenc, 2012; Duong and Zhang, 2008; Liu et al., 2013a; Tatarinov et al., 2002). There is another employment of tags in XML technology in addition to separate the information from the content and that is storage as is explained in the next section.

2.3 What is Markup Language

It is natural that computer data is vulnerable to destruction and the reliability of computer language is limited to the ways it can be retrieved. For example, files of Lotus 1-2-3 which is a computer language required a lot of time and investigation in retrieving its data as explained by (Harold and Means, 2004). This is because Lotus language is a machine language and it is diffcult to read for non-experts. The need for technology which is human and machine readable as mentioned in previous section is important in this situation and XML is a possible solution. (Wilde, 2012) defined markup as a special character or sequence of characters to be distinguished from the content by the program interpreter. Markup can carry any kind of information which may help for information representation such as determining the space between characters, adding page breaks, and changing the font size. (Wilde, 2012) and (Ethier, 2008) found that there is another exploitation of markup in identifying the treeform hierarchical structure of information based on elements. Each element in an XML document is within markup delimiters called tags which indicate the start and the end of the markup for the element. In this way, (Wilde, 2012) illustrated that the markup makes it possible to code the document content (which will classify XML document in section 1-7) and its structure in a technique which allows both people and machines to interpret it clearly as shows in example (1).

Example 1.

```
<?xml version="1.0">
<Shop>
<Fruit> Apple </Fruit>
<Fruit> Pineapple </Fruit>
<Juice> Orange </Juice>
```

</Shop>

In another context, a library, it is clear for non-experts users to understand that the title of a book is XML Data. The expert user delimited the XML element by start tag <Book-Title> and end tag < /Book-Title> and its content is XML Data. There will be more detail about the context of XML elements in later sections. HTML and SGML are also markup languages, but XML was been adopted in many domains as mentioned in the earlier section and has become a de facto standard technology for transfer and for representing information. The reasons will be explained in the next section.

2.4 XML Evolution and Importance

There are two factors that motivated researchers to invent XML technology and have made XML technology adopted by many applications: the complexity of SGML and the inflexibility of HTML (Wilde, 2012).

The ancestor of XML is *Standard Generalised Markup Language* SGML which was proposed at IBM in 1970s and it was developed in many ways to be adopted by ISO standard 8879 in 1986. (Wilde, 2012) explained the tags of SGML which are difficult to understand. It was succeeded by HyperText Markup Language HTML which has a small number of predefined tags and those tags relate only to the layout of the document.

HTML's restricted set of tags, whilst ideal for the purposes for which they were designed, can only be used to determine presentation. (Wilde, 2012) and (Ethier, 2008) claimed that the XML data was developed to have the same power as SGML in but greater flexibility in that it permitted user defined tags and so could be used for purposes other than simply defining an appropriate layout.

In addition benefit of XML documents, (Drol, 2008) explained is that an XML document is a text file and it is easy to transfer through a network. However, XML data transferred among applications needs to be understood by the destination application. This can be only done when XML document is well-formed and valid as will be explained in the next section.

2.5 Well-Formed and Valid of XML Documents

(Wilde, 2012),(Drol, 2008; Ethier, 2008),(Lee and Foo, 2008), and(Harold and Means, 2004) defined an XML document which conform the rules that describe the syntax of XML as well-formed. They considered these rules to be basic because they relate to the structure of XML document. A well-formed XML document means that the XML elements are nested and a collection of characters which is known as *entities* (Wilde, 2012) are correctly referenced by the document then the XML document will be syntactically correct. If an XML document is not well-formed, the application will not work properly and internet browser will display an error message (Lee and Foo, 2008). On the other hand, an XML document said to be a valid when it matches the rules that describe the structure of the document known as an XML schema (Drol, 2008; Ethier, 2008; Harold and Means, 2004; Lee and Foo, 2008; Wilde, 2012).

(Hegewald et al., 2006) stated that a valid XML document has a number of advantages: it enhances the effectiveness of the operational data by detecting invalid data during data transmission. An XML schema improves the query processing through providing the element's semantics as will be shown in the chapter 4.

2.6 The Context of XML elements

(Wilde, 2012) described the structure of XML document as an hierarchical structure of elements which may have one or more elements of additional information. This additional information is saved in the start tag of the element and called an attribute. (Wilde, 2012) and (Algergawy et al., 2010) categorised XML elements into four classes according to their context: sibling, ancestor, child, and leaf as illustrated in figure (1).

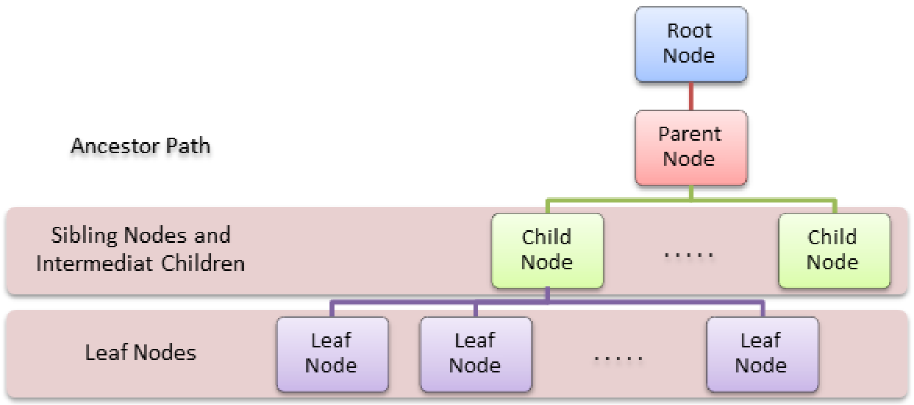


Figure 1: The Context of Element of XML Schema Tree, Adapted from (Al-Shaikh et al., 2010).

The sibling context of an element contains the preceding siblings and the following siblings. The ancestor context of an element is the path from the root element of tree of XML schema to the element. The child context of an element is a set of intermediate child nodes including attributes which is information relating to the node (w3schools, 2016) and sub-elements, while the leaf context of an element is a set of leaf nodes of a sub-tree rooted at the element. The hierarchical structure of XML elements needs to be stored in a data storage model (Win et al., 2003) that provides XML document processes such as accessing, storing, and retrieving (Kim et al., 2007). The next section will explain the common models storing XML document.

2.7 Models for XML Database

To understand the models of XML database, it is necessary to comprehend the instances of XML document.(Kurt and Atay, 2002), (Gulbransen, 2002), and (Powell, 2007) said XML documents exist in two instances: document-centric and data-centric. (Gulbransen, 2002) and (Powell, 2007) defined a document-centric XML as a document that is edited and read by humans such as a user manual. However, they explained data-centric XML document as a machine readable file used to transfer messages between machines such as Web Service Description Language (WSDL) (Christensen). Based on this classification, (Kurt and Atay, 2002) and (Bellahsène, 2003) categorised XML document into two types: XML-enable databases and Native-XML databases (NXD).

An XML-enabled database is a conventional database management system that supports XML documents as defined by (Win et al., 2003) and (Kurt and Atay, 2002) such as Oracle XDK (Kurt and Atay, 2002) and Microsoft SQL server (Microsoft, 2016. This approach uses an existing, mature database management systems to manage XML (Win et al., 2003) and is normally applied to data-centric XML documents (Bellahsène, 2003; Kurt and Atay, 2002).

Data in relational databases is stored in tables which represents one entity type and consists of rows and columns. Rows (e.g. records) represent instances of that entity type and columns which represent the value attributes of that instance and is provided with an indexing mechanism to access data in the table citep Codd:1970:RMD:362384.362685. Because an XML tree has an hierarchical structure (Assefa and Ergenc, 2012) and (Win et al., 2003) and (Kurt and Atay, 2002) adapting it for this type of database management system has several limitations. The limitation includes the necessity to convert the data format from XML into a relational form to save it in relational database. This process of mapping data from XML into relational data and vice versa to retrieve it is complex and time and space consuming when dealing with a large data. In addition, any XML query also needs to be converted into a format that is understood by the underlying database system which takes time and decreases the performance of query processing.

Another model of XML document storage is the Native-XML database which is defined by (Win et al., 2003) and (Kurt and Atay, 2002) as a data model of storage that saves the XML documents and preserves its hierarchical structure thus avoiding the mapping process. (Win et al., 2003) mentioned the restriction of this database is that it needs more I/O resources to retrieve information with a unique path. Moreover, (Assefa and Ergenc, 2012) clarified that an XML tree's hierarchical structure and needs an indexing technique similar to that used in the relational database management. The new indexing system should be designed for hierarchical structures and capable of answering XML queries efficiently. An XML labelling scheme is the key of that technique

To process XML document efficiently, it is important to study XML queries and this is the subject of next section.

2.8 XML Query Languages

In the Section 2.3 it was explained that XML has user defined tags which makes it a common standard for many application in different domains. The heavy use of Native-XML databases (NXD) to store data may lead to an increase the size of the data and a decrease the performance of XML database management because it does not have a single well-formed data model (Salminen and Tompa, 2001). To keep an XML database at the highest level of performance, (Chamberlin et al., 2000) suggested that the XML query language should simulate the flexibility of XML and preserve the hierarchical structure of the document. Moreover, they expect the XML database language to have the similar operations (enumerated by (Tian et al., 2011)) such as storing, searching, and retrieving to that of the Structured Query Language (SQL) in relational databases. Many languages were designed for NXD such as: XML Link (XLink) explained by (DeRose, 2001) as a language that declares some elements in XML to be links between two or more resources.

The next XML language is XPath which was defined by (Berglund) and (Robie and Snelson) as a language that provides an expression which represents the hierarchical structure of a node in the XML tree and atomic values such as integers, string, Boolean, and series that may contain references to nodes in the XML tree and atomic values. Another XML language is XQuery which was defined by (Boag and Simon) as a flexible language that is able to work with a wide range of XML information sources.

To permit a user to structure an XML document known as XML Schema ((Mlýnková and Nečaský, 2013), W3C designed two XML schema languages: Document Type Definition (DTD)(Tim Bary, 2008) and XML Schema Definition (Paul V Biron, 2004). The former was described by (Gong and Yao, 2013) in their paper as using a few data types which are used to define the types of an XML element as explained in example (2).

In addition, (Harold and Means, 2004) claimed that DTD is good at defining the structure of the XML documents in a narrative style documents such as

Books, web pages, and reports. However, XML improves the representation of simple documents: trading, and banking are examples of areas where a DTD is too simple (Harold and Means, 2004). (Harold and Means, 2004) explained the limitation of DTD as follows: the first, limitation of DTD can be found through data type definition of element, for example DTD cannot define the data type of a price element as an integer greater than zero. Second, an XML parser can not read a DTD because they have different syntax. Third, a DTD cannot order the children of an element, for instance DTD cannot order the children: FirstName, MiddleName, and Surname for the ancestor element *Parent*. Finally, it is not easy to combine two DTDs seamlessly to form a big database in contrast to XML documents.

Their solution was XML schema (XSD) which is scalable, extensible, and has a powerfull data type definition. (Harold and Means, 2004) explained that the word schema means form or shape in Greek language and it entered to the computer science through database technology.(Harold and Means, 2004) clarified that the meaning of schema has grown from defining tables and data type of each field in the table to define the content of the documents as in the example (3) which is the XML schema for the documents of example (1).

Example 2.

```
<!DOCTYPE Shop
[
<!ELEMENT Shop(Fruit, Juice)>
<!ELEMENT Fruit(#PCDATA)>
<!ELEMENT Juice(#PCDATA)>
<!ENTITY citrus "Orange" >
]>
```

```
    <?xml version="1.0">
<Shop>
<Fruit> Apple </Fruit>
<Fruit> Pineapple </Fruit>
<Juice> &citrus </Juice>
</Shop>
```

where,

DOCTYPE defines the Shop as a root element,

ELEMENT Shop define the children of the root Shop,

ELEMENT Fruit and ELEMENT Juice are defined as a type of PCDATA,

ENTITY defines citrus as a reusable resource in XML file and replace it with Orange to be displayed in the output file.

#PCDATA means parse character data between the tags.

Haw and Lee (2011) stated that an XML document can be queried using two types of query processing: *full text queries* (keyword-base search) which is close to the content retrieval in the technology of information retrieval or *structural queries* (structural search) which are a complex queries that find matches on the tree that have a similar structure and tags to those specified in the query. Structural queries are categoriesed by Haw and Lee (2011) into: a *path query* (Simple Path Expression) that defines a single node (leaf) at a time based on either Parent-Child (P-C) or Ancestor-Descendant (AD) relationships, or a *twig query* (Branching Path expression) which defines two or more nodes based on P-C, A-D and sibling relationships.

Example 3.

```
<xs:element name="Shop">
```

```
<xs:complexType>
<xs: sequence>
<xs:element name="Juice" type="xs:string"/>
<xs:element name="Fruit" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

where,

xs : *element* name="Shop" is a definition of the root element Shop.

xs : *complexType*, it means the element Shop is a complex type (has children).

xs : *sequence*, the complex type element shop has a sequence of elements.

xs : *elementname* = "*Juice*", is the definition of element Juice as a string.

xs : *elementname* = "*Fruit*", is the definition of element Juice as a string.

The use of XML schema is considerable through some XML applications such as information retrieval. Providing users with a precise answer relies on the accuracy of the similarity comparison. The accuracy can be accomplished by including semantic information that can be found in the schema and structure within the comparison process ans will be seen in the following chapters.

2.9 Conclusion

The flexibility of XML technology comes from the ability of XML users to define XML elements as they need them in different areas (Harold and Means, 2004). These features made XML technology a standard format for data transmission and representation through different domains (Wilde and Glushko, 2008). Domains which adopted XML include: Bioinformatics where (Achard et al.,

2001) found that the flexibility of XML is a suitable environment to represent Bio information. For the Geography domain, (Peng and Zhang, 2004) asserted that XML was a better technology to transfer geographical information between Geo-graphical Information systems GIS. In mathematical work, W3C released Mathematical Markup Language MathML as one of the XML applications to deploy the mathematical expression over the web (Carlisle, 2014). Distributed Learning Transferring ADL has adopted XML to release e-learning application dubbed Shareable Content Object Reference Model SCORM (ADL). This application allows users to share and reuse teaching materials based on a distributed approach (Shih et al., 2006). (Tekli et al., 2009) added additional domains such as: data warehousing, management and version control of documents, semi-structure document integration by determining the similar XML document which created by different sources to provide more information to the user, clustering and classification of XML documents, XML document retrieval. The performance of these applications depends on the accuracy of the similarity measurement which will be explained in detail in the next section.

3 XML Documents Similarity (Representation and Measurement

3.1 Introduction

(Tekli et al., 2007) claimed that the efficiency of the similarity measures has be-come the core to the performance of some technologies. For instance, data man-agement and warehousin (Chawathe et al., 1996, 1999; Cobena et al., 2002), XML query processing (Lin et al., 2014; Liu et al., 2013b; Schlieder; Zhou et al., 2012a,b), XML document clustering (Aïtelhadj et al., 2012; Algergawy et al., 2011; Costa and Ortale, 2012, 2013; Piernik et al., 2016; Posonia and Jyothi, 2013; Wang et al., 2012), XML document validation (Gal, 2007; Smiljanić et al., 2005; Solimando et al., 2014; Tekli and Chbeir, 2012; Zerdazi and Lamolle, 2008). To measure the similarity between XML documents, (Algergawy et al., 2011) and (Asghari and KeyvanPour, 2015) argued they need a uniform representation as will be explained in the next section.

3.2 XML Data Representation

(Algergawy et al., 2011) and (Asghari and KeyvanPour, 2015) claimed that the XML data model is an important step to measure the similarity of XML documents. This is because it identifies the semantic and structural features of the document. In this section, the recent studies will be explained which adopted two kinds of modelling: Tree and Vector.

3.2.1 Similarity approaches based on Tree-based Representation

As explained in the section (2.3), (Wilde, 2012) and (Goldfarb and Prescod, 2001) stated that hierarchical structure of XML data has attracted many researchers to compute XML document similarity based on their tree-like structure. (Piernik et al., 2015) defined the tree-like structure of XML data as follows:-

$$T = (N_T, E_T, L_T, a)$$

Where,

N_T={$n_{root}, n_1, , n_n$}, is a group of nodes where n_{root} is a root node of a tree.

E_T={(n_1,n_2),: n_i,$n_j \in N_T$}, is a finite group of edges that connect between the parent (n_1) and its child (n_2).

L_T, is a group of node labels that correlated with the names of the corresponding elements and attributes of an XML data.

a: $N_T \rightarrow L_T$, is a function that maps each node into a label.

This structure was employed by many researchers such as (Nayak and Xu, 2006), (Antonellis et al., 2008), (Alishahi et al., 2010), (Guzman et al., 2013) to measure the similarity between XML documents as will be studied starting from the model of (Nayak and Xu, 2006).

(Nayak and Xu, 2006) proposed the LevelStructure algorithm to compute the structural similarity between heterogeneous XML documents. They classified XML documents into two groups: homogeneous documents and heterogeneous documents. The latter are documents which are derived from different XML schemas or Document Type Definitions (DTD) which may have different tree structures but may have the same information included in the tag and known as semantics (Kim et al., 2007) or may have not (i.e. synonyms rather than exact names). The former are documents that are derived from sub-trees of the same DTD. (Antonellis et al., 2008) claimed that the rise in the use of heterogeneous has had an impact on the performance of many applications such as data mining, mathematical, and communication. This is because the difference in structure between heterogeneous documents makes the measurement of similarity costly in terms of both time and storage. Therefore, (Nayak and Xu, 2006) proposed a new model called LevelStructure.

(Nayak and Xu, 2006) assigned a distinct number for each named node in an XML tree, grouped nodes from each level of the XML tree by number and arranged them as a list of levels. They did not take node relationships into consideration. Therefore, it is possible to find XML documents with the same LevelStructure representation with different semantics as is shown in figure (2).

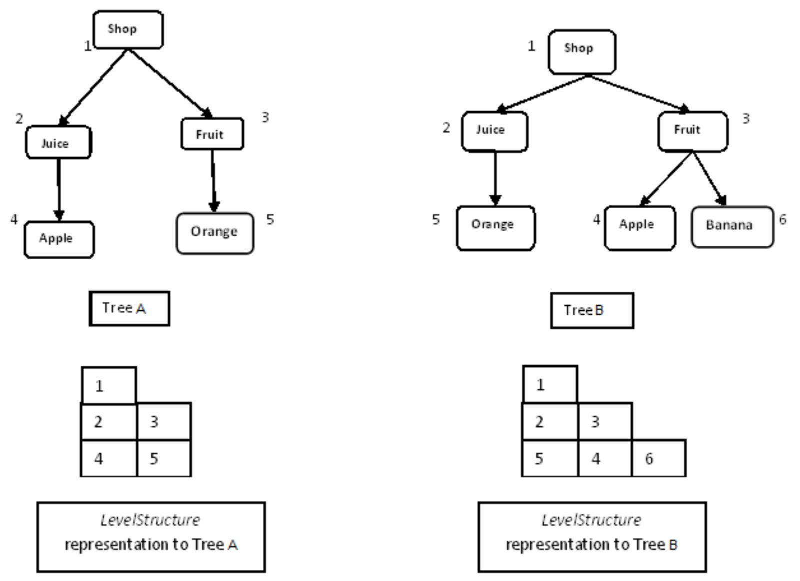

Shop
Juice
Fruit
Apple
Orange
Banana
Tree A
Tree B
LevelStructure representation to Tree A
LevelStructure representation to Tree B

Figure 2: LevelStructure representation to heterogeneous XML documents, Adapted from (Nayak and Xu, 2006).

The parent of node Apple in tree 'A' is Juice and its number is 2, however, the parent node of Apple in tree 'B' is Fruit with number 3. The LevelStructure representation for level 3 of both trees seems to be similar, but they are semantically different.

To overcome the drawback of this approach of (Nayak and Xu, 2006), (Alishahi et al., 2010) added another number to the number of node. The number of the parent node is added to the current node as shown in figure (3).

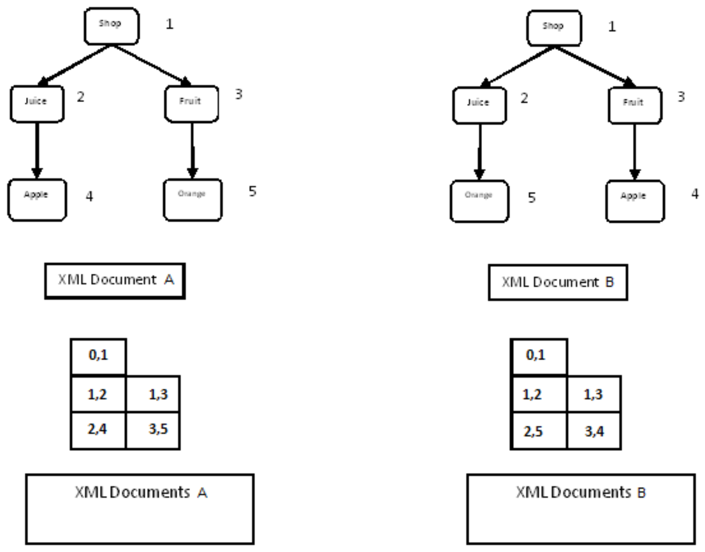


Figure 3: Tree representations of XML Documents, Adopted from (Alishahi et al., 2010).

In contrast to (Alishahi et al., 2010) and (Antonellis et al., 2008) assigned a number to each edge in the tree rather than assigning an additional number to each node. The latter proposed another approach which is dubbed LevelEdge Structure to improve the similarity computation of LevelStructure algorithms.

(Antonellis et al., 2008) assigned a distinct number to each edge that connects specific nodes. They argued that their proposed algorithm calculates the structural similarity of homogeneous XML document in addition to heterogeneous document. The approach of (Antonellis et al., 2008) organizes the information as a tree of levels and each level consist of a set of numbered edges that connect two consecutive nodes as shows in figure (4).

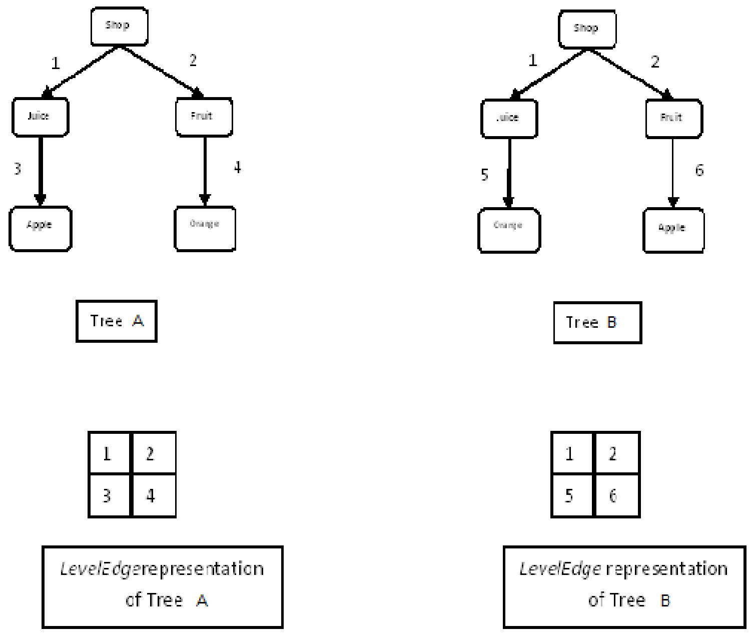


Figure 4: LevelEdge representation of homogeneous XML documents, Adapted from (Antonellis et al., 2008).

The algorithm of Antonellis summarized the information about each level by edge numbers instead of node numbers and that increased the performance when computing structural similarity because they included the relationship between nodes in the similarity evaluation. (Antonellis et al., 2008) then compared the structure of two XML trees through the comparison of edges at different levels but they reduced the quality of similarity measurement because they excluded semantic computation. For example, if we replace the tag name 'orange' with 'citrus', the result will be different as showed in figure (5).

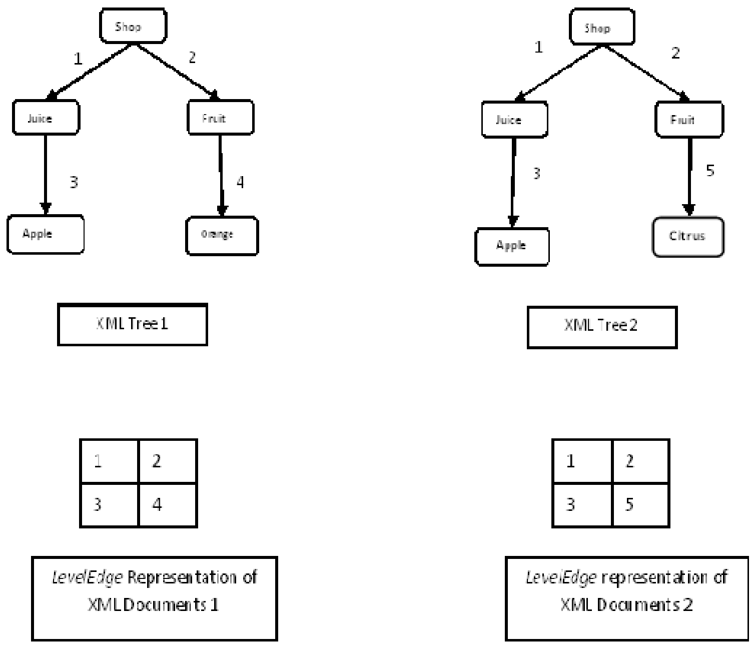


Figure 5: LevelEdge representation of XML documents, Adapted from (Antonellis et al., 2008).

(Guzman et al., 2013) argued that the model of (Antonellis et al., 2008) may have some drawbacks with the similarity computation of heterogeneous XML documents The former claimed LevelEdge theory compares identical edges which have the same semantics but they are structurally different (different levels).

Therefore, Guzman suggested a novel algorithm to measure the structure and semantics of XML documents as will be explained in the next model.

(Guzman et al., 2013) addressed the importance similarity measurement based on semantics and proposed a new algorithm to measures semantic and structural similarity between XML documents. They measure structure similarity between XML documents based on the LevelEdge algorithm and compute the semantic similarity using an online lexical English dictionary called WordNet that finds synonyms, such as 'worker' and 'laborer' (Alqarni and Pardede, 2013).

In spite of this exploitation of the tree representation to evaluate the similarity between XML data, it has some limitation as claimed by (Algergawy et al., 2011). For example, tree cannot represent a structural relationship called association which identifies that two nodes are not in the tree, they are conceptually in the same level in the tree (Algergawy et al., 2011). For example, the association relationship has a substitute mechanism to represent student information in English or in French, the tree cannot represents this kind of relationship. Therefore, another group of researchers (Hagenbuchner et al., 2005; Zhang et al., 2012) adopted a vector representation to measure the similarity between XML documents as will be explained in the next section.

3.2.2 Similarity Approaches Based on Vector-Based Representation

(Asghari and KeyvanPour, 2015) described a vector as an XML element feature that represents the element in the document such as, element name, element label, and element level in an XML tree. There are a good number of researchers who adopted that adopted vectors to calculate the similarity between XML documents. For instance (Nayak and Xu, 2006) assigned a number to each node in the XML tree and(Antonellis et al., 2008) in used vectors to compare the structural similarity as clarified in the previous section. (Zhang et al., 2012) used vectors to represent the structural and semantic features of XML documents as will be explained in section (3.3.3)

In this section, the common representations were described to facilitate the computation of similarity between XML documents. The similarity is evaluated based on three aspects which will be explained in the next section.

3.3 XML Similarity Measures

Measuring the similarity between XML documents has become the most important issue for some domains: XML document validation, XML schema matching (Algergawy et al., 2010) chemical compound differentiation (Deshpande et al., 2005), finding similar genetics in DNA sequences (Bell and Guan, 2003; Guan et al., 2004).

In this section three classes of model compute similarity based on three principles: Efficiency (Antonellis et al., 2008; Nayak and Xu, 2006; Yang et al., 2012), Effectiveness (i.e. XML node semantics) (Algergawy et al., 2010; Alqarni and Pardede, 2013; Zhao and Tian, 2013), and Efficiency (i.e similarity of XML node level) and Effectiveness (similarity of semantics) (Guzman et al., 2013; Mota et al., 2013; Thuy et al., 2013) are discussed. The models that employed the first approach will be explained in the next section.

3.3.1 Structure Similarity Measures

The size of data storage for some applications that adopted XML data has increased sharply with heavy use and measuring of similarity will become an issue. This issue is addressed by (Yang et al., 2012) who designed a new approach to enhance the similarity computation for these kinds of applications quickly.

(Yang et al., 2012) addressed the effect of structural similarity between XML documents on the performance of some applications such as clustering XML documents based on structural comparison (Choi et al., 2007; Lian et al., 2004) , chemical compound, similarity (Dehaspe et al., 1998), and extracting a similar genetics in DNA sequences (Bell and Guan, 2003; Mannila et al., 1997). In contrast to (Antonellis et al., 2008) in section (3.2.1) who matched edges based on numbers, (Yang et al., 2012) considered three aspects in their paper to evaluate structural similarity: first, the level of the node, repeated substructures and the type of node: simple node (i.e. terminal) node or complex node (i.e. sub-root node). They classified edges in XML trees as follows: Normal Edge NE, an edge which connects the parent node with its children. Topological Edge TE, an edge which connects the ancestor node and its descendant node. In contrast to (Antonellis et al., 2008), (Yang et al., 2012) assigned numbers to edges according to the edge classification. The former assigned number to each edge based on relationship between two nodes to compare the structural similarity. According to the edge classification of (Yang et al., 2012), the edges are weighted in similarity calculations depending on the edge classification. They assigned a weight to each type of edge as follows: the weight of complete or topological matching is 1, the weight of repeated matching is 1/2. Figure (6) is an example of the method of (Yang et al., 2012) for assigning weights to the edges of two trees. The edge $Apple \rightarrow Red$ in the XML tree 'A' is a completely matched edge to the edge $Apple \rightarrow Red$ in the XML tree 'B' and its weight is 1. The edge $Fresh \rightarrow Apple$ in the tree 'A' is topologically matched to the edge $Fresh \rightarrow Apple$ in the XML tree 'B' and its weight is 1 as well. The edge $apple \rightarrow Red$ in the XML tree 'A' is matched to number of $apple \rightarrow Red$ edges in the XML tree 'B' and its weight is ($\frac{1}{2}$).

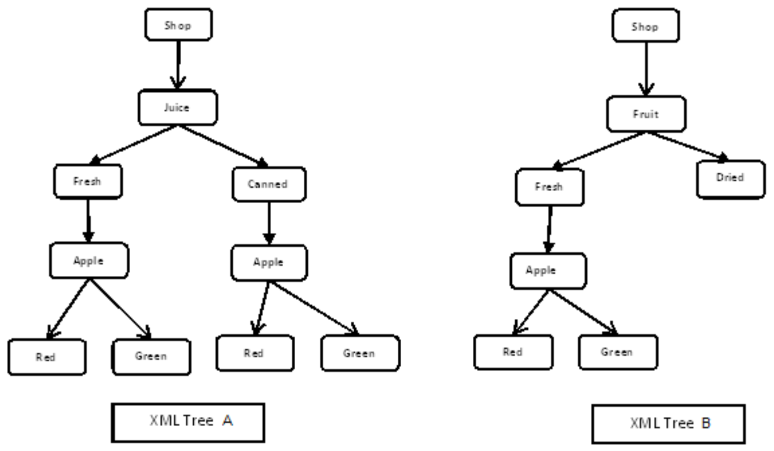


Figure 6: Tree representations of XML Documents.

In the second step, (Yang et al., 2012) defined two trees: pattern tree, that has the edges which are complete matching or topological matching or repeated matching. Another tree which has repeated matching edges in the pattern tree. For the last step they designed a mathematical operation to evaluate the structural similarity between trees based on the edge weights.

Similarity computation of XML document based on structure may result in incorrect information. For example, a user query "//Yellow-Apple" in the tree of figure (7) and the answer will be true if the semantic information is ignored. Other researchers analysed the efficiency of the XML similarity computation and proposed some algorithms to increase the quality of the similarity measurement as will be demonstrated in the next section.

3.3.2 Semantic Similarity Measures

Applications such as keyword search which will be detailed in chapter (4) (Bao et al., 2009; Chen and Papakonstantinou, 2010; Cohen et al., 2003; Guo et al., 2003; Kong et al., 2009; Li et al., 2007, 2004; Liu and Chen, 2007; Liu and Cher, 2008; Schmidt et al., 2001; Sun et al., 2007; Tatarinov et al., 2002; Xu and Papakonstantinou, 2005b, 2008; Zhou et al., 2010) suggest that XMLs schema matching depends on the effectiveness of the similarity computation which is expressed by the XML element semantics. In this section, recent models that were proposed to improve the effectiveness of similarity evaluation which relies on the XML element semantics will be explained starting from the approach of (Zhao and Tian, 2013).

(Zhao and Tian, 2013) considered the significant role of semantic similarity measurement between XML data and XML query languages. Similarity evaluation can enhance the performance of XML database management, for example in duplicate data detection which is one of the database management activities. (Mota et al., 2013) employed similarity measurement to improve the retrieval of information. In figure (7) for example, when a user queries Apple in the tree, then the returned answer will be error and may miss returning a potentially relevant answer to the user which is Red-Apple.

(Zhao and Tian, 2013) presented a new keyword query algorithm based on semantics which is called (SKSA). The proposed the algorithm works as illustrated in the figure (8).

They exploited a common semantic dictionary which is known as WordNet (see section 3.2.1) that is used to find synonyms through the interface JWI (MIT Java WordNet Interface) and save them in Semantic Space represented as a linked list. The approach of (Zhao and Tian, 2013) is to parse and label an

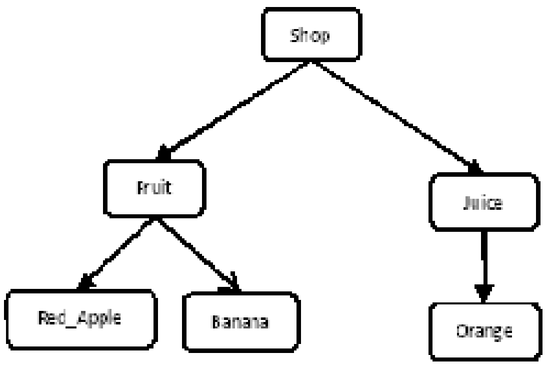


Figure 7: Tree Representation of XML Document.

XML document using the *Dewey Labelling Scheme* which will be explained in the chapter (5). (Zhao and Tian, 2013) analysed the user query into keywords and compared them with XML node labels. If the model does not find a matching keyword in the XML node labels, it will use 'JWI' to find the synonyms in the WordNet dictionary.

(Zhao and Tian, 2013) analysed the enhancement of the XML query processing by improving the similarity evaluation between the keyword query and XML document based on semantics. (Alqarni and Pardede, 2013) adopted a different method to improve the similarity measurement between XML schemas based on semantics as will be explained in the next approach. (Alqarni and Pardede, 2013) argued that it is possible to improve the efficiency of the schema similarity measurement by excluding dissimilar elements based on semantic features. They suggested that the involvement of dissimilar elements in the process of schema similarity evaluation is time consuming and reduces the effciency of matching process. So they developed a filtering system dubbed Internal Filtering

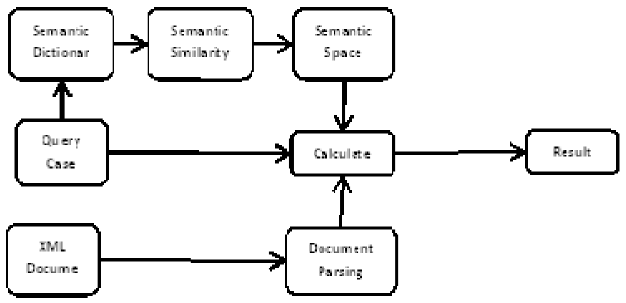


Figure 8: The Models of SKSA, (Zhao and Tian, 2013).

Threshold (IFth) to discard dissimilar elements from the similarity computation process based on the differences of internal features (semantics). (Alqarni and Pardede, 2013) found that a range of threshold values from 0.1 to 0.6 are the reasonable because it accounted for 40% to 60% of the total similarity value. Moreover, a threshold value higher than 0.6 may discard candidate elements from the similarity measurement. Excluding dissimilar elements may enhance the performance of the matching process but it impacts on the effectiveness of the process because of the inverse relation between them.

(Alqarni and Pardede, 2013) measured the quality of the system matching using *Precision, Recall, and F-measure* which depend on four principles which relate on the results of the manual matching (R_m) and automatic matching ((A_m) as shown in the figure (9).

(Algergawy et al., 2010) defined four principles to measure system matching effectiveness between XML schemas: *False negative* $(A) = (R_m) - (A_m)$, is the information which is relevant (manually identified) but not identified by the system. *True positive* is the information which is relevant (identified manually)

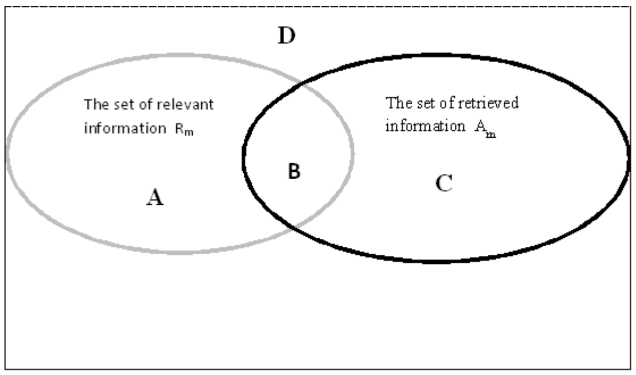

D
The set of relevant information R_m
The set of retrieved information A_m
A
B
C

Figure 9: Quality Measures Principles.

and was identified automatically $(B) = (R_m) \cap (A_m)$. Fa*lse positive* is the information which is retrieved by the system but is not relevant $(C) = A_m - R_m$. *True negative* (D), is the information which is not relevant and is retrieved by the system

Based on these principles, (Alqarni and Pardede, 2013) calculated *Precision* as the percentage of relevant information 'B' to the total retrieved information by the system 'B+C', $P = \frac{B}{B+C}$. For *Recall* evaluation, they computed the ratio of retrieved relevant data to the total relevant data $R = \frac{B}{A+B}$. (Alqarni and Pardede, 2013) assessed the *Fmeasure* using the formula

$$F - measure = 2 - \frac{P * R}{P + R} = \frac{2 * |B|}{(|B|+|C|) + (|A|+|B|)}$$

.

A combination of semantic and structural information in the similarity measurement is required in some applications such as the health care domain (Thuy et al., 2013) where, semantics is required for effciency of measurement whilst structure improves the efficiency of computation. An XML schema has both structural and semantic information and this has attracted the attention of a group of researchers to the consideration of XML schemas in similarity evaluation of XML documents as will be explained in the next section.

3.3.3 Structural and Semantic Similarity Measures (Hybrid)

High quality similarity measurement is required in some domains, for instance, searching for the accurate information from XML databases (Chen and Papakonstantinou, 2010; Cohen et al., 2003; Guo et al., 2003; Li et al., 2007, 2004; Schmidt et al., 2001; Tatarinov et al., 2002; Xu and Papakonstantinou, 2005b, 2008) and a healthcare domain (Thuy et al., 2013). This issue is analysed by (Guzman et al., 2013) as demonstrated in section (3.2.1) and much effort has been spent in the domain of similarity measurement based on structure and semantic computation starting from (Algergawy et al., 2010).

(Algergawy et al., 2010) modelled an XML schema as tree and considered two properties of elements in the XML schema: internal properties (element semantics) and external properties (element structure). The internal properties represent the semantics of XML elements and they adopt four characteristics: name, data type, constraints, and annotation.

(Algergawy et al., 2010) tokenized each element name into set of tokens and used the following formula to compute the token similarity of two elements

$$Nsim(T_1, T_2) = \frac{\sum_{t_1 \in T_1}[maxsim_{t_2 \in T_2}(t_1, t_2)] + \sum_{t_2 \in T_2}[maxsim_{t_1 \in T_1}(t_2, t_])]}{|T1|+|T2|}$$

where the first *maxsim* compared each token which belongs to the first element name with the set of tokens of the second element and vice verse with the second *maxsim*. To measure the token similarity, (Algergawy et al., 2010) also adopted the WordNet dictionary to measure the semantics similarity and tree edit distance to compute the syntactic similarity between the element names.

Table 1: Table of Data type Similarity, Adapted from (Algergawy et al., 2010).

XML $Type_1$	$Type_2$	Tsim
string	string	1.0
string	decimal	0.2
decimal	float	0.8
float	float	1.0
float	integer	0.8
integer	short	0.8

To evaluate the data type similarity, they exploited table (1) which was published on the website of (Biron, 2004).

The constraints of the XML element was a third factor measured by (Algergawy et al., 2010) to evaluate the internal similarity. The indicators *minOccurs* and *maxOccurs* represent the number of appearance the element in the document (w3schools, 2016). (Algergawy et al., 2010) adopted the following formulas to compute the (max,min) cardinality constraints of two elements.

$$CSim_(minOcurs) = 1 - \frac{|minOccurs\ of\ T_1 - minOcuurs\ of\ T_2|}{|minOccurs\ of\ T_1|+|minOccurs\ of\ T_2|}$$

$$CSim_(maxOccurs) = 1 - \frac{|maxOccurs\ of\ T_1 - maxOcuurs\ of\ T_2|}{|maxOccurs\ of\ T_1|+|maxOccurs\ of\ T_2|}$$

The last characteristic calculated is the element annotation (Asim) (Algergawy et al., 2010). This was done using a statistical study of the importance of the word in the document. The last step in computing the similarity of internal features between two elements was to combine the similarity measures $Nsim, Tsim, Csim, Asim$

(Algergawy et al., 2010) assessed the similarity of features of pairs of elements by evaluating the element in four contexts: child, leaf, sibling, and parent (2.6). They measured the child ($ChSim(El_1, El_2)$ and leaf similarity $LeafSim(El_1, El_2)$ between two elements using the following formula:

$$ChSim(El_1, El_2) = \frac{\sum_{i=1}^{i=k}[_{j=i}^{j=k'} maxInterSim(El_(1i), El_(2j)]}{max(|K|, |k'|}$$

.

where $maxInterSim(El_1, El_2)$ is the maximum similarity between two childrenEl_1 and El_2 and k and k' are the number of children of both elements El_1 and El_2 respectively. Then (Algergawy et al., 2010) computed the average similarity between the two sets of elements children using one of mathematical methods such as *cosine measure* or *Euclidean Distance*. The measurement of the sibling similarity was done by (Algergawy et al., 2010) through the same equation which were used previously to compute the children and leaf similarity, but it was used to compare the similarity between siblings of two elements. To evaluate the similarity of the ancestor contexts of two elements $PSim(P_1, P_2)$, (Algergawy et al., 2010) employ tree edit distance method to measure the similarity of each ancestor in the path of the element from the root. The final step in the computation of the external feature similarity of element is combining the similarity of the element's context:

$$\begin{aligned} ExterSim(El_1, El_2) = Combine_E(ChSim(El_1, El_2)), LeafSim(El_1, El_2), \\ SibSim(El_1, El_2), PSim(El_1, El_2)). \end{aligned}$$

Where $Combine_E$ is a function used to combine the values of external feature similarity. Finally, (Algergawy et al., 2010) computed the similarity of two elements based on semantic and structural similarity by combining the internal and external similarity measures.

In addition to (Algergawy et al., 2010) there is another group of researchers who were attracted to computing the similarity between XML Schemas based on semantics and structure. They thought that the involvement of semantic measures with structure may increase the effectiveness of the similarity evaluation as will be explained in the approach of (Thuy et al., 2013).

(Thuy et al., 2013) addressed the problem of measuring the resemblance of the two schemas in a health care domain. They considered a similarity measurement based on structure and semantic characteristics of the elements in both schemas. In contrast to the model of (Algergawy et al., 2010) who employed four factors in the semantic similarity computing, (Thuy et al., 2013) compared the semantic similarity of XML element from three views: data type, cardinality constraint, and element name. (Thuy et al., 2013) analysed the element type as either complex, which has children, or simple which is leaf. For complex type elements, they compared the structural similarity of their children as will be explained in the structural similarity measurement part. They assigned a value 0 to the data type similarity. If one of the elements is complex and the other is simple their attributes are different. To compute the data type similarity, (Thuy et al., 2013) used 12 constraint facets which are in the table (2) where 'u' indicates that the data type has this facet.

Based on this information, (Thuy et al., 2013) designed the following formula to find the values of the data types similarity which were saved in the table (3).

$$DSim(d_1, d_2) = \frac{|\{cf_i | d_1[cf_i] = d_2[cf_i], 1 \leq i \leq n_(cf)\}|}{n_(cf)}$$

Where, d_1 and d_2 are arbitrary data types presents in the table (2); *cf* is the list of constraining facets: *length*, *minLength*, *maxLength*, *pattern*, *enumeration*, *whitespace*, *maxInclusive*, *maxEnclusive*, *minExclusive*, *minExclusive*,

Table 2: Constraint Facets of Data Types, Adapted from (Dan, 2003).

Simple Data Type	length	minlength	maxlength	pattern	enumeration	white space	maxInclusive	maxExclusive	minExclusive	minInclusive	totalDigits	fractionDigits
String	u	u	u	u	u	u						
Date				u	u	u	u	u	u	u		
Decimal				u	u	u	u	u	u	u	u	u
Integer				u	u	u	u	u	u	u	u	u
Float				u	u	u	u	u	u	u		
Language	u	u	u	u	u	u						

totalDigits, *fractionDigits*. $n_(cf)$ is the number of constraining facets which in this case is twelve

For example, to compute the similarity value between integer and string data types the equation proposed by ((Thuy et al., 2013)) was:

$$DSim(integer, string) = \frac{3}{14} = 0.25$$

where string and integer data types have 3 similar facts out of 12 facets.

The next factor that (Thuy et al., 2013) took into consideration for the computation of semantic similarity is the cardinality constraint (i.e. occurrence of the element). They employed the occurrence indicators which are maxOccurs and minOccurs to define the maximum and minimum number of occurrences of the XML element may occur in XML instance.

$$CSim(e_1(min, max), e_2(min, max)) = \frac{(1 - \frac{|e_1.min - e2.min|}{|e_1.min + e2.min|}) + (1 - \frac{|e_1.max - e2.max|}{|e_1.max + e2.max|})}{2}$$

Table 3: Constraint Facets of Data types, Adapted from (Dan, 2003).

	XML String	Date	Decimal	Integer	Float	Language
String	1.00	0.25	0.25	0.25	0.25	0.50
Date	0.25	1.00	0.58	0.58	0.58	0.25
Decimal	0.25	0.58	1.00	0.75	0.58	0.25
Integer	0.25	0.58	0.75	1.00	0.58	0.25
Float	0.25	0.58	0.58	0.58	1.00	0.25
Language	0.50	0.25	0.25	0.25	0.25	1.00

Where $CSim$ specifies the cardinality constraint of the elements e_1 and e_2, min and max refer to minOccurs and maxOccurs respectively. (Thuy et al., 2013) assigned minOccurs 0 or 1 and maxOccurs 1 or 'unbound'. They found that the usual value of maxOccurs is undetermined 'unbound' and the number of occurrences of maxOccurs = 'unbound' is 5 times greater than maximum value of the definite maxOccurs. To find the value of 'unbound', (Thuy et al., 2013) designed the following equation:

$$d_1[maxOccurs = bound] = 5 * MAX(d_2[maxOccurs])$$

Based on the previous equations, (Thuy et al., 2013) computed the cardinality constraints for two elements as shown in the table (4).

Similarly, (Algergawy et al., 2010)) employed a mathematical equation to measure the cardinalty constraints between elements which was simpler than that in (Thuy et al., 2013). Therefore, the values in the tables are different.

(Thuy et al., 2013) adopted WordNet to compute the names of tags in order to increase the precision of semantics similarity measurement between two

Table 4: The similarity Values of Cardinality Constraints, Adapted from (Thuy et al., 2013).

	Min=0 Max=unbound	Min=1 Max=unbound	Min=0 Max=1	Min=1 Max=1
Min=0, max = unbound	1.00	0.5	0.67	0.17
Min=1, max = unbound	0.5	1.00	0.17	0.67
Min=0, max = 1	0.67	0.17	1.00	0.5
Min=1, max = 1	0.17	0.67	0.5	1.00

elements. They exploited a breadth-first search algorithm to find the synonyms of the element in the WordNet. Breadth first search is a search strategy that explores the tree from the root node outward in all directions level by level. The algorithm starts exploring vertices of tree graph that have distance (i+1), then vertices that have distance (i+2) and so on, until the target node is reached which is the shortest path to the required node (Korf, 2010). The model of (Thuy et al., 2013) use the Breadth first search is starting from the synonym set on WordNet of the element e_1 to the synonym set of the element e_2, and so on, until e_2 is matched. The approach will return one of two results: '0' if the names of the elements are not matched, or $0.9^{distance}$ if they are, where $Distance$ refers to the level of the element.

(Thuy et al., 2013) combined the values of $CSim, DSim, elementname$ of two elements to produce the semantic similarity using the following formula:

$$SeSim(e_1, e_2) = \alpha * NameSim(e_1, e_2) + \beta * DSim(e_1, e_2) * CSim(e_1, e_2)$$

Where, $SeSim$ represents semantics similarity, and and are weighted constants which are assigned values according to the consideration of the linguistic similarity in the semantics similarity computation. (Thuy et al., 2013) restricted their evaluation of the similarity computation by assigning 0.32 and 0.34 to the weighted constants and respectively. (Thuy et al., 2013) gave users a role in similarity measurement because they thought that it may lead to user satisfaction with the measurement process because the result will reflect the priorities of the user in the similarity computation.

In comparison to the approach of (Algergawy et al., 2010) who depend on the semantic information for structural similarity measure, (Thuy et al., 2013) computed structural similarity taking into account the linguistic similarity of the elements. They setup a threshold to decide whether the value of element names similarity is acceptable to be considered in the structural similarity. (Thuy et al., 2013) designed the following equation which considers the context of the element in the tree and tags similarity:

$$StSim(e_1, e_2) = \frac{sum\_links(e_1, e_2) + sum\_links(e_2, e_1)}{leaves(e_1) + leaves(e_2)}$$

where $leaves(e_1)$ and $leaves(e_2)$ are the number of descendants which are rooted in e_1 and e_2 respectively. $sum\_links(e_1, e_2)$ and $sum\_links(e_2, e_1)$ are the total number of descendants that have a similar tags from tree(e_1) to tree(e_2) and vice versa. For example, the rate of the structural similarity between 'Apple' in 'tree A' and 'tree B' in figure (10) can be computed as follows:

$$StSim(Apple, Apple) = \frac{1+1}{2+2} = \frac{2}{4} = \frac{1}{2}$$

After computing the elements structural similarity, (Thuy et al., 2013) computed structural and semantic similarity between two elements using the following equasion:

$$ESim(s,t) = \delta * SeSim(s,t) +)1 - \delta) * StSim(s,t)$$

where δ is a weighted value between 0 and 1.

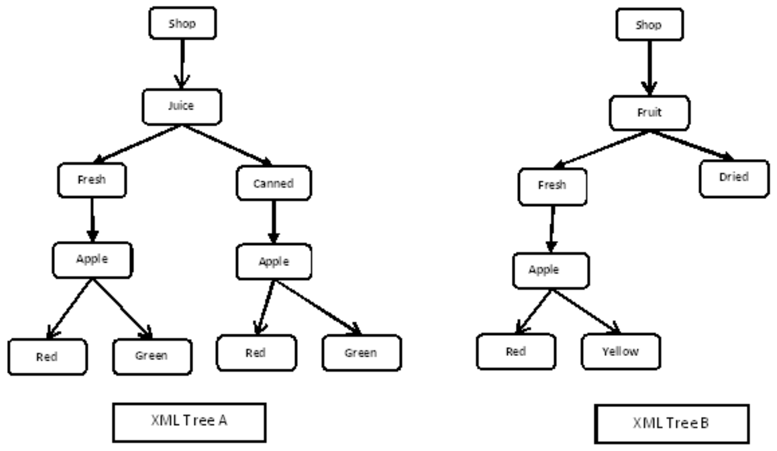


Figure 10: Tree Representations of XML Documents.

Finally, (Thuy et al., 2013) employed the following equation to evaluate the similarity between two schemas:

$$SchеSim(T_1, T_2) = \epsilon * \sum_{i=1}^{k} SeSim(e_1, e_2) = (1 - \epsilon) * TreeSim(T_1, T_2)$$

Where $ScheSim$ is the schema similarity of T_1 and T_2, ϵ is a weighted value between 0 and 1, k is the minimum number of elements in either tree T_1 or T_2, $SeSim$ is the semantic similarity between e_1 and e_2. (Thuy et al., 2013) did not clarify the role of $TreeSim$ in the equation, but it is possible that it is the structural similarity between trees T_1 and T_2.

Conclusion

A number of algorithms to investigate the recognition of the XML tree similarity have been presented in this chapter. To compare between XML documents, needs a uniform the representations of both documents. Therefore, two representation were explained: tree and vector, which are common forms that were exploited by researchers to evaluate the similarity of XML documents. After that, a third measurements was employed to compute the similarity to increase efficiency and effectiveness 'hybrid'. Some researchers focused on improving the similarity measurement by comparing the structures of the XML documents. Other researchers considered the semantics of the XML elements and adopted an online thesaurus such as WordNet to improve the semantic similarity by finding the synonyms of XML nodes. However, different groups of researchers adopted both structural and semantic similarity comparison to improve the performance of XML database management system in some domains such as health care.

As mentioned previously in this section, similarity measurement was exploited by (Zhang et al., 2003) to improve the query processing and fetch an accurate answer to user queries and this may lead enhance the performance of XMLdatabase management system. Increasing the performance of XML database management system by improving XML query processing has attracted the attention of many researchers as will be explained in the next section

4 Querying XML Documents

4.1 Introduction

There is extensive use of XML for data representation and transformation in different domains such as data warehousing (Chawathe et al., 1996, 1999; Cobena et al., 2002) cited in (Tekli et al., 2007), mathematics (Mathematics Markup Language (MathML)) (Carlisle, 2014), healthcare (Thuy et al., 2013). This high degree of adoption of XML data is attractive to software venders who have developed different kinds of XML databases such as: Timber, Oracle XML DB, MarkLogic Server, and the Toronto XML Engine (Jagadish et al., 2002; MarkLogic, 2016; of Toronto, 2002) as cited in (Le et al., 2015). Proposing a user friendly approach to answering user queries has attracted a good number of researchers, for instance, (Cohen et al., 2003; Guo et al., 2003; Li et al., 2004; Schmidt et al., 2001; Tatarinov et al., 2002).

In this chapter, structural and semantic similarity between a user query and an XML document will be studied, the type of XML query will not be considered. A number of query processing approaches will be explained which rely on a the Lowest *Common Ancestor LCA* which is a node in the XML tree that contains all query keyword as a descendant nodes (Liu and Chen, 2007). The drawbacks of this approach were addressed by a group of researchers who tried to improve the query processing and fetch the intended information to the user as will be seen later beginning with (Schmidt et al., 2001).

(Schmidt et al., 2001) proposed an algorithm based on the semantics of LCA to process XML query. The target of this model is to provide information for a user who is unaware of the XML schema and query structure. Based on this many researchers suggested other semantics to increase the meaningfulness of the answer returned as will be explained starting from the model of (Liu et al., 2013b).

4.2 Query Processing Based on Smallest Common Ancestor SLCA

(Liu et al., 2013b) claimed that the intent of a user query can be identified through the query structure which may play an important role in retrieving the information for a user precisely. So, they concentrated in their study on query matching with the XML document based on element content, that is the information between start and end tags (Microsoft, 2016) and query structure. However, they did not consider attributes which are the information that relates to the element (Microsoft, 2016) and data type. The approach of (Liu et al., 2013b) starts by splitting the user query into units based on element (tag) and content. For example a user writes the query 'Shop/ Juice/Orange' intending to find the node in the tree in the figure (11). The user query will be decomposed into three units: 'Shop', 'Juice', 'Orange' considering the node order from the root. Based on the constructed group, (Liu et al., 2013b) defined a term Keyword Query With Structure QWS which is a combination of two or more of groups and describes the structure of the candidate answer to the user query. For example, the candidate QWS which consist of two groups are: 'Shop, Juice', 'Shop, Orange', and 'Juice, Orange', A similar procedure will be implemented to construct QWS of three groups 'Shop Juice Orange'.

To evaluate the results of QWSs, (Liu et al., 2013b) exploited the XSeek algorithm which was designed by (Liu and Chen, 2007) and is based on a semantic entity called *Smallest Lowest Common Ancestor* SLCA which is based on LCA. A SLCA node is a LCA node in the tree which contains all query keywords and

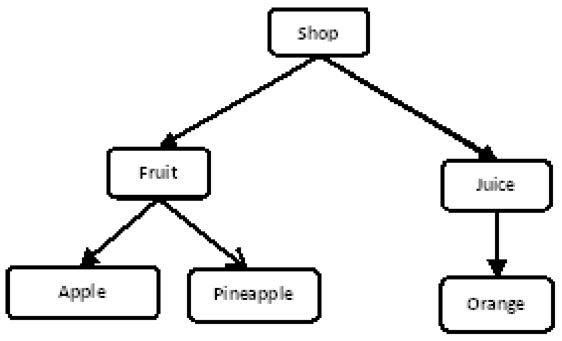


Figure 11: Tree Representation of XML Document.

none of its descendant is an LCA (Xu and Papakonstantinou, 2005b)). After determining the SLCA node of the QWSs, the approach of (Liu et al., 2013b) then computes the matching answers of QWSs and return the most common answer to the user.

More work has been done based on the use of the SLCA node to retrieve the required answers for the users in addition to (Liu et al., 2013b). (Lin et al., 2014) proposed a new model to cover the weaknesses of the previous work (Lin et al., 2011) called *Relevant Matches with NOT* 'RELMN' that did not work properly in some cases in processing a user query as will be explained in the next method.

4.3 Processing XML query contains NOT Operator

(Lin et al., 2014) described that RELMN has two steps: first, it searches for an SLCA node in a tree that satisfies a user query but its descendant nodes do not by a 'NOT' operator which known as 'Negative Keyword'. In the second step, RELMN will prune a subtree of the identified SLCA to a sibling subtree which has more positive keywords or less negative keywords. Let us consider the result of the user query " 2015 ∧ Product ∧ ! 350ml" from the tree in figure (12). The SLCA node is 'Juice1' and the subtree 'Canned Juice1' will be excluded by the subtree 'Canned Juice2' because the latter has more positive keywords than the former.

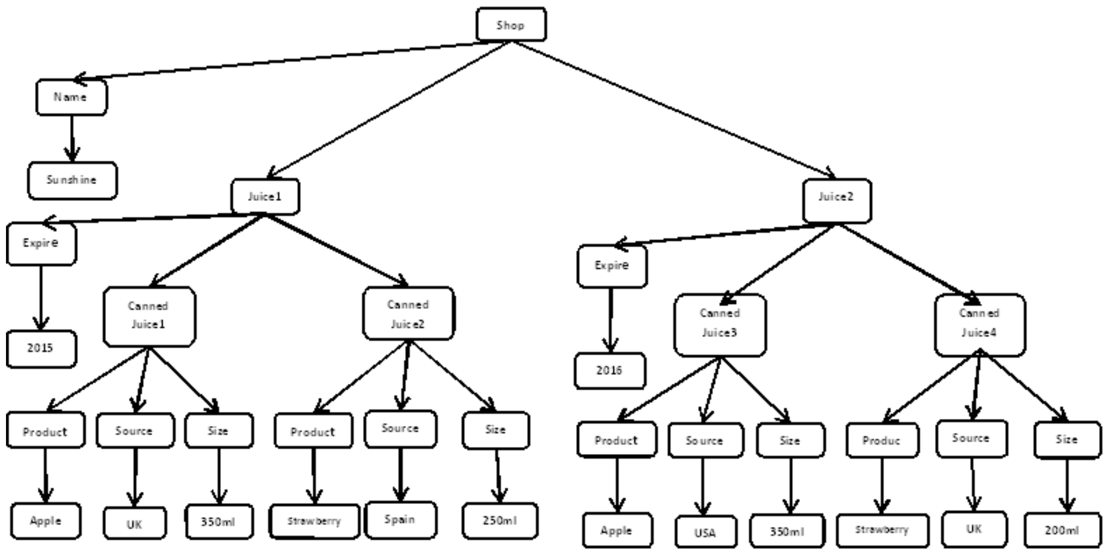


Figure 12: Tree Representation of XML Document, adapted from (Lin et al., 2014).

(Lin et al., 2014) illustrated that RELMN has drawbacks in the processing of a user query that has a 'NOT' operator which may cause a 'false negative' answer, an answer that is detected manually but not by the system (section 3.3.2). For example, let us suppose user writes the query 'Sunshine ∧ Product ∧ UK ∧ !250ml', the manual determination of the SLCA node is 'Shop' and the query keywords are leaves of the nodes 'Canned Juice1' and 'Canned Juice4' repectively. However, RELMN will fetch 'Canned Juice4' only, because the node 'Juice1' has a negative word '250ml' as a descendant of 'Canned Juice2' and the node 'Juice1' will be discarded by the sibling 'Juice2'. (Lin et al., 2014) proposed a novel approach called *Valid SLCA* to to cope with the defect in RELMN by excluding the descendant of negative elements rather than a subtree that has a negative child. For instance, the result of the query 'Sunshine $\wedge$ Product $\wedge$ UK $\wedge$! 250ml' will be 'Canned Juice1' and 'Canned Juice4' which match the manual answer.

(Lin et al., 2014) analysed the enhancement of the processing of queries that have a NOT operator based on the semantics SLCA. However, the determination of the SLCA node is considered by another researchers to improve query efficiency as will be seen in the next approach by (Zhou et al., 2012a).

4.4 Improving of XML Query Processing Using XML Labelling Scheme

(Zhou et al., 2012a) studied the issue of computation of the SLCA node quickly to responde to the query efficiently. They adopted a *Dewey Labelling Technique* (explained in the chapter (5)) to label tree nodes and define the SLCA node which has the query keywords. (Zhou et al., 2012a) partitioned the label list of the XML tree into lists and for each list they calculated LCA node which was a root node of a subtree that has all query keywords (Guo et al., 2003) one of its children may be a SLCA node (Zhou et al., 2012a). The novel approach is that they inspect each keyword 'W_i' in each subtree and examine the subtree that has 'W_i' to see if it has minimum number of query keywords and return the SLCA node if it does. If the tree 'T' has all query keywords and none of its subtrees the algorithm will fetch the root node of the tree 'T' as an SLCA node.

(Zhou et al., 2012a) studied the efficient response to a user query based on SLCA which is based on LCA. Another team of researchers addressed the determination of the LCA node based on range value queries.

4.5 Using of XML Grammar to Enhance the XML Query Processing

(Zeng et al., 2013) stated that the use of LCA and its variant SLCA fetch exact match subtrees for the user query but they do not support range queries. They defined two terms in their algorithm: *Normal Term* 'T_n' which refers to the exact match elements of the user query and *Range Term* T_r which refers to a range of values specified in the user query. To answer a range query, (Zeng et al., 2013) proposed a new approach which consists of two steps: they adapted the formal existence grammar of keyword search methods which is :

$$T \rightarrow anonemptystringwithoutspace$$

$$Q \rightarrow T \mid QT$$

where T represents a 'Term', Q represents a keyword query, $\rightarrow$ represents 'as defined', and | represents 'or'. The adapted grammar is

$$T \rightarrow anonemptystringwithoutspace$$

$$T_r \rightarrow T :< T \mid T :> T \mid T :<= T \mid T :>= T \mid T : T - T$$

$$T_n \rightarrow thoseTarenotT_r$$

$$Q \rightarrow T_n \mid T_r \mid QT_n \mid QT_r$$

where T_r represents a range term, T_n represents a normal term, and the symbols :<, :<, :<=, :>= have the same meaning as the conventional symbols

$<, >, <=, >=$ but they are preceded by : to distinguish them from the predefined symbols. The second step in the model is that they designed a novel index called *Point and Range Hybrid-Entrance Index* which has two entrances to support matching of both normal and range terms. For the point match entrance, they used a B+ tree index as seen in the figure (15) which contains the relevant node labels of the XML tree in figure (13). For the range terms, (Zeng et al., 2013) used a hash table in which each entrance represents the range node name and the relevant labels in the XML tree as shown in figure (14) that contains the node names and its labels in the XML tree.

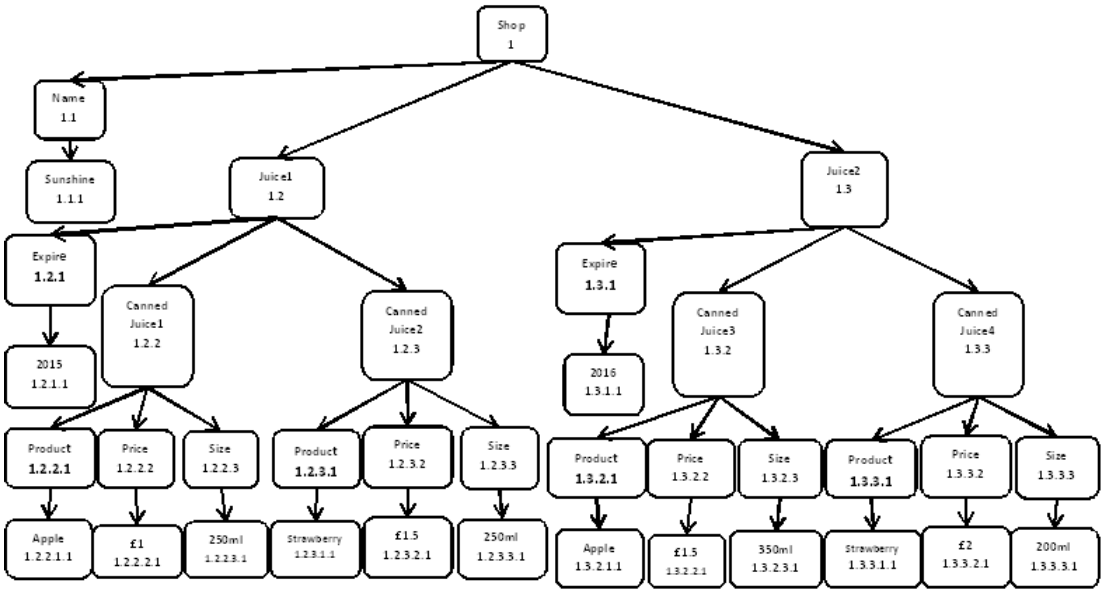
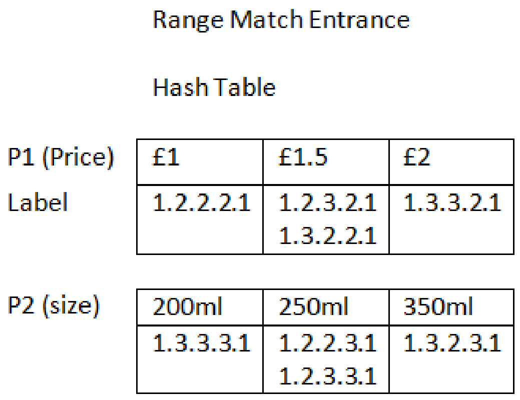


Figure 13: Ordered Labelled XML Tree.

For instance, when a user writes the query 'Strawberry price :< £2'. The novel approach will look for the node name 'Strawberry' in the point match entrance and retrieve the label(s) '1.2.3.1.1' and '1.3.3.1.1' for the intended

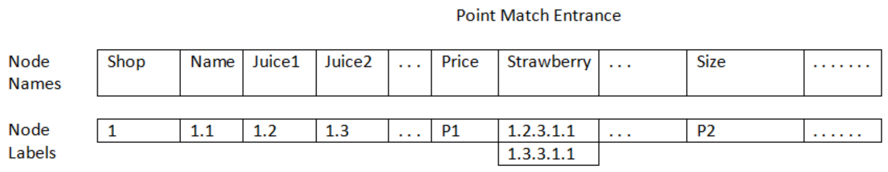


Range Match Entrance

Hash Table

P1 (Price)	£1	£1.5	£2
Label	1.2.2.2.1	1.2.3.2.1 1.3.2.2.1	1.3.3.2.1
P2 (size)	200ml	250ml	350ml
	1.3.3.3.1	1.2.2.3.1 1.2.3.3.1	1.3.2.3.1

Figure 14: Range Match Entrance.



Point Match Entrance

Node Names	Shop	Name	Juice1	Juice2	. . .	Price	Strawberry	. . .	Size	. . . .
Node Labels	1	1.1	1.2	1.3	. . .	P1	1.2.3.1.1 1.3.3.1.1	. . .	P2	. . . . . .

Figure 15: Point Match entrance.

node name. Based on the adaption in the grammar in the first step, the range value of price will follow the pointer 'P1' and fetch the node label whose value matches the user requirement in the query which shares the prefix node label of 'Strawberry' to provide the LCA node which is 'Canned Juice2' that has the label '1.2.3'.

(Zeng et al., 2013) designed a new grammar to deal with range queries. However, another team of researchers has a different view of the accuracy of LCA in providing the user with the intended answer.

4.6 Dominance Semantics to Improve the Precision of XML Query Processing

(Nguyen and Cao, 2012) claimed that the semantics of LCA has low precision as it returns a large number of irrelevant answers. To increase the precision, they designed a novel algorithm which starts from computing the mutual information between two nodes which is a measure of the relationship between two nodes. The mutual information consist of two parts: the probability of accessing the node in the XML tree based on the prefix path of the nodes. For instance, the probability of accessing the node '200ml' in the figure (13) based on the path '/Shop/Juice/Canned Juice' is $\frac{1}{2}$, where, the other path leads to '350ml'. The second part is finding the join probability between two nodes. For example, the join probability between '2' and '200ml' from the path '/shop/Juice/Canned Juice/' is '$\frac{1}{2}$2, but the join probability between '£2' and '250ml' through the path '/Shop/Juice/' is '$\frac{1}{4}$. (Nguyen and Cao, 2012) processed the the resulting information to measure the mutual information between two nodes. After they measured the relationship between two nodes in XML tree, they employed the evaluation and suggested a new semantics dubbed Dominance Lowest Common Ancestor DLCA which is based on mutual information. (Nguyen and Cao, 2012) define DLCA answers as a tree whose nodes are have a strong relationship. (Nguyen and Cao, 2012) did not depend on the top-k ranking algorithm which is defined by the same author as a ranking function used to rank the answers to user's queries by a measure of the relevance of the answer to the query keywords. Instead they ranked them based on the measurement of the relevance information. They claimed that the top-k model is costly, so they proposed a new ranking strategy based on the score of the dominance relationship to provide the user a more relevant answer.

(Nguyen and Cao, 2012) addressed the issue of providing a large number of irrelevant answers to the user and proposed a new approach to solve it. On the other hand, a different groups of researchers analysed the problem of returning an error message when an expert user query the XML database as will be demonstrated in the next model.

4.7 Processing Error Messages in the Query Processing

During a normal process of XML tree update, some of information in the tree will be changed. When a user tries to query the database for a word which has been deleted, the result will be an error message and this problem was investigated by (Bao et al., 2015) and is called the *MisMatch* Problem. (Bao et al., 2015) followed a number of steps in their approach starting from finding the data type of the intended node in the user query from the schema for the XML tree in contrast to the approach of (Nguyen and Cao, 2012) which is based on mutual information. They matched the query keywords with the nodes in the subtree starting from the LCA node which made it possible to predict the data type of the target keyword. Forexample, when a user wants to find the price of canned apple juice with the size 250ml in the figure (13), he/she will write the query '/Canned Juice/Product/Size = 250ml/ price'. The new algorithm will predict the data type of the user's intended answer by comparing the path of each keyword in the query beginning from LCA node 'Canned Juice', predict the targeted node and then extract the node's data type from schema. Depending on the data type of the intended element, (Bao et al., 2015) evaluated the user query and detected the MisMatch problem. To give a reason for the MisMatch problem to the user, they defined a term called *Distinguishability* which compares the similarity of the keywords in the descendant nodes of the

LCA node. If the distinguishability of a keyword is higher than a predefined threshold, it is important to be match it, but if it is less than the threshold it is the reason for the MisMatch problem and it needs to be replaced.

The novel model tries to provide an approximate as an alternative answer by replacing the MisMatch node with a node of the same type in different subtree. For instance, the query '/Canned Juice/Apple/Size = 350ml/price' will result in the MisMatch problem because this product not available in the shop. Thus, (Bao et al., 2015) suggested a substitute answer which is '/Canned Juice/ Apple/Size = 250ml/price' where all keywords matched the user query except 'Size' and this is the suggested to the user. (Bao et al., 2015) relied on the approximate answer where its distinguishablility is less than the threshold to reduce the number words replaced and provide the user fewer answers in a short time. Similar to (Nguyen and Cao, 2012)), (Bao et al., 2015) suggested a new ranking method to evaluate proposed answers which are based on three factors: the number of keywords need to be changed, a comparison of the dimensions of the suggested answer and the intended answer from the LCA node and the distinguishability score.

(Bao et al., 2015) studied giving the user the benefit of information in case of a missing. However, another team of researchers analysed the assistance to a non expert user in quering XML documents as will be illustrated in the next approach.

4.8 Semantics for Non-Expert Users for Querying XML Documents

(Agarwal and Ramamritham, 2015) studied the problems that relate to existing query algorithms based on LCA method and its variants: SLCA and ELCA.

SLCA semantics is defined in the approach of (Liu et al., 2013b). ELCA semantics which is a node in the tree that has all occurrences of query keywords after excluding all descendant subtrees that have the same set of key-words (Xu and Papakonstantinou, 2005a). They claimed that these semantics do not always provide useful answers to non-expert user where, they determine the LCA node and retrieve the descendant nodes that match the query keywords. In contrast, (Agarwal and Ramamritham, 2015) designed a novel theory called *Generic Keyword Search* 'GKS' which retrieves useful information from any node in XML tree based on part of query keywords. They called the nodes that contain part of the query keywords as *Least Common Entity* 'LCE' and they used variable 's' to determine a least number of query keywords (either 2 or 3) to be searched for in the XML tree. When the number of keywords in the subtree of the LCE nodes is equal to the number of query keywords, then the LCE node will be considered as a SLCA. For example, three users write the three queries Q1, Q2, Q3 to retrieve information from an XML tree in the figure (16) and 's=2'.

$Q1 = (Price, Apple, Local)$

$Q2 = (Price, Apple, Imported)$

$Q3 = (Price, Apple, Local, Import)$

The answers of these will be as follows in the table (5).

It is clear that the LCE approach fetched a larger number of results than either ELCA or SLCA.

The model of (Agarwal and Ramamritham, 2015) is recently published, so it has not yet been analysed by other researchers. From the previous example the results of Q2 and Q3 in the table (5) demontrate that the LCE approach brought a large number of results but some are irrelevant. The user requested the price of an imported apple in the query Q2, but the new semantics fetched

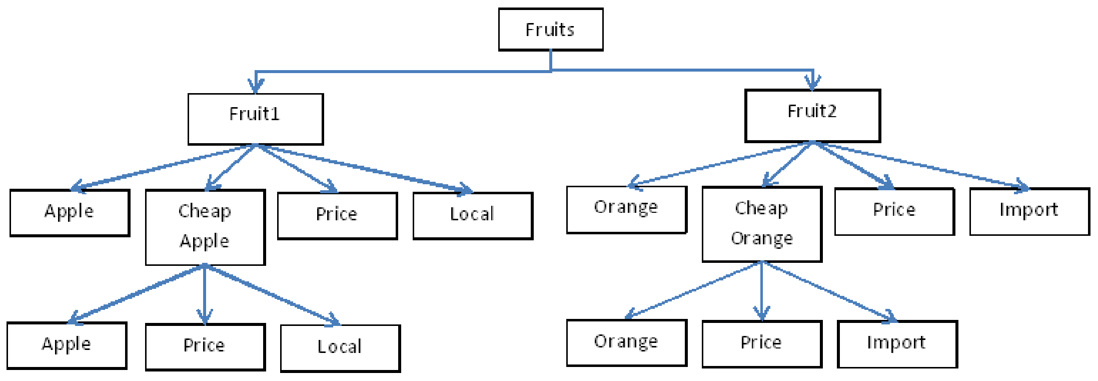


Figure 16: Tree Representation of XML Document.

Table 5: The results of the queries based on LCE, ELCA, and SLAC.

	LCE	ELCA	SLCA
Q1	Fruit1, Cheap Apple	Fruit1, Cheap Apple	Cheap Apple
Q2	Fruit1, Cheap Apple, Fruit2, Cheap Orange	Null	Null
Q3	Fruit1, Cheap Apple, Fruit2, Cheap Orange	Fruits	Fruits

information about the local apple and imported orange. In Q3, where the user requests data which relates to the imported and exported apple the proposed approach supplied the user with information related to the orange. This leads to reduced recall and increased precision of the query processing algorithm.

Conclusion

Many methods were studied in this chapter which relied on a comparison of structure between a tree or subtree and the user queries in order to supply the users with the information they would like. As can be shown, most of the query processing approaches adopted the semantics LCA which bring the user a root node which has all query keywords. Exploiting this approach, researchers found it has drawbacks and they proposed variants such as SLCA and ELCA which are derived from LCA to increase the performance the query processing. To overcome these faults, a number of alternative approaches were proposed for instance LCE and DLCA. Another team of researchers addressed the problems that relate to the user query where the 'NOT' operator is included in the query.

However, another group of researchers studied simplifying query processing for users who are unfamiliar with information retrieval. They did not include semantics in the query processing, but focused on a structural comparison to simplifying the query writing for users who are unfamiliar with XML query processing and they designed methods to help these users access the information they wanted. More studies have aimed to improve the XML query processing based on a structural comparison between the query and the XML document using numbers, characters, and a combination of both. These numbers and characters describe the path from the root node to the required information as will be explained in the next chapter.

5 XML Labelling Schemes

5.1 Introduction

(Assefa and Ergenc, 2012; Duong and Zhang, 2008; Fu and Meng, 2013) defined XML labelling as a process of assigning each node in the XML tree a unique label which expresses information about the node, such as, its position and the relationships. Thus the labels are both an index to the nodes of the XML tree and a sumary of relationships between them. (Assefa and Ergenc, 2012; Duong and Zhang, 2008; Fu and Meng, 2013) state that a labelling scheme has become an important requirement for an XML database due to the amount of data involved. They argued that a good labelling scheme can be assessed in terms of: compactness, the labels should be small to fit in main memory, dynamism, updates to the XML should not require relabelling existing nodes and flexiblibility the ablity to represent all kind of node relationships.

In this chapter three labelling techniques will be demonstrated: the Interval-Based Labelling Scheme, the Prefix-Based Labelling Scheme and the Vector-Based Labelling Scheme. Early labelling schemes were developed for static XML documents where the XML tree was not expected to change (Dietz, 1982; Fu and Meng, 2013; Haw and Lee, 2011). In trees that are updated, a relabelling process is required to maintain the performance of query processing (Yun and Chung, 2008)). In this chapter, a group of labelling schemes will be explained for central XML documents (neither compressed nor distributed) to clarify the process of both statically and dynamically labelling a document. It will be noticed that in what follows the nodes in some figures contain numbers; these numbers are to help distinguish one node from another and are not labels.

5.2 Interval Labelling Schemes

(Haw and Lee, 2011) claimed that the earliest labelling scheme is the model of node encoding proposed by (Dietz, 1982). (Dietz, 1982) proposed an algorithm which assigns each node a label that consist of two integers based on the node's position in the preorder and postorder of traversal of a tree data structure. In this approach a node labelled (x, y) is ancestor of node labelled (v, w) when it appears before (v, w) in a preorder traversal tree and after in postorder travesal as shown in figure (17).

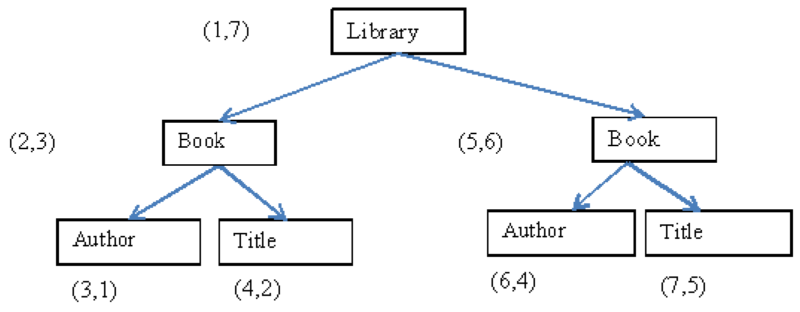


Figure 17: Preorder/Postorder-Based Labelling Scheme.

The labelling scheme of (Dietz, 1982) can thus represent Ancestor-Descendant 'A-D' relationships (see section (2.6)) but cannot represent other structural relationships such as Parent-Child 'P-C' (section (2.6)). So another technique called Containment Labelling was designed to identify this relationship (Xu et al., 2007). The new technique identifies each node with a label which consist of (start position, end position) which is sufficient to identify the range of its descendant nodes (Subramaniam and Haw, 2014; Zhuang and Feng, 2012). A common example of this labelling category is the approach of (Zhang et al., 2001). For two nodes, x labelled $(x.start, x.end)$ and y labelled $(y.start, y.end)$, x is an ancestor of y if $x.start < y.start$ and $x.end > y$.end. (Xu et al., 2007) also labelled nodes with their level and so encoded another property. A node labelled $(x.start, x.end, x.level)$ is a parent of a node labelled $(y.start, y.end, y.level)$ if x is an ancestor of y and $x.level = y.level - 1$. This is illustrated in figure (18).

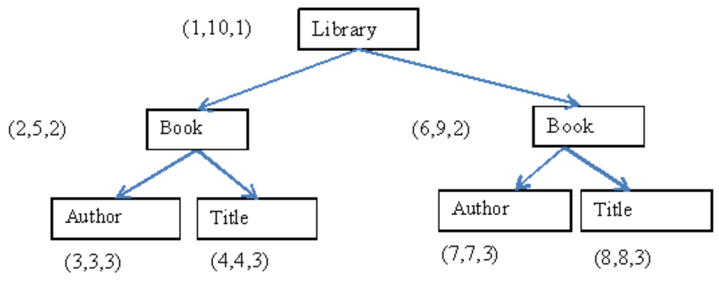


Figure 18: Interval-Based Labelling Scheme (Using Containment Property).

More effort in the defining the node relationships in an XML tree was made by (Subramaniam et al., 2014) who developed a new labelling scheme to represent structural relationships. (Subramaniam et al., 2014) designed the labelling scheme called *ReLab*. Each node is labelled (*level, ordinal, rID*) where level is the node level starting from '0' at the root node. ordinal is a unique identification number assigned to a node in preorder tree traversal of the tree. Finally rID, the region identifier is the ordinal of the rightmost descendant node except for leaf nodes who take the rID of their parent. The scheme is illustrated in figure (19).

The node labelled (2, 3, 4) in figure (19) is the child of the node (1, 2, 4) because the ordinal of the child node is between the ordinal of the parent and its rID and so it is a descendant and the level is one more than the parent's

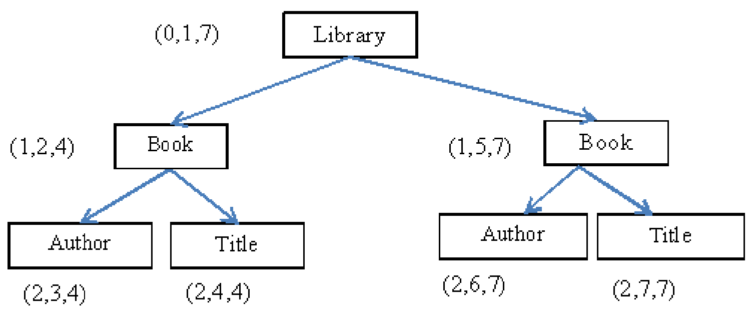


Figure 19: ReLab Labelling Scheme.

level so it is a child.

A common drawback of this category of labelling scheme was addressed by (Fu and Meng, 2013). They identified that although they identify *Ancestor-Descendant* relationships and *Parent-Child* they are unable to determine *Sibling Relationships* SR (see section (2.6)) and the *Lowest Common Ancestor* LCA node. A node in XML tree is said to be the LCA when all query keywords occur in its descendants (Schmidt et al., 2001). (Fu and Meng, 2013) designed a new labelling scheme intended to identify this case in the next labelling scheme approach.

(Fu and Meng, 2013) proposed a new labelling scheme model called *iTriple Code* that assigns a quaternary label to each node during a single Depth-First traversal of the tree. The label consists of (*start, end, parentid, pt*) where *start* and *end* are as explained previously, *parentid* is the identifier of the parent node, and *pt* is a table that contains a list labels of ancestors of the parent node. The novel algorithm can determine the LCA node as it will starts the traversal from a leaf node and follow the order of parent's 'ids' which are saved in a table.

The labelling schemes described so far are static interval-based labelling schemes which are unable to maintain the nodes relationships during the update process as nodes are assigned numbers sequentially (Yun and Chung, 2008). They pointed out that the preservation of gaps when numbring nodes for the insertion of further nodes in the future may not be sufficient to avoid relabelling of nodes as consequence rapid change of XML document. (Yun and Chung, 2008) explained that if the preserved space is not sufficient for the inserted nodes overflow problems cannot be avoided. To solve this problem, (Yun and Chung, 2008) proposed a new data structure to label trees called a *Nested Tree Structure* suitable for dynamic XML documents but still based on the interval labelling scheme.

The (Yun and Chung, 2008) labelling scheme is another interval-based labelling scheme, so it labels nodes with a tuple of (*DocID, StartPos, EndPos, LevelNum*). The DocID is the identifier of the document, StartPos and EndPos are assigned either by counting the number of words from the start element to the end element in the document or by depth first traversal of the XML tree. LevelNum is the depth of an element in a document. The positions can include space for new nodes.

A node N1=(D1, S1, E1, L1) is parent of the node N2=(D2, S2, E2, L2) when D1=D2 so both nodes are part of the same document, S1<S2, E1>E2 and L2=L1 +1. The node N1 is an ancestor to the node N2, when D1=D2, S1<S2, E1>E2 and L1<L2.

(Yun and Chung, 2008) studied three cases of XML tree update: the reserved space sufficient for the newly inserted tree, the reserved space is not sufficient for the inserted tree, and the reserved space is zero. In the first case of the tree update, the model of (Yun and Chung, 2008) assigns labels to the inserted tree sequentially and the labels of the original nodes will not be effected as shows in figures (20) and (21).

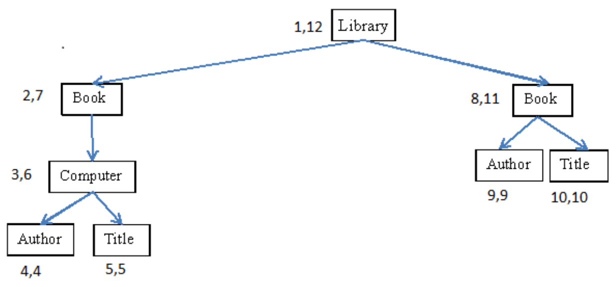


Figure 20: The Reserved Space is sufficient and Before Tree Update.

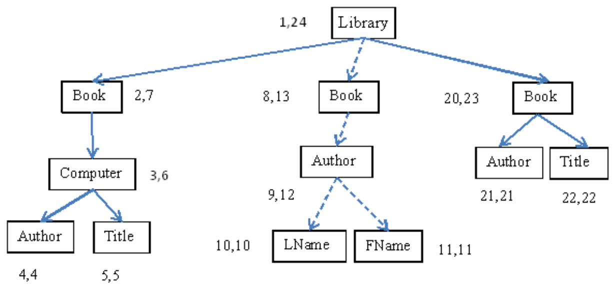


Figure 21: The Reserved Space is sufficient and After Tree Update.

In the second case, only one integer value is needed to number the inserted tree and the labels of the original tree do not need to be changed as shown in figures (22) and (23).

As can be seen from figure (23), the labels of original nodes of the tree are not affected by the update node because the novel algorithm assigns a new label which is 1,3 to the root node (Author) of the new subtree that is considered as a child and the nodes of the inserted tree are allocated with a new labels: 2,2 and 3,3.

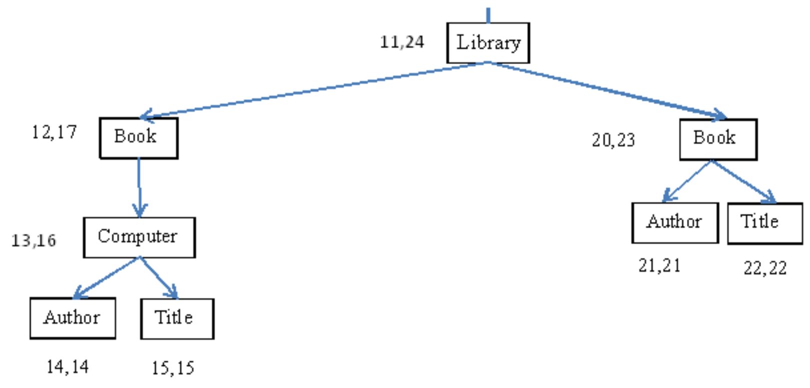


Figure 22: The Reserved Space is not Sufficient for the Whole Tree Before the Tree Update.

The last case is the most complex because the reserved space is zero and some nodes of the original tree have to be relabelled after tree update. The model of (Yun and Chung, 2008) solved this problem by combining the original tree with the inserted sub-tree and relabelling the nodes of the combination of trees rooted from the parent of the inserted sub-tree. For example, the label 11, 22 of the root node *Library* in figure (24) was changed to 1,18 in figure (25).

(Haw and Lee, 2011; Lu and Ling, 2004) claimed that an interval-based labelling scheme is flexible and represents all type of node relationship. However, XML trees sometimes change frequently and node relabelling cannot be avoided, so it is not an ideal approach in this case. Therefore, a number of labelling schemes were proposed to represent all nodes relationships and to avoid relabelling nodes during tree update as will discussed in the next sub-section.

Furthermore the time taken for label generation in Interval schemes grows exponential because they visit each XML node twice (Sans and Laurent, 2008). The need for a labelling scheme that produces labels in linear time and with the

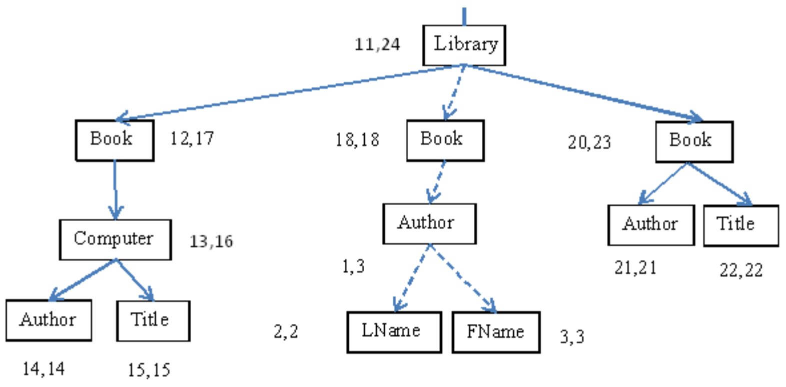


Figure 23: The Reserved Space is not Sufficient for the Whole Tree After the Tree Update.

representation of the hierarchical structure of an XML tree is badly needed as will be explained in the next section.

5.3 Prefix Labelling Scheme (Dewey)

The technique underlying this category of labelling scheme is similar to the Dewey Decimal Coding which is used by librarians (Sans and Laurent, 2008). Using this scheme it is possible to find structural relationship from the label (Assefa and Ergenc, 2012). This category of labelling scheme encodes the label of a node's parent as a prefix to its own label. The parent's and nodes own labels are separate by a delimiter ',' or '.' (Assefa and Ergenc, 2012; Liu and Zhang, 2016; Tatarinov et al., 2002).

The ordering of XML database is attracted many researchers (Duong and Zhang, 2008), (Assefa and Ergenc, 2012), (Liu et al., 2013a), and (Subramaniam et al., 2014) to improve the performance. (Tatarinov et al., 2002) and (Wu et al., 2004) classified XPath queries based on the order-sensitivity of the query:

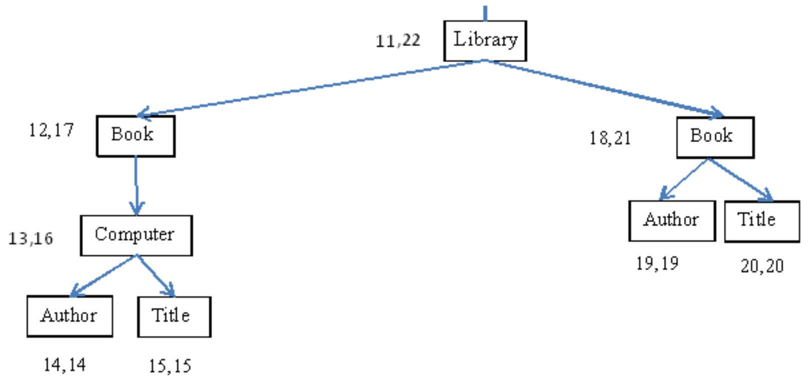


Figure 24: The Reserved Space is zero Before the Tree Update.

Preceding, Following. Queries in this class targeted all nodes before or after the context node excluding any descendant or ancestor nodes.

For Example, in figure (26) the query *Library/Book[2]/Following::Author[2]* will retrieve the information following *Library/ Book[2]* in figure (26). *Preceding-sibling* and F*ollowing-Sibling* queries will fetch the following or preceding siblings of the context node. For instance, the *query Library/ Book[2]/Following-sibling::** will select all following sibling nodes of Book 2 in figure (26). Position queries will select the information from a specific node. For example, the query *Library/Book[2]* will retrieve all the information related to the element Book 2 in figure (26).

(Tatarinov et al., 2002) analysed the labelling of an XML document to answer a different kind of query and proposed an algorithym which is known as *Dewey Order*. The novel model combines two techniques of node labelling that are designed by the same author: *Global Order*, where the node is assigned a label which is a node's global order in an XML tree as shown in the figure (27) and Local Order, where the node's label represents the node's order among its

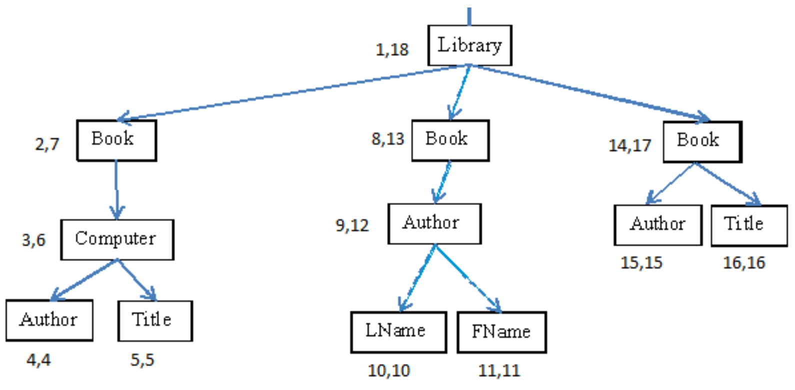


Figure 25: The Reserved Space is zero After the Tree Update.

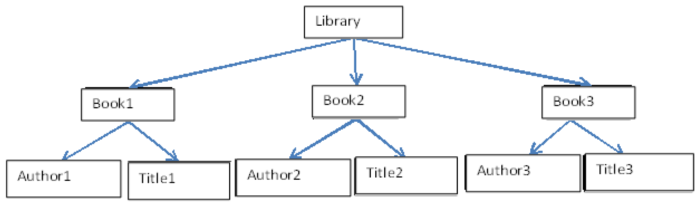


Figure 26: Tree Representation of XML Document.

siblings as shown in the figure (28). The *Dewey order* model merged these two labelling schemes as shown in figure (29).

Dewey Order encoding represents the nodes relationship in the nodes' labels expressively. However, (O'Neil et al., 2004) argued that the the model of (Tatarinov et al., 2002) is unsuitable for dynamic XML documents. So that, the former suggested a new labelling scheme to cover the drawback of Dewey Order labelling scheme as will explained in the next model. (O'Neil et al., 2004) proposed a new labelling scheme called *ORDPATHs* to maintain the order of of XML tree during update and to improve the query performance.

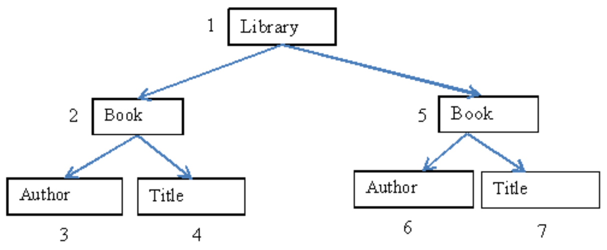


Figure 27: Global Labelling Scheme.

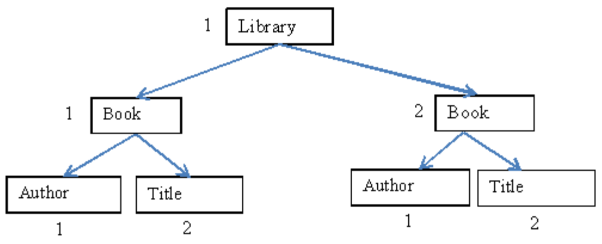


Figure 28: Local Labelling Scheme.

They employed only odd numbers to encode the initial nodes in the XML tree as shown in figure (30) where the children of the root node *Library* were assigned the labels (1,1) and (1,3). (O'Neil et al., 2004) exploited the negative numbers to label a new node inserted before the leftmost of the siblings as shows in figure (31). They encode a new node inserted in between siblings using an even numbers as shown in figure (32).

(O'Neil et al., 2004) labelled a tree nodes using odd numbers with even numbers for the update nodes which may increase the label size and as a consequences may lead to an increase storage size and reduce query process and labelling performance. More work has been done in prefix labelling schemes to update an XML tree with lower storage cost by reducing the size of label as will

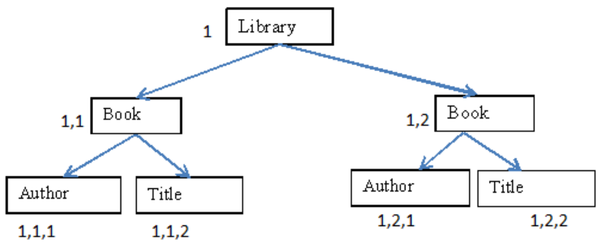


Figure 29: Dewey Labelling Scheme.

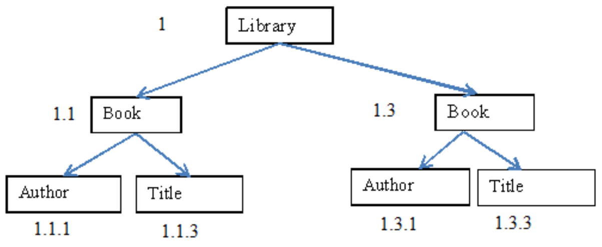


Figure 30: ORDPATHs Labelling Scheme.

be explained in the approach of (Liu et al., 2013a).

(Liu et al., 2013a) attempted the preservation of the relationship between XML nodes after update by proposing a new method called *Dynamic Float-Point Dewey* 'DFPD'. Their novel approach starts by labelling the XML nodes from the root node with the parent node label as a prefix of its child's using depth-first tree traversal with the components separated by dot '.' and where the last component is the local level of the child. To update an XML tree dynamically, (Liu et al., 2013a) studied four cases of node insertion: before the first child, after the last child, a new child and between two siblings. In the case of an insert before the first child, the algorithm adds a negative component to label of the new node '1.-1' similar to ORDPATHs algorithm. When inserting

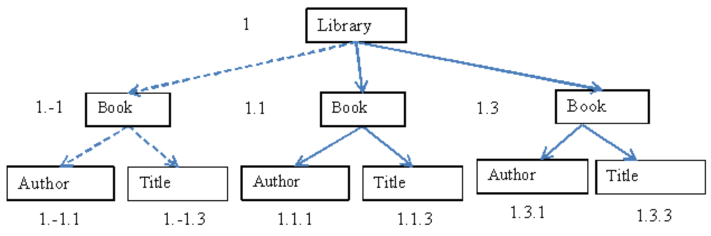


Figure 31: Insert Node Before the Leftmost in ORPATHs labelling scheme.

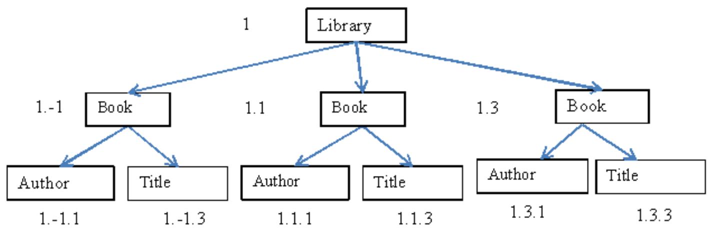


Figure 32: Insert Node Between two Nodes in ORPATHs labelling scheme.

a node after the last child of the root node, the approach will follow the Dewey labelling order. For example, if a label of the last child is '1.5', then the label of the inserted node is '1.7'. When inserting a new child node, the label of the new child node will have an additional component. For instance, if the label of the parent node is '1.22', then the label of the new node will be '1.22.1'. To insert a new node between two existing nodes, the new model DFPD exploited 'nominator/ denominator' as a floating-point number to label a new node as shown in the figure(33).

(Liu et al., 2013a) computed the label of the inserted node as follows:

$$Parentlabel.Z = \frac{X * \frac{previoussiblinglabel}{siblinglabeldemonimator} + Y * \frac{nextsiblinglabel}{nextsiblingdemonimator}}{previoussibloinglabel + nextsiblinglabel}$$

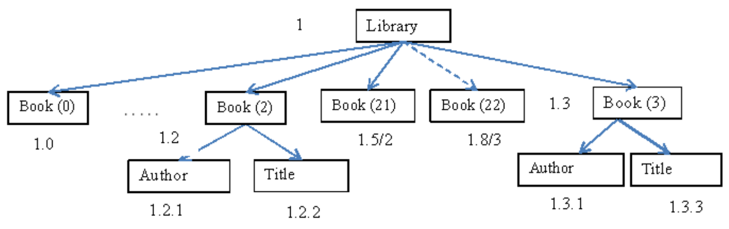


Figure 33: Insert node between two nodes in DFPD.

Where

Z is the label of the update node,

X and Y are the denominators of previous and next sibling respectively.

For example, in figure (33) the label (1, 5/2) of the inserted node Book 21 is computed as follows:

$$1.Z = \frac{2 * \frac{5}{2} + 1 * \frac{2}{1}}{1+1} = 1.\frac{5}{2}$$

The label of the node Book 22 in figure (33) is computed similarly and is assigned the following label:

$$1.Z = \frac{2 * \frac{5}{2} + 1 * \frac{3}{1}}{2+1} = 1.\frac{8}{3}$$

The labelling scheme of (O'Neil et al., 2004) and (Liu et al., 2013a) addressed the issue of labelling a dynamic tree using numbers which may increase the label size and consume the storage space with continuous update. There is another dynamic labelling scheme based on prefixes which assigns labels to updated nodes using a combination of letters and numbers as will be explained in the next labelling scheme.

(Duong and Zhang, 2005) argued that a label such as '1.2.13.24.3' may confuse a user about the depth of a node in an XML tree and with four delimiters to figure out the node's relationship may require a lot of space for storage. Therefore, they designed a new approach they dubbed *Labelling Scheme for Dynamic XML data* (LSDX) to reduce a use of the delimiter and to support a dynamic labelling of XML during update. (Duong and Zhang, 2005) combined alphabetic letters with numbers to represent the node's label starting from the root node which is assigned '0a' where 'a' refers to the node label and '0' represents the node level where the top level, the root, is zero. They attached the label '1a.b' to the first child of the root node where 'b' represents the node's label, '1' is a level of the node 'b' and 'a' is the label of the parent as shown in figure (34).

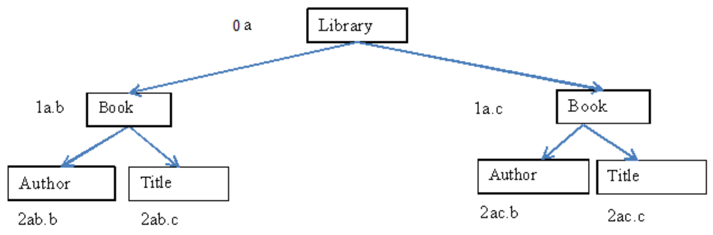


Figure 34: Labelling Scheme for Dynamic XML update (LSDX).

(Duong and Zhang, 2005) started labelling the child node from 'b' to a letter 'a' to an inserted node before the node 'b'. For example, if a user updates an XML document by adding a node before the leftmost node 'b', LSDX approach will assign a label '1a.ab' to maintain the alphabetic order of the updated node as shows in figure (35).

To insert new node between two siblings, the LSDX will generate a label between the preceding node's label and the following node's label as demonstrate in figure (36). The novel approach assigns the label '1ab.bb' to the new node which is between '1ab.b' and '1ab.c' alphabetically and it generates the label

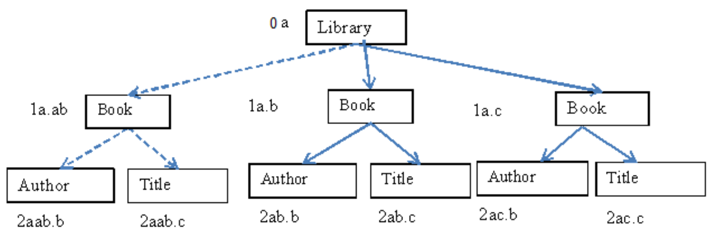


Figure 35: Insert New Node Before the leftmost node.

'1ab.bc' for the new node inserted after the node '1ab.bb'.

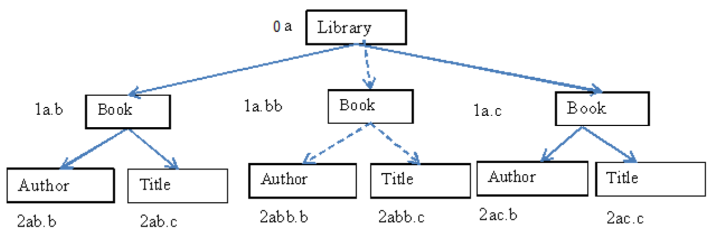


Figure 36: Insert New Node Between two Siblings.

The labelling scheme of (Duong and Zhang, 2005) follows alphabetic order when a user insert a new node after the rightmost node and a new child to an existing node. The repetition of letters to represent a position of a node between siblings may impact on the storage size. For instance, if the user wants to insert another node between node '1a.bb' and node '1a.c' in figure (36), LSDX will generate the label '1a.bbb' for the new node. (Duong and Zhang, 2008) addressed this case and suggested a novel techniques for labelling of XML tree as will be explained in the next labelling scheme.

(Duong and Zhang, 2008) proposed a new labelling scheme called *Com-*

pressed Dynamic Labelling Scheme 'Com-D' to cover the drawback of LSDX. The former replaces the repetition of letters in the node's label with a number to save a storage space. For example, the label '2abb.b' in the labelled tree shown in figure (36) is represented as '2a2b.b' in the model of (Duong and Zhang, 2008). Where, the first 2 represents the node's level, 'a' is the parent node's label, the second 2 is the number of 'b's, and the 'b' after the delimiter is the current node's label.

More work has been done on labelling XML documents based on prefix techniques and they employ a combination of letters and numbers to encode each label in the XML tree as in the approach of of letters and numbers to encode each label in the XML tree as the approach of (Duong and Zhang, 2005, 2008). However, they claimed that the scheme of (Duong and Zhang, 2008) may not achieve its ambition to enhance the performance of query processing and tree update because of the cost of the compressing and decompressing processes for the labels. So, they designed a new scheme which employed a combination of letters and numbers to label XML nodes without as much storage as will be explained in the next scheme of (Assefa and Ergenc, 2012).

The (Assefa and Ergenc, 2012) labelling scheme consist of three sections *'Level, Order, Parentorder'*, where *Level* is a number that represents the level of a given node in the XML tree starting from the level '0' for the root node level. *Order* which gives the horizontal distance of the node from the most left node on the level. (Assefa and Ergenc, 2012) exploited alphabetic order to manage the order of element at the same level as will be shown in figure (37) *Parentorder* is order of the node's parent.

They exploited numbers to give the information about parent/child relationships and the alphabetic order to provide information about the siblings

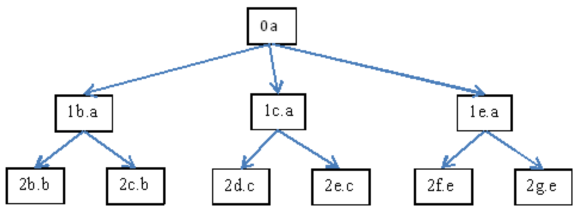


Figure 37: OrderedBased Labelling Scheme, adapted from (Assefa and Ergenc, 2012).

relationships. To find ancestor/descendant relationships, they needs to trace the parent of the parent/child recursively until they reach the intended ancestor or the root.

The model of(Assefa and Ergenc, 2012) labels the root node '0a' where 0 refers to a level number and 'a' to the character label of the node. Their novel approach follows alphabetic order to label the nodes of a lower level starting from 'b' as shown in figure (37). When the number of node in same level exceeds 25, it concatenates an extra character to the node label after every 25th node. For instance, if the label of a 25th node is 'z' then the label of the 26th node will be 'zb' and so on. This concatenation will increase the storage size and may have a negative effect on the performance of the labelling scheme and query processing.

To label 100 nodes, requires $1(25) + 2(25) + 3(25) + 4(25) = 250$ characters. The firrst 25 nodes will be labelled starting from 'b' to 'z', the second will be labelled with a concatenation of two characters starting from 'bb' to 'bz' and third 25 nodes will be labelled 'bbb' to 'bbz' and so on.

(Assefa and Ergenc, 2012) designed the following formula to optimise the calculation of the number of characters that are needed for each level.

$$Y = Ceil(\frac{logM}{log25})$$

where M is the number of nodes of each level, Y is the number of characters required to label M nodes, $Ceil$ is a function which returns the smallest integer equal to a given expression. For example, to label 100 nodes using the optimized formula, requires 2 characters as shown in the following equation:

$$Y = Ceil(\frac{log100}{log25}) = 2$$

After determining the number of characters needed for node encoding, (Assefa and Ergenc, 2012) studied update in an XML tree in three cases: inserting of a node before the first node in a level, inserting a node between two nodes, and inserting a node after the last node of a level.

To insert a node before the first node of a given level, they added 'a' before 'b' to give a label to the inserted node greater than to the next sibling node whilst retaining alphabetic order as shown in figure (38).

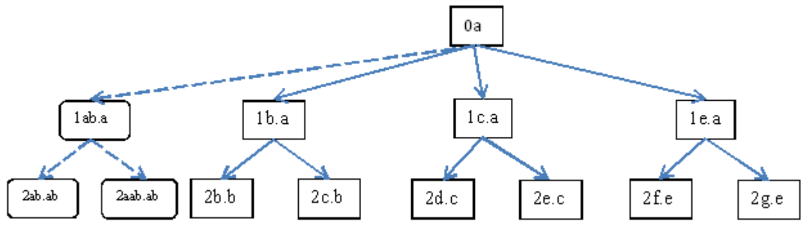


Figure 38: Insert a Node Before the First Node in the OrderedBased Labelling Scheme, Adapted from (Assefa and Ergenc, 2012).

For insertion of node in between two nodes, (Assefa and Ergenc, 2012) added a new character to the label of the new node without affecting the labelling order of the nodes. The code of the new node is greater than the previous sibling and less than the next sibling as we can see in figure (39). The label of the previous sibling is 'b', so (Assefa and Ergenc, 2012)) added another 'b' to the label of the new node to maintain the order of the nodes of within the level.

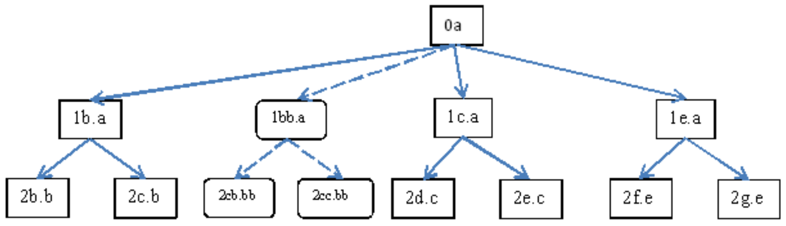


Figure 39: Insert a Node Between Two Nodes in OrderedBased Labelling Scheme, Adapted from (Assefa and Ergenc, 2012).

(Assefa and Ergenc, 2012) addressed the insertion of a new node after the last node by incrementing the label order of the new node alphabetically as shown in figure (40). The label of the new node is 'e' which greater than that of the previous sibling which is 'd'.

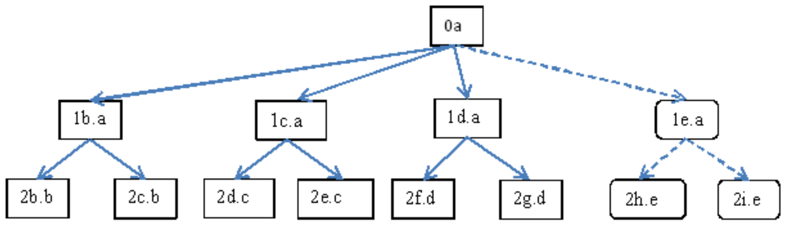


Figure 40: Insert a Node After the Last Node in the OrederedBased Labelling Scheme, Adapted from (Assefa and Ergenc, 2012).

(Assefa and Ergenc, 2012) employed numbers and characters to encode the node labels of XML tree to define the structural relationships based prefix order and to maintain these relationships despite dynamic update of the XML tree whilst preserving the efficiency of query processing.

Prefix labelling schemes requires more storage space with the insertion of more nodes in the depths of the XML tree, and this may lead to a lack of storage space (Haw and Lee, 2011; Xu et al., 2012). Another group of researchers adopted different technique to label XML nodes in based on mathematical operations to assign labels to nodes in minimal cost in time and storage as will be explained in the next labelling scheme category.

5.4 Multiplicative Labelling Scheme

This kind of labelling schemes exploits atomic numbers to label XML nodes and determine the node relationships through the arithmetic properties of the node's label (Al-Shaikh et al., 2010; Haw and Lee, 2011; Wu et al., 2004). Researchers, for instance (Wu et al., 2004) and (Al-Shaikh et al., 2010) argued that the interval labelling scheme needs a lot of storage for dynamic tree labelling and sometimes the relabelling process cannot be avoided. For the prefix labelling schemes, they claimed that label size will increase with any increase in the depth of a tree and a consequences of this the storage cost will also increase. So they studied the encoding of XML trees using mathematical operations to determine the node relationships in the minimal storage size and fast query processing as will be clarified in the labelling scheme approach of (Wu et al., 2004).

(Wu et al., 2004) exploited a mathematical property of prime numbers which are numbers that are only divisible by one and themself (Schaffer, 2001) to label XML nodes. They defined two properties to identify the A-D relationship (see section 2.6). Property 1 Divisibility: "if an integer number 'A' has a prime factor which is not a prime factor of another integer 'B', then 'B' is not divisible by 'A'. (Wu et al., 2004) exploited this property to define the labels of the nodes in a top-down tree traversal. The label of a child is produced by multiplying the node's self label with its parent's label. Based on the Divisibility property, the child's node label must be divisible by the parent label as can be seen in figure (41).

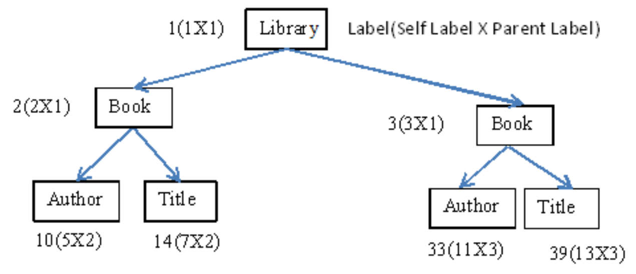


Figure 41: Top-Down Prime Number Labelling Scheme, Adopted from (Wu et al., 2004).

The second property defined by (Wu et al., 2004) is used to number the nodes of the XML tree using a bottom-up technique which is *"In a bottom-up prime number labelling scheme, for any node 'x' and 'y' in an XML tree, 'x' is an ancestor of 'y' if and only if label(x) mod label(y)=0.* The label of the parent node is determined by the multiplication of the labels of its children. For instance, the label of the parent node 15 in figure (42) is the result of the multiplication of its child node labels 3 and 5.

(Wu et al., 2004) mentioned in their paper that the size of the label depends on the depth of the XML tree. For instance, in figure (42) an XML tree is shown

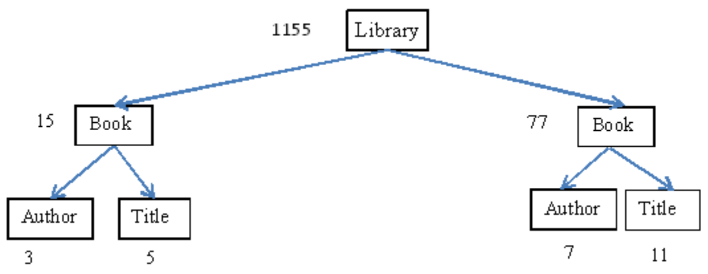


Figure 42: Bottom-Up Prime Number Labelling Scheme Adapted from (Wu et al., 2004).

which is labelled using prime number labelling top-down. (Wu et al., 2004) optimize the label size in four steps: first, they reserve a small prime numbers for the root's children because they have an influence on the label size. Second, the number 2 is the only even prime number and its exploited to encode the leaf nodes based on abel size. Second, number 2 is the only even prime number and its exploited to encode the leaf nodes based on $2^1, 2^2, 2^3, ..., 2^n$. Third, they collapse the paths that occur multiple times as the path in figure (43) which is a combination of paths in figure (41).

Final, they suggested decomposing a large tree into a number of sub trees which are labelled separately with the roots of these sub trees labelled in a global tree.

To maintain the order of nodes in an XML tree during update, (Wu et al., 2004) employed *Chinese Remainder Order* which is a mathematical formula used to find a relationship between a set of integers which are relatively prime with a set of integer numbers (Zheng and of South Carolina, 2007). The map-

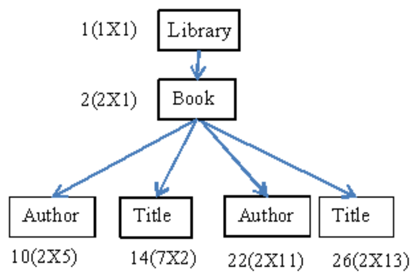


Figure 43: Combine Paths in The Prime Number Labelling Scheme.

ping between these two sets of is done by defining a factor called Simultaneous Congruence SC which is generated from the self labels and also used to determine node order in the from the self labels. For example, let 1523 be the 'SC' of the prime numbers '1,3,5,7', the order numbers of the relevant prime numbers '1,2,3,5,7' are '0,1,2,3,4' which are produced based on the equation:

$$1523 \; mod \; 1 = 0$$

$$1523 \; mod \; 2 = 1$$

$$1523 \; mod \; 3 = 2$$

$$1523 \; mod \; 5 = 3$$

$$1523 \; mod \; 7 = 4$$

as is explained in figure (44).

To add a new node that has a self label 17 in order 3 in figure (44), the approach of (Wu et al., 2004) will search for the largest prime number and update the values of 'SC' to keep the order of nodes in the tree. For example, the values of 'SC' in figure (44) will be updated to 1139 and 20 to contain the new node and maintain the node order as clarified in figure (45).

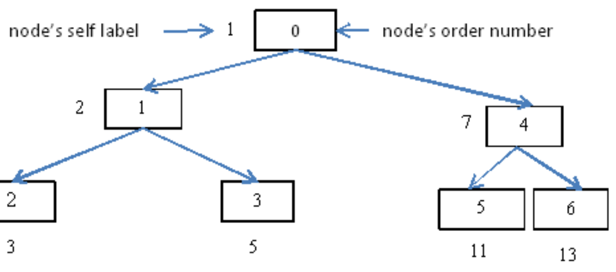


Figure 44: XML tree ordered with SC=1523 from the ordered nodes (0-5) and SC=6 for the order node (6).

As can be seen in figure (45), the burden of the nodes order preservation lies on the 'SC' by changing its values rather than updating the nodes order.

The labelling scheme of (Wu et al., 2004) may consume a lot of storage in labelling a deep XML tree to determine the node relationships. This is because the node label is produced by multiplying the descendant prime number with its self label 'prime number' and the result will increase with increase in the tree depth (Al-Shaikh et al., 2010). Therefore, (Al-Shaikh et al., 2010) designed a new labelling scheme to label XML nodes based on prime modulo and modular multiplication as will be explained in the next approach.

The approach of (Al-Shaikh et al., 2010) proposed a novel labelling scheme which allocates each node a label that consist of two elements 'L, E' as show in figure (46) where 'L' is a prime number self label given to the node using the breadth first technique and 'E' is constructed by the formula:

$$E = (S * L) mod P$$

, where 'E' is a label of a new node, 'S' is parent's label, 'L' is the node's self-label, and 'P' a globally large prime number modulo of the tree as clarified in

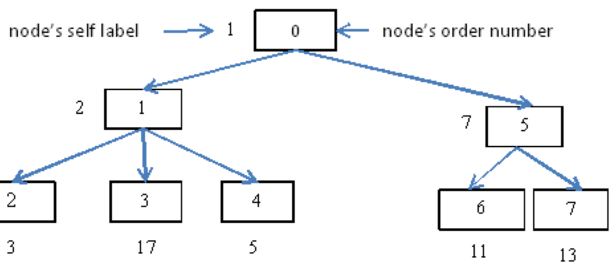


Figure 45: Insert a New Node in The Prime Number Labelling Scheme, adapted from (Wu et al., 2004).

the figure (46).

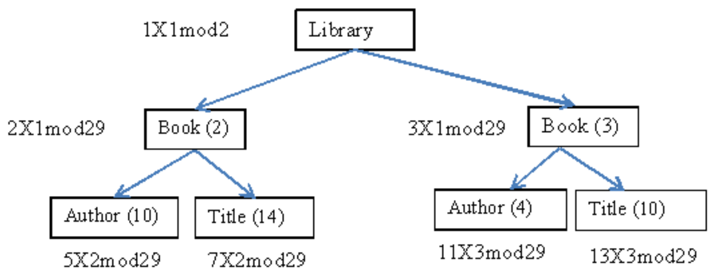


Figure 46: Modulo-Based labelling Scheme with P=29, adapted from Al-Shaikh et al. (2010).

To insert a new node into an XML document in figure (46), (Al-Shaikh et al., 2010) calculated the label of the inserted node based on the formula above. In addition, they developed another equation to determine the structural relationship by defining the parent of the new child:

$$E = (S * \overline{L}) \mod P$$

where '$\overline{L}$' is the multiplicative inverse of 'L' and it can be defined through the inverse equation (*Extended Euclidean Algorithm*) EEA

$$(L * \overline{L}) \mod P = 1$$

must be true.

For example, let us suppose that a new node has been inserted with the self label 13 and a parent with the self lable 3 in figure (46). The label 'E' of the new nodes can be computed through the formula:

$$3X13 \mod 29 = 10$$

The structural relationship of the insert a node with its parent can be found as follows: first, find the inverse number of the label of the new node 13. The inverse number of 13 can be calculated using EEA through the equation:

$$13X\overline{13}mod29 = 1$$

If the computation of the left side of the equation equals to the right side which is 1, then the number $\overline{13} = 9$ is the inverse number of 13. The parent's label of the new node can be computed by the equation: number of the label of the new node 13. The inverse number of 13 can be calculated using EEA through the equation:

$$13X\overline{13}mod29 = 1$$

with the 'E' label of the parent node of the node 13.

Al-Shaikh et al. (2010) maps a self label of node to an order label which is a natural order of node using 'SC' values. During tree update, the approach of

Al-Shaikh et al. (2010) needs to update the values of 'SC' to maintain the node order in the tree which may affect the labelling performance as claimed by (Xu et al., 2012). Therefore, they proposed a new labelling scheme which exploits vectors to encode XML nodes as will be explained in the next model of labelling scheme.

(Xu et al., 2012) was attracted by graph vectors and their relationships which are ordered by the tangent of the angle between the vector and x-axis. They proposed a Vector Order-Centric encoding approach for labelling XML nodes based on a vector which consist of a binary tuple (x, y) where x and y are positive integers.

It is used to 'encode' a term in the method which means the representation of a node's label as vector which consist of (x, y) . (Xu et al., 2012) applied their vector encoding method to update of the XML documents dynamically to avoid a costly relabelling process which may affect the performance of the query and labelling scheme.

(Xu et al., 2012) adopted the vectors where the x value is positive (as it is adopted in this explanation). They produce vectors based on the relation$A + B = (x_1 + x_2), (y_1 + y_2)$, where the vector of A is $'x_1, y_1'$ and the vector of B is $'x_2, y_2'$. However, A represents a first label and B represents the last label and the new label is generated through the equation $ceil\frac{A+B}{2}$ which is equal to $A + B$.

They designed a set of principles to order vectors such as: for two vectors $A(x_1, y_1)$ and $B(x_2, y_2)$, A precedes B '$A \leq B$' either $\frac{y_1}{x_1} \leq \frac{y_2}{x_2}$ or $x_1 = x_2$ and $y_1 \leq y_2$. The vectors '$A = B$' when $x_1 = x_2$ and $y_1 = y_2$.

They applied the new method of label encoding on a containment labelling scheme (which is a class of interval labelling scheme) called *V-Containment* with the preservation of the containment properties explained in section (5.2). This will be discussed in detail later.

(Xu et al., 2012) employed an linear transformation to demonstrate the V-Containment labelling scheme as follows:

$$f(i) \ = \ (1, i) \ for \ i \in Z$$

where Z is set of integers as an example of the transformation process.

In addition, they proposed a factor called Granularity Sum 'GS' to maintain the order of the vectors during the tree update. The 'GS' of a node A(x,y) can calculated by the formula:

$$x + y$$

(Xu et al., 2012) addressed updating trees encoded by the Vector Order-Centric method and they determined a set of principles for the tree modification: the range of the new node should be inside the range of its parent, the start of the new node must be less than the end of the closest preceding sibling, the end of the inserted node must greater than the start of the next sibling.

(Xu et al., 2012) studied the modification of XML trees from four perspectives: inserting node before the first node of the root, between siblings, after the last child of the root, and adding a new child to a node in the tree.

In the first case of adding new node, the start and end of the inserted node should be bound to the start of the root and the following sibling. For example, if Book 1 in figure (47) is inserted before Book 2, the label of Book 1 should bound between the root and the following sibling and it will be computed as follows:

$$(2 * 1 + 1, 2 * 1 + 2) = (3, 4), (1 + 1, 1 + 2) = (2, 3)$$

because the GS(parent) (1+1=2) < GS(child) (1+2=3).

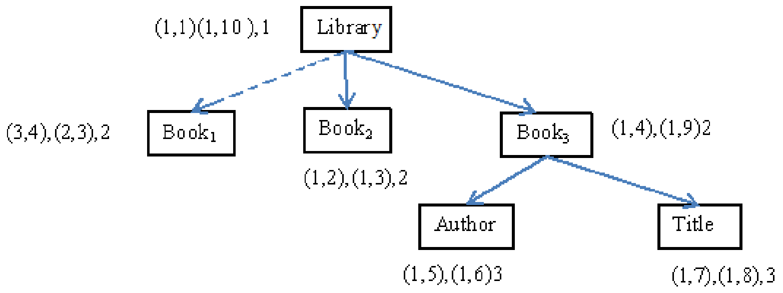


Figure 47: Insert a New Element as a First Child in V-Containment labelling Scheme.

To insert a node after the last child node of the parent, (Xu et al., 2012) proposed the equation:

$$((2*x1+x2,2*y1+y2),(x1+x2,y1+y2))$$

to bound the start and end of the new node with the end of the preceding sibling and the end of the parent. For instance, insert Book 4 into the XML tree in figure (48) and the the novel algorithm will assign the following label to the new node:

$$(2*1+1,2*10+9)=(3,29),(1+1,9+10)=(2,19)$$

The next circumstance is the insertion a new node *Edition* between two nodes. The start and end of the new node should be between the end of the preceding node and the start of the following sibling as shown in figure (49) and they used the formula of the previous case to calculate the new node vector.

The child node is assigned a label '(3,17),(2,13)' which is a result of the following equation:

$$(2*1+1,2*4+9),(1+1,4+9)=(3,17),(2,13)$$

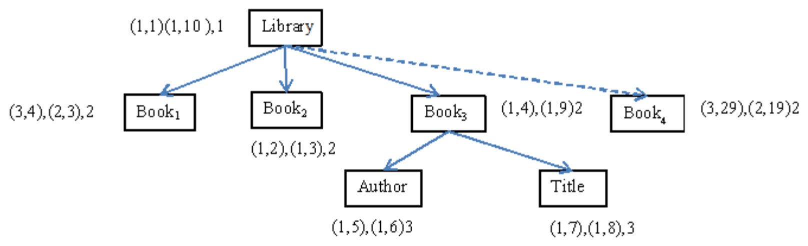


Figure 48: Insert a New Element as a Last Child in V-Containment labelling Scheme.

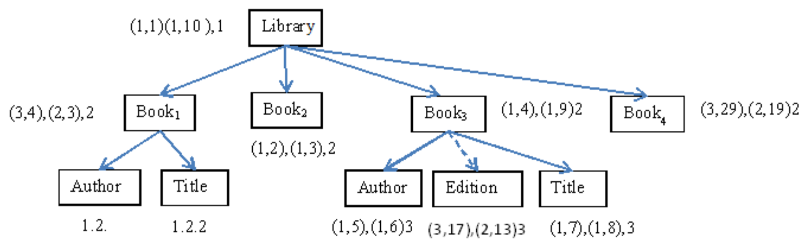


Figure 49: Insert a New Element Between two Nodes in V-Containment labelling Scheme.

and the inserted child is bounded by the preceding and following siblings.

The last case analysed by (Xu et al., 2012) is the insertion of a child to XML tree. The novel approach will take into consideration the range of the parent label and computes a label of the new node using the equation of the preceding cases to find the vector of the updated node. For example, the label of the new node First in figure (50) is extracted from the information of the parent's label as can be seen from the result of the formula:

$$((2*1+1),(2*7+8)),(1+1,7+8)=(3,22),(2,15)$$

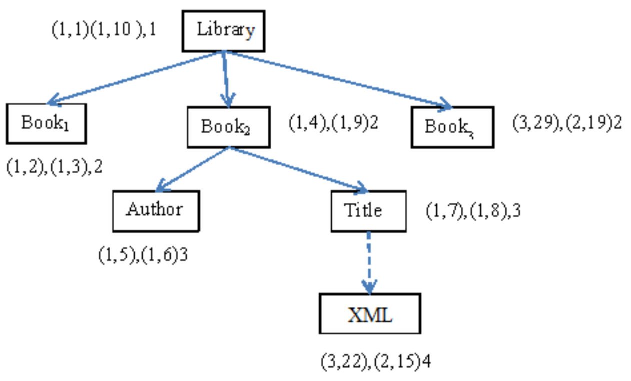


Figure 50: Insert a New Child in The V-Containment labelling Scheme.

As (Xu et al., 2012)) applied the Vector Order-Centric approach to both an interval labelling scheme, and a prefix-based labelling scheme and called it *V-Prefix Labelling Scheme*. They initialize their approach on a prefix version by transforming each component of the node's label into a vector 'x,y'.

(Xu et al., 2012) studied the update of XML tree in four cases similar to the V-Containment labelling scheme: insert a node before the first node of the root as node Book 11 in figure (51) which is assigned a label '(1,1).(1,0)' that is less than the label '(1,1).(1,1)'.

Insert a node between siblings as when inserting the node publish in figure (51) which is assigned a label '(1,1).(1,2).(2,3)' where the vector '(2,3)' is results from the formula:

$$((1+1),(1+2))$$

which are the labels of its siblings and '(1,1).(1,2)' is the prefix path. Adding a node after the last child of the root as shows in figure (51) where the node Book 3 is labelled '(1,1).(1,3)' because it is after the node Book 2. The last case is inserting a node as child to an existing node in the tree. For example, the node (XML) in figure (51) is coded with label '(1,1).(1,2).(1,2).(1,1)' which is its location and starts with the parent label '(1,1).(1,2)(1,2)'.

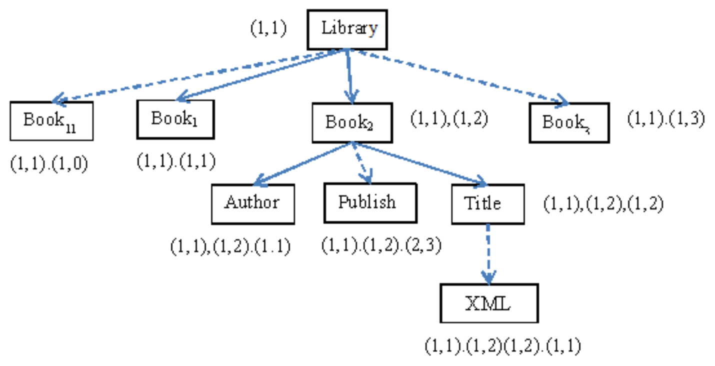


Figure 51: Insert a New Nodes in V-Prefix labelling Scheme.

In this chapter, three classes of XML labelling schemes have been explained: Interval, Prefix, and Multiplicative. A number of experiments are discussed in the next chapter to show the effect of XML datasets characteristics such as number of siblings and depth of the tree on these labelling schemes.

6 Experiments and Statistical Analyses

6.1 Introduction

The dimensions of XML trees are different from the depth and width point of view. To investigate and compare static labelling schemes, three real XML databases have been chosen Nasa, dblp, and Treebank-e, whose characteristics are shown in the table (6). These databases available on the website of the University of Washington for research purposes (UOW). These experiments have been executed to determine the appropriate labelling scheme for specific

XML databases based on the time and space needed for labelling the XML tree nodes. The Vector Order-Centric labelling scheme was chosen to update Nasa dataset dynamically because its characteristics lie between dblp and Treebank-e databases. To address the overflow problem in the dynamic labelling schemes, six experiments were executed on update Nasa: three experiments using V-PreËİx and another three using V-Interval as will be explained in the section 6.3.

The experiments were encoded using JavaSE-1.8. To ensure accurate timing results in the experiments the timed part of the experiment was placed in a loop and the first 10 iterations were discarded. Java source code is compiled into Java bytecode to be executed in the computer using the Java Virtual Machine. During frequent executions of Java bytecode statements, JVM will generate complied code for the iterated statements, as a result, this will skip the lookup for those repeated statements (Oaks, 2014).

The integrated development environment IDE known as Eclipse 'Release 4.4.0RC1' is exploited to run Java codes on a machine has Intel (R) Core (TM) i5-3570t CPU 2.30 GHz, RAM 4 MB, and windows 7 Enterprise.

Table 6: XML Databases.

XML database	No. of elements	Max Depth (Level)	File Size
Nasa.xml	476646	8	23MB
dblp.xml	3332130	6	127MB
Treebank-e.xml	2437666	36	82MB

6.2 Execution of Static Labelling Schemes

Two labelling schemes were executed in these experiments: the Dewey Encoding labelling scheme designed by (Tatarinov et al., 2002) and is chosen to represent the prefix labelling technique; and the containment labelling scheme proposed by (Xu et al., 2007) to represent the interval labelling schemes. To address which labelling scheme: Prefix or Interval is suitable for labelling the XML documents in table (6) from time perspective, six experiments were executed to measure execution time for the generation of labels as will be explained in the next section.

6.2.1 Time for XML Database Labelling

In this part of the experiments, two variables were measured: *standard deviation* STD, which represents the distribution of the results around the mean, where, a small value of STD means a repeatable experiment and a high level of confidence in the result and vice versa. The second variable will be a significant measurement which evaluates the mean of the speed of the labelling schemes. These were executed 110 times for each database in the table (6) and the first 10 results were disgarded to ensure reliability.

Two experiments were executed to label the Nasa XML database using Prefix and Interval labelling schemes to analyse the time spent for labelling the nodes of the database. As mentioned in the previous chapter, the Prefix labelling scheme labels XML nodes using depth-first traversal of a tree which visits each node once. However, the Interval labelling scheme visits each node in a tree twice: first to assign a value to the *start* part of the node's label and the second visit to allocate a value to the *end* part of the node's label.

Figure (52) shows the time consumed for labelling the Nasa dataset using both the Prefix and Interval labelling schemes. The y-axis represents the time in millisecond spent to label the database and the x-axis represents the type of scheme.

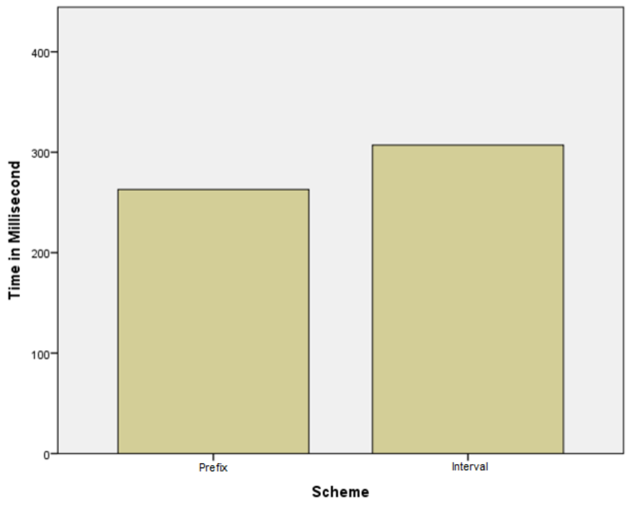


Figure 52: The Time Consumed for Static Labelling Nasa database using Prefix and Interval.

Table (7) includes statistical information about the labelling of the Nasa database based on the time measurements where 'No of Variables' represents the number of experimental executions.

Table 7: Time Consumed for Static Labelling Nasa using Prefix and Interval.

Scheme	**No. of Variables**	**Mean**	**std**
Prefix	100	262.96	14.716
Interval	100	307.07	5.883

It is clear that the mean time for the Preifx scheme is shorter than that for the Interval scheme which means that labelling Nasa using the Prefix scheme is faster than the Interval encoding scheme because the technique of the former generates label in linear time and the latter scheme produce labels in exponential time (Sans and Laurent, 2008).

The next two experiments were executed to label the dblp database which was wider and shorter than Nasa as shown in the table (6). Figure (53 shows the time spent for labelling dblp using Prefix is faster than that using Interval for the same reason as for the Nasa. The axes are the same as those on the Nasa figure.

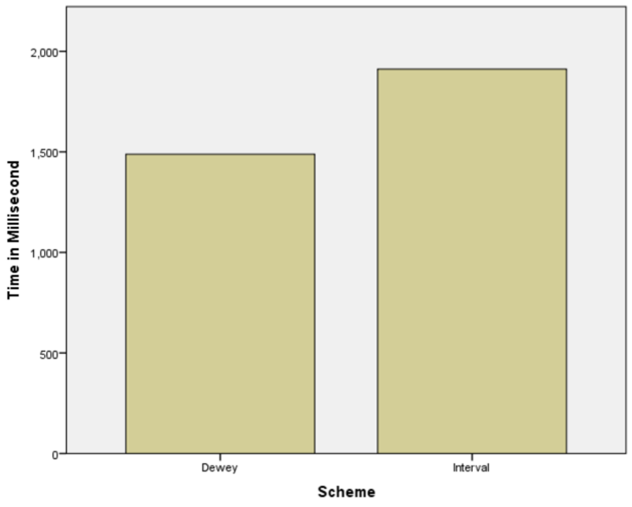


Figure 53: The Time Consumed for Static Labelling dblp database using Prefix and Interval.

The table (8) demonstrates that the difference of the mean time execution for Prefix is shorter than that for Interval.

An additional two experiments were performed on the Treebank database which was wider than Nasa and narrower dblp but which was the deepest tree

Table 8: Time Consumed for Static Labelling dblp using Prefix and Interval.

Scheme	**No. of Variables**	**Mean**	**std**
Prefix	100	1488.15	29.255
Interval	100	1911.89	24.688

as shown in table (6). The Prefix and Interval labelling schemes were executed to analyse the time needed to label a tree such Treebank-e. Figure (54) shows that the time consumed for labelling Treebank-e using Prefix is shorter than that in Interval which a similar to the previous databases.

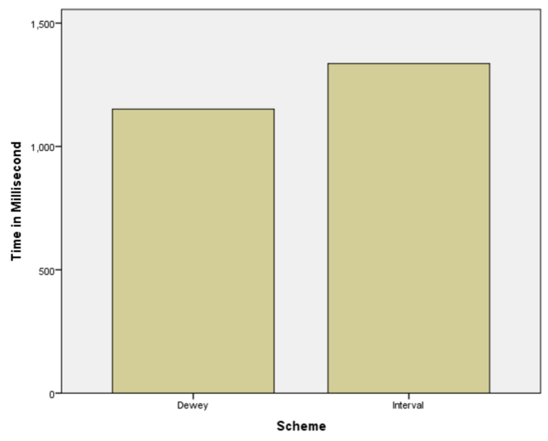


Figure 54: The Time Consumed of Static Labelling Treebank-e database using Prefix and Interval.

In spite of the fact that the Treebank-e is the deepest tree in the table (6), the mean time for the Prefix labelling scheme was shorter than for the Interval labelling scheme and for the same reason of the experiment of dblp.

Table 9: Time Consumed for Static Labelling Treebank-e using Prefix and Interval.

Scheme	**No. of Variables**	**Mean**	**std**
Prefix	100	1175.09	42.050
Interval	100	1362.24	17.280

Conclusion

To summarise, the figure (55) represents the information of the table (10) which includes statistical information about the time consumed for labelling Nasa, dblp, Treebank-s databases using the Prefix and Interval labeling schemes. In spite of the different characteristics of these databases, the time spent for labelling the three datasets using Prefix is shorter than that using Interval technique. As mentioned previously, the process of label generation takes linear time in Prefix labelling scheme because the algorithm will visit the tree nodes only once. However, the algorithm of Interval labelling scheme will visit nodes of XML tree more than once and this will take exponential time which is longer than linear time (Sans and Laurent, 2008). As a result, the time spent for labelling nodes of the XML databases which have different structures using the Prefix technique is shorter than the Interval.

The statistical information in the table (10) demonstrates that Prefix is more suitable than Interval for labelling both wide (dblp) and deep XML databases.

Additional experiments were conducted to measure the space required to store the label nodes of the databases: Nasa, dblp, and Treebank-e databases as will be

Table 10: Time Consumed for Static Labelling Nasa, dblp , and Treebank-e using Prefix and Interval.

Scheme	Nasa		dblp		Treebank-e	
	Mean	STD	Mean	STD	Mean	STD
Prefix	262.96	14.716	1488.15	29.255	1175.09	42.050
Interval	307.07	5.883	1911.89	24.688	1362.24	17.280

6.2.2 The Space required for Static Labelling of XML Database

Storage space is another factor which is used to measure the performance of labelling schemes. Two sets of experiments were performed: one set was executed to label the nodes of Nasa, dblp, Treebank-e databases using Prefix and Interval to analyse the space needed to store the labels. A series of experiments were implemented to analyse the space required for labelling 'Nasa, dblp, Trebank-e' using Prefix and Interval labelling schemes.

Prefix and Interval labelling schemes were implemented on Nasa to examine the space required for node labelling. The node label which is encoded by a prefix label consists of a number of sections separated by delimiters that represent the Parentchild P-D structural relationships of nodes or Ancestor-Descendant A-D (Liu and Zhang, 2016) defined in the section (2.6).

To add a new label the Prefix scheme will allocate 1 within a new section which is preceded by the parent's label (Tatarinov et al., 2002). In contrast, the Interval scheme assigns a new label which is in the range of the sequence from 1 to '2n', where 'n' is the number of tree nodes (Xu et al., 2012). Two experiments were executed to measure the space needed to store the node labels of Nasa using Prefix and Interval. Figure (56) shows that the space required for storing labels

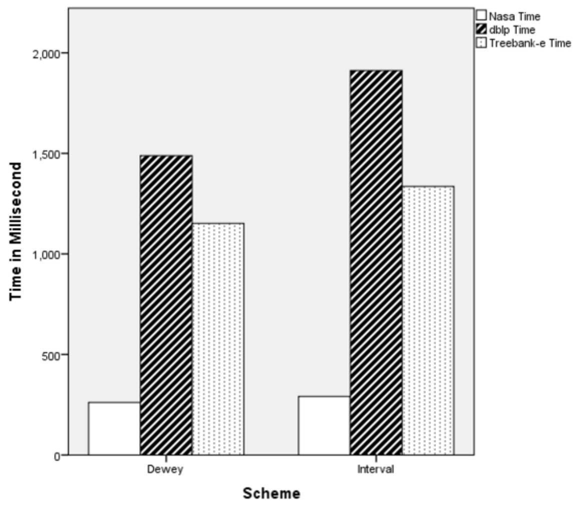


Figure 55: Time Required for Static labelling XML Databases.

which were generated using Interval is larger than that using Prefix. The size of labels generated by the Interval scheme increases exponentially with the increase of the tree level to keep the range label of the new child within the range of its ancestor label (Tatarinov et al., 2002). On other hand, the label size increases linearly using Prefix because it assigns a new label section to the new child.

To analyse the space required to label the nodes of dblp database, the same set of labelling schemes were implemented. The depth of the dblp tree is shallow in comparison with Nasa, but figure (57) shows that the space needed for storing labels which are generated by Interval is higher than that Prefix which matches the figure of Nasa. However, the former is a wide tree which means the Prefix labelling scheme is preferable because of its linear property for generating labels. The labels that are generated by an interval labelling scheme are computed exponentially based on the ancestor labels and the node size will increase exponentially as well.

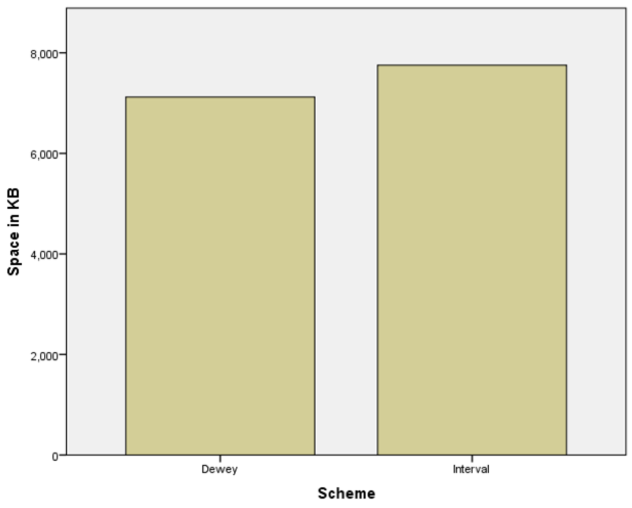


Figure 56: Space Required for Static Labelling Nasa XML Databases.

The same labelling schemes, Prefix and Interval, were executed to analyse the time and space needed for labelling the Treebank-e database which the deepest tree in the table (6). It is shown in figure (58) that the space required to store the labels that were produced by Prefix is larger than Interval. As was explained previously the Prefix technique produces labels sequentially and the label size depends on the node level in the tree. The tree depth of Treebank-e is 36 levels which means the label of a node at level 36 will consist of 36 sections. In spite of the exponential property of Interval, the label sections of the same node consist of 3 sections and should be smaller.

Conclusion

Table (11) listed the space needed to save the labels of the Nasa, dblp, and Treebank-e databases.

The statistical information in the table (11) shows that interval labelling is

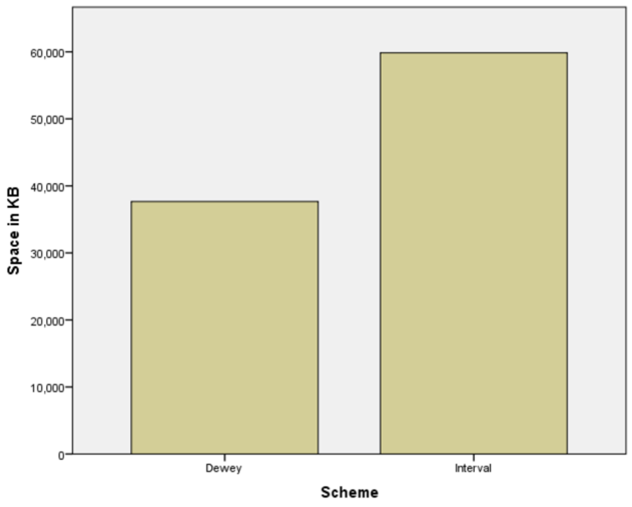


Figure 57: Space Required for Static Labelling dblp XML Databases.

Table 11: Space Required for Saving the Labels of Nasa, dblp, and Treebank-e using the Static Prefix and Interval Labelling Scheme.

Scheme	**Nasa**	**dblp**	**Treebank-e**
Prefix	7119.90	37664.14	48921.83
Interval	7754.21	59858.03	44368.03

more space-efficient for Treebank-e, the deepest tree. For shallow and wide trees, interval labelling is more expensive in space, for deep trees, it is less expensive in space, than prefix labelling. For instance, the deepest tree in the table (6) is Treebank-e which consists of 36 levels. The space required to save the labels of the Treebank-e database produced by Prefix labelling is 37,664 KB and this number is doubled when saving the labels generated by Interval labelling as shown in figure (59). In contrast to Treebanke, Nasa and dblp have 8 and 6 levels respectively and the space needed to save their labels using the Prefix

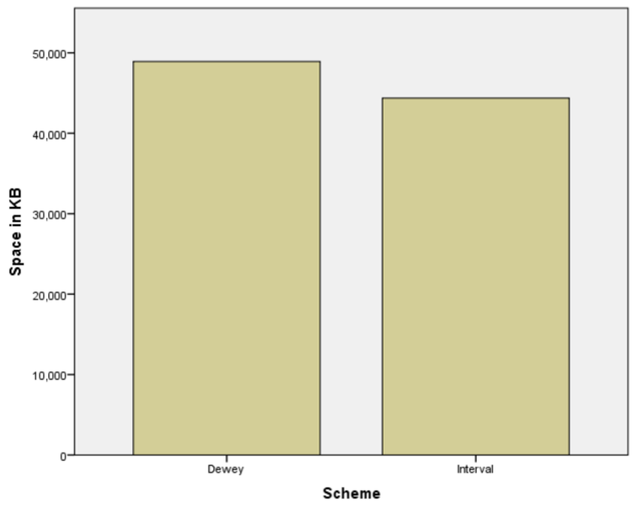


Figure 58: Space Required for Static Labelling Treebank-e XML Databases.

method is smaller than that using Interval.

A group of experiments were conducted to measure the storage space required to label XML and the time that was spent doing it. The next experiments were done to compute the space and time needed for labelling the nodes dynamically during database update as will explained in the next section.

6.3 Execution of Dynamic Labelling Schemes

In the dynamic labelling experiments, the Nasa database was chosen because its depth and width are between those of the dblp and Treebank-e datasets. Three experiments were conducted using V-Prefix and another three using the V-containment (vector order-centric) labelling schemes to insert 200, 500, and 1000 elements respectively. These experiments were employed to measure the time and space required for dynamic labbelling of the Nasa database as will be explained in the next sections.

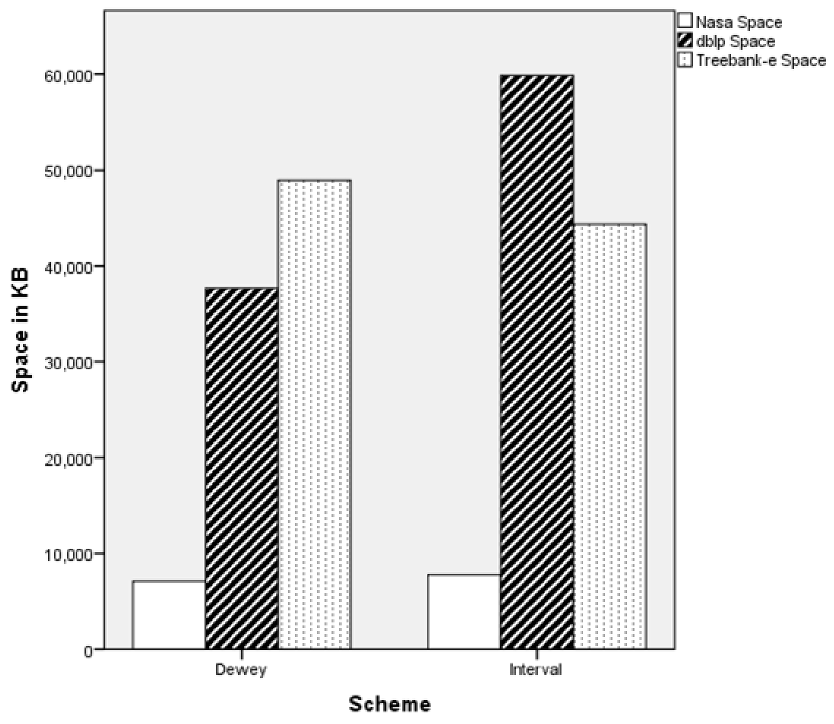


Figure 59: Space Required in (KB) for Static labelling XML Databases.

6.3.1 Time Required for Dynamic Labelling of XML Database

Figure (60) clarifies the values of table (12) which show that the time spent to label 200 and 500 elements using V-Prefix is less than using V-Interval which is similar to static labelling for Nasa in figure (55). This is because V-Prefix and V-Containment are based on the Prefix and Interval labelling scheme respectively where the latter labelling scheme generates labels in exponential time as mentioned in the section (5.2).

However, less time was required to update 1,000 elements in the Nasa database using V-containment than V-Prefix which contradicts the fact that exponential time is longer than the linear time. This contrast will be explained in the next

Table 12: The Mean of Time Required for Update three groups of Elements in Nasa Database using Vector Order-Centric Labelling Scheme.

Vector Scheme	**200 Element**	**500 Element**	**1000 Element**
V-Prefix	541.40	1404.40	4078.60
V-Containment	786.40	1930.70	3166.30

section.

6.3.2 The Space Cost for Storing Dynamic XML Database

Three experiments were executed to measure the space required to store the labels of the Nasa database after the addition of 200, 500, and 1,000 elements as shown in table (13). The information of the table (13) is illustrated in figure (61) which shows that the storage space required to store labels generated using V-Containment for 200 and 500 new elements is larger than that using V-Prefix.

However, when labels for 1,000 elements were produced by V-Containment the new space consumed was 3,166 KB which is smaller than 4,078 KB, the space required for the same number of labels generated by V-Prefix. A common logarithm is used to calculate the logarithm values of figure (61) to amplify the value of the space required to store the updated 1,000 elements as shown in figure (62).

The problem of space decreasing during the update of 1,000 elements illustrates the problem of overflow which was mentioned in the section (5.2) which occurs when a node label is over a predefined size as demonstrated by (Assefa and Ergenc, 2012; O'Connor and Roantree, 2012; Yun and Chung, 2008)). The reason for the overflow is the expensive computation of the vector and the use of the UTF- 8 encoding mechanism (Yergeau, 2003) to label the nodes (i.e.

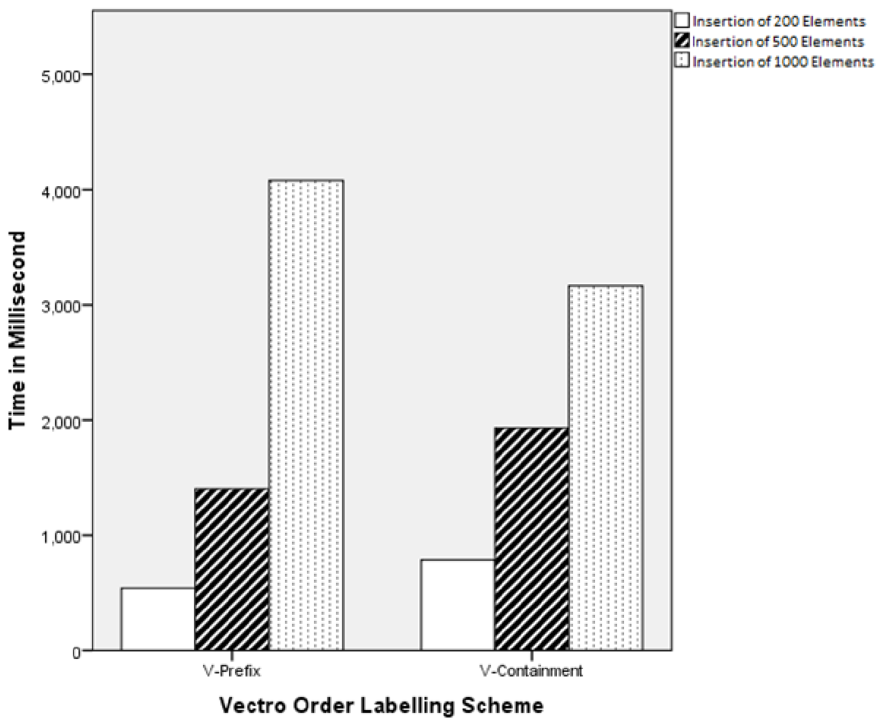


Figure 60: Time Spent for Update three groups of elements in Nasa.

represent the structural relationship) which increases the label size over the storage (Ghaleb and Mohammed, 2015; Xu et al., 2012). The explanation for time decreasing during 1,000 node updates is because of this overflow problem, therefore, the results of the time computation match the result of (O'Connor and Roantree, 2012) and were ignored by (Assefa and Ergenc, 2012) because they were not reliable.

Conclusion

To conclude, the Vector Order-centric scheme is not consistent due to the problem of overflow. After inserting 1,000 elements into the Nasa dataset, time and

Table 13: The space needed in KB to insert three groups of Elements in Nasa Database using Vector Order-Centric Labelling Scheme.

Vector Scheme	**200 Element**	**500 Element**	**1000 Element**
V-Prefix	109457.00	192053.00	852514.00
V-Containment	9024557.00	9368019.00	9499.55

space values decreased and the database needed to be relabelled. So, Vector Order- Centric is not suitable when updating an XML databases with more than 500 new nodes due to the costly relabelling process.

6.4 Experiment Results Evaluation

The results of the practical work in the dissertation can be compared with the work of (Xu et al., 2012) who employed the three datasets in the table (6) and the ((Ghaleb and Mohammed, 2015) who exploited dblp and Treebank-e. The results of the experiments of static labelling schemes in this dissertation are in agreement with the existing works of (Xu et al., 2012) and (Ghaleb and Mohammed, 2015) when they initializing the node labels. The results of the dynamic XML tree update using Vector Order-Centric labels match the results of (O'Connor and Roantree, 2012) who did not mentioned the XML datasets in their experiments. Moreover, the results match the outcome of (Assefa and Ergenc, 2012) who employed xmlgen of the XMark database (Bench Mark standard for XML Database Management) (Schmidt et al., 2002).

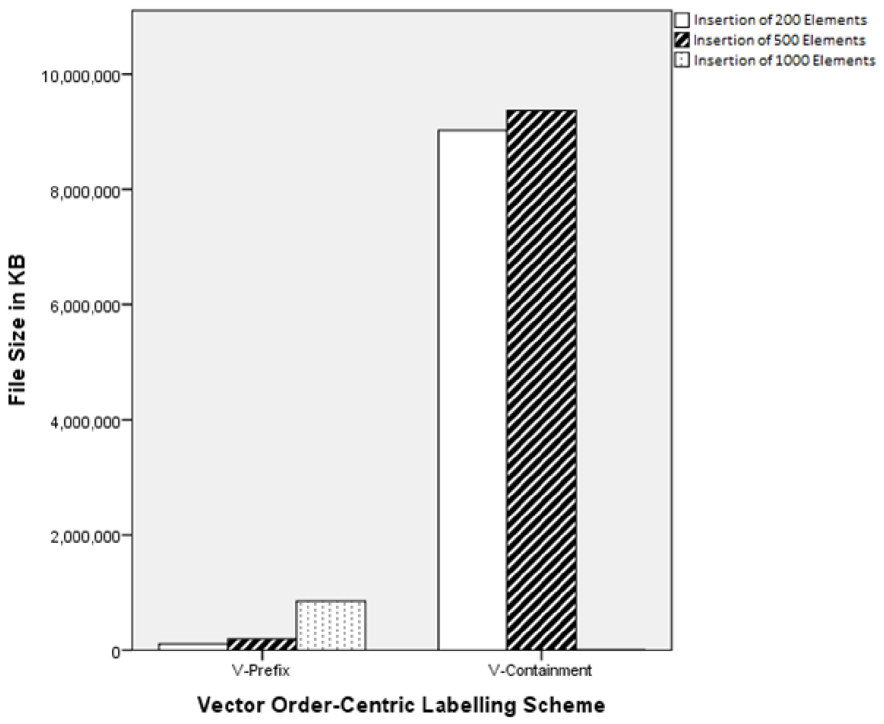


Figure 61: The Space required in KB to add three groups of elements in Nasa.

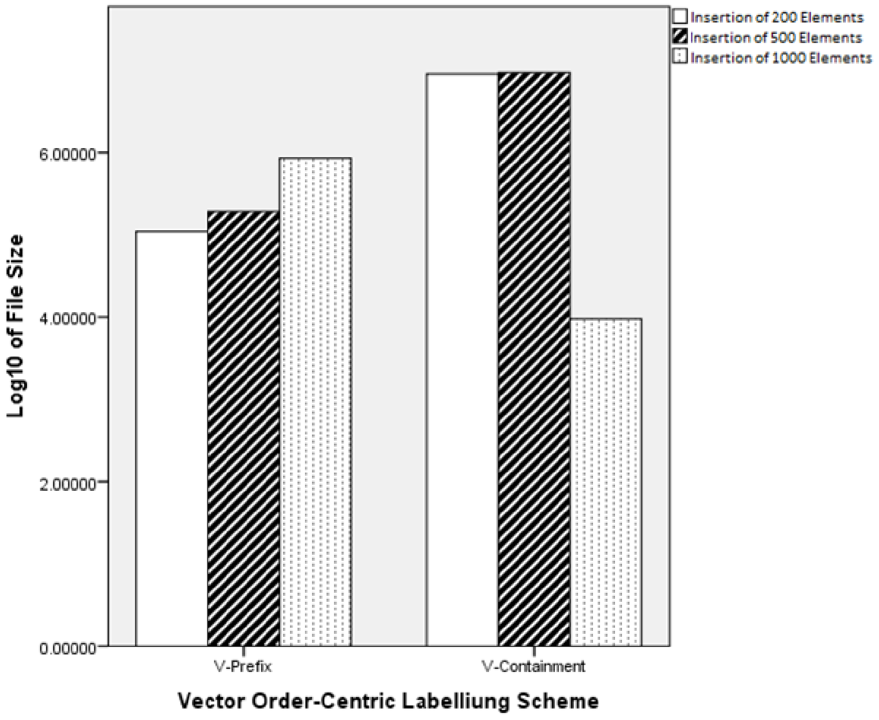


Figure 62: Logarithm Calculation of the space required for the Update three groups of elements in Nasa.

7 Conclusion and Future Works

7.1 Introduction

In this dissertation, a number of basic XML techniques which have exploited different technologies to enhance XML performance and are described in the last chapter which includes 3 parts : dissertation summary, the results of the experiments and evaluation, and future work.

7.2 Dissertation Summary

The dissertation started from the computation of the XML tree similarity in chapter 3 which is fundamental to the applications of XML documents and can play an important role in document performance. To measure the XML tree similarity, two forms were exploited to unify the document representation: tree (Alishahi et al., 2010; Antonellis et al., 2008; Guzman et al., 2013; Nayak and Xu, 2006) and vector (Asghari and KeyvanPour, 2015; Nayak and Xu, 2006; Zhang et al., 2012). These representations were adopted to measure the XML documents in terms of three characteristics XML: semantics, structure, and hybrid (semantics and structure).

In chapter 3, an algorithm for (Yang et al., 2012) was explained as an example for efficient similarity computation for XML documents. This computation is required for some techniques and applications where structure must be measured such as clustering based on structure (Choi et al., 2007; Lian et al., 2004), Chemical compounds (Dehaspe et al., 1998), and similar genetics (Bell and Guan, 2003; Mannila et al., 1997). Another team of researchers (Zhao and Tian, 2013) and (Alqarni and Pardede, 2013) studied the semantic similarity for applications such as query processing and data management to avoid duplicated data.

To increase the effectiveness of similarity measures, they adopted online dictionaries such as WordNet to find synonyms when the names of two element have the same meaning but different words, for instance, teacher and lecturer. Some applications need to measure the similarity from both semantic and structural principles. For instance in XML schema matching (Algergawy et al., 2010) and the healthcare domain (Thuy et al., 2013).

An XML schema describes the structure of the relevant XML documents and has the semantic information of the elements. To measure the similarity between two schemas efficiently, it is necessary to compare the semantics and structural similarity of the schemas. The information in the healthcare domain is sensitive because it relates to peoples health.(Guzman et al., 2013) researched the comparison of elements which have the same structure which have a different semantics such as data type. Therefore, they exploited WordNet as well to enhance the effectiveness of the similarity measurement by measuring two words that have the same meaning but different names.

In chapter 4 the similarity technique was exploited to find useful information for the user represented by a user query. The Chapter did not considered the query types that are mentioned in the section 2.8, but it considered the algorithms that used to fetch the required information based on a semantics called LCA, a node which has all query keywords. (Liu et al., 2013b) researched the query processing problem and proposed a method which split the user query into groups based on the content and the tag. The resulting units were compared with the XML document and returned the SLCA node which is based on the LCA (Xu and Papakonstantinou, 2005b).

Another employment of the semantics of SLCA was represented by the model of (Lin et al., 2014) who addressed the problem of a negative word which is preceded by the NOT operator in the user query and which may fetch 'false negative' results. The approach of (Lin et al., 2014) was an improvement on the model of (Lin et al., 2011) who excluded subtrees that have a child as a negative word and they may miss a number of useful answers. However, (Lin et al., 2014) excluded the negative elements only and fetched more subtrees which might have a required answer. More employment to the semantics of SLCA was made by(Zhou et al., 2012a) to fetch a useful answer for the user using Dewey labelling scheme. (Zhou et al., 2012a) labelled the XML tree by the Dewey technique and scattered the tree into blocks and examined each subtree that has a minimum number of keywords to be a candidate answer to the query.

As many researchers relied on the semantics SLCA to process user queries effectively and efficiently, another team of researchers such as (Zeng et al., 2013) argued that the LCA and its variants is not capable of processing range queries. Therefore, they designed a new grammar that has symbols $<, >, <=, >=$ to handle this kind of query and retrieved the required LCA node. Another investigation of the drawbacks of LCA was made by (Nguyen and Cao, 2012) who claimed that this semantics fetches a large numbers of irrelevant answers. Therefore, they suggested a new semantics called DLCA which is based on a strong relationship between the root to the intended node. Another approach was suggested by (Bao et al., 2015) to answer a query where its answer has been deleted from the database. The algorithm of (Bao et al., 2015) analysed the user query to give the user a reason the information missing and propose an alternative answer based on the data type of the required element. Another approach was designed by (Agarwal and Ramamritham, 2015) for users who do not have experience of how to query the data in the XML database. They propsed a new semantics called LCE that retrieves answer that contains at least two keywords of the query. Another support for non expert users came from (Gao et al., 2012) who presented an extension to the XPath query language that enables the user to restructure their query based its answers.

In chapter 5, three classes of labelling schemes were explained: Interval, Prefix and Multiplicative. In the interval labelling scheme, the earliest scheme designed by (Dietz, 1982)) was explained. It assigned labels to each node in an XML tree which consisted of two numbers (start, end). Allocating labels can be done based on a containment property: the labels of child nodes should be contained in the range of the labels of its ancestors. The main drawback of this scheme is determining the relationships among the nodes. Therefore, a number of labelling schemes were proposed to cover this disadvantage.

The approach of (Dietz, 1982) did not cover the P-C relationship (see section 2.6) which may reduce the effectiveness of the query processing (Subramaniam et al., 2014). So (Zhang et al., 2001) suggested a new labelling scheme that allocated an additional section in the label, the node level, to express P-C relationships. The label in the approach of (Zhang et al., 2001) is (start,end,level), where the level of the parent is one higher than that the level of the child. On the other hand, (Subramaniam et al., 2014)) defined the relationship of the nodes differently. Their node label consists of three sections (level, ordinal, rID) where level can be used to define P-C relationships and ordinal, is the node identifier. rID, is the number of the last descendant. The numbers 'ordinal' and 'rID' can be seen as an interval range for the labels. Another model in this chapter was proposed by (Fu and Meng, 2013) to determine siblings relationship and LCA semantics which important in query processing (see Chapter 4). They designed a node label based on four sections (start, end, parentid, pt), where start and end have the same meaning as in the previous methods, parentid is the identifier of the parent node, and pt is the label list of the children.

These algorithms allocated labels statically to XML tree nodes, but XML database can be updated frequently and a dynamic labelling scheme is very important to avoid the costly relabelling process. (Yun and Chung, 2008) suggested a new labelling scheme based on Interval to label XML nodes dynamically. The labels of the proposed scheme consist of (DocID, StartPos, EndPos, Level-Num), where DocID is the document ID (maybe a subtree ID in the document). StartPos is the start number of the first element in the subtrre. EndPos is the end number of the last element in the subtree. LevelNum is the node's level number. (Yun and Chung, 2008) studied insertion into XML trees in three cases: where there is enough for space the new tree, there is not enough space for the new tree, and there is no space.

The second section of this chapter was Prefix Labelling scheme which is similar to Dewey Decimal Coding used by librarians (Sans and Laurent, 2008). This model define the relationship of the nodes by delimiters that separates the node label from the prefix path which starts from the root by a delimiter (Assefa and Ergenc, 2012; Liu and Zhang, 2016; Tatarinov et al., 2002).

In this section, the best known labelling scheme which is based on Prefix and is called Dewey order encoding and was proposed by (Tatarinov et al., 2002) was explained. This scheme is a combination of two schemes of global order and local order designed by the same author. However, this scheme cannot avoid the relabelling process during update of the document. Therefore, (O'Neil et al., 2004) proposed ORDPATHs labelling scheme as a dynamic model for Dewey encoding. (O'Neil et al., 2004) initialised the node labels using odd numbers and exploited even number to label inserted nodes in the middle of the tree and negative numbers to label the nodes that are inserted at the rightmost end of the tree.

Another dynamic labelling scheme based on Prefix was demonstrated in this chapter is DFPD. DFPD was designed by (Liu et al., 2013a) and used floating-point numbers based on 'nominator/denominator' to generate a label for a node inserted between nodes. The generated label must follow the labelling order (i.e. less than the next label and greater than the previous label) and otherwise follows ORDPATHs technique.

An additional dynamic labelling scheme explained is LSDX which was invented by (Duong and Zhang, 2005). This model used numbers to represent the node level and characters rather than numbers to represent the node label to save the storage. The label of an updated node will have characters that follow alphabetic order of the siblings and represents the node level in the tree. But, this scheme has a drawback which is character repetition that may increase the size of the label. This drawback was covered as was demonstrated by (Duong and Zhang, 2008) who modelled a dynamic labelling scheme to overcome the repetition of characters in the approach of ((Duong and Zhang, 2005)). The new scheme replace repeated characters with a number equal to the repeated character, for example, '2abbb.a' will be '2a3b.a'.

The last dynamic labelling model which illustrated in chapter 5 was the model of (Assefa and Ergenc, 2012) and called OrderBased. OrderBased reduced the use of a number characters to represent the order of the node using a mathematical equation rather than of using character concatenation. Then, (Assefa and Ergenc, 2012) addressed the problem of node update in the tree and they assigned a character to the new node in a way which maintains alphabetic order of the node labels at that level.

The third section was about schemes that adopted mathematical principles to represent the structural relationship among the XML tree nodes. (Wu et al., 2004) and (Al-Shaikh et al., 2010) claimed that the preceding schemes need increasing storage as the depth and width the tree increased. So, they relied on the mathematical concepts to represent the node relationship as discussed in the Multiplicative labelling section.

(Wu et al., 2004) initialised the node labelling by exploiting two mathematical operations to represent the node relationships and suggested two properties: divisibility of label nodes in top-down technique and modulo of label the node in bottom-up tree traversal. To keep the node label's order during tree update, they employed the chinese reminder theorem to determine the relationship between the self label which is an integer value and a prime number. Moreover, they used a value called Simultaneous Congruence SC to define the order of the self label to maintain this order during the tree update. If the self label of the new node affects the labels order, the method of (Wu et al., 2004) will change the value of SC to contain the label of the new node rather than relabel all nodes of the tree.

Modulus is another mathematical principle which was exploited by (Al-Shaikh et al., 2010) and their theorem was explained in this chapter. The model of (Al-Shaikh et al., 2010) is based on two factors: a prime self label and a factor that can be achieved using an equation which based on the node's self label, nodeÂś parent label, and a large prime number which is employed to order the node labels. To update the tree, they employed the same formula used to generate a label for the new node. To determine the P-C relationship of the new node with its parent, they exploited an inverse operation to find the label of the parent from the new node.

The last multiplicative labelling scheme explained in the chapter 5 was a vector order-centric labelling scheme which was designed by (Xu et al., 2012). It is adapted to work with different labelling schemes to label the XML nodes dynamically based on a graphic vector which consists of (x,y). The first application of vector order-centric labelling was on a Containment labelling scheme that is one of the Interval schemes and the combination scheme called V-Containment. (Xu et al., 2012) used the Containment labelling scheme as an initial scheme to produce labels based on interval property. In their next step, they generated vectors where 'x' is positive. After that, they transformed each number of the node label into a correspondence vector in which the containment property is maintained. To keep the order of vectors in the interval method during the tree update, they defined the factor 'GS' and a set of rules. When the tree is updated, the vector order-centric model will generate a vector for the new node based on the node's location among its siblings (rightmost, middle, leftmost) and must be under the parent vector range. (Xu et al., 2012) applied vector order-centric approach on Dewey labelling scheme which is based on a prefix scheme. They used the Dewey algorithm to initialise the labels and generated the corresponding vectors for the labels. After that, (Xu et al., 2012) transformed each number of the node label into the correlating vector (i.e. each number correlates to a vector). A new node will have a new vector based on the prefix method, i.e. it has the prefix's path from the root and its vector will be generated based on its position among its siblings. A new child will be given a new section (vector) to follow the prefix method of labelling schemes.

7.3 The Results of the Experiments and Evaluation

In the chapter 6, two sets of six experiments were executed to measure the time and space needed to label three datasets statically: Nasa, dblp, and Treebank-e which have different width and depth. These experiments were executed to compute the time and space required to label the three databases statically. A stack of integers was employed to generate the labels of Interval method and two stacks: of integer and strings were exploited to produce the labels in the prefix labelling scheme. It is shown from the experiments that the prefix labelling scheme is better than Interval for labelling different XML tree structures from the time point of view. However, Interval is preferred for labelling deep XML trees such as Treebank-e which has 36 level which required 44,368 KB in comparison to Nasa and dblp that consumed 775,421 KB and 59,858 KB respectively. The results of these experiments matched the results of (Xu et al., 2012) and (Ghaleb and Mohammed, 2015).

Another twelve experiments were executed to label Nasa dynamically using Vector Order-Centric labelling scheme based on Prefix (V-Prefix) and Interval (V-Containment). In addition to the data structures that were used in the static labelling schemes, it used a hash table to make the transformation between the node labels and the corresponding vectors. Using the UTF-8 encoding for labelling nodes and the complexity of label computation (Gal, 2007) caused the overflow problem when labelling upto 1,000 elements. So, the results are not reliable and were ignored as they reported by (Assefa and Ergenc, 2012) and (Schmidt et al., 2002).

7.4 Future Work

XML databases are exposed to frequent updates and these changes should maintain the node label structure which is exploited by other applications such query processing. An effective labelling scheme is one which is capable of keeping the order of nodes in the tree during the dynamic update. As a result of the tree update, the XML tree will increase in size and so will the label size which may effect data access and processing. Therefore, the representation of a node label can improve the performance of the labelling scheme during the update and avoid the overflow problem where the label size runs over the storage size (O'Connor and Roantree, 2012). The representation of the node context through the label need more investigation as future work to improve dynamic labelling schemes.

References

Xml data repository. URL `http://www.cs.washington.edu/research/xmldatasets/www/repository.html`.

Xml schema- data types quick reference. website, 2003.

Frederic Achard, Guy Vaysseix, and Emmanuel Barillot. Xml, bioinformatics and data integration. *Bioinformatics*, 17(2):115–125, 2001.

ADL. Scorm 2004 4th edition version 1.1 overview. Accessed: 2016-05-23.

Manoj K Agarwal and Krithi Ramamritham. Enabling generic keyword search over raw xml data. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1496–1499. IEEE, 2015.

Ali Aïtelhadj, Mohand Boughanem, Mohamed Mezghiche, and Fatiha Souam. Using structural similarity for clustering xml documents. *Knowledge and Information Systems*, 32(1):109–139, 2012.

Raed Al-Shaikh, Ghalib Hashim, AbdulRahman BinHuraib, and Salahadin Mohammed. A modulo-based labeling scheme for dynamically ordered xml trees. In *Digital Information Management (ICDIM), 2010 Fifth International Conference on*, pages 213–221. IEEE, 2010.

Alsayed Algergawy, Richi Nayak, and Gunter Saake. Element similarity measures in xml schema matching. *Information Sciences*, 180(24):4975–4998, 2010.

Alsayed Algergawy, Marco Mesiti, Richi Nayak, and Gunter Saake. Xml data clustering: An overview. *ACM Computing Surveys (CSUR)*, 43(4):25, 2011.

Mohamad Alishahi, Mahmoud Naghibzadeh, and Baharak Shakeri Aski. Tag name structure-based clustering of xml documents. *International Journal of Computer and Electrical Engineering*, 2(1):119, 2010.

Alaa Almelibari. *Labelling Dynamic XML Documents: A GroupBased Approach*. PhD thesis, University of Sheffield, 2015.

Alaa Abdulbasit Almelibari and Siobhan North. Query processor for native xml databases.

Ahmad Abdullah Alqarni and Eric Pardede. Internal filtering approach toward efficiency optimization of matching large scale xml schemas. In *Proceedings of the 2013 16th International Conference on Network-Based Information Systems*, NBIS '13, pages 464–469, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-1-4799-2510-0. doi: 10.1109/NBiS.2013.77. URL `http://dx.doi.org/10.1109/NBiS.2013.77`.

Panagiotis Antonellis, Christos Makris, and Nikos Tsirakis. Xedge: clustering homogeneous and heterogeneous xml documents using edge summaries. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1081–1088. ACM, 2008.

Elaheh Asghari and MohammadReza KeyvanPour. Xml document clustering: techniques and challenges. *Artificial Intelligence Review*, 43(3):417–436, 2015.

Beakal Gizachew Assefa and Belgin Ergenc. Orderbased labeling scheme for dynamic xml query processing. In *Multidisciplinary Research and Practice for Information Systems*, pages 287–301. Springer, 2012.

Zhifeng Bao, Tok Wang Ling, Bo Chen, and Jiaheng Lu. Effective xml keyword search with relevance oriented ranking. In *2009 IEEE 25th International Conference on Data Engineering*, pages 517–528. IEEE, 2009.

Zhifeng Bao, Yong Zeng, Tok Wang Ling, Dongxiang Zhang, Guoliang Li, and H. V. Jagadish. A general framework to resolve the mismatch problem in xml keyword search. *The VLDB Journal*, 24(4):493–518, 2015. ISSN 0949-877X. doi: 10.1007/s00778-015-0386-1. URL `http://dx.doi.org/10.1007/s00778-015-0386-1`.

David A Bell and JW Guan. Data mining for motifs in dna sequences. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 507–514. Springer, 2003.

Z. Bellahsène. *Database and XML Technologies: First International XML Database Symposium, XSYM 2003, Berlin, Germany, September 8, 2003, Proceedings.* Lecture Notes in Computer Science. Springer, 2003. ISBN 9783540200550. URL `https://books.google.co.uk/books?id=2387N4nnB-IC`.

Boag S. Chamberlin D. FernÃąndez M.F. Kay M. Robie J. SimÃľon J. Berglund, A. W3c, xml path language (xpath) 2.0. Accessed: 2016-06-20.

Elisa Bertino and Elena Ferrari. Xml and data integration. *IEEE internet computing*, 5(6):75–76, 2001.

Permanente K. Malhotra A. Biron, P.V. Xml schema part2:datatypes seconde edition. website, 8 2004. URL `http://www.w3.org/TR/xmlschema-2/`. Last access: 13.2.2016.

Chamberlin D. Fernndez M. F. Florescu D. Robie J. Boag, S. and J. Simon. Xquery 1.0: An xml query language, technical report. Accessed: 2016-05-20.

Ion P. Miner P. Carlisle, D. W3c, mathematical markup language (mathml) version 3.0., 2014. URL `http://www.w3.org/TR/MathML/`. Accessed: 2016-05-21.

Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An xml query language for heterogeneous data sources. In *International Workshop on the World Wide Web and Databases*, pages 1–25. Springer, 2000.

Sudarshan S Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *ACM SIGMOD Record*, volume 25, pages 493–504. ACM, 1996.

Sudarshan S Chawathe et al. Comparing hierarchical data in external memory. In *VLDB*, volume 99, pages 90–101. Citeseer, 1999.

Liang Jeff Chen and Yannis Papakonstantinou. Supporting top-k keyword search in xml databases. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 689–700. IEEE, 2010.

Ilhwan Choi, Bongki Moon, and Hyoung-Joo Kim. A clustering method based on path similarities of xml data. *Data & Knowledge Engineering*, 60(2):361–376, 2007.

Curbera F. Meredith G. WeeraWarana S. Christensen, E. Web services description language (wsdl) 1.1. URL `http://www.w3.org/TR/wsdl`.

Gregory Cobena, Serge Abiteboul, and Amelie Marian. Detecting changes in xml documents. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 41–52. IEEE, 2002.

Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. Xsearch: A semantic search engine for xml. In *Proceedings of the 29th international con-*

ference on Very large data bases-Volume 29, pages 45–56. VLDB Endowment, 2003.

Gianni Costa and Riccardo Ortale. On effective xml clustering by path commonality: An efficient and scalable algorithm. In *2012 IEEE 24th International Conference on Tools with Artificial Intelligence*, volume 1, pages 389–396. IEEE, 2012.

Gianni Costa and Riccardo Ortale. Developments in partitioning xml documents by content and structure based on combining multiple clusterings. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 477–482. IEEE, 2013.

Luc Dehaspe, Hannu Toivonen, and Ross D King. Finding frequent substructures in chemical compounds. In *KDD*, volume 98, page 1998, 1998.

Maler E. Orchard D DeRose, S. W3c, xml link language (xlink) version 1.0., 2001. Accessed: 2016-06-20.

Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. on Knowl. and Data Eng.*, 17(8):1036–1050, August 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.127. URL `http://dx.doi.org/10.1109/TKDE.2005.127`.

Paul F Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122–127. ACM, 1982.

W. Drol. *Object-Oriented Macromedia Flash MX*. Expert's voice. Apress, 2008. ISBN 9781430208389. URL `https://books.google.co.uk/books?id=wvIMcuyc1cgC`.

Maggie Duong and Yanchun Zhang. Lsdx: a new labelling scheme for dynamically updating xml data. In *Proceedings of the 16th Australasian database conference-Volume 39*, pages 185–193. Australian Computer Society, Inc., 2005.

Maggie Duong and Yanchun Zhang. Dynamic labelling scheme for xml data processing. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 1183–1199. Springer, 2008.

K. Ethier. *XML and FrameMaker*. Apress, 2008. ISBN 9781430207191. URL `https://books.google.co.uk/books?id=74X3suYvyoUC`.

Lizhen Fu and Xiaofeng Meng. Triple code: An efficient labeling scheme for query answering in xml data. In *Web Information System and Application Conference (WISA), 2013 10th*, pages 42–47. IEEE, 2013.

Avigdor Gal. The generation y of xml schema matching panel description. In *International XML Database Symposium*, pages 137–139. Springer, 2007.

Yingfei Gao, Hui Tang, and Kuipeng Xue. Effectively extracting useful information from complex structured xml databases. In *Computer Science and Information Processing (CSIP), 2012 International Conference on*, pages 1121–1125. IEEE, 2012.

Taher Ahmed Ghaleb and Salahadin Mohammed. A dynamic labeling scheme based on logical operators: A support for order-sensitive xml updates. *Procedia Computer Science*, 57:1211–1218, 2015.

Charles F Goldfarb and Paul Prescod. *XML Handbook with CD-ROM*. Prentice Hall PTR, 2001.

Wei Gong and Peng Yao. Based on grammar analysis's expressiveness among the different xml-schema languages. In *Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on*, pages 116–119, May 2013. doi: 10.1109/ICSESS.2013.6615268.

Ji W Guan, David A Bell, and Dayou Liu. Discovering maximal frequent patterns in sequence groups. In *International Conference on Rough Sets and Current Trends in Computing*, pages 602–609. Springer, 2004.

D. Gulbransen. *Using XML*. SPECIAL EDITION USING. Que, 2002. ISBN 9780789727480. URL `https://books.google.co.uk/books?id=aqfvLQzR-GkC`.

Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 16–27. ACM, 2003.

Renato Guzman, Irvin Dongo, and Regina Ticona Herrera. Structural and semantic similarity for xml comparison. In *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*, pages 177–181. ACM, 2013.

Markus Hagenbuchner, Alessandro Sperduti, Ah Chung Tsoi, Francesca Trentini, Franco Scarselli, and Marco Gori. Clustering xml documents using self-organizing maps for structures. In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 481–496. Springer, 2005.

E.R. Harold and W.S. Means. *XML in a Nutshell*. In a Nutshell (O'Reilly). O'Reilly Media, 2004. ISBN 9781449379049. URL `https://books.google.co.uk/books?id=NBwnSfoCStAC`.

Su-Cheng Haw and Chien-Sing Lee. Data storage practices and query processing in xml databases: A survey. *Knowledge-Based Systems*, 24(8):1317–1340, 2011.

Jan Hegewald, Felix Naumann, and Melanie Weis. Xstruct: Efficient schema extraction from multiple and large xml documents. In *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, pages 81–81. IEEE, 2006.

Hosagrahar V Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks VS Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu, et al. Timber: A native xml database. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 11(4):274–291, 2002.

Tae-Soon Kim, Ju-Hong Lee, Jae-Won Song, and Deok-Hwan Kim. Similarity measurement of xml documents based on structure and contents. In *International Conference on Computational Science*, pages 902–905. Springer, 2007.

Lingbo Kong, Rémi Gilleron, and Aurélien Lemay Mostrare. Retrieving meaningful relaxed tightest fragments for xml keyword search. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 815–826. ACM, 2009.

Richard E Korf. *Artificial intelligence search algorithms.* Chapman & Hall/CRC, 2010.

Atakan Kurt and Mustafa Atay. An experimental study on query processing efficiency of native-xml and xml-enabled database systems. In *International Workshop on Databases in Networked Information Systems*, pages 268–284. Springer, 2002.

Thuy Ngoc Le, Zhifeng Bao, and Tok Wang Ling. Exploiting semantics for xml keyword search. *Data & Knowledge Engineering*, 99:105–125, 2015.

W.M. Lee and S.M. Foo. *XML Programming Using the Microsoft XML Parser*. Apress, 2008. ISBN 9781430208297. URL `https://books.google.co.uk/books?id=6bgYAAAAQBAJ`.

Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective keyword search for valuable lcas over xml documents. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 31–40. ACM, 2007.

Yunyao Li, Cong Yu, and HV Jagadish. Schema-free xquery. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 72–83. VLDB Endowment, 2004.

Wang Lian, Nikos Mamoulis, Siu-Ming Yiu, et al. An efficient and scalable algorithm for clustering xml documents by structure. *IEEE transactions on Knowledge and Data Engineering*, 16(1):82–96, 2004.

Rung-Ren Lin, Ya-Hui Chang, and Kun-Mao Chao. Identifying relevant matches with not semantics over xml documents. In *International Conference on Database Systems for Advanced Applications*, pages 466–480. Springer, 2011.

Rung-Ren Lin, Ya-Hui Chang, and Kun-Mao Chao. Locating valid slcas for xml keyword search with not semantics. *ACM SIGMOD Record*, 43(2):29–34, 2014.

Jian Liu and XX Zhang. Dynamic labeling scheme for xml updates. *Knowledge-Based Systems*, 2016.

Jian Liu, ZM Ma, and Li Yan. Efficient labeling scheme for dynamic xml trees. *Information Sciences*, 221:338–354, 2013a.

Xiping Liu, Lei Chen, Changxuan Wan, Dexi Liu, and Naixue Xiong. Exploiting structures in keyword queries for effective xml search. *Information Sciences*, 240:56–71, 2013b.

Ziyang Liu and Yi Chen. Identifying meaningful return information for xml keyword search. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 329–340. ACM, 2007.

Ziyang Liu and Yi Cher. Reasoning and identifying relevant matches for xml keyword search. *Proceedings of the VLDB Endowment*, 1(1):921–932, 2008.

Jiaheng Lu and Tok Wang Ling. Labeling and querying dynamic xml trees. In *Asia-Pacific Web Conference*, pages 180–189. Springer, 2004.

Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

MarkLogic. New generation big data requires a new generation database. website, 2016. URL `http://www.marklogic.com/`. Last access: 4.3.2016.

Microsoft. Xml elements' attributes, and types (xml designer). website, 2016. URL `https://msdn.microsoft.com/en-us/library/7f0tkwcx(v=vs.80).aspx`. Last access: 8.9.2016.

Irena Mlỳnková. An analysis of approaches to xml schema inference. In *SITIS*, pages 16–23, 2008.

Irena Mlỳnková and Martin Nečaskỳ. Heuristic methods for inference of xml schemas: Lessons learned and open issues. *Informatica*, 24(4):577–602, 2013.

Marta Mota, Paulo Caetano da Silva, and Sidney Viana. Similarity evaluation in xml schema and xlink. In *Proceedings of the 19th Brazilian symposium on Multimedia and the web*, pages 153–156. ACM, 2013.

Richi Nayak and Sumei Xu. Xcls: a fast and effective clustering algorithm for heterogenous xml documents. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 292–302. Springer, 2006.

Khanh Nguyen and Jinli Cao. Top-k answers for xml keyword queries. *World Wide Web*, 15(5-6):485–515, 2012.

S. Oaks. *Java Performance: The Definitive Guide: Getting the Most Out of Your Code*. O'Reilly Media, 2014. ISBN 9781449363543. URL `https://books.google.co.uk/books?id=aIhUAwAAQBAJ`.

Martin F O'Connor and Mark Roantree. Scooter: A compact and scalable dynamic labeling scheme for xml updates. In *International Conference on Database and Expert Systems Applications*, pages 26–40. Springer, 2012.

University of Toronto. Toronto xml server. website, 2002. URL `http://www.cs.toronto.edu/tox/`. Last access: 4.3.2016.

Patrick O'Neil, Elizabeth O'Neil, Shankar Pal, Istvan Cseri, Gideon Schaller, and Nigel Westbury. Ordpaths: insert-friendly xml node labels. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 903–908. ACM, 2004.

Ashok Malhotra Paul V Biron, Kaiser Permanente. Xml schema part 2: Datatypes second edition. website, 10 2004. URL `http://www.w3.org/TR/xmlschema-2/`. Last access: 31.1.2016.

Zhong-Ren Peng and Chuanrong Zhang. The roles of geography markup language (gml), scalable vector graphics (svg), and web feature service (wfs) specifications in the development of internet geographic information systems (gis). *Journal of Geographical Systems*, 6(2):95–116, 2004.

Maciej Piernik, Dariusz Brzezinski, Tadeusz Morzy, and Anna Lesniewska. Xml clustering: a review of structural approaches. *The Knowledge Engineering Review*, 30(03):297–323, 2015.

Maciej Piernik, Dariusz Brzezinski, and Tadeusz Morzy. Clustering xml documents by patterns. *Knowledge and Information Systems*, 46(1):185–212, 2016.

A Mary Posonia and VL Jyothi. Structural-based clustering technique of xml documents. In *Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on*, pages 1239–1242. IEEE, 2013.

G. Powell. *Beginning XML Databases*. Programmer to programmer. Wiley, 2007. ISBN 9780471791201. URL `https://books.google.co.uk/books?id=-9vNi1tqpP0C`.

Chamberlin D. Dyck M. Robie, J. and J. Snelson. W3c, xml path language (xpath) 3.0. Accessed: 2016-05-20.

Airi Salminen and Frank Wm Tompa. Requirements for xml document database systems. In *Proceedings of the 2001 ACM Symposium on Document engineering*, pages 85–94. ACM, 2001.

Virginie Sans and Dominique Laurent. Prefix based numbering schemes for xml: techniques, applications and performances. *Proceedings of the VLDB Endowment*, 1(2):1564–1573, 2008.

F. Schaffer. *Number Chart 1-100*. Carson Dellosa Publishing Company Incorporated, 2001. ISBN 9780768212327. URL `https://books.google.co.uk/books?id=hvZAAAAACAAJ`.

Torsten Schlieder. Similarity search in xml data using cost-based query transformations.

Albrecht Schmidt, Martin Kersten, and Menzo Windhouwer. Querying xml documents made easy: Nearest concept queries. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 321–329. IEEE, 2001.

Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J Carey, Ioana Manolescu, and Ralph Busse. Xmark: A benchmark for xml data management. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 974–985. VLDB Endowment, 2002.

Wen-Chung Shih, Chao-Tung Yang, and Shian-Shyong Tseng. A methodology for retrieving scorm-compliant teaching materials on grid environments. In *International Conference on Asian Digital Libraries*, pages 498–502. Springer, 2006.

Marko Smiljanić, Maurice Van Keulen, and Willem Jonker. Formalizing the xml schema matching problem as a constraint optimization problem. In *International Conference on Database and Expert Systems Applications*, pages 333–342. Springer, 2005.

Alessandro Solimando, Giorgio Delzanno, and Giovanna Guerrini. Validating xml document adaptations via hedge automata transformations. *Theoretical Computer Science*, 560:251–268, 2014.

Samini Subramaniam and Su-Cheng Haw. Me labeling: A robust hybrid scheme for dynamic update in xml databases. In *Telecommunication Technologies (ISTT), 2014 IEEE 2nd International Symposium on*, pages 126–131. IEEE, 2014.

Samini Subramaniam, Su-Cheng Haw, and Lay-Ki Soon. Relab: A subtree based labeling scheme for efficient xml query processing. In *Telecommunication Technologies (ISTT), 2014 IEEE 2nd International Symposium on*, pages 121–125. IEEE, 2014.

Chong Sun, Chee-Yong Chan, and Amit K Goenka. Multiway slca-based keyword search in xml data. In *Proceedings of the 16th international conference on World Wide Web*, pages 1043–1052. ACM, 2007.

Igor Tatarinov, Stratis D Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 204–215. ACM, 2002.

Joe Tekli and Richard Chbeir. A novel xml document structure comparison framework based-on sub-tree commonalities and label semantics. *Web Semantics: Science, Services and Agents on the World Wide Web*, 11:14–40, 2012.

Joe Tekli, Richard Chbeir, and Kokou Yetongnon. Structural similarity evaluation between xml documents and dtds. In *International Conference on Web Information Systems Engineering*, pages 196–211. Springer, 2007.

Joe Tekli, Richard Chbeir, and Kokou Yetongnon. An overview on xml similarity: Background, current trends and future directions. *Computer science review*, 3(3):151–173, 2009.

Joe Tekli, Richard Chbeir, Agma JM Traina, Caetano Traina, and Renato Fileto. Approximate xml structure validation based on document–grammar tree similarity. *Information Sciences*, 295:258–302, 2015.

Pham Thu Thu Thuy, Young-Koo Lee, and Sungyoung Lee. Semantic and structural similarities between xml schemas for integration of ubiquitous healthcare data. *Personal and ubiquitous computing*, 17(7):1331–1339, 2013.

Zongqi Tian, Jiaheng Lu, and Deying Li. A survey on xml keyword search. In *Asia-Pacific Web Conference*, pages 460–471. Springer, 2011.

C.M. Sperberg-McQueen Eva Maler FranÃğois Yergeau Tim Bary, Jeaqn Paoli. Extensible markup language (xml) 1.0 (fifth edition). website, 11 2008. URL `http://www.w3.org/TR/2008/REC-xml-20081126/`. Last access: 31.1.2016.

w3schools. Xsd idicatore. website, 2016. URL `http://www.w3schools.com/xml/schema_complex_indicators.asp/`. Last access: 20.2.2016.

Guoren Wang and Mengchi Liu. Query processing and optimization for regular path expressions. In *International Conference on Advanced Information Systems Engineering*, pages 30–45. Springer, 2003.

Xu Wang, Jinmao Wei, Baoquan Fan, and Ting Yang. Voting affinity propagation algorithm for clustering xml documents. In *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, pages 1907–1913. IEEE, 2012.

Erik Wilde. *Wildeś WWW: technical foundations of the World Wide Web*. Springer Science & Business Media, 2012.

Erik Wilde and Robert J Glushko. Xml fever. *Queue*, 6(6):46–53, 2008.

Khin-Myo Win, Wee-Keong Ng, and Ee-Peng Lim. Enaxs: efficient native xml storage system. In *Asia-Pacific Web Conference*, pages 59–70. Springer, 2003.

Xiaodong Wu, Mong-Li Lee, and Wynne Hsu. A prime number labeling scheme for dynamic ordered xml trees. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 66–78. IEEE, 2004.

Liang Xu, Zhifeng Bao, and Tok Wang Ling. A dynamic labeling scheme using vectors. In *International Conference on Database and Expert Systems Applications*, pages 130–140. Springer, 2007.

Liang Xu, Tok Wang Ling, and Huayu Wu. Labeling dynamic xml documents: an order-centric approach. *Knowledge and Data Engineering, IEEE Transactions on*, 24(1):100–113, 2012.

Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 527–538, New York, NY, USA, 2005a. ACM. ISBN 1-59593-060-4. doi: 10.1145/1066157.1066217. URL `http://doi.acm.org/10.1145/1066157.1066217`.

Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 527–538. ACM, 2005b.

Yu Xu and Yannis Papakonstantinou. Efficient lca based keyword search in xml data. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 535–546. ACM, 2008.

Ting Yang, Jinmao Wei, Baoquan Fan, Xu Wang, and Haiwei Zhang. Structural similarity computation based on extended edge matching method. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, pages 1201–1205. IEEE, 2012.

Francois Yergeau. Utf-8, a transformation format of iso 10646. 2003.

Jeffrey Xu Yu, Daofeng Luo, Xiaofeng Meng, and Hongjun Lu. Dynamically updating xml data: numbering scheme revisited. *World Wide Web*, 8(1):5–26, 2005.

Jung-Hee Yun and Chin-Wan Chung. Dynamic interval-based labeling scheme for efficient xml query and update processing. *Journal of Systems and Software*, 81(1):56–70, 2008.

Yong Zeng, Zhifeng Bao, and Tok Wang Ling. Supporting range queries in xml keyword search. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 97–104. ACM, 2013.

Amar Zerdazi and Myriam Lamolle. Computing path similarity relevant to xml schema matching. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 66–75. Springer, 2008.

Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On supporting containment queries in relational database management systems. In *ACM SIGMOD Record*, volume 30, pages 425–436. ACM, 2001.

Xue-Liang Zhang, Ting Yang, Bao-Quan Fan, Xu Wang, and Jin-Mao Wei. Novel method for measuring structure and semantic similarity of xml documents based on extended adjacency matrix. *Physics Procedia*, 24:1452–1461, 2012.

Zhongping Zhang, Rong Li, Shunliang Cao, and Yangyong Zhu. Similarity metric for xml documents. In *Knowledge Management and Experience Management Workshop*, 2003.

Guofeng Zhao and Shan Tian. Research on xml keyword query method based on semantic. In *Information Science and Cloud Computing Companion (ISCC-C), 2013 International Conference on*, pages 806–811. IEEE, 2013.

X. Zheng and University of South Carolina. *Chinese Remainder Theorem Based Single and Multi-group Key Management Protocols*. University of South Carolina, 2007. ISBN 9780549212294. URL `https://books.google.co.uk/books?id=BbWuczr-7SAC`.

Junfeng Zhou, Zhifeng Bao, Ziyang Chen, Guoxiang Lan, Xudong Lin, and Tok Wang Ling. Top-down slca computation based on list partition. In *International Conference on Database Systems for Advanced Applications*, pages 172–184. Springer, 2012a.

Junfeng Zhou, Zhifeng Bao, Wei Wang, Tok Wang Ling, Ziyang Chen, Xudong Lin, and Jingfeng Guo. Fast slca and elca computation for xml keyword queries based on set intersection. In *2012 IEEE 28th International Conference on Data Engineering*, pages 905–916. IEEE, 2012b.

Rui Zhou, Chengfei Liu, and Jianxin Li. Fast elca computation for keyword queries on xml data. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 549–560. ACM, 2010.

Canwei Zhuang and Shaorong Feng. Full tree-based encoding technique for dynamic xml labeling schemes. In *International Conference on Database and Expert Systems Applications*, pages 357–368. Springer, 2012.

Appendices

A Appendix: Prefix Labelling Scheme

```
Prefix Labelling Scheme ()
  Input: XML File;
  Output: Set of Labels;
  begin
        - Read the root element of the XML file;
        - stack = parent, child;
        While (not End of File)
        begin
          - If (the tag is start tag)
            begin
              - if (child is empty)
                begin
                  - child.push = 0;
                  - parent.push = 1;
                end;
              - else
                begin
                  - read start tag;
                  - x = child.get(child.size-1);
                  - x++;
                  - child.push(((child.size) - 1), x);
                  - push.child (0);
```

```
                        - parent.get((parent.size) - 1);
                        - parent.push (parent,x);
                    end;
                end;
            - else
                begin
                    - child.pop;
                    - parent.pop;
                end;
            end;
end.
```

B Appendix: Interval Labelling Scheme

```
Interval Labelling Scheme ()
    Input: XML File;
    Output: Set of Labels;
    begin
            - Initialisation: counter = 0;
            While (Not End of File)
                begin
                    - Read the root element of the XML file;
                    - If (the tag is start tag)
                        begin
                            - push counter to stack;
                            - counter++;
                        end;
                    else
                        begin
                            - pre = stack.pop;
                            - post = pre;
                            - if (pre != counter)
                                begin
                                    - post = counter ++;
                                end:
                            - level = element depth in the stack;
                            - assign the label (pre.post.level) to the element;
                            end;
    end.
```

C Appendix: Vector Order-Centric Labelling Scheme Based on Prefix (V-Prefix)

```
V-Prefix ()
  Input: max No. of Elements;
  Output: Set of Vectors;
  begin
      - initialisation:
        - max = max No. of XML elements;
        - first = 1;
        - last = max;
        - HT = Hashtable;
        Create (first, last);
          begin
            - if (HT.size == max )
              - return;
            - else
              begin
                - mid = ceil ((first+last) div 2);
                - if (mid is not in HT)
                  begin
                    - $x_mid = x_first + x_last$;
                    - $y_mid = y_first + y_last$;
                    - HT.put (mid, $(x_mid, y_mid)$);
                    Create (mid, last);
                  end;
                - else
                    - Create (first, mid);
                end;
                    end;
Prefix ()
Input: XML File;
Output: Set of Vector-Label;
begin
        - Read the root element of the XML file;
        - stack = parent, child;
        While (not End of File)
            begin
                - If (the tag is start tag)
                    begin
                        - if (child is empty)
                            begin
                                - child.push = 0;
                                - parent.push = HT.get(1);
                            end;
                        - else
                            begin
                                - read start tag;
                                - x = child.get((child.size) - 1);
                                - x++;
                                - child.push((child.size) - 1, x);
                                - push.child (0);
                        - parent.get((parent.size) - 1);
                        - parent.push (HT.get(parent),HT.get(x));
                    end;
                end;
            - else
                begin
                    - child.pop;
                    - parent.pop;
                end;
            end;
end.
```

D Appendix: Vector Order-Centric Labelling Scheme Based on Containment (V-Containment)

```
V-Containment ()
  Input: max No. of Elements;
  Output: Vectors;
  begin
      - initialisation:
        - max = max No. of XML elements;
        - first = 1;
        - last = max;
        - HT = Hashtable;
        Create (first, last);
          begin
            - if (HT.size == max )
              - return;
            - else
              begin
                - mid = ceil ((first+last) div 2);
                - if (mid is not in HT)
                  begin
                    - x(mid) = x(first) + x(last);
                    - y(mid) = y(first) + y(last);
                    - HT.put (mid, (x(mid),y(mid)));
                    Create (mid, last);
                  end;
                  - else
                      - Create (first, mid);
                  end;
              end;
Interval Labelling Scheme ()
Input: XML File;
Output: Set of Labels;
begin
- Initialise counter;
      While (Not End of File)
          begin
              - Read the root element of the XML file;
              - If (the tag is start tag)
                  begin
                      - push counter to stack;
                      - counter++;
                  end;
              else
                  begin
                      - pre = stack.pop;
                      - post = pre;
                      - if (pre != counter)
                          begin
                              post = counter ++;
                          end:
                      - level = element depth in the stack;
```

```
                    - assign the label (HT.get(pre).HT.get(post).level) to
the element;
                end;
            end;
        end.
```