PENTAGONAL SCHEME FOR DYNAMIC XML PREFIX LABELLING

E.A.M. TAKTEK

PhD

2020

PENTAGONAL SCHEME FOR DYNAMIC XML PREFIX LABELLING

Ebtesam TAKTEK

Submitted for the Degree of

Doctor of Philosophy

(PhD Thesis)

Faculty of Engineering and Informatics

Department of Computer Science

University of Bradford

2020

Abstract

Ebtesam Taktek

PENTAGONAL SCHEME FOR DYNAMIC XML PREFIX LABELLING

Keywords: XML Labelling, Prefix Scheme, Dewey labelling, Dynamic Scheme

In XML databases, the indexing process is based on a labelling or numbering scheme and generally used to label an XML document to perform an XML query using the path node information. Moreover, a labelling scheme helps to capture the structural relationships during the processing of queries without the need to access the physical document. Two of the main problems for labelling XML schemes are duplicated labels and the cost efficiency of labelling time and size. This research presents a novel dynamic XML labelling scheme, called the Pentagonal labelling scheme, in which data are represented as ordered XML nodes with relationships between them. The update of these nodes from large-scale XML documents has been widely investigated and represents a challenging research problem as it means relabelling a whole tree. Our algorithms provide an efficient dynamic XML labelling scheme that supports data updates without duplicating labels or relabelling old nodes. Our work evaluates the labelling process in terms of size and time, and evaluates the labelling scheme’s ability to handle several insertions in XML documents. The findings indicate that the Pentagonal scheme shows a better initial labelling time performance than the compared schemes, particularly when using large XML datasets. Moreover, it efficiently supports random skewed updates, has fast calculations and uncomplicated implementations so efficiently handles updates. Also, it proved its capability in terms of the query performance and in determining the relationships.

Acknowledgements

Firstly, I praise God for providing me with the perseverance, courage and strength to complete this research.

I wish to thank my principal supervisor, Dr Dhavalkumar Thakker, for his advice, feedback, support, and encouragement. He is truly an excellent supervisor. I am incredibly grateful.

Also, I wish to thank my associate supervisor Prof Daniel Neagu.

A special thanks go to my parents (Abdullatif and Bchira) who I am beyond grateful to, for their endless support, encouragement and prayers were what sustained me this far.

I am extremely grateful to my husband (Sulaiman) for his encouragement and for giving me the motivation I needed to keep going throughout this journey. I am incredibly appreciative of my lovely daughters (Malak, Mais and Goud) for being considerate and helpful; I feel blessed because of you all. Also, my baby son (Omar) for bringing more happiness to our lives. Many thanks to my incredible siblings (Ahmed, Khiria, Eman and Mahamed) and all my family members for their prayers and their unwavering support. I am beyond thankful to all my colleagues and friends for their encouragement and their help in any way they can.

I am also obliged to the Libyan government, which gave me the chance and funding to complete my PhD degree.

Publications and Presentations

Some of the contents of this thesis were published in a conference and some others in a Journal paper by Knowledge-Based Systems Journal.

- Taktek, E., Thakker, D. and Neagu, D. “Comparison between Range-based and Prefix Dewey Encoding”. DOI: 10.5220/0007229003640368 In Proceedings of the 14th International Conference on Web Information Systems and Technologies (WEBIST 2018), ISBN: 978-989-758-324-7, pages 364-368. Science and Technology Publications. Seville, Spain.
- E. Taktek and D. Thakker, Pentagonal scheme for dynamic XML prefix labelling, Accepted for the Elsevier Knowledge-Based Systems (2020) 106446, https://doi.org/10.1016/j.knosys.2020.106446. (impact factor: 5.921).
- Oral Presentation at the 1st Annual Innovative Engineering Research Conference (AIERC 2017), Labelling fuzzy spatiotemporal XML documents using IFDewey postfix code. 17th July 2017, University of Bradford, UK.
- Poster at the 1st Annual Innovative Engineering Research Conference (AIERC 2017), University of Bradford 17th July 2017.

- Poster at the 2nd Annual Innovative Engineering Research Conference (AIERC 2018), University of Bradford 17th October 2018.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

The development of XML was originally expected to help website designers. Today scientists, publishers and database development managers, archive administrators and different analysts utilize XML to handle their data [1]. Through inductive systems and distributed databases systems, XML data management has been adapted for many applications which range from geographical, bioinformatics and engineering data to customer services and cash flow improvements [2]. This is because XML data afford a simple format that both humans and machines can understand [3-5]. XML facilitates the ability to define the content of an XML document separately from its format; this makes it simple to both share and reuse data in different applications [1].

The increasingly widespread use of eXtensible Markup Language (XML) for storage and exchange of data due to its self-describing and its ability of organizing data has led to greater interest in the further development of systems that are able to store and query XML data [6].

A labelling scheme assigns a unique code to each node in an XML tree to create the relationship that exists between the nodes in the tree and to facilitate query processing [7-12]. In order to enable the determining of the relationships among nodes; different labelling schemes were proposed to process queries efficiently. Figure 1.1 shows an example of a labelling XML tree using the interval labelling scheme [13].

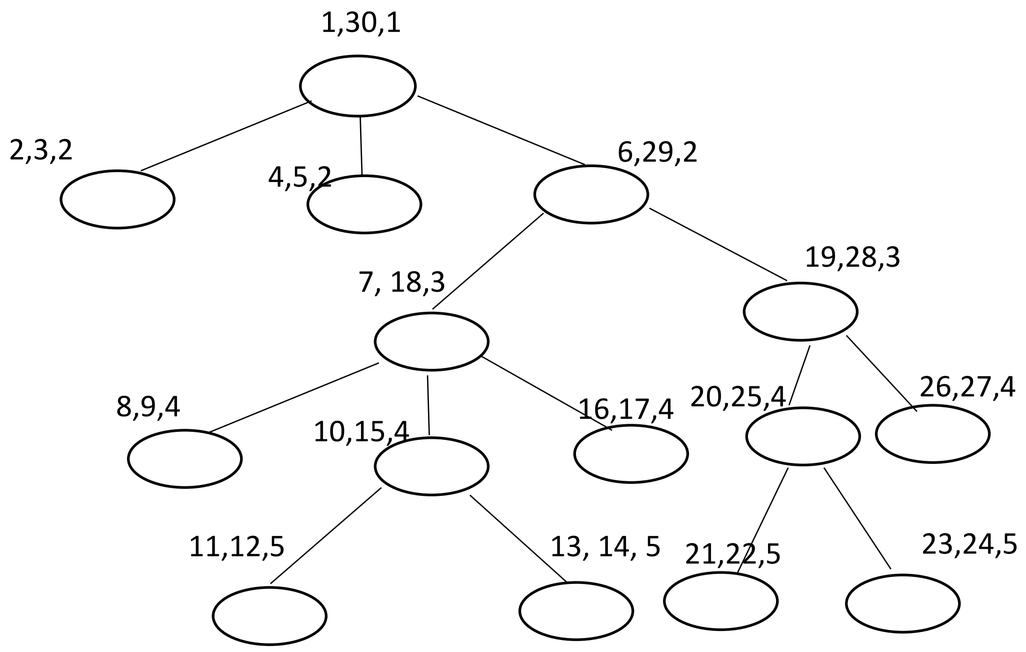


Figure 1.1 An example of a labelling XML tree using the interval labelling scheme.

In Figure 1.1 the interval labelling scheme stores a combination of values to each node. Each label is represented as a 3-tuple < pre-order, post-order, depth >, which is used to identify the exact position of an element.

The pre-order traverses across the ordered tree starting from the root and handing each level from left to right, while the post-order traverses by visiting the leaf nodes from left to right, and then processing their parent level from bottom to top. Also, the depth determines the level of the nodes [14].

This approach to labelling thoroughly manages XML data as the labels study the position of the node to sort the order of the document. This improves storage, updates and queries on XML data [15]. Labelling schemes have the ability to provide identification in order to maintain the structural relationships between elements as parent-child, ancestor-descendent and sibling. The order of the nodes is based on a comparison to their labels.

An XML query based on a labelling scheme is the same as relational database queries that depend on indexing. Consequently, an ordered XML Tree and structural information, such as parent/child or ancestor/descendant, are encoded into highly compressed labels by labelling schemes.

An XML labelling scheme requires less storage space and provides more flexibility compared with other XML query techniques [16-19], for example, the structural indexing techniques, where each node of an XML database is referenced within the index along with its path summary from its root to the designated node. This results in a larger storage space [16].

This thesis proposes a new labelling scheme and considers the restrictions of the existing XML labelling schemes to improve the efficiency of XML data management systems. It focuses on the size of XML labels and the time taken for the labelling process. It also evaluates the query performance and the labelling scheme’s ability to handle different types of update.

The remaining sections of this chapter are organized as follows: Section 1.1 presents the importance of XML labelling schemes, while section 1.2 describes the research motivation and hypothesis. Moreover, the chapter explains the research aims and objectives in section 1.3, whilst section 1.4 discusses the structure of the thesis. Section 1.5 concludes the chapter.

1.1 Overview and Importance of XML labelling schemes

The challenge of indexing techniques lies in the query processing performance [7-11, 19]. A labelling scheme helps to capture the structural relationships during query processing without the need to access the physical document; this helps to reduce the query processing time [20, 21]. In addition, the challenges associated with labelling-based methods for dynamic XML data involve the support for data updates without duplicating labels or relabelling old nodes [22]. Figure 1.2 illustrated an example of relabelling nodes (denoted by the dashed circle) when a new node is inserted (denoted by the black circle) [13].

An efficient labelling scheme should have the following significant properties: labelling should be dynamic, which means avoiding the need to relabel XML tree nodes when the XML files are updated. Furthermore, the XML label size should be compact, which means optimizing the performance of the label size and producing more compact labels that lead to decreased storage costs in both the initial labelling and after the skewed node insertions [23-26]. Labelling should support all kinds of structural relationship queries, as it is important for query processing in XML database management systems [6, 27, 28]. A labelling scheme generally is limited in one or more of the essential properties and this is our problem identification.

The challenges include the reduction of time and size taken to generate the labels [29]. Large label sizes can lead to negative impacts on both update and query performances [30]. For example, the authors [13] used a fraction fragment to represent the last component of the label where the two following siblings are labelled as A and B.

$A = a_1. a_2 \ldots a_{m-1} . (a_m/k_a)$ and $B = b_1. b_2 \ldots b_{m-1} . (b_m/k_b)$.

If A and B are inserted labels, then the newly inserted label is

$a_1 .a_2 \ldots a_{m\_1} . ((a_m + b_m)/(k_a + k_b))$.

In the example, the scheme generates floating-point numbers, which can lead to limited accuracy [31]. Moreover, the mantissa is denoted as a fixed number of bits. It can then be extended by two bits for each insertion, which can cause overflow problems [26, 32]. Also, The querying in this technique is slow as decoding process is based on ORDPATH which is time consuming [33].

Previous research relied on the assumption that using XML parsers without node labels was sufficient to read and explore XML datasets [34, 35]. Furthermore, most existing research was based on the retrieval and navigation of data [36, 37]. However, the need for labelling schemes has become essential to efficiently support XML queries and update nodes [7, 13, 38, 39]. Moreover, it is useful to adopt dynamic XML labelling schemes to avoid relabelling existing XML nodes when conducting updates [13, 27, 28].

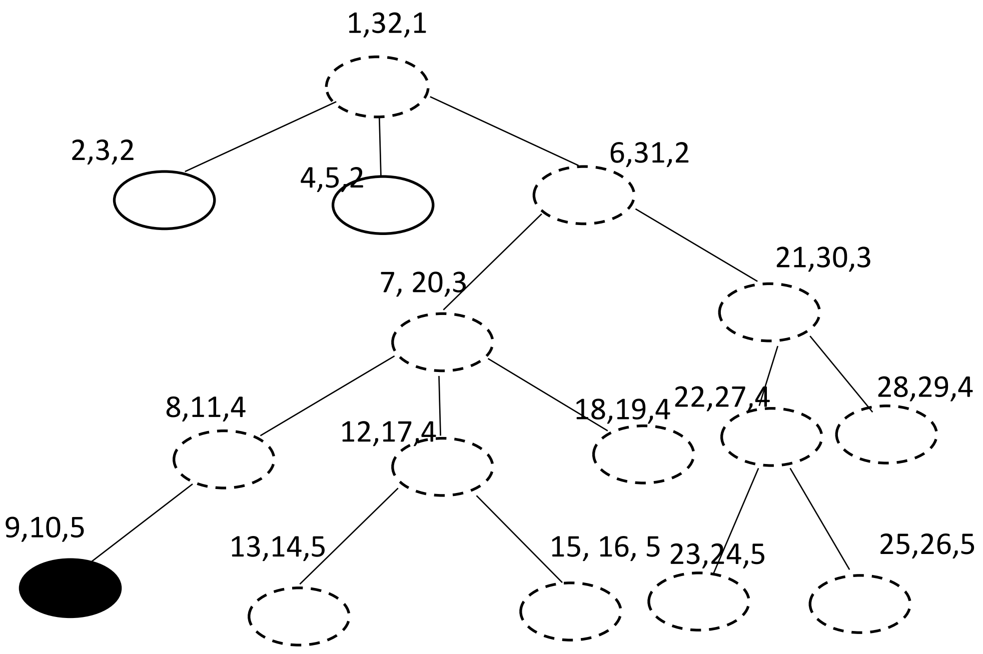


Figure 1.2 An example of relabelling nodes.

As shown in Figure 1.2, the relabelled nodes are denoted by the dashed circles and when a new node is inserted, it is denoted by a black circle. The relabelled nodes indicated that more than one node is required for the relabelling process as shown in Figure 1.2 [40, 41]. We need to relabel all the dashed circles nodes based on the interval labelling scheme that stores a combination of values to each node. Each label is represented as pre-order, post-order and depth. The pre-order starting from the root and handing each level from left to right, while the post-order visiting the leaf nodes from left to right, and then processing their parent level from bottom to top. In addition, the depth determines the level of the nodes [14].

In updating new XML nodes, two forms of insertion are mainly used. The first form is the random skewed insertion, which means frequently inserting between two random nodes selected. The second form is order skewed insertion which is a frequent insertion before or after a specific node [32, 42]. Some labelling scheme were tested over skewed insertions [13, 23, 33]. Labelling schemes consider four cases of insertion: inserting before the leftmost sibling, inserting after the rightmost sibling, inserting between two siblings, and inserting a child into a leaf node [21, 22, 26, 32, 35, 43-45].

1.2 Research motivation and hypothesis

Dynamic labelling schemes have been developed to support efficient XML updates; however, each of the existing schemes is limited in one or more aspects.

Firstly, an example of a noticeable limitation of an existing scheme is the work in [34] which is a Prefix labelling scheme and has proven to be unsuitable for dynamic XML documents as updating a new node using this scheme requires the relabelling of all its existing right sibling nodes along with their relatives in the entire XML tree [44]. This is time-consuming and inefficient for dynamic XML data. Another example of this limitation is found in the Region based labelling scheme [46] as this also supports static XML documents [13, 47], meaning that these two schemes are limited in terms of only being appropriate for non-updatable XML documents.

Dynamic prefix-based labelling schemes also have limitations. For example, the scheme that is used in [44] only allows limited updates [16, 22], as just the even and negative integer values are reserved for updating the XML tree. Also, The decoding technique is time- consuming [33].

Moreover, the extended prefix Dewey [20] is approximate 10%–30% larger in size compared to the original Dewey. This is due to the large size generated by applying the extended Dewey technique, which produces a large label size at the cost of extra storage. This is considered as the limitation of this scheme [13, 21, 48].

Furthermore, In regards to the Dynamic Dewey scheme (DDE) [22] which is an update of the Dewey scheme. The main weakness is that the labelling scheme results in the production of a large label size [21, 49]. This is due to the scheme storing the level information as part of components in that label. Also, frequent insertions occur between two siblings by applying the midpoint technique, which results in increased storage costs as the depth increases [21, 49]. Moreover, The DFPD scheme [21] generates floating-point numbers, which can lead to limited accuracy [31]. In addition, The querying in this technique is slow due to the reason that its decoding process is based on ORDPATH which is time-consuming [33]. This scheme also causes overflow problems [26, 32].

We have compared the Pentagonal scheme with DDE and DFPD as their internal model is based on dynamic XML and are both based on the prefix labelling approach as well as that they completely avoid relabelling in XML updates. In addition, they support the loading of different XML document Sizes and support XPath query language. However, in spite of their advantages, they suffer from multiple problems and so the Pentagonal Scheme was proposed to address these limitations and achieve better results in comparison to them. Specifically, our aim was to obtain a small label size, support dynamic updates without relabeling nodes, support frequent insertions without overflow problems, generate improved labelling time performance and evaluate query performances.

The main contribution of our labelling scheme is that it is efficiently supports updates in all the cases of insertion, it performs best when a vast number of random skewed nodes has been updated. Also, it proved its capability in terms of the query performance and in determining the relationships. Our scheme also supports frequent insertions without overflow problems.

In terms of originality, the Pentagonal scheme has been applied for the first time to label XML data. The storage mechanism in our scheme is based on the Pentagonal numbering and prefix labelling scheme. Our labelling scheme considers the restrictions of the existing XML labelling schemes to improve the efficiency of XML data management systems. It focuses on the size of XML labels and the time taken for the labelling process. It also evaluates the query performance and the labelling scheme's ability to handle different types of update. In terms of the implementation and the design of the proposed scheme, we applied the SAX parser due to its improved performance in relation to handling large XML documents.

We tackled these limitations in our approach by providing a fully dynamic labelling scheme that supports frequent insertions by assigning integer pentagonal numbers to each node in order to obtain a small label size. In addition, each label has a variable-length label to further avoid overflow problems.

This thesis aims to dkesign a novel labelling scheme that supports dynamic XML documents,and is based on Pentagonal numbers in prefix labelling scheme to represent the new label. The scheme should support updates in XML tree without duplicating labels or needing to relabel old nodes.

Our motivation for using the pentagonal scheme was to optimise the performance of the labelling time and to produce more compact labels that lead to decreased storage costs in both the initial labelling and after the insertions. By using the Pentagonal Scheme method, we can avoid using float-point numbers, this allows for fast labelling time and avoids overflow problems. Based on the research motivation, the research hypothesis is specified as follows:

“Applying Pentagonal numbers for dynamic XML documents, based on prefix labelling approach to generate the new labels may improve the labelling time performance, providing labels without extreme growth or any overflow problems in the label size, and supporting insertions in dynamic XML databases as well as facilitating the query performance.”

1.3 The research aims, objectives and the research questions.

Section 1.2 highlighted the research hypothesis and the limitations of current labelling schemes from which the aims and research objectives were developed. We aim to design a novel scheme to support updates in a dynamic XML tree without duplicating labels or needing to relabel old nodes. Our scheme aims to generate labels based on the prefix labelling scheme. It also evaluates the labelling process in terms of size and time and the ability to handle different types of update. Furthermore, the scheme aims to support frequent updates with fast calculations and uncomplicated implementation. It will also effectively extract from the labels' structural information to accomplish a high-performance query.

The research questions are as follows:

RQ1: How to design a novel labelling scheme that in comparison to the state-of-the-art will support compact labelling size, low execution time, avoid relabeling while inserting new nodes, and offer efficient XML query processing?

RQ2: How to achieve low execution time and compact labels with this novel labelling scheme even after frequent, large and random skewed insertions in different positions such as inserting between two siblings and into a leaf node?

1.4 Structure of the Thesis

This section highlights the structure of the thesis, which is divided into three parts. The first part consists of three chapters which introduce the related background and literature reviews that influenced the creation of the hypothesis. The second part conducts a comparison between range-based and prefix encoding, with a focus on reductions to labelling time and memory size; this is discussed in detail in chapter four. Based on both theoretical and practical points of view, the main concept of this research is discussed in detail in chapter five, which also covers the experimental results. The third part of the thesis consists of chapters six to eight which cover the queries' experimental results, a comparison between native database systems and labelling schemes, the thesis conclusion and future work. The following section describes the thesis chapters:

- Chapter 1 – Introduction.

This chapter has introduced the research work that influenced the creation of the hypothesis in general, the research motivation and hypothesis, and the research aims and objectives. It also outlined the structure of the thesis.

- Chapter 2 - XML Data Background.

This chapter provides an overview of XML and its XML tree structure. It also provides a description of its syntax and illustrates the parsing techniques.

- Chapter 3 - Literature Review on XML Labelling Scheme.

This chapter presents an overview of labelling schemes and discusses the structure, strengths, weaknesses and restrictions of several existing XML labelling schemes.

- Chapter 4 - Comparison between Range-based and Prefix-Dewey Encoding.

This chapter compares two XML labelling schemes, namely range-based encoding and prefix encoding. The study aims to achieve the fastest labelling time and to ensure the generation of short labels in terms of memory size

- Chapter 5 - Pentagonal Labelling Scheme for Dynamic XML Data.

This chapter explains the underpinning theory of the proposed scheme by offering a definition that illustrates the rules of the algorithms and describes the structure of the scheme. Also, the chapter describes the practical design and implementation of the Pentagonal scheme, which is based on the definition and the algorithm rules. In addition, to evaluate the Pentagonal labelling scheme, several experiments are performed on different datasets. This chapter illustrates the experimental results in order to evaluate the proposed scheme's reliability, scalability and performance of the proposed scheme, while graphical diagrams are presented to evaluate it.

- Chapter 6 - Query Experiments

This chapter illustrates the experimental results in order to evaluate the query performance of the proposed scheme. The experiment compared the ability of the Pentagonal dynamic labelling scheme to handle query response times and the time spent determining different relationships.

- Chapter 7 - A Comparison Between Native Database Systems and Pentagonal Labelling Schemes.

This chapter explains the concept of Native XML database systems and compares the proposed scheme with two Native XML databases systems. Also, the experiment compared the ability of the Pentagonal dynamic labelling scheme, eXist database and BaseX database to handle different dataset sizes and the execution of different queries.

- Chapter 8 - Conclusion and future work.

This chapter summarises the whole thesis, the main findings and key contributions. Moreover, the recommendations for future work are emphasised.

1.5 Conclusion

This chapter presented a brief introduction to the thesis and explained the importance of XML labelling schemes. The research motivation, hypothesis, research aims and objectives were introduced. Lastly, the structure of the thesis was underlined.

2 XML DATA BACKGROUND

This chapter provides an informative discussion based on XML data background covering all aspects of XML and its XML tree structure. Also provides a description of its syntax and illustrates the concepts of XML parsing techniques.

Extensible Mark-up Language (XML) is developing as a de facto standard for data exchange among several applications on the World Wide Web due to its self-describing and the ability to organise data [16, 50-52].

XML files are demonstrated as a tree, and labelling schemes encode the structural tree information to answer queries without having to access the original XML file [38, 53-57].

This chapter presents a brief overview of XML, starting with XML Overview then goes on to describe the Storage and the Structure of XML in Section 2.2 and Section 2.3 respectively. Next, in Section 2.4 XML parsing are explained. The chapter will be concluded in Section 2.5.

2.1 XML Overview

XML facilitates the ability to define the content of an XML document separately from its format; this makes it simple to both share and reuse data in different applications [1]. XML is beneficial for several reasons. First, it allows users to propose their own tags as a self-describing language, which construct it extremely flexible[58]. Furthermore, the XML language is uncomplicated and Text-based user interfaces, with a transportable data format that read by most of the platforms [59, 60].

HTML (HyperText Mark-up Language) produce a standard to display, create and access web pages. However, HTML does not provide tag information to describe the content so systems cannot recognise the structure of the data [61, 62]. XML was developed in 1996 to address the limitation in HTML, which was sponsored by the World Wide Web Consortium, W3C [51].

XML is especially relevant in the context of Big Data, as XML is a highly flexible structure. It describes the structure of the text meaning the user can design their own tags and separate the content from its format [58]. Also, XML has the ability to cover different types of data. XML data can implant any possible type of data as either complex information, e.g. living organisms and biological systems, or as multimedia data, e.g. image, video, and sound [2].

2.2 XML Storage

XML has used for retrieval data over the Web in heterogeneous and homogeneous platforms, exchange data, transformation data and information representation and represent semi-structured data [10, 16, 32, 39, 63]. The main approaches to store XML data are XML Enabled Database (XED) and a Native XML Database (NXD). XED is used to store data-centric documents that contain well-structured information. Therefore, the data can be transfer into a traditional relational database [64, 65]. NXD is used to store document-centric XML that contain semi-structured XML document and stored in the hierarchical structure [66]. In addition, hybrid storage has been proposed; this technique simply mapped some parts of the structured XML into relational data and other parts can be kept in XML data type itself as NXD format [16, 67].

This thesis will emphasis facilitating native XML database to process XML queries and focusing on evaluation XML query that relies on document-centric XML. To start with clear comprehension, a description of the concept and the structure of XML is in the next section.

2.3 The concept and the structure of XML

The basic concept of an XML document can be defined as an element. Elements can contain other elements and can be nested at any depth. Each element of the document surrounded by two tags. Start tag will be at the beginning as <*tag-name*>, and end tag will be at the end as </*tag-name*>. Also, the form <*tag-name*/> can be tag of an Empty element [68-70].

Figure 2.1 shown an example of XML document holding data on a store. This document provides information for customers such as first-name, last-name, full-address, mobile and the email of the customers of the technology store. For each customer, the document also records information on his/her salesman. The contact element is an example of an element with sub-elements in that it contains mobile, email element. The full-address is an example of an element containing text, whereas email is an example of an empty element. The attributes can be specified for the elements. The attribute is in the form of *name* = *value*, wherever the *name* is a label and the *value* will be a quoted string. The *attributes* location in the document is in the start tag of the element, it has different types to specify an element identifier, IDRE type is containing to a single target, and IDREFS type is containing to multiple targets. In addition, CDATA type is referred to as textual information [68].

```
<store id="Technology">
    <customer id="C201">
       : : :
    </customer>
    <customer id="C223"  salesman="S201">
      <name>
           <firstname> Muhammad </firstname>
           <lastname> Ibrahim</lastname>
      </name>
      <full-address> 96, Hindley Street, Bolton </full-address>
      <contact>
          <mobile> 984 589 482 </mobile>
          <mobile> 785 942 468</mobile>
          <email reference="MuIbrahim123@yahoo.com"/>
      </contact>
      <preference>
          <contact-way> email </contact-way>
         <contact-time> morning </contact-time>
         <branch> liverpool </branch >
         <pay-method> cash </pay-method>
      </preference>
    </customer>
    <customer id="C250" salesman="S223">
       : : :
    </customer>
</store>
```

Figure 2.1. An example of XML document.

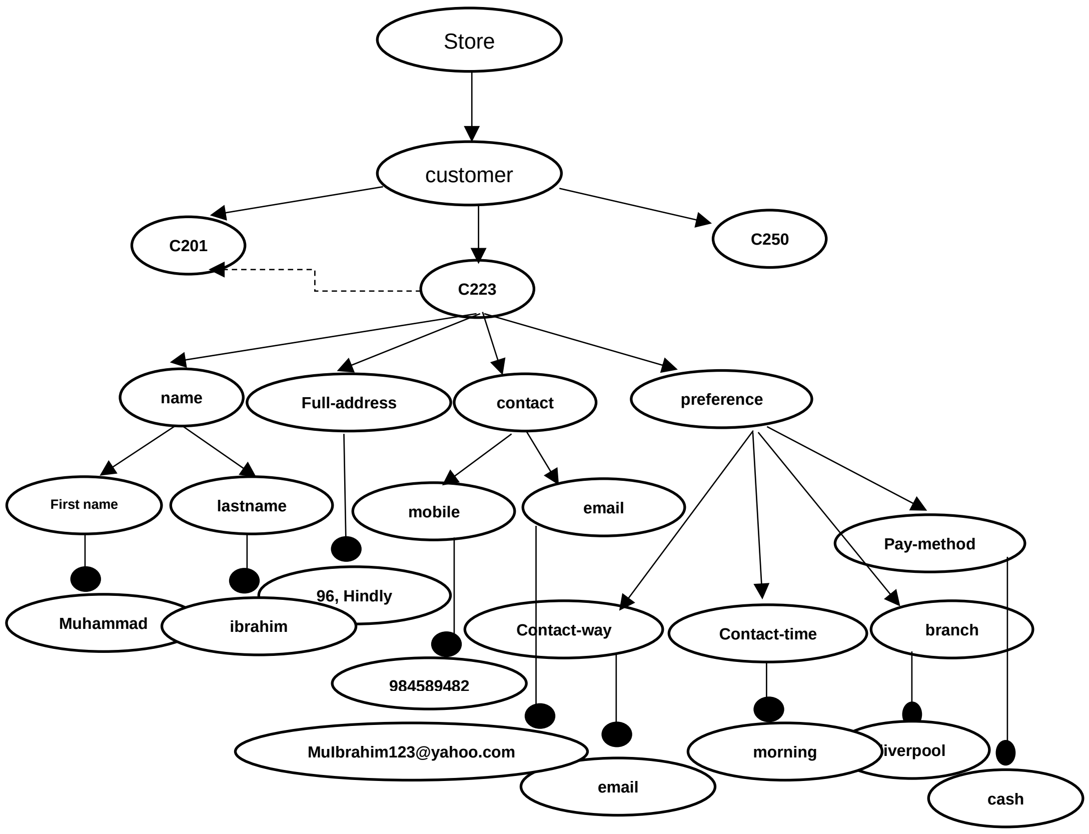


Figure 2.2. Graph representation of the document of Figure 2.1.

In Figure 2.3, the relationship existing between a customer and the salesman is demonstrated by an IDREF attribute named salesman in the customer element, where the value is the *id* of the customer salesman [68]. The graph in Figure 2.2 represented XML document that reported in Figure 2.1. The graph is representing the element-attribute and the element- subelement relationships, and the edges representing relations between elements. Edges are signified by lines. A document type declaration is known as DTD. The DTD can be attached to XML documents, identify the rules that XML documents need to follow [68].

```
<!DOCTYPE store[
<!ELEMENT store (customer)>
<!ELEMENT customer(name,full-address,contact,preference>
<!ELEMENT name (firstname,lastname)>
<!ELEMENT contact (mobile,email)>
<!ELEMENT preference(contact-way,contact-time,
                                        branch?,paymethod*)>
<!ELEMENT full-address(#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT mobile (#PCDATA)>
<!ELEMENT email EMPTY>
<!ELEMENT contact-way(#PCDATA)>
<!ELEMENT contact-time (#PCDATA)>
<!ELEMENT branch (#PCDATA)>
<!ELEMENT paymethod (#PCDATA)>
<!ATTLIST store id ID #REQUIRED>
<!ATTLIST customer id ID #REQUIRED salesman  IDREF
#IMPLIED>
<!ATTLIST email reference CDATA #IMPLIED> ]>
```

Figure 2.3. An example of document type declaration (DTD).

For example, Figure 2.3 illustrations the DTD for the document in Figure 2.1. A DTD is collected of two parts: the element statements part and the attribute list statements part. The element statements part defines the structure of all elements included in the document. Each element specifies its subelements and their order. In addition, for each subelement it specifies either they are optional ("?") or Not. Also, they might occur zero, one or more times (“+” or “*”), Also if the subelements are alternative or not to another subelement (“|”). Moreover, the type #PCDATA which allowed only data content; ANY is allowed all kind of content, EMPTY if no content is allowed. The attribute list statements part requires, for each element, the list of its attributes, in terms of names, types, optionality parts #IMPLIED is signified an optional attribute, #REQUIRED to signify a mandatory attribute and #possibly is denote to default values [68].

The current XML source has been classified into two main types: that is, *valid* and *well-formed* documents. A well-formed document is defined as a document that is written under the grammar rules of XML [World Wide Web Consortium 1998a] [68]. A valid document is known as a document which follows a given DTD. As a result, valid documents can be understood as illustrations of a matching DTD. For instance, Figure 2.2 is an example of a valid document, as it follows the DTD in Figure 2.3 [68].

The XML schema language, unlike DTDs, affords different data typing correlated with type in the programming languages. The XML schema description defines numerous different types of data, such as integer, string, date, time, and, boolean. Despite the built-in data types, XML schema also offers the ability to introduce new types. Developers not just using the element as plain text in an XML document, but they could define their data types. Therefore, they can effectively use and dealing with elements and attributes in an XML document [71].

```
<?xml version=“1.0” encoding=”UTF-8”?>
<xsd:schema
xmlns:xsd=“http://www.w3.org/2016/10/XMLSchema”>
<xsd:complexType name=“nameType”>
  <xsd:sequence>
     <xsd:element name=“firstname” type=“nameType”/>
     <xsd:element name=“lastname” type=“nameType”/>
  </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name=“full-addressType”>
  <xsd:restriction base=“xsd:string”>
      <xsd:maxLength value=“100”/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name=“contactType”>
 <xsd:sequence>
    <xsd:element name=“mobile” type=“Type”/>
         <xsd:restriction base=“xsd:string”>
             <xsd:pattern value=“(d{3})-d{3}-d{4}”/>
         </xsd:restriction>
    <xsd:element name=“email” type=“emailType”/>
 </xsd:sequence>
</xsd:complexType>
<xsd:element name=“customer” type=“customerType”/>
</xsd:schema>
```

Figure 2.4. Sample schema for the XML document shown in Figure 2.1.

Figure 2.4 illustrations a sample schema, and Figure 2.3 illustrations a sample DTD for the XML document in Figure 2.1. In comparing Figures 2.2 with Figures 2.3, the DTD defined the value of mobile as character data. However, in the schema, the data type can be defined as mobileType, to represent mobile and make it under restriction to be a valid number that represents a standard US mobile number [71].

2.4 XML Parsers

A parser is an interface between the application program and the XML document. XML parsers can identify if the XML document is valid and well-formed through interpreting its content by the Application Programming Interfaces (APIs) [71]. This process takes place with the use of the parser, which with the access to the documents' internal structure and contents can read the XML documents and then provide the application programs [71].

There are two types of XML parser: Document Object Model (DOM) and Simple API for XML (SAX). These are discussed in the following sections.

2.4.1 Document Object Model (DOM)

The DOM parser, which was issued as a W3C recommendation in 1998 is ultimately a tree-structure-based API. The DOM parser is a language-neutral interface. The DOM platform allows programs to access and update the structure and the content of XML documents. The DOM parser demonstrates the nodes of the XML document as a tree containing elements, attribute and text [72-75]. An XML parser creates the DOM tree of XML document and then send it to an application program, it provides a set of APIs to manipulate these nodes in the tree. However, Using the DOM-based XML processor requires the entire structure of an XML document to be built within main memory [71, 76].

2.4.2 Simple API for XML (SAX)

The SAX parser does not generate a data structure. An XML processor with SAX scans an input XML document and then creates events, for example, an element start or element end [77, 78]. The application programs implement the handlers which receive these events in order to process them correctly. The SAX parser is most suited for dealing with large documents which not fit in the main memory [72, 77-79]. Moreover, the SAX parser is best suited for extracting the contents of specific elements [71].

2.5 Conclusion

An overview of the fundamental aspects of XML data has been presented in this chapter. The aspects provided are sufficient to cover the essential background to this thesis. The focus of this thesis is XML labelling technology; the next chapter outline the literature review of XML labelling schemes.

3 Literature Review of labelling schemes

This chapter provides an informative discussion based on extensive literature reviews covering the aspects of XML with regards to labelling schemes. It has been structured to provide a thorough scientific understanding of the study of labelling schemes.

XML labelling scheme has been recommended for speedy query processing of massive XML documents [22, 80-82]. Nonetheless, even with the wide-ranging of labelling approaches, extensive problems have been faced in developing a suitable labelling scheme for effective management of XML data, since the Dewey Order labelling [34] and ReLab [46] does not support dynamic XML data and only supports static

XML documents. Also, the NLSXU scheme [83] does not support any node insertion and does not show any query performance testing. Moreover, DDE and CDDE [22] both support dynamic updates, but they produce a large label size [21, 49]. Furthermore, another problem faces is that in SCOOTER [23] the large labels' quaternary strings slows down the query processing [27, 84].

In section 3.1 we cover an overview of labelling schemes, in Section 3.1.1 we discussed different labelling schemes, the Prefix labelling scheme is detailed in section 3.1.2. The limitation of existing labelling schemes presented in Section 3.2. Lastly, Section 3.3 concludes the chapter.

3.1 Overview of labelling schemes

Labelling, or numbering, the scheme is generally used to label an XML document in order to perform an XML query using path node information [85]. This captures the structural relationships during the query processing with no need to access the physical document [20, 86]. Labelling schemes reduce the query processing time and hence make the retrieval and indexing of XML data more efficient [34, 86]. In simple terms, labels in such schemes present relationships between nodes in XML trees [12, 29, 46, 87, 88] and are used for retrieval purposes. They achieve this by relying on XML labelling schemes, keyword searches and XML data queries [20, 89].

3.1.1 labelling schemes

The current labelling schemes have been classified into four main types: Interval based schemes (also known as Range based labelling schemes; Region encoded labelling schemes; Subtree based labelling schemes or Containment labelling schemes) [7, 14, 32, 39, 46, 85, 90, 91]); Prefix based schemes [7, 9, 13, 16, 21, 22, 34, 92], Multiplicative based schemes [39, 43, 49, 93-99] and Hybrid based schemes [7, 16, 47, 48, 100].

XML Documents can be classified into two types: static [14, 34, 85], which is sufficient for non-updatable XML documents, and dynamic, which is regularly updated [22, 26, 43, 46, 101, 102]. A dynamic labelling scheme has been proposed based on the mathematical principles of vector order [95, 103]. A vector order has been proposed to avoid relabelling, which is applied to both interval-based and prefix-based labelling schemes. In their algorithm, the midpoint has been calculated, which is applied to the interval-based labelling scheme (Region Labelling

Scheme). A vector, V, is an object with weight and a path that can be represented as a binary tuple, $V= (x)$, where x and y are positive integers. The Vector-based labelling method is adapted from vector-encoding [28, 45]; it represents interval-based labels in a vector form. The nodes are labelled as <*start vector, end vector, and level value*>, whereas the vectors' gradient values are used to preserve the order of the assigned vectors. By inserting node C between nodes A and B, a vector value is allocated to C according to vectors A and B. $A=(x_a,y_a)$, $B=(x_b,y_b)$, and $C=A+B=(x_a+x_b,ya+y_b)$, then $G(A)>G(C)>G(B)$, where $G(A)>G(B)$ if, and only if, $(y_{a*b})>(x_a*y_b)$. However, their work did not perform any experiment to test the performance of their scheme or any query testing.

An Improved Binary String Labelling (IBSL) scheme [55] been proposed. It is a binary, string-based encoding approach, and the IBSL label is a sequence of numbers 0 and 1. Their scheme avoids relabelling when updating XML documents and reuses the deleted label at the same position. However, it increases the cost of storage in the case of frequently skewed insertions [13, 47]. Moreover, the IBSL scheme tests the leaf node insertions only.

A Prime-based Middle Fraction Labelling Scheme PMFLS [48] has been designed in which a series of algorithms are proposed to obtain the structural relationships among nodes and to support updates. PMFLS is a hybrid labelling scheme; it combines the advantages of both prefix and region schemes. PMFLS also supports updates without recalculation. However, prefix labels naturally extend when XML data are updated during frequent insertions, causing overflow problems.

In [46], ReLab is Region-based Labelling scheme; their experimental evaluation denotes the schemes in terms of the time taken to generate labels for each XML node. This is not only used for the unique identification of XML nodes, but also structural relationship purposes.

In [39], ME labelling provided a roust hybrid scheme for dynamic updates in XML databases. They proposed an XML labelling scheme that helps a quick determination of the structural relationships among XML nodes and supports dynamic updates without relabelling nodes in the case of update occurrences. Due to the simplicity of the ReLab [46] scheme, it has generated labels faster than other Region-based schemes [47]. However, ReLab [46] does not support dynamic XML data but only static XML documents [13, 47].

The NLSXU scheme [83] has generated labels using digits (0-9), uppercase and lowercase letters and a few characters in the Unicode character set. It provided a greater varied range of characters. The Unicode value of the characters is considered to preserve the order of the siblings in the XML document. NLSXU reduced the space for synthetic data and reduced the index size compared to the NLSX scheme [83]. In addition, the NLSXU scheme reduced the time taken to generate labels compared with LSDX and NLSX. However, the NLSXU scheme did not support any node insertion and did not show any query performance testing.

The Clustering labelling scheme [104] is a hybrid approach that has been proposed based on the interval and prefix labelling schemes. This scheme is based on the clustering technique and the levels of the nodes in XML trees. The approach is to divide the whole data tree into small groups where two labels are used for every node and the cluster (group) is linked to the entire tree using the label of that cluster.

The RLP-Scheme [105], is a hybrid approach of multiplicative and prefix labelling schemes. This is similar to the Dewey scheme [20]. However, each node label in the RLP-Scheme contains more information compared to the node labels in the Dewey Scheme. The RLP scheme is divided into several groups, each node is allocated an *ID* of the formula [*G,P,S*] where *G* represents the group number of the node, *P* represents the self-label of all its ancestors, and *S* denotes its self-label.

The Branch map labelling scheme [106] records the correspondence between a parent and child nodes, unlike other schemes discussed in this thesis, where the schemes assign a label to each node. This scheme is suitable for structural summary indexing. It uses the SAX parser to parse XML documents. Moreover, a hash key is used to represent the path index, where the information regarding the path and structure with the tag name is stored. Each node is assigned (*label*, *branch*, *count*, *children*), where the *label* represents the location of the node following the Dewey label scheme. The *Branch* represents the branch map and each "1" signifies a node in the XML tree. *Count* denotes the number of appearances of the tag, and *children* represents the tag names of the children of the node.

In [107], the labelling scheme holds the containment information and represents it as a 4-tuple: (Did, start, end, level). Did represents the XML encoding of the XML document. Start represents the occurrence position and is created by the pre-order traversal. end records the beginning maximum number of the current node in the sub-tree. The Level is used to determine the structural relationship between the nodes.

[108] They applied a structure named partial tree structure. This was appropriate for large XML documents where multiple computers can be used for processing. They have used the BaseX database for their comparison of loading and execution times. In [108] two index sets were used to execute XPath queries in large XML documents, in order to achieve a faster evaluation time of the structural relationships between nodes.

3.1.2 Prefix labelling scheme

The prefix-based labelling scheme is considered a suitable approach for dynamic XML data [92]. Several prefix labelling schemes have been proposed. In prefix labelling schemes each node label contains a unique label, which is the parent's label and concatenated with the node self-label. Notable research in developing a prefix labelling scheme to improve the storage, retrieval and query into XML data is the Dewey Order labelling scheme. This is based on the Dewey decimal classification system for libraries [34]. Even though the Dewey Order is popular; In [44] authors argue that the model of [34] is unsuitable for dynamic XML documents since updating a new node requires the relabelling of all its right sibling nodes with their relatives in the whole XML tree. The labels studied the position of the node and needed to sort the order of the document during an update.

In [34], the authors proposed a prefix encoding using Dewey coding to label XML trees. In this method, a vector presents each node. The root in an XML tree is labelled by an empty string ε and the non-root element u is labelled as a combination of its parent label and a postfix integer number (x_i). If u is the x^{th} child of s in an XML tree then the label u, label(u), is q, a concatenation of label s and x, which is presented as a label(s).x., where s is the parent of u. For example, if the label for node u is 2.5.3 then its 4th child label will be 2.5.3.4. The advantage of this, for any element labels, we can easily extract the node labels of its ancestors. For example, if an element label is 5.1.3.1, then its parent label is 5.1.3, and its first ancestor label is 5.1.

Due to the simplicity of the Dewey Order labelling scheme [34], it has become common amongst indexing schemes [57, 109, 110]. However, this mechanism is not appropriate for dynamic XML data. For example, to insert a new sibling node into an XML tree, the Dewey Order labelling scheme requires the relabelling of all its right sibling nodes along with their descendants.

In [20], the Extended Dewey code has been proposed to address this limitation of Dewey Order labelling scheme. Each element is a grouping of its parent label and a postfix integer number (x_i). For any element e_i with name t_i, the extended Dewey assigns an integer number, x_i, to e_i such that x_i mod n_i =i. Extended Dewey Encoding needs some scheme information for labelling. Moreover, the element tag names are added as part of their Dewey labels. Scheme information can be extracted from DTD. Otherwise, before assigning XML tree nodes labels, the whole XML document must be scanned at least once to know the document's scheme information [111]. They suppose that the element name of u is k^{th} tag in $CT(ts)$ (k=0,1,...,n-1). $CT(ts)$ = tn-1, to express all child nodes of t from the DTD structure information of an XML document. Where $CT(ts)$ is the child names of tag t, and ts denotes the tag of element s. Here, label(u) = label (s).x is used to express the code of node u and s is the parent node of u. If u is a text value, then x = -1; Otherwise, we assume that the element name of u is the k^{th} tag in $CT(ts)$ (k=0,1,...,n-1), where t_s denotes the tag of element s. If u is the first, then $x = k$, otherwise, if we assume that y is the last component of the left sibling label u, then X ={$\lfloor yn \rfloor.n+k$ $if(y \ mod \ n)<k; \lceil yn \rceil.n+k \ otherwise.$

The extended Dewey labelling scheme also does not support dynamic updates in XML trees; this requires the reconstruction of the child name clue data after insertion [13, 111]. Furthermore, adding XML tree element names within their labels increases the label size and makes the computation process methods very expensive [16, 112]. However, it performs well in evaluating query processing by accessing only the leaf nodes that contain the labels. This speeds up the process and satisfies queries [7, 13, 23, 38, 39].

Also, [22] have proposed a Dynamic Dewey scheme (DDE), which is an update of the Dewey scheme. This transforms it into a fully dynamic labelling scheme based on the mathematical operations of a Dewey label. In DDE, the label has different lengths, starting with a byte for the first level and increasing in depth relative to the level value. This can be appropriate for avoiding overflow problems. In addition, it has the ability to avoid a complete relabelling and supports a high query performance. Figure 3.1 shows the initial label for DDE. The advantage of the DDE compared to previous works is that the scheme has shown an improved performance when new XML nodes are inserted. The main weakness is that the labelling schemes are making a large label size [21, 49]. It implicitly stores the level information as the number of components in that label, and frequent insertions occur between two siblings by applying the midpoint technique, which needs extra storage costs when the depth increases [21, 49]. This property remains true after random insertions. Given two labels X: $x_1.x_2. \ldots.x_n$ and Y: $y_1.y_2. \ldots\ldots y_m$, the following properties can be extracted from the labels: X is the parent of Y only if X is an ancestor of Y and $n=m$-1. X is an ancestor of Y only if $n<m$ and (X_1/Y_1) = (X_2/Y_2) = ……… = (X_n/Y_m). X is a sibling of Y only if $n=m$. Figure 3.2 shows how a DDE labelling scheme manages multiple insertions within an XML document. As X is the first child of a node when a new node is inserted before node X: where the label of the new node will be $x_1.x_2. \ldots.x_{(n-1)}$. As X is the last child of a node when a new node is inserted after the node X: $x_1.x_2. \ldots\ldots x_n$ the label of the new node will be X: $x_1.x_2. \ldots\ldots x_{(n+1)}$. Below a leaf node X: $x_1.x_2. \ldots\ldots x_n$ the label of the new node will be X: $x_1.x_2. \ldots\ldots x_{n.1}$ when a new node is inserted. However, between two continuous siblings, X and Y, a new node is inserted, and the label of the new node becomes $A+B$.

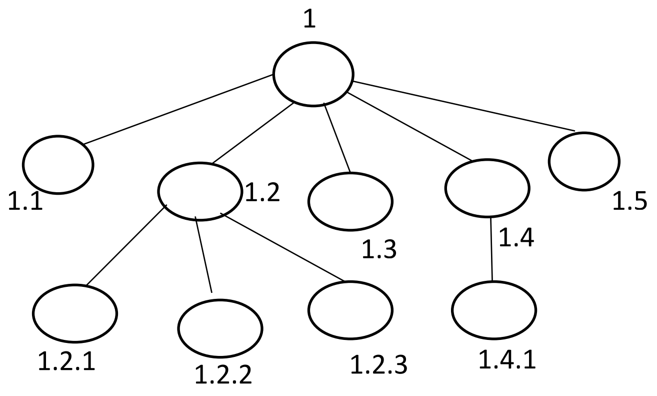


Figure 3.1. Dynamic Dewey scheme (DDE) initial labels

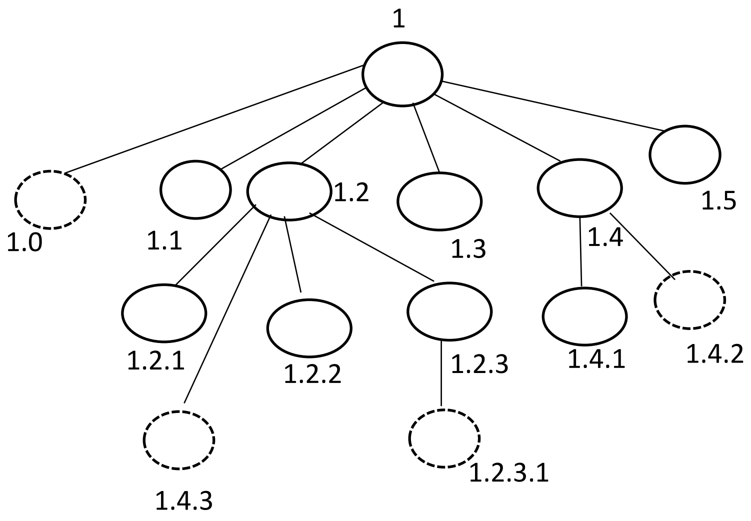


Figure 3.2. Processing insertions with DDE labels.

Dynamic Dewey Labelling considers four cases of insertion, as illustrated in Figure 3.2 Firstly, inserting before the leftmost sibling, the new label is created by reducing the local order value of the leftmost sibling by 1; in this case, negative values are acceptable. Secondly, inserting after the rightmost sibling, the new label is created by incrementing the local order value of the rightmost sibling by 1. Thirdly, inserting between two siblings (giving X and Y), the new label, e.g., node *v*, is assigned as the midpoint vector, $X+Y$, which is equal to $x_1+y_1.x_2+y_2.....x_m+y_m$. Finally, inserting a child into a leaf node where the new label is created by concatenating the parent label and the digit "1". CDDE is an improved version of DDE [22]; it has been presented to recover the performance of DDE when updating

XML documents by allowing initial labels to be negative values. The improvement in CDDE is insignificant in terms of updating time and label sizes, as clarified in their work [22].

DDE and CDDE support dynamic updates, although they produce a large label size at the cost of extra storage [21, 49]. This mainly occurs when frequent insertions occur between two siblings due to the large size generated by applying the midpoint technique. DDE is not appropriate for defining the structural relationships in multiple XML documents and requires an additional document to differentiate the labels in several XML documents [13, 21, 113].

Many dynamic schemes have been proposed based on the Dewey, Dynamic Float-Point Dewey [21]. The authors proposed a DFPD labelling scheme, and the initial labels were based on Dewey labels and handle updates to XML documents by considering the same three cases in DDE techniques: Inserting before the leftmost sibling, inserting after the rightmost sibling, and inserting a child into a leaf node. However, when inserting between two following siblings, the new label is calculated, and the result is a float-point number. The decimal part is 0; in this situation, the last component of the new label is a float-point number.

Assume that the two following siblings are labelled as ($a_1. a_2 \ldots a_{m-1} .a_m$ *and* $b_1. b_2 \ldots b_{m-1} .b_m$) individually, then the new node can be calculated using the Equation: $a_1 .a_2 \ldots a_{m\_1}.((k_a \times a_m + k_b \times b_m)/(k_a + k_b))$. Hypothetically, the values of k_a and k_b can be considered a set of positive integer numbers. Recently, the authors have introduced the DPLS labelling scheme [13] to improve the DFPD performance [21]. DPLS is a dynamic prefix-based labelling scheme and supports updates in XML trees. The newly inserted label between two siblings, A and B, can be calculated using this equation: $a_1 . a_2 \ldots a_{m-1} .((a_m + b_m)/(k_a + k_b))$. The authors used a fraction fragment to represent the last component of the label, and both numerator and denominator are integers. Both DFPD and DPLS schemes are generated floating-point numbers, and this can lead to limited accuracy [31]; moreover, the mantissa is denoted as a fixed number of bits. Then, it can be extended by two bits for each insertion, which can cause overflow problems [26, 32]. To sort this problem, they implemented a successive, variable-length storage format by adopting the ORDPATH technique [44], even though complicity in the ORDPATH has a negative effect on XML query processing [22, 55].

In [43], a GroupBased Approach is based on the prefix GroupID labelling scheme. The labelling mechanism is based on dynamic Dewey labelling and can be divided into two phases: Each label has a local and global fragment. The local label can be duplicated, although not within the same group. The global label uniquely identifies a group of local labels. The XML nodes, except the root within the tree, are first clustered in a way that each group of nodes is a sub-tree that has its root, and child nodes, and is given a global label. Each node in a group has a local label, starting from the parent node to the child nodes. It has flowed the GroupID prefix label and sizes rise rapidly as the XML tree goes deeper [48].

The SCOOTER labelling scheme [23] has been proposed based on quaternary strings and represents the node order lexicographically. In addition, the scheme supporting node insertion suffers from overflow problems in certain situations [22, 56]. Also, decoding large labels' quaternary strings slows down the query processing [27, 84].

Furthermore, [44] introduced a prefix labelling scheme, called ORDPATH. The main goals of ORDPATH are to gratefully handle the insertion of XML nodes in the database and to avoid relabelling. The main idea is to use only positive, odd integers to label elements in an initial load, and even and negative integer component values are reserved for later insertions into an existing tree. However, the ORDPATH technique allows just a limited number of insertions [16, 22], although, the complexity of the decoding mechanism has a negative effect on XML query processing [22, 55].

[114] illustrates a new prefix labelling scheme. The labelling mechanism is based on a mapping function that converts the integers allocated to the parameters Start, End, and Parent_Start to the binary bit string. The method takes advantage of the Fibonacci sequence to implement a variable-length storage format. In updating new XML nodes, a new section appears in the label in order to avoid relabelling the old nodes; this also keeps the order of the nodes and captures the structural relationships. However, the scheme has tested the different cases of insertion except for leaf node insertions.

A comparison of the existing labelling schemes is provided in table 3.1.

XML Labelling Scheme	**XML Document Type**	**Labelling Scheme Type**	**Data Type**
Improved Binary String Labelling (IBSL) scheme [55]	Dynamic	Prefix labelling scheme	binary, string
Prime-based Middle Fraction Labelling Scheme PMFLS [48]	Dynamic	Hybrid labelling scheme (prefix and region schemes)	Prime numbers

ReLab [46]	Static labelling scheme	Interval-based labelling scheme	Integers
ME labelling [39]	Dynamic updates	Hybrid scheme	Odd numbers
NLSXU scheme [83]	Static (did not support any node insertion)	Prefix labelling scheme	Digits (0-9), uppercase and lowercase letters and Unicode characters
[34] Dewey coding	Static	Prefix encoding	Integer numbers
[20] the Extended Dewey code	Static	Prefix labelling schemes	Integers, letters
Dynamic Dewey scheme (DDE) [22]	fully dynamic labelling scheme	Prefix labelling scheme	Integer numbers
Dynamic Float-Point Dewey DFPD [21].	Dynamic labelling scheme	Prefix labelling scheme	Float-point number
dynamic prefix-based labelling scheme DPLS [13]	Dynamic labelling scheme	Prefix labelling scheme	Floating-point numbers
GroupBased Approach [43]	Dynamic Dewey labelling	Prefix GroupID labelling scheme	Integers
SCOOTER labelling scheme [23]	Dynamic Skewed insertions	Prefix labelling scheme	Quaternary strings, order lexicographically
ORDPATH [44]	limited number of insertions	Prefix labelling scheme	Integers
new prefix labelling scheme [114]	Dynamic labelling scheme	Prefix labelling scheme	Integers
Clustering Labelling Scheme[104]	Dynamic labelling scheme	Hybrid labelling scheme	Numerical data

Table 3.1: A comparison of the existing labelling schemes.

3.2 The Limitation of Existing Labelling Schemes

Dynamic labelling schemes have been developed to support efficient XML updates. Each of the existing labelling schemes is limited in one or more aspects. The standard limitation we have found is that some of the mechanisms do not support dynamic updates. [17] is work on Prefix type of labelling scheme, and only supports static updates since it requires regeneration of the child name data after each insertion, as the element tag names are added as part of their Dewey labels [13, 111]. And this increases the label size [16, 112]. On the contrary, it performs well in evaluating query processing by accessing only the leaf nodes that contain the labels. This speed up the process and satisfies the queries [7, 13, 23, 38, 39].

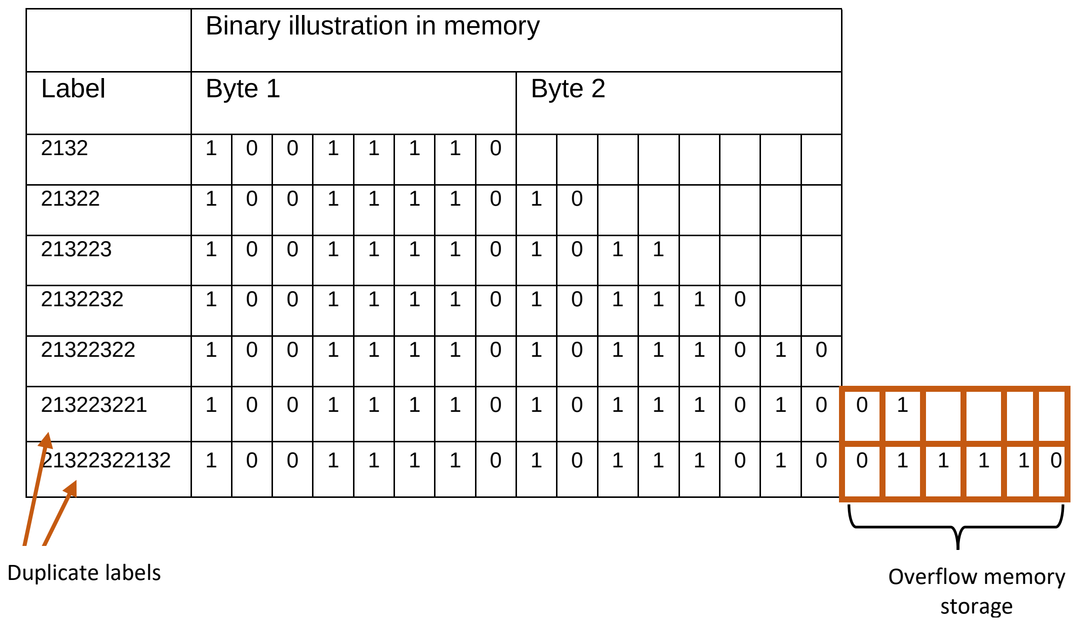
Similarly, the work in [34] is also based on the Prefix labelling scheme and supports static updates, which is unsuitable for dynamic XML documents as updating a new node requires the relabelling of all its existing right sibling nodes with their relatives in the entire XML tree [44]. [46] is a Region-based Labelling scheme and only supports static XML documents [13, 47], on the positive side, it is capable of generating labels faster than other Region-based schemes [47].

In [44] the dynamic prefix-based labelling scheme only allows limited updates [16, 22], as just the even and negative integer values are reserved for updating XML tree.

In addition, some schemes produce a large label size at the cost of extra storage. For example, [20] the size of extended Dewey is approximate 10%–30% more than that of original Dewey. This is due to the large size generated by applying their technique, which causes overflow problems [13, 21, 48].

Some labelling schemes suffer from overflow problems, where the node labels are stored as fixed-length binary numbers. The cause of the overflow is the fixed-length labels where the frequent insertions can lead to overflow problems [35, 84]. When an overflow occurs, a fragment of the new label can be lost, which can lead to the creation of duplicate labels. Figure 3.3 is an example of overflow problems.

studies such as [10, 19] did not evaluate the query performance experimentally but only presented it theoretically. On the other hand, others evaluated the XML query process only through determining the relationships over a large number of randomly selected label pairs. This has been done by [13, 21, 22].



	Binary illustration in memory																					
Label	Byte 1								Byte 2													
2132	1	0	0	1	1	1	1	0														
21322	1	0	0	1	1	1	1	0	1	0												
213223	1	0	0	1	1	1	1	0	1	0	1	1										
2132232	1	0	0	1	1	1	1	0	1	0	1	1	1	0								
21322322	1	0	0	1	1	1	1	0	1	0	1	1	1	0	1	0						
213223221	1	0	0	1	1	1	1	0	1	0	1	1	1	0	1	0	0	1				
21322322132	1	0	0	1	1	1	1	0	1	0	1	1	1	0	1	0	0	1	1	1	1	0

Duplicate labels
Overflow memory storage

Figure 3.3 Example of overflow problem [35]

3.3 Experimental Setting.

All experiments in chapter 4,5,6,7 and 8 were performed on a processor of an Intel Core i7 with 8GB of main memory and 64-bit Operating System, running Windows 10 system. We run different algorithms in Java IDE 8.2. The experiments evaluated the scheme's performance in terms of the label time and size. The experiments were conducted on the initial label and the handling insertions. We carried out the experimental process on different datasets [115]. The selection of XML datasets represents various features of XML trees, such as the number of nodes, file sizes, maximum depth and the degree of fan-out. It was vital to consider the variety of different datasets in order to reflect the scalability of the dataset in our results and evaluation. The real datasets we have used are DBLP, TreeBank, NASA, Reed, UWM, eBay, Sigmod and XMark. TreeBank datasets are designed by the University of Pennsylvania’s Department of Computer and Information Science, and the size of a TreeBank XML file is 82 MB with maximum breadth 144,493 [115]. The Digital Bibliography Library Project (DBLP) database is a large XML file related to computer science publications, conferences, series and books. The DBLP dataset is used by a wide variety of XML database applications, and it was used due to the ability to provide a wider range of sibling nodes with its maximum breadth being 328,858 [116]. The NASA database contains reliable astronomical data and has been developed from a flat-file format by a NASA XML Project. The size of the XML file is 23 MB with maximum breadth 80,396. The breadth describes the number of nodes on the same level whereas the depth is the number of levels from any node to its root and the maximum depth describes the maximum number of levels in an XML tree [115].

A Sigmod record is generally used to present and evaluate small XML databases [115]. eBay is an auction data converted to XML from web sources, Reed and UWM datasets store university courses’ data derived from university websites [115].

The XMark dataset is a well-known dataset and the most common benchmark for XML data management [117, 118]. It contains a scalable document database and is a large file with 111MB with a deep recursive ancestor structure. Moreover, the decedent nodes have a depth of 12 and a wide range of fan-out nodes which have different breadths at each level with the maximum being 25,500 nodes.

We adapted XPathMark queries that include the main aspects of the XPath language [119] and different relationships [117, 118]. XPath queries are widely used in other research, such as [120-123]. XPathMark was designed for the XMark Benchmark, which is a well-known and the most common benchmark for XML data management [117, 118].

3.4 Conclusion

In this chapter, we have reviewed the state of the art in terms of labelling schemes to design a novel algorithm that addresses some of the limitations of existing techniques as highlighted in section 3.2, namely: 1) to obtain a small label size, 2) supports dynamic updates without relabelling nodes, 3) support frequent insertions without overflow problems and 4) evaluate the query performance.

In order to investigate these aspects, two typical XML labelling schemes were applied [91, 124]. In chapter 4 we employed Range-based and Prefix Dewey Encoding in order to label different XML datasets that represent different features of XML trees. Various experiments were carried out to investigate the time and storage space required for each scheme.

4 COMPARISON BETWEEN RANGE-BASED and PREFIX DEWEY ENCODING

XML has become an increasingly important area in data storage and communication over the web. XML data labelling plays a significant role in the management of XML data since it allows the unique allocation of XML content in order to improve the query performance. This chapter focuses on two typical XML schemes for labelling native XML databases where the data is represented as ordered XML trees and contains relationships between nodes.

The remaining sections of this chapter are organized as follows: In section 4.1 we cover an overview on labelling schemes, in Section 4.2 we presented the existing related work in this area while Section 4.3 describes the Prefix Dewey encoding and the Range-based encoding methods. In section 4.4 the experimental results and evaluation are discussed while Section 4.5 concludes the chapter.

4.1 Overview

XML data has become one of the most important issues in the field of databases. Existing research has been conducted to improve the storing, retrieving and querying of XML data [34]. The main approaches for facilitating query processing based on native XML databases are structural indexing and labelling scheme. Labelling schemes focus on assigning a unique code to each node in XML trees as encoding for the documents to reduces the query processing time [12, 46, 88]. However, one of the criticisms of most of the encoding techniques is that they contain a large label size [38, 39].

This chapter compares Range-based encoding and Prefix Dewey encoding in order to achieve the fastest labelling time and to ensure the generation of short labels in term of memory. We used utf-8, utf-16, utf-23 to control the bits subsequent of the label value. The experiments evaluated the scheme’s performance in terms of the label size and initial label time.

4.2 Related Work

There are different labelling schemes which have been proposed for efficiently processing native XML databases. This section reviews and address issues related to the most common XML labelling schemes.

[34] have proposed a method called Local Order Encoding scheme, each node is assigned an integer number, which represents its relation position among its siblings. It is appropriate to reconstruct document order. The advantage is that it does not result in large label sizes and therefore each label has a fixed length, which is one byte for each node and uses UTF-8-character encoding scheme. However, fixed-length in labelling is leading to overflow problems. Also, the local encoding does not support all kinds of structural relationship queries, such as to determine the relationship between the following and preceding nodes.

In addition, [34] have proposed Dewey encoding scheme for labelling XML trees based on Dewey decimal classification system, it is one of the prefix labelling schemes. In this method, each label is presented as an integer number and delimiter “.” [8]. Each node (u) is labelled as a combination of its parent label and postfix integer number (x_i). If u is the x^{th} child of s in XML tree then the label of u, label (u) is the concatenation of label of s and x which is presented as the label(s). x, where s is the parent of u. For example, if an element label for u is 3.6.4, then its 5^{th} child label will be 3.6.4.5. If an element label is 6.2.4.1, then its parent label is 6.2.4, its first ancestor label is 6.2. An advantage of this method is that for any element label, we can easily extract node labels of its ancestors and determine the relationship between nodes. However, the drawback of the Dewey scheme is not appropriate for dynamic XML data; inserting a new sibling node requires relabelling all the right sibling nodes along with their descendants.

[22] have proposed a Dynamic Dewey encoding scheme (DDE), which is an update of the Dewey encoding scheme to transform the original Dewey into a fully dynamic labelling scheme. The advantage of the DDE is that the label has different length; starting with a byte for the first level and increases in depth concerning the level value. So that can be appropriate for avoiding overflow problems. In addition, it has the ability to avoid relabelling completely and support high query performance. The main drawback is that a large label size, especially when the depth increases and frequent insertions occur between two siblings by applying the midpoint technique.

[125] have proposed VLEI encoding scheme. VLEI scheme is applied to XML labelling, and the data type is binary string. The VLEI encoding has used number 9 for the identifier. For example, when a child node is inserted, the label for the node becomes the label of its parent node + 9 + VLEI code. However, VLEI encoding used eight bytes for the VLEI code. The VLEI main drawback is that lead to overflow problem, especially with skewed insertion. t is the new VLEI sequence code.

t = 1 . {0|1}$*$, If t.0.{0|1}$*$ < t < t.1. {0|1}$*$

For example, 10 < 1 < 11 and 100 < 10 < 101 < 1 <110 < 11 < 111.

The authors in [126] used Range based labelling scheme which aims to determine the structural relationships between nodes by using the related containment information. Each label is represented as a 3-tuple and has fixed-length. 10 scheme leads to overflow problems. *Start, end* and *depth* are used to identify exactly the position of an element. *Start* is generated by a pre-order traversal of the document trees exactly finds the occurrence position. While *end* is the maximal start of elements in the sub-tree of the current element and *depth* gives additional information to determine the parent-child relationship.

Following from the related work in this Section, the main drawback we have identified in the existing labelling schemes is the growth of the label sizes in response to that, in Section 4.3 we presented a comparison between two schemes with a focus on achieving labelling time and memory size.

4.3 Comparisons Between Prefix Dewey Encoding and The Range-Based Encoding.

In the Dynamic Dewey encoding scheme, each label has a different length; starting with a byte for the first level and it increases in relation to the level value. The length of labels can vary widely depending on the position of the nodes within the XML tree. However, prefix labels naturally extend when XML data is updated during frequent insertions, causing overflow problems. However, in the Local Order Encoding scheme**,** each node is assigned an integer number ,and each label has a fixed-length label; which is one byte for each node and used UTF-8-character encoding [127]. Furthermore, in Dewey encoding, each label is presented as a combination of its parent label and postfix integer number by delimiter "." [8]. In contrast, in Rang then its parent label is 6.2.4, its first ancestor label is 6.2. The based labelling scheme, each label presented as a combination of the *start, end, depth* values using "," as a delimiter. Furthermore, in Quaternary encoding QED [35] and SCOOTER encoding [23] proposed different delimiter storage scheme; they used number "0" as delimiters and consequently, these schemes increase the decoding time because of the extra comparison operation to identify the 0 whether a bit or a delimiter.

In the following experiment, we controlled the bits subsequent of the label value [127] for both Range based scheme and Prefix Dewey labelling scheme. We have done this to aid the generation of short label size and achieve the fastest labelling time.

4.4 Experimental Work and Results.

All experiments were performed on processor of an intel Core i7 with 8GB of main memory and 64-bit Operating System, running Windows 10 system. We run Range-based algorithm and Prefix Dewey labelling algorithm in Java IDE 8.2. We used utf-8, utf-16, utf-23 to control the bits subsequent of the label value. The experiments evaluated the scheme's performance in terms of the label size.

We carried out the experimental process on different datasets [115]. The XML datasets represent various features of XML trees such as the number of nodes, file sizes, maximum depth, the degree of fan-out. It was vital to consider the variety of different datasets in order to reflect the scalability of the dataset in our results and evaluation. The real datasets we have used are DBLP, TreeBank and NASA. TreeBank dataset is designed by the University of Pennsylvania's Department of Computer and Information Science, and it has a maximum breadth 144,493 with its size being 82 MB [115]. The Digital Bibliography Library Project (DBLP) database is related to computer science publications, books, series and conferences. The DBLP dataset is a large XML file with its maximum breadth being 328,858. It is used by a variety of XML database applications as it provides a wider range of sibling nodes [115]. The NASA database has been developed from a flat-file format by a NASA XML Project and contains reliable astronomical data This dataset has a size of 23 MB with a maximum breadth of 80,396. The breadth describes the number of nodes on the same level whereas the depth is the number of levels from any node to its root and the maximum depth describes the maximum number of levels in an XML tree [115].

XML dataset	File Size	Max Depth	Max breadth	Number of nodes
TreeBank	82 MB	36	144493	2437666
DBLP	127 MB	6	328858	3332130
NASA	23 MB	8	80396	476646

Table 4.1: Features of the existing real-life XML dataset.

Table 4.1 gives the properties of these datasets and summarises their characteristics. We described and specified the chosen datasets and the platform used.

4.4.1 Experimental Evaluation.

The label initialisation experiment for both Prefix Dewey labelling and Range-based were implemented successfully. The focus of this evaluation is the comparative of fastest labelling initial time and generation of short labels in term of memory. The outcome of this experiment was also aimed to compare the labelling size based on utf-8, utf-16 and utf-23. This experiment was intended to evaluate the Prefix Dewey labelling against Range-based schemes; the results showed that the experiment met its objective.

In this section, we illustrated the experiments that were used to gauge the Prefix Dewey Labelling Scheme compared to the Range-based scheme. This experiment examined two parameters: the initial labelling time and the total label size.

In our work, each of the schemes was individually executed, and the number of runs had to total at least 10 [128, 129]. In our work, the first three runs were omitted to validate the accuracy and the reliability of the results as well as to avoid cache memory.

Prefix Dewey labels

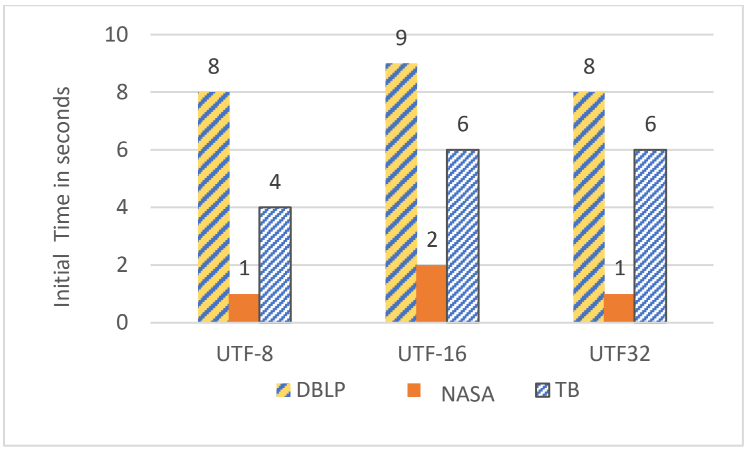


Figure 4.1 Initial labelling time for Prefix Dewey labelling scheme.

Range-based labels

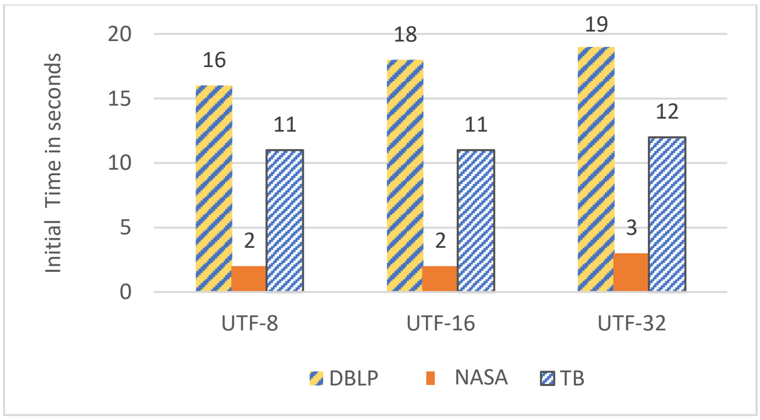


Figure 4.2 Initial labelling time for Range-based scheme

Prefix Dewey labels (Total label Size)

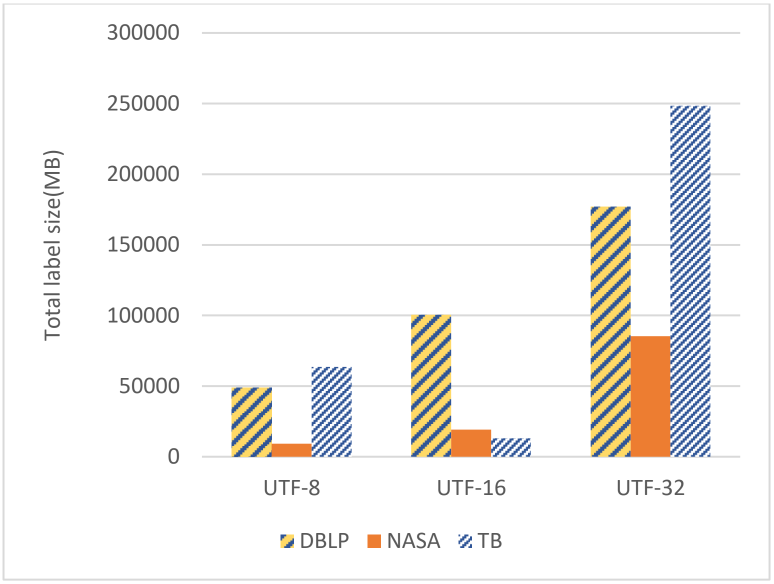


Figure 4.3 Total label Size (MB) for Prefix Dewey labels.

Range-based labels (Total label Size)

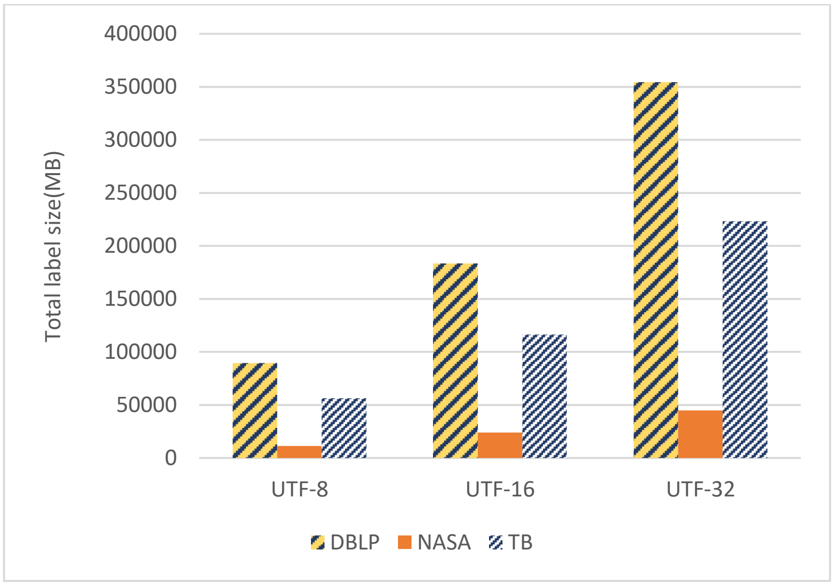


Figure 4.4 Total label Size (MB) for Range-based scheme.

From Figure 4.1 and 4.2 the Dewey Labelling scheme performs better in Treebank, DBLP and NASA Datasets. The results have shown that for the Prefix Dewey Labelling scheme, when applied in DBLP Datasets, its initial labelling time was 8 seconds in comparison to the Range-based scheme in which their execution time was longer by 50%. Therefore, leading to the conclusion that the time required to label the document in Prefix Dewey Labelling scheme is more efficient compared to Range-based scheme.

All schemes were executed 10 times to progress the accuracy of the initial label times in different datasets. It was necessary to identify the number of runs in order to gain considerable results.

From Figure 4.3 and 4.4 the Prefix Dewey Labelling scheme has a smaller label size in comparison to the Range based coding methods, except in the utf-32 test, it has shown that Range-based scheme performed better than Prefix Dewey scheme in NASA and TreeBank datasets.

4.5 Conclusion.

This chapter compares two XML labelling schemes, namely Range-based encoding and prefix Dewey encoding. The work was aimed to compare these schemes concerning the fastest labelling time and to ensure the generation of short labels in term of memory size and also to control the bits subsequent of the label value using utf-8, utf-16 and utf-23. Our experimentation has shown that the overall label size for Prefix Dewey has a smaller label value in comparison to the Range based encoding scheme, except in the utf-32 test, where Range-based scheme performed better in NASA and TreeBank datasets. In addition, when the schemes were applied to DBLP, TreeBank and NASA, their initial labelling time was 4,6 and 6 seconds respectively in comparison to the Range-based scheme in which its execution time was longer by more than 50 %.

Leading to the conclusion that the time required to label the document in Prefix Dewey Labelling scheme is more efficient compared to Range-based scheme. Also, the overall label size for Prefix Dewey has a smaller label value in comparison to the Range based encoding scheme.

In the next chapter, we present a novel labelling scheme by combining the advantages of the Prefix Dewey Labelling scheme. In order to build our scheme which aims to achieve the fastest labelling initial time and to ensure the generation of short labels in term of memory size.

5 PENTAGONAL LABELLING SCHEME FOR PREFIX DYNAMIC XML DATA

In this chapter we propose a novel Pentagonal scheme using Pentagonal theorem for assigning initial labelling and Handling XML updates. Various XML labelling schemes have been proposed to improve the storage, insertion and retrieval in dynamic XML data. Unlike other labelling schemes, our scheme preserves pentagonal numbers theorem [130] for insertions when updates occur.

The numbers n(3n - 1)/2 are called pentagonal numbers [115]. The first five pentagonal numbers are 1, 5, 12, 22 and 35. These numbers represent points that can be arranged to form regular pentagons [131].

In section 5.1 we cover an overview on the proposed Pentagonal labelling schemes, in Section 5.2 we discussed assigning the initial labelling for the Pentagonal Scheme, Section 5.3 presents handling XML updates that describe the Insertion Before the Leftmost sibling (Section 5.3.1), Insertion After the Rightmost Node (Section 5.3.2), Insertion Between Two Nodes (Section 5.3.3), Insert a child into a leaf node (Section 4.4.4) and Illustrates of node insertions (Section 5.3.5). The experiments of proposed labelling schemes presented in Section 3.4. the Results Analysis detailed in section 5.5. Lastly, Section 5.6 concludes the chapter.

5.1 Overview

At the development stage of our scheme, many other schemes were considered as a start point and offered inspiration for the Pentagonal labelling scheme whose strengths and weaknesses we evaluated. The following are some of the schemes that formed the basis to develop and improve our scheme.

DDE [22] is based on mathematical equations where the new label is allocated the midpoint and assigned its position between two siblings. The ORDPATHs [44] labelling scheme was designed to avoid relabelling by reserving negative-even integers for the insertion of new nodes. DFPD [21] is also based on mathematical equations where float-point numbers represent the new label. For more detail about these and other schemes, see chapter (3).

Based on our review of labelling schemes, we proposed the Pentagonal labelling scheme as a novel algorithm, since the use of pentagonal numbers to add new labels has not previously been attempted. Other aims, were to address some of the limitations of existing techniques which are to obtain a small label size, support dynamic updates without relabelling nodes, support frequent insertions without overflow problems, and evaluate query performances. More information on the limitations of existing techniques is provided in section 3.2.

The initial label mechanisms of nodes are generated based on the Pentagonal approach. The Pentagonal scheme can be denoted by (d, f), where d is synchronized with the Prefix labelling scheme and f is the Pentagonal test function for reserving the pentagonal number.

We used the prefix-based labelling scheme and preserved the pentagonal exponents for dealing with the insertion schemes. In the prefix XML labelling scheme, each node is associated with a node-id path from the root to the last component [20]. The last component is named as a self-label, and the other components are named as parents. The numbers n(3n - 1)/2 are called pentagonal numbers [130].

The following formula is for pentagonal numbers for dealing with the insertion schemes (see Equation 1).

$$\prod_{n=1}^{\infty}(1 - x^n) = \sum_{-\infty}^{+\infty}(-1)^n x^{n(3n-1)/2} \qquad (1)$$

$$(1 - x)(1 - x^2)(1 - x^3) \ldots = 1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15}$$

Table 5.1 gives the first twenty of Pentagonal numbers.

N	*f (n)*	*n*	*f (n)*
1	1	11	176
2	5	12	210
3	12	13	247
4	22	14	287
5	35	15	330
6	51	16	376
7	70	17	425
8	92	18	477
9	117	19	532
10	145	20	590

Table 5.1. Pentagonal sequences.

5.2 Assigning initial labelling for the Pentagonal Scheme

As presented in Figure 5.1, the initial labelling of our Pentagonal scheme is based on the DDE labelling schc1 and uses the prefix-based labelling scheme. In the Pentagonal scheme, the root node “Universities” is assigned a label value “1” and called the parent label. The child node labels form a sequence of components separated by ’.’. It assumes that node *X* in the XML tree has the label $x_1.x_2....x_m$; thereby, the labels of its children are $x_1.x_2....x_{m.i}$. Moreover, *i* in the initial label is not a pentagonal number so we cannot start with i = 1. The first child “Department” is assigned a label value “1.2”. The second and third children, “Employee” and ” Student” are assigned the label values “1.3” and “1.4” respectively. The fourth child *“Id”* is assigned a label value “1.6”, and the last component is 6 instead of 4. This aims to preserve the pentagonal numbers and retains them to support the XML updates. We illustrated the proof of updates in Figure 5.5. The labels of the remaining child nodes are generated by incrementing the label and avoiding the use of pentagonal numbers.

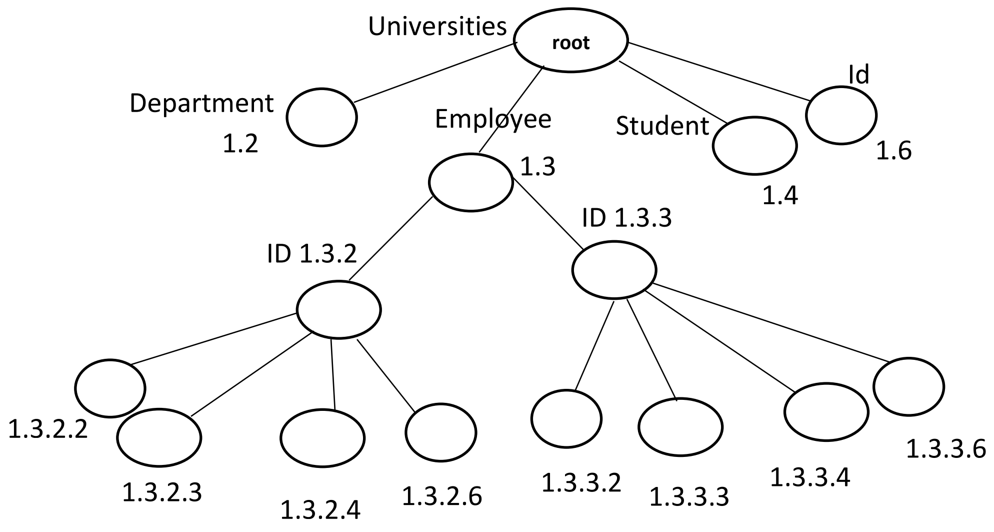


Figure 5.1. The initial labelling for Pentagonal scheme.

Definition 1. Given two labels *X* and *Y*, a node *X* is the root of a subtree containing the node *Y*. If there is a linked path of nodes from the root *X* to node *Y* such that $Z_1,\ldots,Z_m$ is a linked path of nodes and node $X=Z_n$, and $Y=Z_m$, where $n<m$ then node *X* is an ancestor of the node *Y* and node *Y* is a descendent of *X*.

Definition 2. Given two labels, X and *Y*, a node *X* is a parent of node *Y*; if *X* and *Y* are directly linked in an XML tree, and *X* appears exactly one level above *Y*, then *X* is a parent of a node *Y* and *Y* is a child of *X*.

Definition 3. Nodes *X* and *Y* are siblings if both nodes are at the same level and share the same parent in an XML tree. If *X* appears to the left of *Y* in an ordered XML tree, then *X* is called a pre-order sibling to node *Y*, whereas *Y* is a post-order sibling to node *X*.

Algorithm 1 Assigning the initial labelling for Pentagonal algorithm

```
Input: XML document
Output: Prefix Labels
Comment1: ⊕ denotes concatenation
01  if (n is the root)
02      rootlabel ⟵ 1
03  else
04      prefixLable(n)= label(parent(n))
05      for (n=1; n<=Cound(n); n++)
06          selflable = n;
07          get-original function(selfLable);
08          // if the result is integer means the selflable is a pentagonal
09              if (selflable is a pentagonal number)
10                  selflable ++;
11              endif
12          NewLabel ⟵ prefixLabel(n) ⊕ selflable(n)
13      end for
14  endif
```

Figure 5.2: Illustration of obtaining the initial labels for Pentagonal algorithm.

According to the first two lines, the root label is assigned the digit ‘1’. The first child label is assigned the digit ‘1.2’ and preceded by a sequence of prefixLabel(*n*) ⊕ selflable(*n*). The selflable(*n*) cannot start with 1 as 1 is a pentagonal number. The labels of the remaining child nodes are generated by incrementing the label. In this stage, the scheme avoids using the pentagonal numbers.

Algorithm 2 Insert Labels in Pentagonal Scheme

```
Input: Previous node labels
Output: Insert new nodes using Prefix Pentagonal numbers.
Comment1: ⊕ denotes concatenation
01 if (leftSideNode is empty and rightSideNode is notempty)
02       selflabel⟵lastComponentRightSide – 1
03       get-original function(selflable);
04        if (selflable(n) is a pentagonal number)
05              selflable(n) ⟵ selflable(n)-1
06        endif
07        NewLabel(n) ⟵ prefixLabel(n) ⊕ selflable(n)
08 endif
09 if (leftSideNode is notEmpty and rightSideNode is empty)
10        selflabel⟵lastComponentLifttSide +1
11        if (selflable(n) is a pentagonal number)
12             selflable(n) ⟵ selflable(n)+1
13        endif
14        NewLabel(n) ⟵ prefixLabel(n) ⊕ selflable(n)
15 endif
16 if (leftSideNode and rightSideNode is notEmpty) then
17    if (leftSideNode and rightSideNode are non-pentanal numbers) then
```

```
        selflable(n) ⟵ get-pentagonal (leftSideNode + rightSideNode)
        NewLabel(n) ⟵ prefixLabel(n) ⊕ selflable(n)
    endif
    if (leftSideNode and rightSideNode are pentanal numbers) then
        selflable1(n) ⟵ get-original (leftSideNode)
        selflable2(n) ⟵ get-original (rightSideNode)
        selflable(n) ⟵ get-pentagonal (selflable1 + selflable2)
        NewLabel(n) ⟵ prefixLabel(n) ⊕ selflable(n)
    endif
    if (leftSideNode is pentanal and rightSideNode is non-pentanal) then
        selflable1(n) ⟵ get-original (rightSideNode)
        selflable(n) ⟵ get-pentagonal (leftSideNode + selflable1)
        NewLabel(n) ⟵ prefixLabel(n) ⊕ selflable(n)
    endif
    if (leftSideNode is non-pentanal and rightSideNode is pentanal) then
        selflable1(n) ⟵ get-original (leftSideNode)
        selflable(n) ⟵ get-pentagonal (selflable1 + rightSideNode)
        NewLabel(n) ⟵ prefixLabel(n) ⊕ selflable(n)
    endif
endif
if (leftSideNode and rightSideNode are Empty) then
    prefixLable(n)= label(parent(n))
    NewLabel(n) ⟵ prefixLabel(parent) ⊕ 2
endif
return NewLabel
```

Figure 5.3. Illustration of the insertion of labels for Pentagonal algorithm and explained four types of insertion scenario.

Algorithm 3 Pentagonal-test algorithm

```
Input: XML node
Output: True or false.
01  $x$ ⟵ last component of the node tested
02  $y \longleftarrow (1+\sqrt{1+(24*x)})/6$
03  **if** ($y$ is an integer value) then
04      return true  // the number is pentagonal
05  **else**
06      return false // the number is non-pentagonal
07  **endif**
```

Figure 5.4. Illustration of whether the last component of the tested node is pentagonal or not.

Equation (2) is used for a get-pentagonal function:

$f1 = n(3n-1)/2,$ where *n* is a non-pentagonal value…..……….(2).

Equation (3) is used for a get-original function:

$f2 = (1+\sqrt{1+(24*x)})/6,$ where *p* is a pentagonal value …..(3).

If the result of equation (3) is an integer value, then the last component of the node is a pentagonal value. In comparison, if the result is not an integer, then the last component of the node is a non-pentagonal value. We tested the last component of each node in order to reduce the label size. For example, if the last component is 70, which is a pentagonal value, then equation (3) is applied, and the number will reduce to 7. Moreover, if the other last component is 176 this is a pentagonal value and equation (3) is applied. Then, the number will reduce to 11, as shown in Table 5.1. We add the new two values together; then, we apply the get-pentagonal function in equation (3) to obtain a new self-label, which should be a pentagonal number. The new label is generated as the prefix label for the parent and concatenated with the new self-label.

5.3 Handling XML updates

In this section, we emphasise the issue of handling XML updates, particularly in a dynamic labelling scheme that handles insertions without relabelling existing nodes. Our proposed scheme completely avoids relabelling in XML updates. Labelling schemes consider four cases of insertion [21, 22]. The case of an insert node before the leftmost sibling and after the rightmost sibling. Also, inserting between the two siblings and into the leaf node, the pentagonal labelling scheme provides all four cases when inserting nodes. This section illustrates the implementation of the four scenarios in insertion nodes, and using Algorithm 2 and Algorithm 3 (Figures 5.2 and 5.3) generates the values of the newly inserted labels. Thus, let *X* and *Y* be two nodes, whereby node *X* is labelled as *(x1.x2....xm)*, and node *Y* is labelled as *(y1.y2....yn).*

5.3.1 Insertion Before the Leftmost sibling

The first scenario: insert a new node before the leftmost sibling. Reducing the last component of the leftmost sibling by 1 and then by applying the Pentagonal-test in algorithm 3 (Figure 5.4) to creates the self-label. If the self-label is a Pentagonal number, then reduce another 1 from the self-label. The new label generated as the prefix label is concatenated with the new self-label. For example, the leftmost sibling is labelled as $x_1.x_2....x_m$; the generated new label is $x_1.x_2....(x_{m-1})$. However, if (x_{m-1}) is a Pentagonal number, then the generated new label is $x_1.x_2....(x_{m-2})$.

5.3.2 Insertion After the Rightmost Node

The second scenario: insert a new node after the rightmost sibling. The self-label is created by incrementing the last component of the rightmost sibling by 1 and then applying the Pentagonal-test algorithm. If the self-label is a Pentagonal number, then increase the self-label by 1. The new label is generated as the prefix label and concatenated with the new self-label. For example, the rightmost sibling is labelled as $x_1.x_2....x_m$; the generated new label is $x_1.x_2....(x_{m+1})$. However, if (x_{m+1}) is a Pentagonal number, then the generated new label is $x_1.x_2....(x_{m+2})$.

5.3.3 Insertion Between Two Nodes

The third scenario: insert between two siblings. First of all, if the last component of the left-side node and the last component of the right-side node are both non-pentagonal numbers, the self-label is created by adding the last component of the left-side node to the last component of the right-side node and then apply a get-pentagonal function using equation (2) to obtain the self-label. The new label is generated as the prefix label and concatenated with the new self-label. For example, given two siblings, the left sibling is labelled as (X: $x_1.x_2....x_m$), and the right sibling is labelled as (Y:$y_1.y_2....y_n$) where $m=n$ and x_m, y_n are non-pentagonal numbers, our algorithm is applied to get a pentagonal function for ($x_m + y_n$) in order to get the self-label, and the new label is generated as the prefix label for the parent and concatenated with the new self-label.

Secondly, if the last component of the left-side node and the last component of the right-side node are pentagonal numbers, our algorithm applies get-original function using equation (3) for the last component of the left-side node and for the right-side node and adds the two values together. The get-pentagonal equation (2) is then applied to obtain a new self-label. The new label is generated as the prefix label for the parent and concatenated with the new self-label. Equation (3) used for get-original function

$f2 = (1 + \sqrt{1 + (24 * x)}\,)/\,6,$ where p is a pentagonal value …..(3).

For example, given two siblings, the left sibling is labelled as ($X: x_1.x_2....x_m$), and the right sibling is labelled as ($Y: y_1.y_2....y_n$) where $m=n$ and x_m, y_n are pentagonal numbers. Our algorithm applies the get-original function for x_m and y_n and then applies the get-pentagonal function for the original values $x_{m'} + y_{n'}$ in order to get the new self-label. Then, the new label is generated by concatenating the prefix label for the parent with the new self-label.

Thirdly, if one node is pentagonal and the other is not a pentagonal number, our algorithm applies a get-original function for the pentagonal node and adds the two values together. Then the get-pentagonal function is applied to obtain a new self-label. The new label is generated as the prefix label for the parent and concatenated with the new self-label. For example, given two siblings, the left sibling is labelled as ($X: x_1.x_2....x_m$), and the right sibling is labelled as ($Y: y_1.y_2....y_n$) where m=n and x_m is a pentagonal number, y_n as a non-pentanal number; our algorithm applied the get-original function for x_m and then applied the get-pentagonal function for the original values $x_{m'}+ y_n$ in order to get the new self-label. Then, the new label is generated by concatenating the prefix label for the parent with the new self-label.

5.3.4 Insert a child into a leaf node

Fourth scenario: insert a child into a leaf node, where the new label is created by concatenating the prefix label with the digit "2". For example, the parent is labelled as ($X{:}x_1.x_2....x_z$). The generated new label is ($x_1.x_2....x_{z.2}$)

5.3.5 Illustrates of node insertions

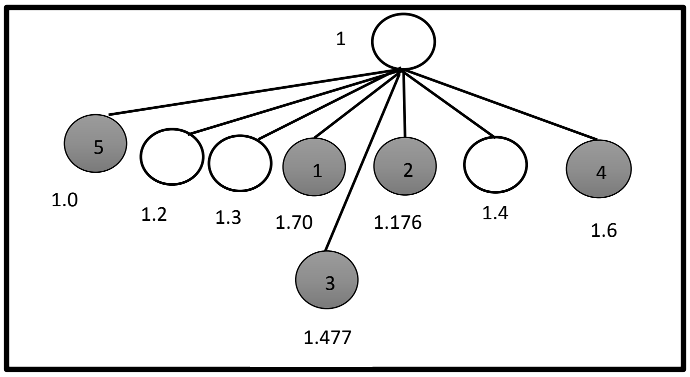


Figure 5.5 Processing labelling for different node insertion.

Figure 5.5 illustrates an example of node insertions, as clarified in the above scenario. The new nodes are inserted into XML trees represented by grey circles, and the numbers inside the circles represent the order of inserted nodes. Node number 1 is inserted between two non-pentagonal numbers, label node 1.3 and label node 1.4 and its 1.pentagonal (3 + 4), which equals 1.70. Node number 2 is inserted between pentagonal and non-pentagonal numbers, which is 1.70 and 1.4 so we returned to the original pentagonal number, which is 7 and its 1.pentagonal (7 + 4) is equal to 1.176. Node number 3 is inserted between two pentagonal numbers, which is 1.70 and 1.176 so we returned to the original pentagonal numbers, which are 7 and 11 its 1.pentagonal (7 + 11) is equal to 1.477. Node 4 is inserted after the rightmost sibling and labelled as 1.4; the generated new label is 1.6. If 1.(4+1) is a pentagonal number, then the generated new label is 1.(4+2), which is equal to 1.6. However, node number 5 is inserted before the first leftmost child; the leftmost sibling is labelled as 1.2, and the generated new label is 1.0.

The following section describes the experiments with their objectives.

5.4 Selection of dynamic labelling schemes.

Labelling schemes are mainly created to store XML documents. They are similar to other storing techniques since they support features such as perform an XML query using the path node information. Moreover, a labelling scheme helps to capture the structural relationships during the processing of queries. Despite all these similarities, Dynamic Labelling schemes differ from other Labelling schemes in that their internal model is based on dynamic XML documents and not based on the static model. Storing data in dynamic XML documents is appropriate for the reason that it is supports data updates without duplicating labels or relabelling old nodes.

It is imperative for the developer of any new labelling scheme to compare against existing labelling schemes and are widely used. This comparison allows us to compare our work with provides confidence in the viability of our solution.

We have selected different labelling schemes for this comparison based on the following selection criteria.

1. The main approach to store XML data is dynamic labelling schemes which is regularly updated [22, 26, 43, 46, 101, 102] and support data updates without duplicating labels or relabelling old nodes. Unlike other approaches to store XML data such as the static labelling schemes [14, 34, 85] which is used to store non-updatable XML documents.

2. Supports the Prefix labelling schemes.

3. Supports Loading of different XML document Sizes.

4. Supports XPath query language.

These criteria ruled out the selection of ReLab [46], NLSXU scheme [83] and Dewey coding [34] as these labelling schemes focus on static XML labelling schemes. In addition, ReLab [46] is an Interval Labelling scheme.

The labelling schemes that satisfy the aforementioned criteria were selected for the comparison, and they are: DPLS scheme [13] and the DDE scheme [22].

The following section describes the experiments with their objectives.

5.5 Experiments

To compare the proposed pentagonal labelling scheme to the DPLS and DDE schemes, several experiments were performed.

These experiments were conducted on the initial label and the handling insertions**.** The comparison has tested different aspects, time and size. These aspects had been facilitated using statistical analysis for pentagonal, DDE and DPLS schemes. All experiments were run on intel Core i7 processor with 8GB of main memory and a 64-bit Operating System, running a Windows 10 system. We run DDE, DPLS and pentagonal algorithms in Java IDE 8.2. The experiments evaluated the scheme's performance in terms of the initial time by seconds and the label size by Kbytes. We carried out the experimental process on different datasets [115].

The selection of XML datasets represents various features of XML trees, such as the number of nodes, file sizes and depth. It is crucial to consider the variety of different datasets in order to reflect the scalability of the dataset in our results and evaluation. The real-life XML datasets that we used are DBLP, NASA, Reed, UWM, eBay, Sigmod and XMark. The Digital Bibliography Library Project (DBLP) database is a large XML file related to computer science publications, conferences, series and books. The DBLP dataset is used by a wide variety of XML database applications, and it was used due to the ability to provide a wider range of sibling nodes with its maximum breadth being 328,858 [116].

The XMark dataset is a well-known dataset and the most common benchmark for XML data management [117, 118]. It contains a scalable document database and is a large file with 111MB with a deep recursive ancestor structure. Moreover, the decedent nodes have a depth of 12 and a wide range of fan-out nodes which have different breadths at each level with the maximum being 25,500 nodes.

TreeBank datasets are designed by the University of Pennsylvania's Department of Computer and Information Science, and the size of a

TreeBank XML file is 82 MB with maximum breadth 144,493 [115]. The NASA database contains reliable astronomical data and has been developed from a flat-file format by a NASA XML Project. The size of the XML file is 23 MB with maximum breadth 80,396. A Sigmod record is generally used to present and evaluate small XML databases [115]. eBay is an auction data converted to XML from web sources, Reed and UWM datasets store university courses' data derived from university websites [115].

Table 5.2 gives the properties of these datasets and summarises their characteristics. We specified and described the chosen datasets and the platform used.

XML datasets	Size of files	Max depth	Total of nodes
DBLP	172 MB	6	3332130
TreeBank	82 MB	36	2437666
NASA	23 MB	8	476646
UWM	2 MB	5	66729
Segmod	467 KB	6	11526
Ebay	34 KB	5	156
XMark	111MB	12	1666315

Table 5.2. Features of the most common XML Benchmarks datasets.

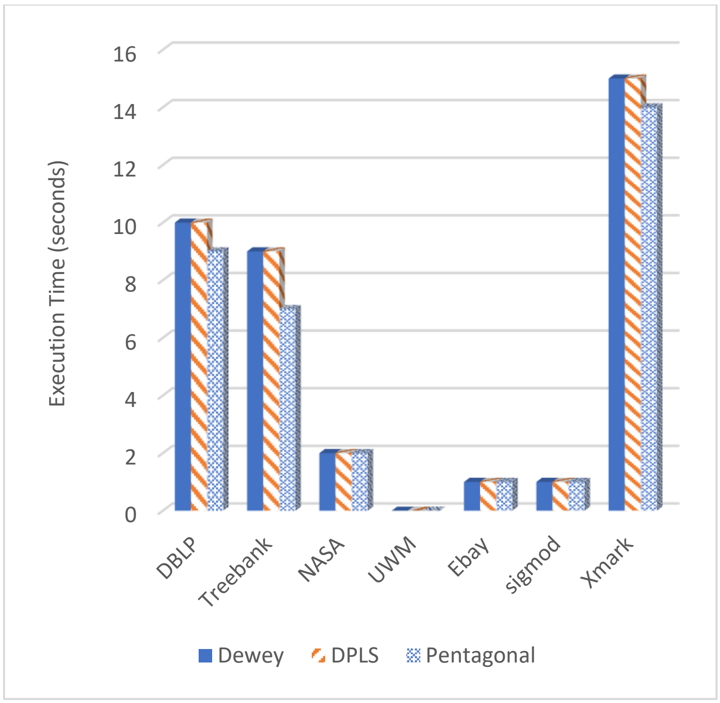


Figure 5.6. The initial labelling time for Dewey, DPLS and Pentagonal Schemes

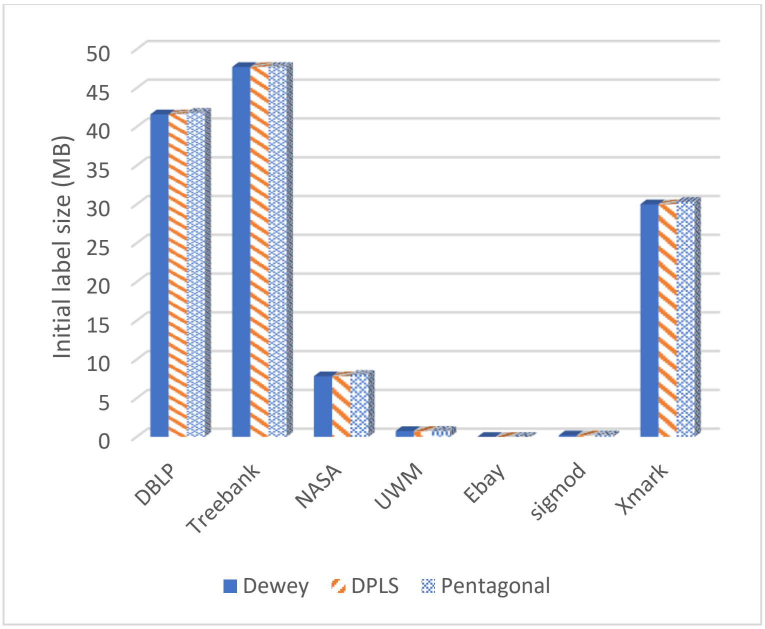


Figure 5.7. The initial Label size for Dewey, DPLS and Pentagonal Schemes

5.6 Results Analysis

In this section, we described the experiments that were used to evaluate the proposed Pentagonal Dynamic Labelling scheme compared to the Dynamic Dewey encoding scheme (DDE) and the Dynamic prefix labelling scheme (DPLS). The first experiment measured the initial label process in terms of time and size. Each of the schemes was individually executed a number of times, and the number of runs had to total at least 10 [128, 129]. However, others suggested using at least 30 runs to progress the accuracy [132]. In our work, the first three runs were omitted to validate the reliability, the accuracy of the results, and to avoid cache memory. From Figure 5.6, the Pentagonal Dynamic Labelling scheme performs best in DBLP, XMark and Treebank Datasets as the results have shown that for the Pentagonal Dynamic Labelling scheme, when applied in Treebank Datasets, its execution time was seven seconds in comparison to the Dewey and DPLS in which their execution time was longer, therefore leading to the conclusion that the Pentagonal scheme shows a better initial labelling time performance than the compared schemes, particularly when using large XML datasets. It is vital to identify the number of runs to gain considerable results; all schemes are executed 12 times to progress the accuracy of the initial label times in different datasets.

The statistical analysis of the results in Figure 5.7 indicated that there was an insignificant difference between the schemes in the initial labelling size; this is reasonable as DPLS and DDE are using the same scheme in the initial labels, and the proposed scheme at this stage avoids using pentagonal numbers. In addition, it would be reasonable to expect the growth of the label's size as the document size increases. From Figure 5.7, we can clearly identify that the size of loading the initial labels for Pentagonal, DPLS and DDE schemes are almost the same as DBLP, XMark, Treebank, NASA, UWM, eBay and sigmod datasets.

The second experiments were to evaluate the scheme's facility to handle XML updates. Four groups of experiments were executed using Pentagonal, DDE and DPLS schemes. The tests have covered the insertion of small and large numbers of nodes into the databases based on different insertion scenarios. The tree update was addressed by [32] through looking at two insertion process: random skewed insertions repeatedly handle new nodes between two consecutive siblings' nodes and ordered skewed insertions repeatedly insert new nodes before or after a particular node.

The first scenario measured the labelling time required using random skewed node insertions. The second was used to evaluate the storage space needed to store their labels. We compare the pentagonal scheme with DDE and DPLS schemes as they show powerful labelling dynamic in XML data. Figure 5.8. shows that the Pentagonal scheme achieved the fastest labelling time when handling random skewed node insertions. Our scheme can effectively support frequent insertions between two siblings', and the reliability of the pentagonal scheme is reasonable. As illustrated in Figure. 5.9(a) the Pentagonal scheme has smaller label sizes than the DPLS scheme. In particular, the Pentagonal performs best when a huge number of random skewed nodes has been updated (see Figure. 5.9(b)).

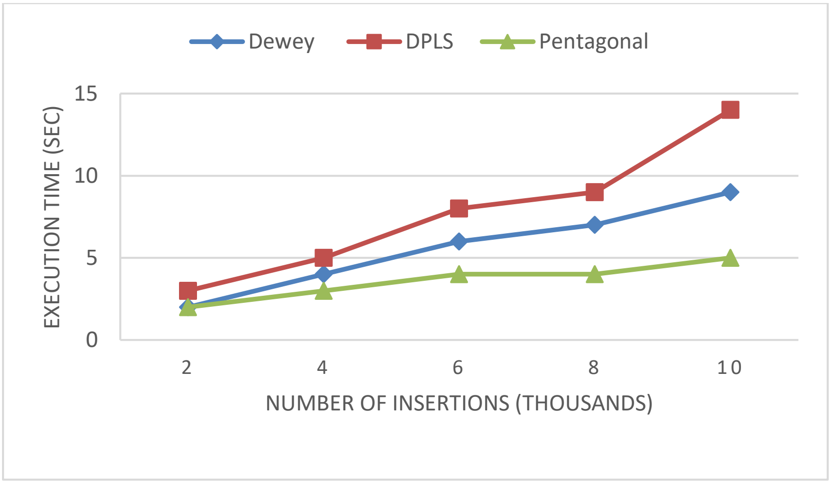


Figure 5.8(a). The labelling time of random skewed node insertions.

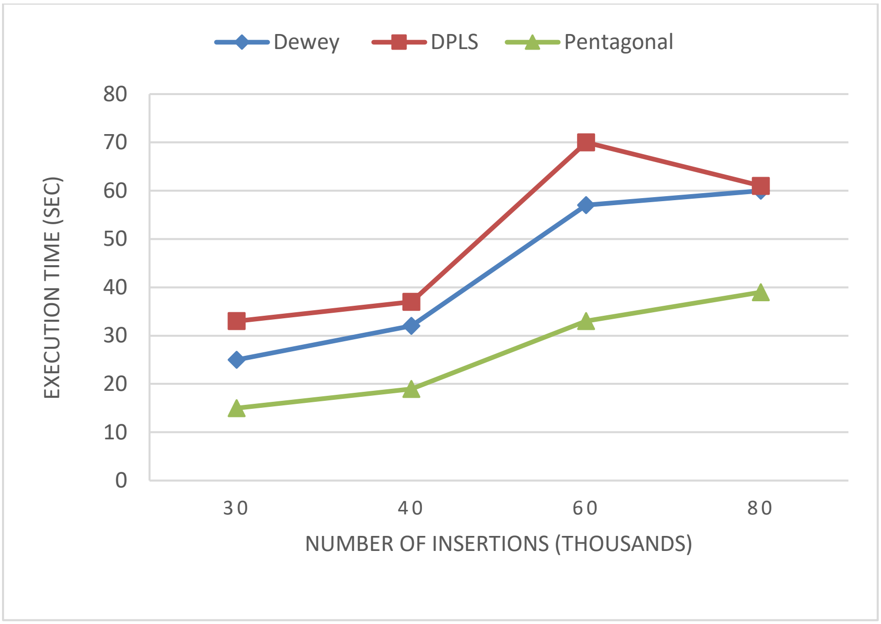


Figure 5.8(b). The labelling time of random skewed node insertions.

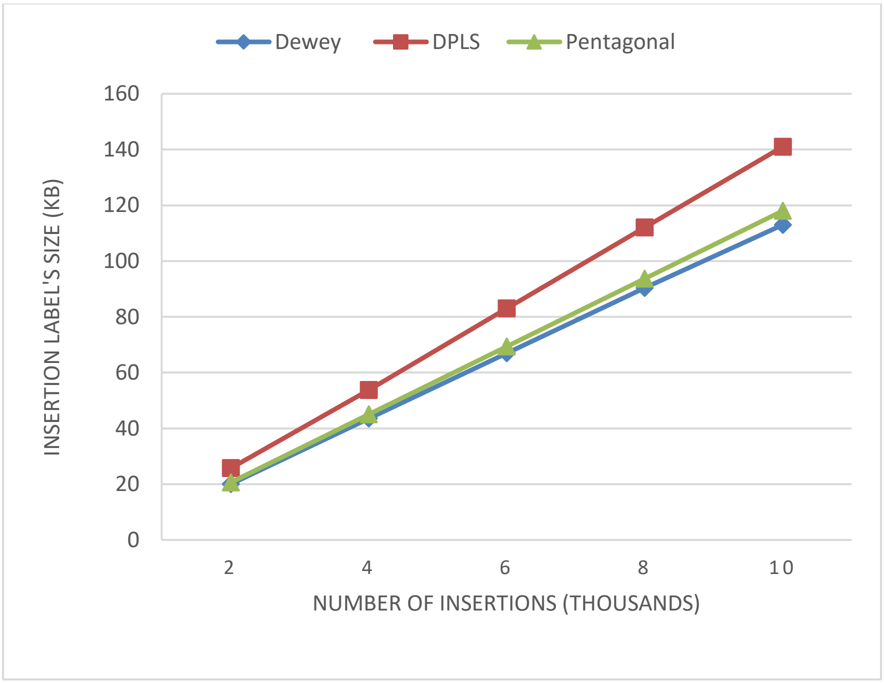


Figure 5.9(a). Label size of random skewed node insertions

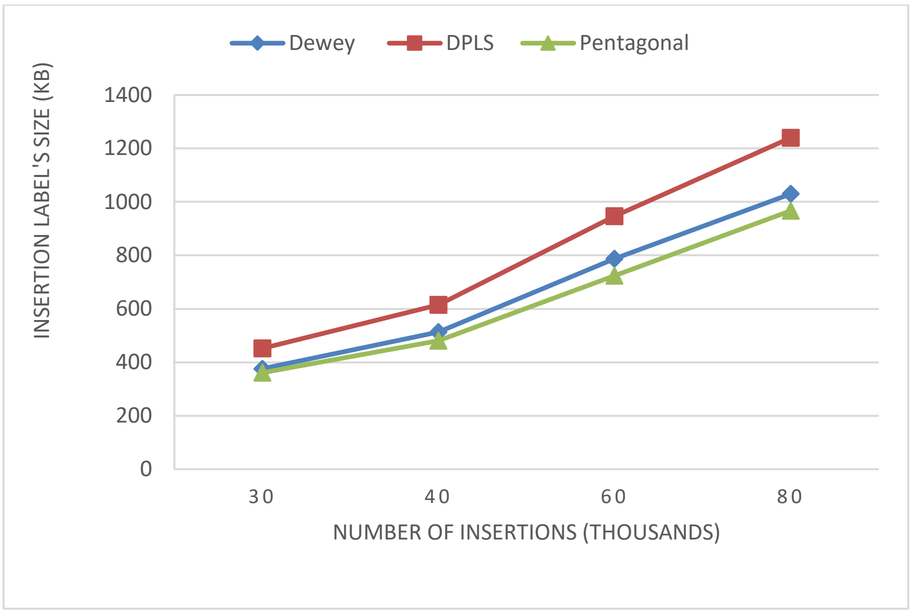


Figure 5.9(b). Labelling size of random skewed node insertions

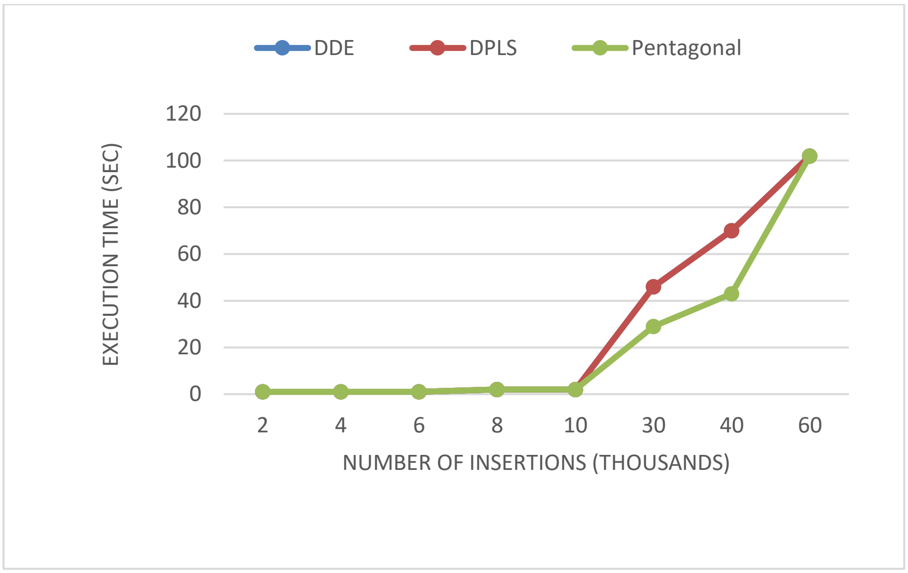


Figure 5.10. Evaluating the scheme's facility for execution times to handle XML updates into leaf node.

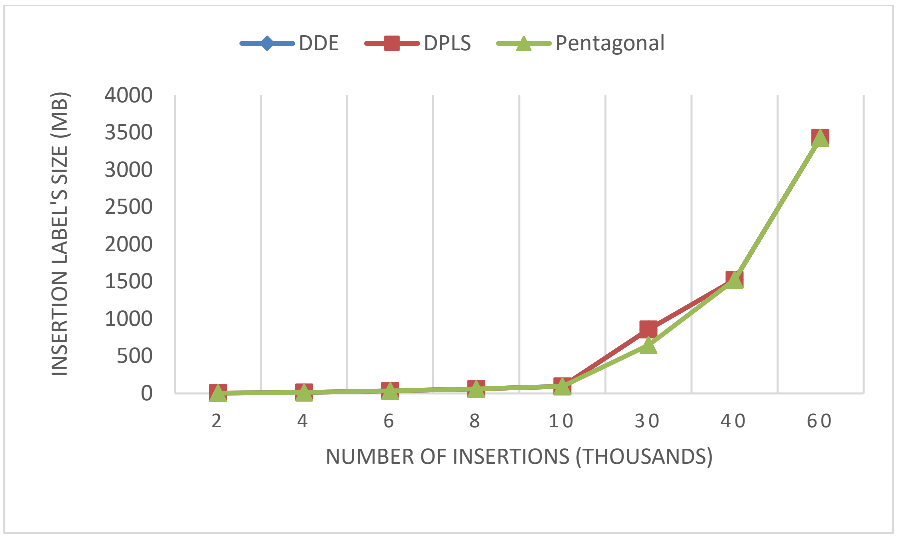


Figure 5.11. Evaluating the label size to handle XML updates into leaf node

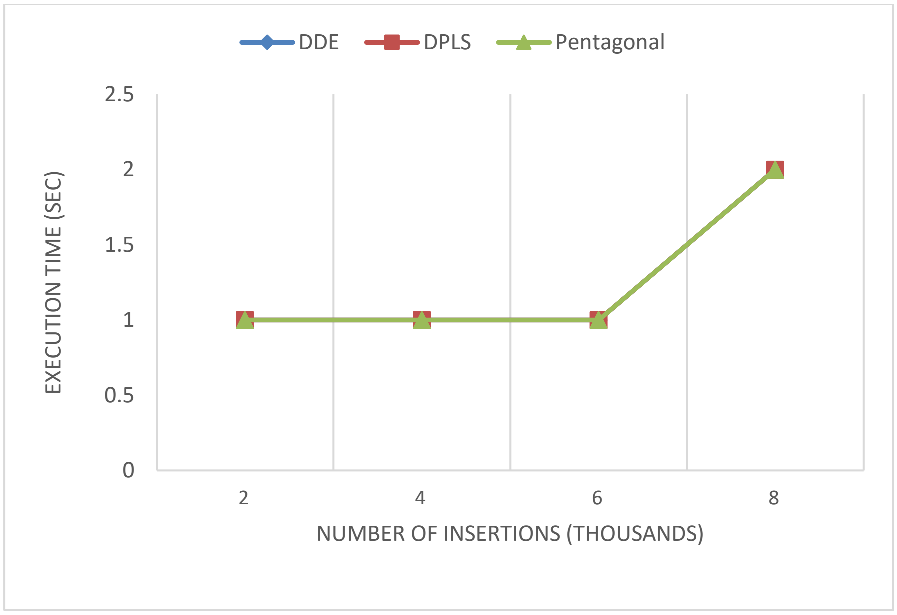


Figure 5.12. Evaluating the scheme's facility for the execution times to handle XML updates after the rightmost sibling.

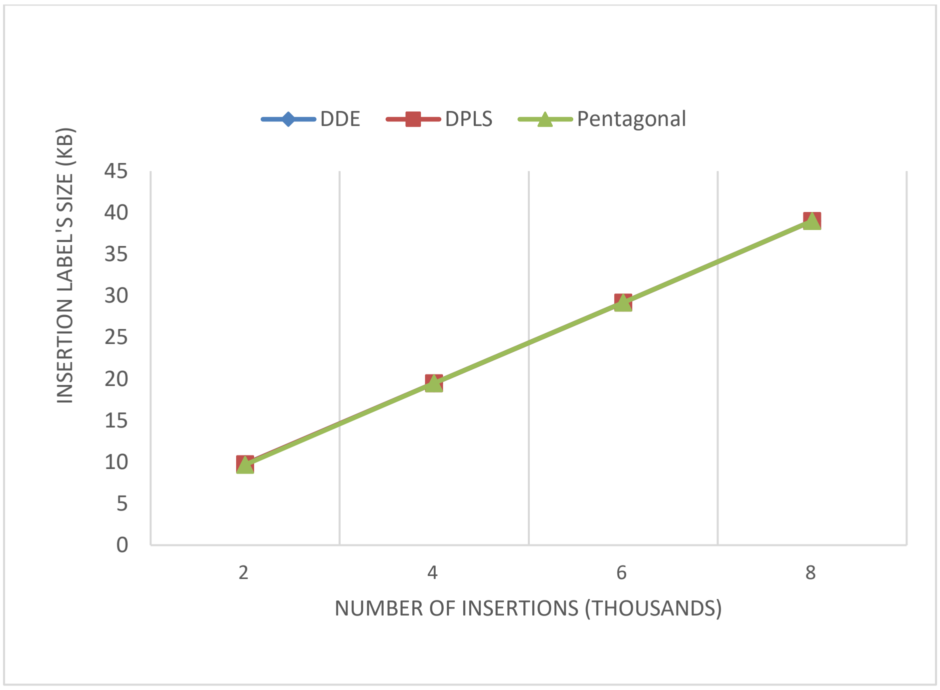


Figure 5.13. Evaluating the label size to handle XML updates after the rightmost sibling.

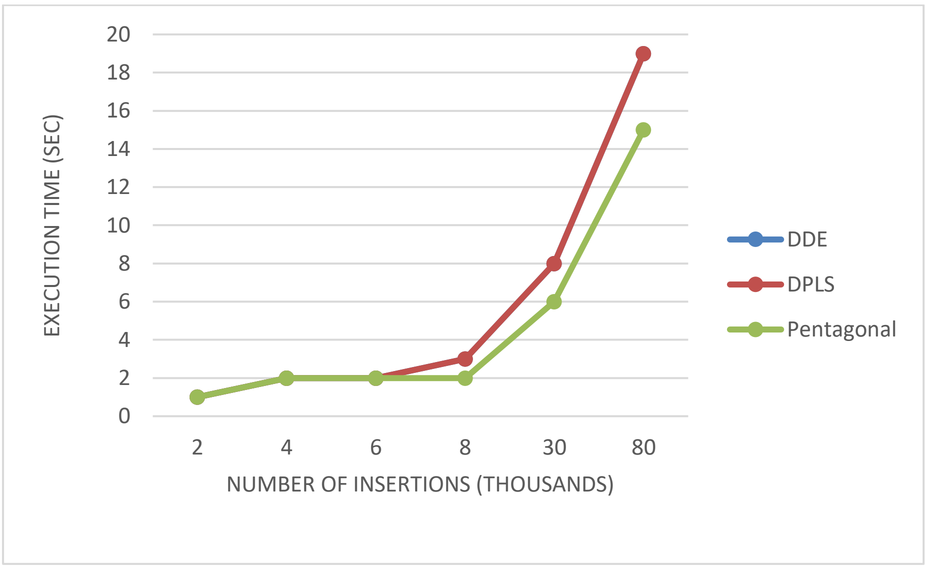


Figure 5.14. Evaluate the scheme's facility for the execution times to handle XML updates before the leftmost sibling.

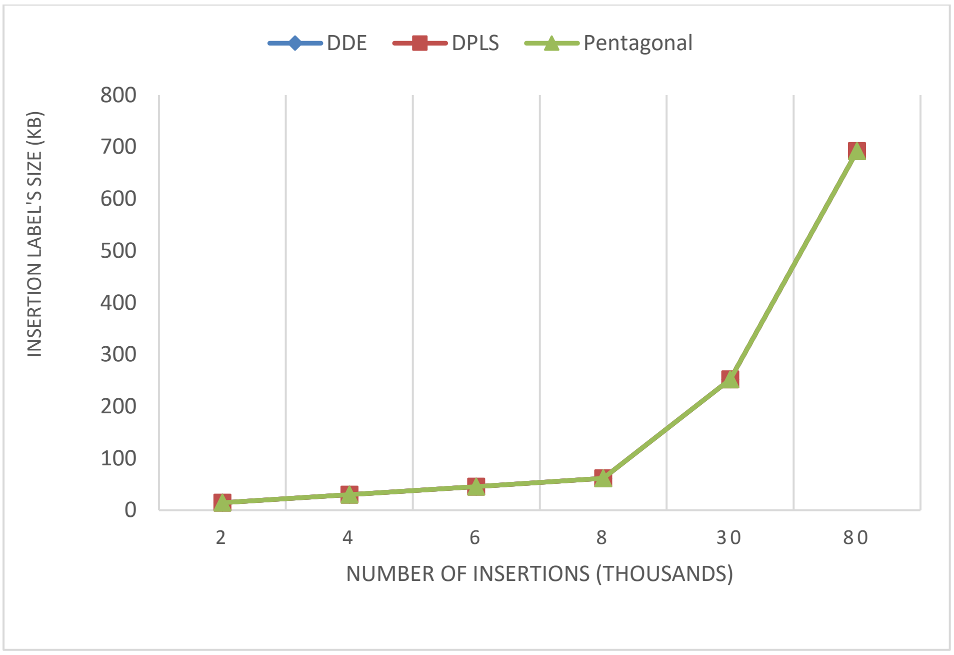


Figure 5.15. Evaluating the label size to handle XML updates before the leftmost sibling.

The results in Figures 5.11, 5.13, 5.15 presented the insignificant difference between the schemes when evaluating the scheme's facility for the label size in handling XML updates into leaf nodes, after the rightmost sibling and before the leftmost sibling. This is reasonable as DPLS and DDE use the same scheme in the update. In the execution times, the results from Figures 5.10 and 5.14 indicated that the Pentagonal scheme was better than DDE and DPLS. The experiments have been discussed, and the data obtained have been reflected on and evaluated. The experiments were carried out to evaluate the proposed scheme and its objectives. All experiments were run on the proposed, DDE and DPLS schemes.

5.7 Conclusion

We presented a novel Pentagonal dynamic labelling scheme to support updates over XML data. The experiment compared the ability of the Pentagonal Dynamic Labelling Scheme to handle insertions. In the experimental datasets, the tests covered the initial labelling in terms of the time and label sizes. Based on the experiments, the Pentagonal Dynamic Labelling scheme is more efficient with large XML documents; it performed best in DBLS, XMark and Treebank Datasets. A conclusion based on these results is that when labelling documents, our scheme has proven to be more efficient, particularly when using large size XML datasets. Also, the statistical analysis of the results indicated that the size of the initial labels for Pentagonal, DPLS and DDE are almost the same. This is reasonable as DPLS and DDE use the same scheme in the initial labels, and the proposed scheme in the initial labels just avoids using pentagonal numbers.

In addition, four types of insertion scenarios were tested. Firstly, the random skewed node insertions; the pentagonal scheme efficiently supports frequent insertions between two siblings. The outcome showed that our scheme achieved the fastest labelling times in term of random skewed node insertions. Moreover, in terms of label size, it has been verified that when handling XML skewed updates between two siblings, our scheme generates more compact labels every time than the DPLS scheme, which leads to decreased storage costs and performs the best when a big number of random skewed nodes is updated. From this, we evaluated the scheme's facility for the execution times and label size to handle XML updates into the leaf node, after the rightmost sibling and before the leftmost sibling.

The aim of the next chapter is to analyses the query performance of labelling schemes over dynamic XML documents. The experiment compared the ability of the Pentagonal Dynamic Labelling Scheme to handle response time queries and the time spent to determine different relationships.

6 QUERY EXPERIMENTS

In chapter 3, we discussed that extant research such as DPLS and DDE schemes evaluated the XML query process only through determining the relationships over a large number of randomly selected label pairs [13, 21, 22]. In addition, other researchers did not evaluate the query performance experimentally but only presented it theoretically [10, 19].

This chapter measured performance for our novel scheme based on query response time and determined structural relationships based on a prefix comparison.

In section 6.1 we cover an overview of datasets and queries for evaluation; Section 6.2 describes the Query performance where section

6.2.1 analysis the results of the query performance on the initial label and Section 6.2.2 analysis the results of the query performance after insertion. Lastly, Section 6.3 concludes the chapter.

6.1 Overview of Datasets and Queries for evaluation

XPath Query retrieves and navigates an XML document based on regular path expressions by appropriate structural relationships [43]. We adapted XPathMark queries that include the main aspects of the XPath language [119] and different relationships [117, 118]. XPath queries are widely used in other research, such as [120-123]. XPathMark was designed for the XMark Benchmark, which is a well-known and the most common benchmark for XML data management [117, 118], and it also presented in more detail in section 5.4. Different queries are specifically designed to validate the scalability of the XMark dataset, and it provides a variety of structural relationships, such as child-parent, ancestor-descendent and following-preceding sibling [117, 118].

We applied the structural joins stack-tree algorithm [133] to perform XPath queries in both the proposed and compared schemes. The structural joins algorithm leads to optimal join performance, and this is key to the efficient implementation of XML queries [134]. Moreover, the

XPath queries works more efficiently under the structural joins algorithm [135]. Therefore, the developers of the XML labelling schemes, such as [33, 136-138], have used structural joins for querying the XML dataset. In their experiments, each query was individually executed several times, with the minimum being 10 times [128, 129]. However, from the statistical test perspective, others prefer at least 30 runs in order to enhance the accuracy of the test [132].

In our work, the first 10 runs were omitted to avoid cache memory and to validate the reliability and accuracy of the results. Queries were executed in 1,666,315 initial labels for each scheme in the XMark dataset as this is the total number of nodes in the Xmark dataset in order to determine the relationships amongst 6,000 pairs of labels that were newly inserted to test the query performance when XML is updated. All tests were performed on the Pentagonal, DDE and DPLS schemes. Tree-structured relationships are parent-child (PC), ancestor-descendent (AD), sibling (S), lower common ancestor (LCA), and document order (DO).

The XPath queries used in our experiments are defined in Table 6.1 [117, 118], where the first column determines the relationship between the nodes; the second column indicates and describes the queries.

Axis name	**Example & description**
Child Axes.	Query 1: /site/regions/*/item - Selects all children of the current node, and all the items.
Parent Axes.	Query2: /site/regions/*/item[parent::namerica or parent::samerica] - The (North or South) American items. Element named items are the children of the world region they belong to. Retrieve all items belonging to either North or South America.
Ancestor Axes.	Query 3: //keyword/ancestor::listitem - Ancestor: Selects all ancestors (parent, grandparent, etc.) of the current node.
Descendent Axes.	Query4: /descendent-or-self::listitem - Descendant-or-self: Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself.
Sibling Axes.	Query5: /site/open_auctions/open_auction/bidder - Children named the bidder of a given open auction are siblings. Following-sibling: Contains the following siblings of the current node.

Table 6.1. XPath Queries

Queries1 and 2 represent parent-child relationships; query1 selects all children of the current node and query2 retrieves all the items belonging to either North or South America, while the elements named item are children of the world region to which they belong. Queries3 and 4 represent the ancestor-descendent relationship; Query3 selects all ancestors (parent, grandparent, etc.) of the current node. Query4 selects all descendants (children, grandchildren, etc.) of the current node and the current node itself. In addition, query5 refers to the following siblings (i.e. post-order siblings) of the current node.

In our work, we evaluate our scheme for query performance by using the most common structural relationships.

6.2 Query performance

In this chapter, we evaluated our scheme for the query performance by using XPathMark queries and the time spent on determining different relationships. In this section, we compared the query performance of different labelling schemes. Also, we performed our experiments by computing the following relationships: parent-child (PC), ancestor-descendent (AD), sibling (S), lower common ancestor (LCA), and document order (DO). In the first experiments (see Section 6.2.1) as we evaluated the response time by using different types in the XPath queries, which are child-parent, ancestor-descendent and following-sibling. The second experiment supports query processing by determining the structural relationship between nodes (see Section 6.2.2).

6.2.1 Query performance and Results' Analysis on the initial label.

In this section, we evaluated the response time for the queries by using different types in the XPath queries, which are child-parent, ancestor-descendent and following-sibling relationship. The queries were executed in 1,666,315 initial labels for each scheme in the XMark dataset as this is the total number of nodes in the Xmark dataset.

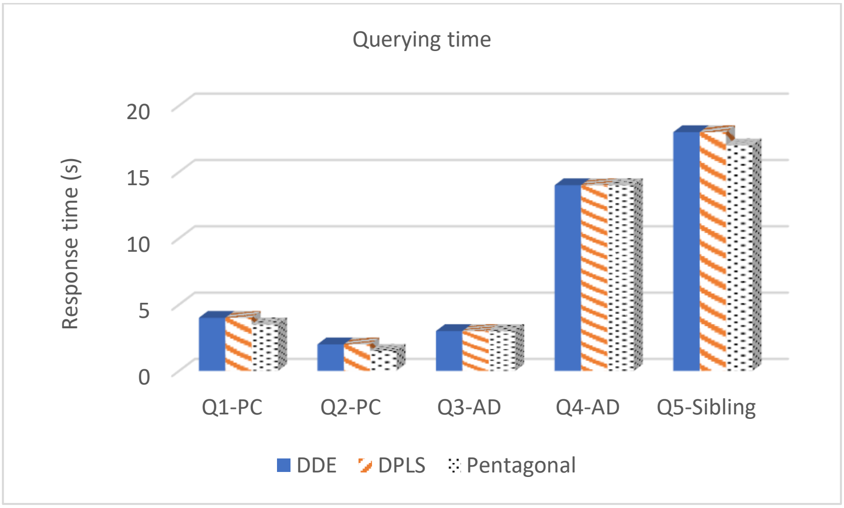


Figure 6.1: Comparison of the query performance over the initial labels

Figure 6.1 reports the results of the query performance execution time when applying different labelling schemes. We applied XPathMark queries in the initial label of the DDE, DPLS and Pentagonal schemes. The queries run over the labels of the XMark dataset; they represent a parent-child relationship for Queries1 and 2, and an ancestor-descendent relationship for Queries3 and 4. In addition, Query5 represents the sibling relationship.

The results in Figure 6.1 indicate that there was no difference between the DDE and DPLS schemes as these schemes used the same Prefix Dewey scheme to generate the initial labels. However, there was a difference in the query performance in the initial labelling between our scheme and the compared schemes. The Pentagonal Labelling scheme performs better in Queries 1, 2 and 5 for parent-child and sibling queries than DPLS and DDE, while Queries 3 and 4 take the same length of time for the ancestor-descendent queries.

This reflexion was investigated statistically in order to obtain the statistical results of the comparison between the algorithms, the Mann Whitney U-test was used [139]. It calculated the U statistic that corresponded to each algorithm by applying the time of each query individually using equations 4 and 5.

$U_A = n_A n_B + ((n_A(n_A+1))/2)-R_A$ ………………… (4)

$U_B = n_A n_B + ((n_B(n_B+1))/2)-R_B$ …………………. (5)

We calculated the R_A, R_B by run each schemes several times and then we sum the the time allocated to the Pentagonal and the DPLS schemes respectively [139]. Therefore: n_A, n_B is the number of observations in the Pentagonal scheme and the DPLS scheme respectively, In this expermint, the number of observations was five. The Mann Whitney U-test provided a p-value less than the significance level as the p-value that obtained was 0.004 when the significance level was 0.05 [139]. This means that the test supports the pentagonal scheme; it had a direct impact on the response time for the parent-child and sibling queries. The Mann Whitney U-test showed that the ancestor-descendent queries did not quantify a significant difference as the p-value was 0.579 when the significance level was 0.05.

6.2.2 Query performance and Results' Analysis after insertion.

This experiment measured the time needed to determine the different relationships between two nodes using their labels. Figure 6.2 presents the time required to determine the relationship after 6,000 pairs of new labels were inserted. The performance was tested on DDE, DPLS and Pentagonal schemes. The results gained from this experiment shows that the Pentagonal Labelling Scheme performed best in the parent-child (PC), sibling (S), lower common ancestor (LCA), and document order (DO). However, the same results were obtained in the ancestor-descendent (AD) relationship.

The results showed that when applying the sibling (S) relationship to the Pentagonal scheme, the execution time was 68 millisecond, which was faster by 35.3% than the DDE and faster by 44.1% than the DPLS. In addition, when applying the parent-child (PC), lower common ancestor (LCA), and document order (DO) relationships, the execution time was 12, 28, 16 milliseconds less respectively in the Pentagonal scheme compared to the DDE and DPLS. This could suggest that the time required to determine the relationship in our scheme is more efficient amongst sibling, parent-child (PC), lower common ancestor (LCA), and document order relationships. Our scheme can effectively support queries after skewed insertions, and the reliability of the pentagonal scheme is reasonable as it has a fast response time, which thus means it handles queries efficiently. As Figure 6.2 shows, the Mann Whitney U-test was used to test for significance to calculate the time spent on determining different relationships. The p-values were 0.001 for the parent-child, sibling, LCA and DO relationships. The tests provided a p-value less than the significance level of 0.05; this means that these queries were faster. For the query ancestor-descendant, the Mann Whitney U-test did not quantify a significant difference as the p-value was 0.531.

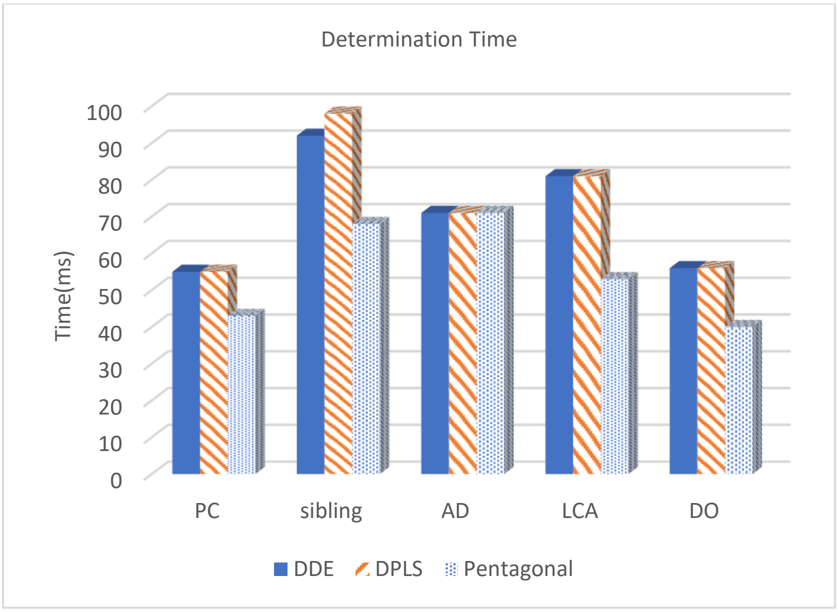


Figure 6.2: Computation time of relationships

According to the test results, there is a difference in the querying time between the compared schemes for the initial label for both parent-child and sibling (p = 0.004). In addition, there are a difference in the determination time for relationships after insertion for parent-child, sibling, Lowest Common Ancestor (LCA), and Document Order relationships as (p = 0.001). The tests provided a p-value less than the significance level of 0.05; this means that these queries were faster and confirmed a better performance.

The findings indicate that the time for parent-child and sibling queries, also determines the time for the parent-child, sibling, Lowest Common Ancestor (LCA), and Document Order relationships. The p-values, which ranged from 0.001 to 0.004, were obtained, which are extremely low values. This means that the test supports the pentagonal scheme; it had a direct impact on the response time and was faster than the compared schemes.

6.3 Conclusion

The aim of this chapter is to analyses the query performance of labelling schemes over dynamic XML documents. The experiment compared the ability of the Pentagonal Dynamic Labelling Scheme to handle response time queries and the time spent to determine different relationships. The results showed that the Pentagonal Dynamic Labelling Scheme has a faster response time than the DPLS and DDE for parent-child, siblings, lower common ancestor, and document order. However, all schemes performed equally in the ancestor-descendent relationship queries.

According to the experimental results, Pentagonal is more efficient than the DPLS and DDE schemes except for the ancestor-descendent relationships where all schemes had similar response time.
However, it proved its capability in terms of the query performance on the initial label and in determining the relationships after insertion.

The next chapter compared the ability of the Pentagonal Dynamic Labelling Scheme with two well-known Native XML databases systems, the eXist database and BaseX database to handle different dataset sizes and executed for different queries.

7 COMPARISON BETWEEN NATIVE XML DATABASE STSTEMS and PENTAGONAL LABELLING SCHEME

The previous chapter described the experiments that were used to measure the performance on the basis of query response time. This chapter presents the results obtained from these experiments and compares the results from our proposed scheme with two well-known Native XML databases systems. In this work, we consider eXist database version 5.2 [66] and BaseX database version 9.4 [140].

The next section identifies the concept of Native XML database systems and Introduction to database systems, followed by non-functional comparison features and attributes in section 7.2. Section 7.3 presents a functional comparison, the experiments based on the comparison of loading time and query response time. Section 7.4 presents the result analysis, Load Time Performance (Section 7.4.1) and Query performance (Section 7.4.2). Finally, Section 7.5 concludes this chapter.

7.1 Selection of Native XML database systems

Native XML databases are mainly created to store XML documents. They are similar to other databases since they support features such as security, transactions, multi-user access, query languages, programmatic APIs and many other vital features. Despite all these similarities, Native XML databases differ from other databases in that their internal model is based on XML and not based on the relational model which what other databases are commonly based on [141].

Storing data in XML documents in a native XML database is appropriate for the reason that it is space-efficient when data is semi-structured. This is where there is a variety in its structure in which mapping it to a relational database causes a large number of tables or a large number of columns with null values [141].

It is imperative for the developer of any new labelling scheme to compare against existing real-world databases that use their labelling schemes and are widely used. This comparison allows us to compare our work with industry-standard systems and provides confidence in the viability of our solution, and its applicability in real-world settings.

We have selected different databases for this comparison based on the following selection criteria.

1. The main approach to store XML data is the Native XML Database (NXD) that is used to store document-centric XML which contains semi-structured XML document and is stored in the hierarchical structure [66]. Unlike other approaches to store XML data such as the XML Enabled Database (XED) which is used to store data-centric documents that contain well-structured information and can be transferred into a traditional relational database [64, 65].

2. Supports XPath query language.

3. Supports Loading of different XML document Sizes.

4. Utilises XML Parser to generate the XML labels that represent the XML tree structure of the XML files.

These criteria ruled out the selection of DB2 and *Oracle XML DB as* these databases focus on XML-enabled database as data-centric documents.

The databases that satisfy the aforementioned criteria were selected for the comparison, and they are: eXist database system version 5.2 [142] and the BaseX database system version 9.4 [140, 143]

7.1.1 Introduction to eXist and BaseX database systems

eXist: XML documents are stored and managed in hierarchical collections. The eXist database uses a numerical indexing scheme in order to speed up query processing [66]. This scheme supports the rapid determination of structural relationships between nodes. For example, ancestor-descendant, parent-child, and following–preceding siblings. In addition, all nodes in the document are indexed. Consequently, the eXist creates full indexing over all nodes [66].

The indexing or the numbering scheme implemented in eXist provides an extension to Lee et al. [97] which presents the document tree as k-ary tree, where k is matching to the maximum number of children nodes of the element in the XML document. A unique number is allocated to each node by traversing the tree in level-order. For two nodes n and m of a tree, size(n) = size(m) if level(n) = level(m), where size(x) is the number of children of node x and where level(y) is the path's length of the from the root node of the tree to y. In addition, at each level, additional information on the number of children needs to be stored in an array. Figure 7.1 illustrates an example of the XML document, and Figure 7.2 shows the labelling generated by eXist [61].

```
<contact>
        <name>John Cage</name>
        <phone>
                <office>19</office>
                <home>1010</home>
        </phone>
</contact>
```

Figure 7.1. An example of XML document.

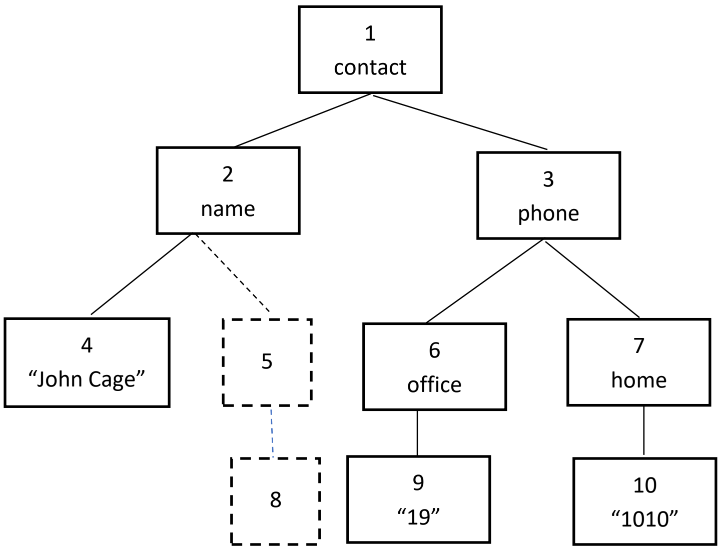
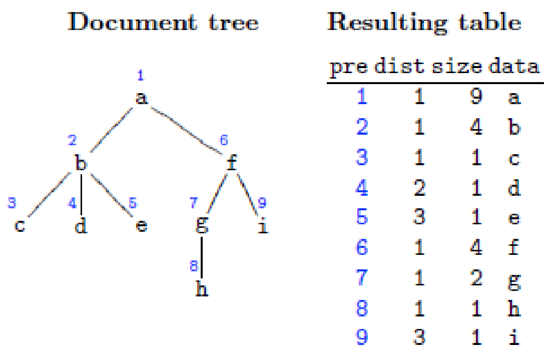


Figure 7.2. Unique identifiers allocated by the level-order labelling scheme [66].

In order to avoid the relabelling node on the case of updating documents, it is possible to leave spare labels between nodes. However, eXist does not afford an advanced update scheme as illustrated in Figure 7.2, the eXist is more suitable for static documents that rarely updates rather than dynamic as it is updates document as a whole rather than to manipulate a single node [66].

BaseX database modification the encoding scheme to speed up the query execution time and optimize the memory consumption [143]. Figure 7.3 shows the encoding scheme generated by BaseX.



Resulting table

pre	dist	size	data
1	1	9	a
2	1	4	b
3	1	1	c
4	2	1	d
5	3	1	e
6	1	4	f
7	1	2	g
8	1	1	h
9	3	1	i

Figure 7.3: Table Encoding in BaseX

The basic of BaseX scheme is the pre-order and post-order plane. All nodes in the document are allocated a *pre* and *post* value, based on the locations they take during a pre and post tree traversal. Each document node persuades a partition of this plane into four separate sections, in order to represent the main XPath axes parent-child, ancestor-descendent and preceding-following [40, 143, 144]. However, using an ordering approach helps with gathering a compact storage. In addition, it helps to hold the fixed-length label to each node [40]. However, as the *pre-order* and *post-order* values signify the hierarchy and the order of the document, performing the update of nodes are expensive [40, 41], particularly when facing the worst-case scenario of updating, that can lead to relabelling of all nodes in the document [40, 41]. In BaseX the *pre-value* serves as node id. In addition, to keep track of the node relationship, BaseX also uses *distance* and *size* values which is for two reasons. Firstly, in comparison to the *parent* value, the distance value is the number of nodes that exist between the parent node and the child node. Whenever the *pre-value* of the parent changes, then all child nodes must be updated. In BaseX, the parent is always equal to "*pre-value* - *distance value*". Secondly, the information on the number of descendent nodes is provided by the *size* value. However, the BaseX scheme has drawbacks relating to updating operations. This weakness can be highlighted when adding a subtree *u* as an only child to an element *f* considering that *t* is the number of nodes in *u*. As a result, three values must be updated as follows:

1. Based on the following axis of *f*, the *pre*-values of all nodes are increased by *t*.
2. All the ancestors of *f* and its size values are increased by *t*.
3. Based on the following sibling axis of *f*, all the distance values of the nodes are increased by *t*

This proves that the node identifiers are not useful when the *pre*-value is changed after an update [40]

7.1.2 selection of datasets

We discuss our results, which were executed by applying XMark dataset [117, 118] and Shakespeare's plays dataset [145] in both the eXist database system, BaseX database system and the Pentagonal labelling scheme. We applied queries in Hamlet, one of Shakespeare's plays, which was stored as an XML ordered dataset with a document size 273KB [145] and in XMark with a document size 111MB, XMark dataset is the most common benchmark for XML data management [117, 118], it is well-known dataset and contains a scalable document database with a deep recursive ancestor structure. In addition, the decedent nodes have a wide range of fan-out nodes and a depth of 12 which have different breadths at each level with the maximum being 25,500 nodes.

7.1.3 comparison methodology

We classify the comparison methodology into non-functional and functional parameters. This section gives more information on these parameters.

The non-functional parameters, such as XML parsers helps in dealing with XML data documents, while the functional parameters such as the query performance aid in evaluating the response time by using different types of XPath queries.

The parser techniques convert the Native XML document into logical representations either as tree-based approach or as events [72, 76, 77, 146]. The most common XML parsers are Document Object Model (DOM) and Simple API for XML (SAX) [72, 73, 77, 147].

The Document Object Model parser is based on the XML tree approach, which requires the entire structure of the document to be built within the main memory, in order for it to represent XML document as a tree structure [71, 76]. The DOM parser demands memory space. This is necessary as the entire XML tree is loaded in memory where the DOM parser could be larger than the original XML document by up to 10 times [73, 148]. Loading the whole XML document into the main memory provides improved performance in terms of XPath Query retrieval, data access, modification and navigation of XML documents [76].

The SAX parser technique scans an XML document and then creates events by treating the XML document as a stream, such as start elements and end elements [77, 78]. It is suited for dealing with large documents which do not fit in the main memory [72, 77-79]. In addition, the SAX parser is best suited for extracting the content of specific elements [71]. The Non-Functional comparison is further highlighted in Section 7.2.

An example of a Functional parameter is the loading time which is a measure of the performance in terms of the time required to load the datasets. A further explanation of the functional comparison of this study is presented in section 7.3.

7.2 Non-Functional comparison: features and attributes

A non-functional comparison is provided in table 7.1.

Feature	Pentagonal	EXist	BaseX
The approach to store XML data	Native XML database Document-centric XML	Native XML database Document-centric XML	Native XML database Document-centric XML
Technology	Java	Java	Java
Query processing	XPath	XPath/XQuery	XQuery
Supported standard	Path expressions	Path expressions	Path expressions
Implementation of xpath query language (structural node relationships)	Parent-child, ancestor-descendent or previous-/next-sibling.	Parent-child, ancestor-descendent or previous-/next-sibling.	Parent-child, ancestor-descendent and previous-/next-sibling.
Document Size	small to large collections of XML documents	Small to large collections of XML documents	Small to large collections of XML documents
Xml parsers	Simple API for XML (SAX)	Document object model (DOM) To speed up query processing,	built-in, SAX parser and a DOM parser
Associated scheme or DTD	Not required	Not required	Not required

Update mechanism	Dynamic documents	Static documents	Static documents

Table 7.1. Features and attributes of eXist, BaseX database systems and Pentagonal labelling scheme. [66, 148-150].

EXist, BaseX and Pentagonal comparison: eXist, BaseX and Pentagonal are Document-centric XML which contain much-mixed content and larger sections of text.

The eXist database did not provide an advanced update mechanism as it means updating a whole tree. This is a limitation for the eXist application as these documents need a frequent update [66]. In BaseX, performing the update of nodes is expensive [40, 41], particularly when facing the worst-case scenario of updating, that can lead to relabelling of all nodes in the document [40, 41]. In our scheme, it is possible to manipulate single nodes and support for dynamic document updates. eXist and BaseX have a user-friendly GUI for both database management and query processing. We applied the SAX parser in our scheme; this is due to the improved performance of the SAX parser in terms of handle large-scale XML documents [26]. The eXist database applied the DOM parser, which demands memory space. This is due to loading the entire XML tree in memory. Since the DOM parse could be larger than the original XML document up to 10 times [73, 148]. BaseX uses different XML parsers, SAX parser, which is covered by the SAX-WRapper class, and a DOM parser covered by DOM-Wrapper class [148].

Documents in eXist, BaseX and Pentagonal scheme are not required to have an associated data type definition (DTD) or scheme [150]. The eXist, BaseX and Pentagonal Scheme automatically build indexes on the loading of XML documents [41, 150]. The supported standard in eXist and BaseX is XPath queries which allow users to query part of the document or even all the documents in the database [40, 149]. Similar to eXist and BaseX, Pentagonal scheme also uses XPath as its database query language.

7.3 Functional Comparison: Experiments based on the comparison of loading time and query response time.

The experiment measured the performance on the basis of query response time. This experiment used Shakespeare's plays dataset [8, 55, 111, 151] and the XMark dataset [117, 118]. We applied queries in Hamlet, one of Shakespeare's plays, which was stored as an XML ordered dataset with a document size 273KB and in XMark with a document size 111MB. See (Chapter 5) for detailed characteristics of the dataset.

Query	Axis name	Dataset	Example & description
Query1:	Child Axes.	XMark dataset	/site/regions/*/item - Selects all children of the current node, and all the items.
Query2:	Parent Axes.	XMark dataset	/site/regions/*/item[parent::namerica or parent::samerica] - The (North or South) American items. Element named items are the children of the world region they belong to. Retrieve all items belonging to either North or South America.
Query3:	Ancestor Axes.	XMark dataset	//keyword/ancestor::listitem - Ancestor: Selects all ancestors (parent, grandparent, etc.) of the current node.
Query4:	Child Axes.	Shakespeare's Hamlet dataset	/PLAY/*/TITLE - Selects all children of the current node, all the title.
Query5:	Parent Axes.	Shakespeare's Hamlet dataset	/PLAY/*/TITLE[parent::PERSONAE] The PERSONAE Title. Element named Title are the children of the world Play they belong to. Retrieve all Title belonging to Personae.
Query6:	Ancestor-Descendent Axes	Shakespeare's Hamlet dataset	/descendant-or-self::SPEECH - Descendant-or-self: Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself.

Table 7.2. XPath Queries for Hamlet - one of Shakespeare's plays – dataset and XMark dataset.

The XPath queries used in our experiments are defined in Table 7.2. for Shakespeare's plays dataset and XMark dataset [117, 118, 152].

7.4 Results' Analysis

In this experiment, we evaluate the eXist, BaseX databases and our scheme for the query performance by using XPath queries [152] on the dataset for Shakespeare's Hamlet and XPathMark queries [118] on the XMark dataset. In this section, we compared the query performance to an indexing scheme and labelling scheme. Each query was individually executed several times, for a minimum of 10 times [128, 129] in order to enhance the accuracy of the test. Our work was based on a personal computer with an Intel Core i7 processor, 8GB of main memory, a 64-bit Operating System, and running a Windows 10 system.

7.4.1 Load Time Performance

In this section, we described the experiments that were used to evaluate the proposed Pentagonal Dynamic Labelling scheme compared to the eXist database and the BaseX Database. The first experiment measured loading time performance. Each of the databases was separately executed several runs in order to gain considerable results and enhance the accuracy of the loading times [123]. In this work, the first three runs were omitted to validate the accuracy, the reliability of the results.

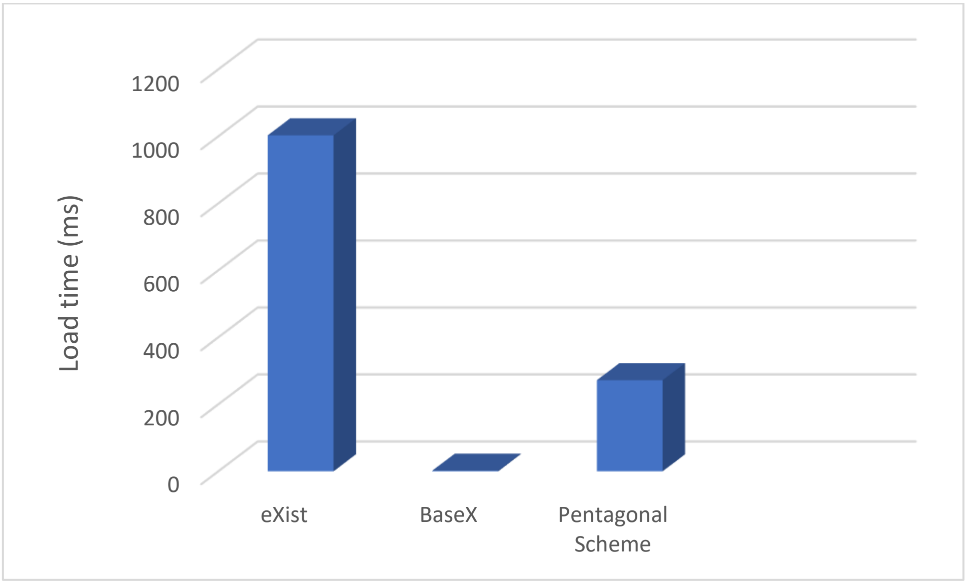


Figure 7.4 Loading time for Hamlet - one of Shakespeare's plays – dataset

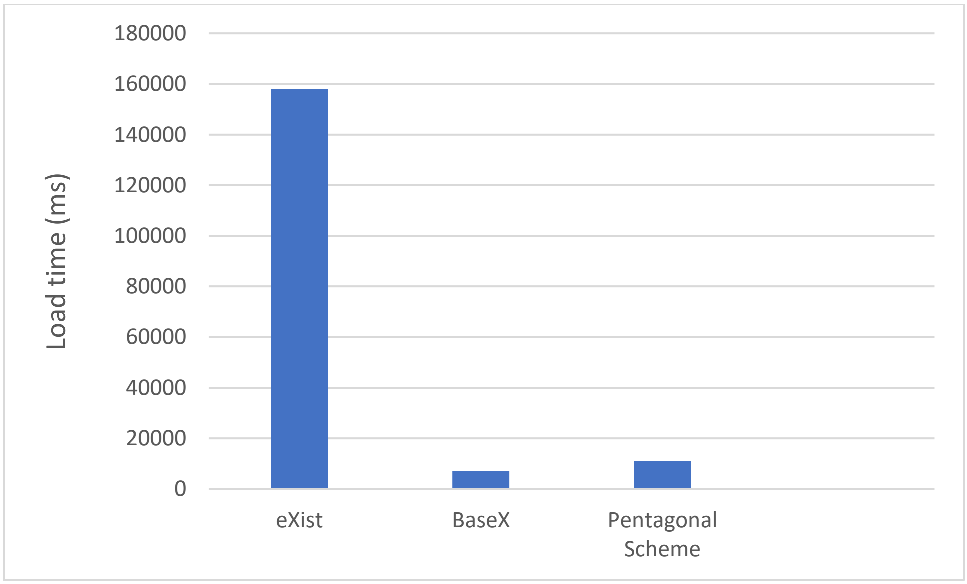


Figure 7.5 Loading time for XMark – dataset

From Figure 7.4, the BaseX database performs best in hamlet dataset; its loading time was 2 milliseconds. The Pentagonal scheme performs better than eXist in Hamlet dataset as the results have shown that for the Pentagonal scheme and eXist database when applied in Hamlet Dataset, the loading time for the Pentagonal scheme was 272 milliseconds. In comparison to eXist database in which its loading time was longer and reached 1 second and 3 milliseconds. From Figure 7.5, the BaseX and the Pentagonal scheme both perform better than eXist database in XMark Dataset as the results have shown that for the BaseX and Pentagonal scheme, when applied in XMark Datasets, their loading time was 7 and 11 seconds respectively in comparison to eXist database in which its loading time was longer and reached 2 minute and 38 seconds, consequently leading to the conclusion that the time required to load XML document in Pentagonal scheme is efficient with larger and small document sizes. In addition, it would be reasonable to expect the growth of the loading time as the document size increases. From Figure 7.5, we can identify that the time of loading the document for XMark dataset, BaseX are almost the same as the pentagonal scheme. The eXist database had the worst loading time; This is due to loading the entire XML tree in memory which can delay the loading time. eXist database applied the DOM parser, which demands large memory space as the parser can be larger than the original XML document by up to 10 times

[73, 148]. In the Pentagonal scheme, the SAX parser was applied, this is due to the improved performance of the SAX parser in terms of handling large XML documents. In addition, BaseX is applying the advantage of both the DOM and SAX parser. This can be seen as the reason for BaseX have faster execution time than the Pentagonal Scheme.

7.4.2 Query performance

In this section, we evaluate the Pentagonal scheme for the query performance compared to the eXist database and the BaseX Database, using Shakespeare’s Hamlet dataset and XMark dataset. In this experiment, we compared the query execution time of eXist database, BaseX database and Pentagonal scheme. Moreover, we executed our experiments by evaluating different XPath queries (see Table 7.2).

The queries run over XMark dataset, they represent a parent-child relationship for Q1 and Q2 and an ancestor-descendent relationship for Q3. In addition, Q4, Q5 and Q6 run over Hamlet Shakespeare's dataset, they represent a parent-child relationship for Q4 and Q5, and Q6 represents the ancestor-descendent relationship.

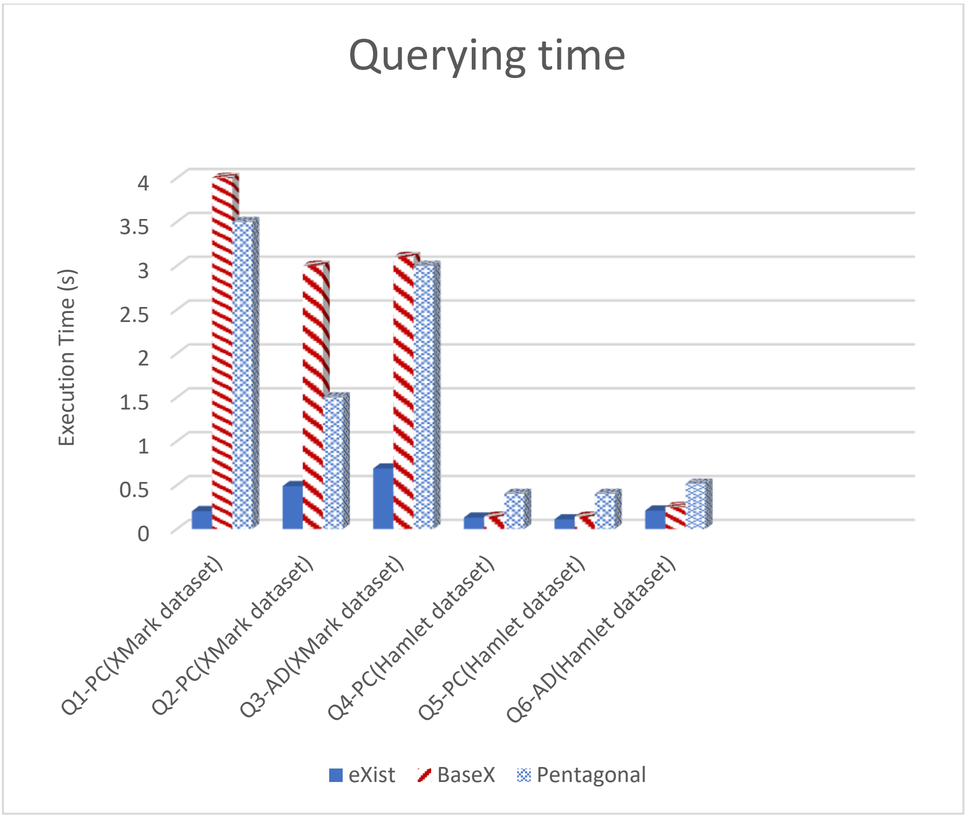


Figure 7.6. Query performance in Shakespeare's Hamlet and XMark datasets

According to the query performance in the datasets for XMark and Shakespeare's Hamlet, the eXist dataset presented a better performance in all queries. This is because determining the structural relationships using DOM is less complicated than in comparison to SAX [26]. In contrast, Pentagonal scheme showed better performance in Q1, Q2 and Q3 compared to BaseX.

Pentagonal is more efficient than the BaseX database with the parent-child and ancestor-descendent relationships which had a faster response in with XMark database. This is because the queries benefit considerably from fast access to the parent-child and ancestor-decedent nodes. This is due to the labelling storage mechanism being based on Pentagonal numbering and prefix labelling scheme, where each node is associated with one value for the node-id path from the root to the last component. In contrast, the BaseX mechanism stores a combination of values to each node based on interval labelling schemes, each node is labelled as a 3-tuple <pre,dist,size>, leading to long labels. Interval labelling scheme suffers from very long labels [26]. As a result of this restriction, the interval labelling scheme is typically not appropriate for applying with dynamic XML data [9, 13, 47, 86, 100].

Our labelling mechanism of nodes is generated based on the prefix labelling approach, which could be seen as the reason for our scheme being more efficient in determining the parent-child and the ancestor-descendent relationships. The query response time in Pentagonal scheme compared with BaseX can perform better in large size XML dataset such as XMark.

Our experiment showed that the Pentagonal scheme and BasX database are powerful in loading time. However, the loading time in the eXist database was very long compared to other schemes. In addition, Pentagonal scheme, eXist and BaseX databases are fully supported the XPath [153] specification. However, the eXist database efficiently performed in all queries. In contrast, Pentagonal scheme showed better performance in parent-child and ancestor-descendent relationships compared to BaseX in large size XML dataset such as XMark.

7.5 Conclusion

The experiment compared the ability of the Pentagonal Dynamic Labelling Scheme, the eXist database and BaseX database to handle different dataset sizes and executed for different queries. In the experiments, the tests covered 111MB and 273KB database sizes, in order to illustrate how the labelling schemes and different native XML databases deal with different XML document sizes. According to our experimental results, BaseX and the Pentagonal scheme both perform better than eXist database as the results have shown that for the BaseX and Pentagonal scheme when applied in XMark Dataset and Hamlet Shakespeare's dataset, their loading time was less in comparison to eXist database in which its loading time was longer and reached 2 minute and 38 seconds.

According to the query response time in the datasets for XMark and Shakespeare's Hamlet, Pentagonal scheme showed better behaviour in XMark dataset compared to BaseX. In contrast, the eXist dataset presented better behaviour in all queries. Our experiment showed that the Pentagonal scheme is powerful in loading and queries large document sizes. XPath queries were run over eXist, BaseX and the Pentagonal scheme; moreover, all the databases loaded the document and fully supported the XPath [153] specification.

8 CONCLUSION and FUTURE WORK

In this thesis, we reviewed state of the art labelling schemes in order to design a novel algorithm that addresses some of the limitations of existing techniques. The study aimed to achieve the target of supporting dynamic updates without relabelling nodes. This was designed to obtain a small label size and support frequent insertions without overflow problems, and to experimentally evaluate the labelling time and query performances. We have illustrated the limitations related to labelling scheme to XML documents. Therefore, the Pentagonal labelling scheme was proposed to resolve these restrictions and limitations. The experimental results and the evaluation of the proposed scheme were discussed in the previous chapters as well as the objectives and implementations.

The remainder of this chapter are organized as follows: Section 8.1 summarises the work, while section 8.2 reflects on the research questions and describes the main contributions of this research. Section 8.3 suggests future work relating to our thesis topic, while section 8.4 concludes the thesis.

8.1 Thesis Summary

This study designed a novel prefix dynamic labelling scheme, named the Pentagonal scheme. The scheme is designed to support updates in a dynamic XML tree without the need to relabel old nodes or duplicate any labels. The Pentagonal scheme generates labels based on the prefix labelling scheme.

The thesis was divided into eight chapters which are organized as follows: the first chapter introduced the research work that influenced the construction of the hypothesis, the research motivation, objectives and research aims. The second chapter provided a brief overview of XM and, the structure of XML documents, and described the storage. In addition, it described its syntax and illustrated the concepts of XML parsing techniques. The third chapter represented different labelling schemes in the literature review and discussed the structure, restriction, strengths and weaknesses of several XML labelling schemes. The fourth chapter compared the most common XML labelling schemes, namely Prefix Dewey encoding and Range-based encoding; furthermore, this chapter has already been published (Taktek, Thakker and Neagu, 2018). In the fourth chapter, different XML datasets were used that represent various features of XML trees, and several experiments were performed to investigate the storage space and labelling time requirement for each scheme. This also enabled a comparison of the relevance of the two schemes to the dataset structures. The aim was to ensure that the generation of short labels in terms of memory size and to achieve the fastest labelling time. Chapter 5 explained the proposed scheme by describing the structure of the scheme and defining the rules of the algorithms. Furthermore, from a practical perspective, this chapter illustrated the design and implementation of the proposed scheme. Several experiments were performed to evaluate the Pentagonal labelling scheme on different datasets. Also, this chapter included the experimental results in order to evaluate the reliability, scalability and performance of the proposed scheme. Chapter 6 evaluated the query performance of the Pentagonal scheme. The experiment compared the ability of the proposed scheme to investigate the queries' response times and determined the relationship between the nodes; the work in chapters 5 and 6 has been published for Elsevier Journal of Knowledge-Based Systems (Taktek, E. and Thakker, D 2020). Chapter 7 identified the concept of Native XML database systems and compared two such systems with the proposed scheme. Furthermore, the experiment compared the ability of the BaseX database, eXist database and Pentagonal dynamic labelling scheme to execute different queries and handle different datasets sizes. The next section discusses the research contributions of this thesis.

8.2 The Research Contributions of this Thesis

This section reflects on the research questions and highlights the main contributions of this thesis.

1. The Pentagonal scheme has been applied for the first time to label XML data. The storage mechanism in our scheme is based on Pentagonal numbering and prefix labelling scheme, where each node is associated with one value for the node-id path. The experiments have covered insertions of small and large numbers of nodes for a range of two to sixty thousands nodes in order to investigate the label size and the execution time. In terms of the implementation and the design of the proposed scheme, we applied the SAX parser due to its improved performance in relation

to handling large XML documents. The outcome showed that our scheme achieved the fastest labelling times in term of random skewed node insertions. In addition, our scheme leads to reduction in the storage costs by applying the get-original function using equation (3). It is also shown that The Pentagonal Scheme performs the best when a significant number of random skewed nodes is updated (for more detail, see chapter 5).

2. The Pentagonal labelling scheme handles insertions without relabelling existing nodes by providing four scenarios of inserting nodes: inserting before the leftmost node, inserting after the rightmost node, inserting between two siblings, and inserting a child into a leaf node. Our scheme was designed effectively for dynamic XML documents and completely avoids relabelling in thses four scenarios based on the mathematical equations for assigning initial labelling and handling XML updates. Unlike other labelling schemes, our scheme preserves the pentagonal numbers theorem [130] for insertions when updates occur. In inserting between two siblings, the new label is generated as the prefix label and concatenated with the new self-label. Our algorithm applies the get-original function using equation (3) for

the last component of the siblings in order to reduce the size of the label.

Equation (3) used for get-original function :

$f2 = (1 + \sqrt{1 + (24 * p)})/ 6,$ where p is a pentagonal value …..(3).

Several experimental works were carried out to ensure that the scheme efficiently deals with insertions when updates occur without duplicating labels or relabelling old nodes. As well as obtaining reduced labels.

3. We have evaluated the Pentagonal scheme's query performance and illustrated the efficiency of determining the relationships between nodes over dynamic XML documents. We applied the SAX parser due to its improved performance in relation to handling large XML documents and using different XPath Queries.

The main strength of our labelling scheme is that it is efficiently supports updates in all the cases of insertion, it performs best when a vast number of random skewed nodes has been updated. Also, it proved its capability in terms of the query performance and in determining the relationships. Our scheme also supports frequent insertions without overflow problems.

Following the work in this thesis, the main contributions are highlighted as follows:

1) The novel Pentagonal labelling scheme supports dynamic updates by avoiding relabelling.

2) The Pentagonal labelling scheme obtains a small label size and performs best when a vast number of random skewed nodes has been updated.

3) The Pentagonal labelling scheme reduces the time taken to generate labels. It has achieved the fastest labelling time when handling random skewed node insertions.

4) The Pentagonal labelling scheme supports frequent insertions without overflow problems.

5) The Pentagonal labelling scheme helps to reduce the query processing time in some cases.

6) Pentagonal labelling scheme was applied for the first time to label XML data.

7) The loading time was quicker for the Pentagonal labelling scheme than for the eXist database.

8) The Pentagonal labelling scheme showed better behaviour in the large dataset compared to the BaseX database.

All the contributions above are supported by evidence obtained experimentally, as demonstrated in the earlier chapters.

8.3 Future Work

The thesis presented a novel dynamic labelling scheme to support updates over XML data. The proposed algorithm of this thesis was based on Pentagonal theory, which labels XML data, handles insertions and query XML data.

Combination of Parsers for Optimum performance trade-off:

In terms of the implementation and the design of the proposed scheme, we applied the SAX parser due to its improved performance in relation to handling large XML documents. In contrast, other labelling schemes, such as the eXist database, applied the DOM parser, which demands large memory space and as shown in the experimental results, had longer loading time. However, the DOM parser had the advantage of better performance in all queries because determining the structural relationships using DOM is less complicated than in comparison to SAX [26]. Therefore, in order to enhance this aspect, it is possible to apply the advantage of both the DOM and SAX parser in the future. In addition, it would be a good improvement for this scheme to investigate and execute more complex queries in order to gain more inclusive results.

Application of compression method with Pentagonal scheme:

Additional investigations could be carried out experimentally to discover the effects of applying compression methods such as the Fibonacci method to our scheme [114], to see if applying them can improve our scheme , in order to get more compact representations of the labels in term of memory size.

Applying different labelling schemes:

Another future work suggestion is to redesign the proposed scheme with other labelling schemes instead of Prefix based labelling such as Interval based schemes [7, 14, 32, 39, 46, 85, 90, 91]), Multiplicative based schemes [39, 43, 49, 93-99] or Hybrid based schemes [7, 16, 47, 48, 100]. This could lead to a new scheme and may improve the efficiency of our scheme.

8.4 Conclusion

The main aim of this research was to improve the efficiency of XML data management, mainly in dynamic XML databases. This thesis proposed a novel prefix dynamic labelling scheme, called the Pentagonal scheme, and considered the restrictions of existing XML labelling schemes to overcome the challenges associated with labelling-based methods for dynamic XML data. This involved our scheme's ability to support data updates without duplicating labels or relabelling old nodes as it is important to use dynamic XML labelling schemes to avoid relabelling existing XML nodes during updates. It also includes the reduction of time and the size taken to generate the labels.

The results concluded that the Pentagonal labelling scheme achieved an efficient performance in many sectors, including the label size, label time, query processing, and structural relationship determination. Moreover, it supported dynamic updates without relabelling nodes. Finally, this chapter summarised the thesis, outlined the research contributions and explained and potential future work.

REFERENCES

1. St.Laurent, S., *Why XML*. 1998.
2. Chaudhri, A., R. Zicari, and A. Rashid, *XML data management: native XML and XML enabled DataBase systems*. 2003: Addison-Wesley Longman Publishing Co., Inc.
3. Khare, R. and A. Rifkin, *XML: A door to automated Web applications.* IEEE Internet Computing, 1997. **1**(4): p. 78-87.
4. Abiteboul, S., P. Buneman, and D. Suciu, *Data on the Web: from relations to semistructured data and XML*. 2000: Morgan Kaufmann.
5. Lloyd, C.M., M.D.B. Halstead, and P.F. Nielsen, *CellML: its future, present and past.* Progress in biophysics and molecular biology, 2004. **85**(2-3): p. 433-450.
6. Wu, X., M.-L. Lee, and W. Hsu. *A prime number labeling scheme for dynamic ordered XML trees*. 2004. IEEE.
7. O'Connor, M.F. and M. Roantree. *Desirable properties for XML update mechanisms*. 2010. ACM.
8. Li, C., T.W. Ling, and M. Hu, *Efficient updates in dynamic XML data: from binary string to quaternary string.* The VLDB Journal—The International Journal on Very Large Data Bases, 2008. **17**(3): p. 573-601.
9. Ghaleb, T.A. and S. Mohammed, *A dynamic labeling scheme based on logical operators: a support for order-sensitive XML updates.* Procedia Computer Science, 2015. **57**: p. 1211-1218.
10. He, Y., *A Novel Encoding Scheme for XML Document Update-supporting.* 2015.
11. Wang, H., et al. *ViST: a dynamic index method for querying XML data by tree structures*. 2003.
12. Alsubai, S. and S.D. North. *A prime number approach to matching an XML twig pattern including parent-child edges*. 2017. SCITEPRESS.
13. Liu, J. and X.X. Zhang, *Dynamic labeling scheme for XML updates.* Knowledge-Based Systems, 2016. **106**: p. 135-149.
14. Dietz, P.F. *Maintaining order in a linked list*. 1982.
15. Bosak, J. and T. Bray, *XML and the second-generation Web.* Scientific American, 1999. **280**(5): p. 89-93.
16. Haw, S.-C. and C.-S. Lee, *Data storage practices and query processing in XML databases: A survey.* Knowledge-Based Systems, 2011. **24**(8): p. 1317-1340.
17. Li, C., T.W. Ling, and M. Hu. *Reuse or never reuse the deleted labels in XML query processing based on labeling schemes*. 2006. Springer.
18. Khaing, A.A. and N.L. Thein. *A persistent labeling scheme for dynamic ordered XML trees*. 2006. IEEE.
19. Duong, M. and Y. Zhang. *LSDX: a new labelling scheme for dynamically updating XML data*. 2005. Australian Computer Society, Inc.

20. Lu, J., X. Meng, and T.W. Ling, *Indexing and querying XML using extended Dewey labeling scheme.* Data & Knowledge Engineering, 2011. **70**(1): p. 35-59.
21. Liu, J., Z.M. Ma, and L. Yan, *Efficient labeling scheme for dynamic XML trees.* Information Sciences, 2013. **221**: p. 338-354.
22. Xu, L., et al. *DDE: from dewey to a fully dynamic XML labeling scheme*. 2009. ACM.
23. O'Connor, M.F. and M. Roantree. *SCOOTER: a compact and scalable dynamic labeling scheme for XML updates*. 2012. Springer.
24. Wong, R.K., F. Lam, and W.M. Shui. *Querying and maintaining a compact XML storage*. 2007.
25. Zou, Q., S. Liu, and W.W. Chu. *Ctree: a compact tree for indexing XML data*. 2004.
26. Al Zadjali, H., *Compressing Labels of Dynamic XML Data using Base-9 Scheme and Fibonacci Encoding.* 2017.
27. Härder, T., et al., *Node labeling schemes for dynamic XML documents reconsidered.* Data & Knowledge Engineering, 2007. **60**(1): p. 126-149.
28. Fu, L. and X. Meng. *Triple Code: An Efficient Labeling Scheme for Query Answering in XML Data*. 2013. IEEE.
29. Cohen, E., H. Kaplan, and T. Milo, *Labeling dynamic XML trees.* SIAM Journal on Computing, 2010. **39**(5): p. 2048-2074.
30. Kay, M.H., *Ten reasons why Saxon XQuery is fast.* IEEE Data Eng. Bull., 2008. **31**(4): p. 65-74.
31. Amagasa, T., M. Yoshikawa, and S. Uemura. *QRS: A robust numbering scheme for XML documents*. 2003. IEEE.
32. Xu, L., T.W. Ling, and H. Wu, *Labeling dynamic XML documents: an order-centric approach.* IEEE transactions on knowledge and data engineering, 2012. **24**(1): p. 100-113.
33. Mirabi, M., et al., *An encoding scheme based on fractional number for querying and updating XML data.* Journal of Systems and Software, 2012. **85**(8): p. 1831-1851.
34. Tatarinov, I., et al. *Storing and querying ordered XML using a relational database system*. 2002. ACM.
35. Li, C. and T.W. Ling. *QED: a novel quaternary encoding to completely avoid re-labeling in XML updates*. 2005. ACM.
36. Chung, C.-W., J.-K. Min, and K. Shim. *APEX: An adaptive path index for XML data*. 2002. ACM.
37. Jiang, Y., et al., *An encoding and labeling scheme based on continued fraction for dynamic XML.* JSW, 2011. **6**(10): p. 2043-2049.
38. Yu, J.X., et al., *Dynamically updating XML data: numbering scheme revisited.* World Wide Web, 2005. **8**(1): p. 5-26.
39. Subramaniam, S. and S.-C. Haw. *ME labeling: A robust hybrid scheme for dynamic update in XML databases*. 2014. IEEE.

40. Kircher, L., *BaseX: Extending a native XML database with XQuery Update.* 2010.
41. Grün, C., *Storing and querying large XML instances.* 2010.
42. Al-khazraji, S. and S. North. *The emergence computation of overflow in dynamic XML tree based on prefix and interval labelling schemes*. 2017. IEEE.
43. Almelibari, A., *Labelling Dynamic XML Documents: A GroupBased Approach.* 2015.
44. O'Neil, P., et al. *ORDPATHs: insert-friendly XML node labels*. 2004. ACM.
45. Xu, L., Z. Bao, and T.W. Ling. *A dynamic labeling scheme using vectors*. 2007. Springer.
46. Subramaniam, S., S.-C. Haw, and L.-K. Soon. *Relab: a subtree based labeling scheme for efficient XML query processing*. 2014. IEEE.
47. Haw, S.-C. and A. Amin, *Node Indexing in XML Query Optimization: A Review.* Indian Journal of Science and Technology, 2015. **8**(32): p. 1-9.
48. Qin, Z., et al., *Efficient XML query and update processing using a novel prime-based middle fraction labeling scheme.* China Communications, 2017. **14**(3): p. 145-157.
49. Xiao, Y., et al. *Branch code: A labeling scheme for efficient query answering on trees*. 2012. IEEE.
50. Gou, G. and R. Chirkova, *Efficiently querying large XML data repositories: A survey.* IEEE Transactions on Knowledge and Data Engineering, 2007. **19**(10): p. 1381-1403.
51. *W3C. Extensible Markup Language (XML)*. 2016.
52. Wu, X. and D. Theodoratos, *A survey on XML streaming evaluation techniques.* The VLDB Journal, 2013. **22**(2): p. 177-202.
53. Lin, R.-R., Y.-H. Chang, and K.-M. Chao. *A compact and efficient labeling scheme for XML documents*. 2013. Springer.
54. Zhuang, C., Z. Lin, and S. Feng. *Insert-friendly XML containment labeling scheme*. 2011.
55. Ko, H.-K. and S. Lee, *A binary string approach for updates in dynamic ordered XML data.* IEEE Transactions on Knowledge and Data Engineering, 2009. **22**(4): p. 602-607.
56. Ghaleb, T.A. and S. Mohammed. *Novel scheme for labeling XML trees based on bits-masking and logical matching*. 2013. IEEE.
57. Zhou, J., et al., *Top-down XML keyword query processing.* IEEE Transactions on Knowledge and Data Engineering, 2016. **28**(5): p. 1340-1353.
58. TIDWELL, D., *Introducti on to Xml [Online].* 2002.
59. Abiteboul, S., P. Buneman, and D. Suciu, *Data on the Web: From Relational to Semistructured Data and XML.* 1999.
60. Harold, E.R., *Effective XML: 50 specific ways to improve Your XML*. 2004: Addison-Wesley Professional.
61. Potok, T.E., et al. *An ontology-based HTML to XML conversion using intelligent agents*. 2002. IEEE.

62. Reis, D.d.C., et al. *Automatic web news extraction using tree edit distance*. 2004.
63. Klaib, A. and J. Lu. *Investigation into indexing XML data techniques*. 2014. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).
64. Nambiar, U., et al. *Efficient XML data management: an analysis*. 2002. Springer.
65. Florescu, D. and D. Kossmann, *Storing and querying XML data using an RDMBS.* IEEE data engineering bulletin, 1999. **22**: p. 3.
66. Meier, W. *eXist: An open source native XML database*. 2002. Springer.
67. Hall, D. and L. Strömbäck. *Generation of synthetic XML for evaluation of hybrid XML systems*. 2010. Springer.
68. Bertino, E., et al., *Specifying and enforcing access control policies for XML document sources.* World Wide Web, 2000. **3**(3): p. 139-151.
69. Deutsch, A., et al., *Querying XML data.* IEEE Data Eng. Bull., 1999. **22**(3): p. 10-18.
70. Milo, T. and S. Zohar. *Using schema matching to simplify heterogeneous data translation*. 1998. Citeseer.
71. Roy, J. and A. Ramanujan, *XML schema language: taking XML to the next level.* IT professional, 2001. **3**(2): p. 37-40.
72. Haw, S.C. and G.S.V.R.K. Rao. *A comparative study and benchmarking on xml parsers*. 2007. IEEE.
73. Kiselyov, O. *A better XML parser through functional programming*. 2002. Springer.
74. Tong, T., et al., *Rules about XML in XML.* Expert Systems with Applications, 2006. **30**(2): p. 397-411.
75. Lam, T.C., J.J. Ding, and J.-C. Liu, *XML document parsing: Operational and performance characteristics.* Computer, 2008. **41**(9): p. 30-37.
76. Wang, F., J. Li, and H. Homayounfar, *A space efficient XML DOM parser.* Data & Knowledge Engineering, 2007. **60**(1): p. 185-207.
77. Nicola, M. and J. John. *Xml parsing: a threat to database performance*. 2003.
78. Pan, Y., Y. Zhang, and K. Chiu. *Hybrid parallelism for XML SAX parsing*. 2008. IEEE.
79. Takase, T., et al. *An adaptive, fast, and safe XML parser based on byte sequences memorization*. 2005.
80. Ahn, J., et al., *A dynamic and parallel approach for repetitive prime labeling of XML with MapReduce.* The Journal of Supercomputing, 2017. **73**(2): p. 810-836.
81. Zhuang, C. and S. Feng. *Full tree-based encoding technique for dynamic XML labeling schemes*. 2012. Springer.
82. Li, C. and T.W. Ling. *An improved prefix labeling scheme: a binary string approach for dynamic ordered XML*. 2005. Springer.
83. Paramasivam, J. and T. Angamuthu, *A New Method of Generating Index Label for Dynamic XML Data.* Journal of Computer Science, 2011. **7**(3): p. 421.

84. O'Connor, M.F. and M. Roantree. *FibLSS: a scalable label storage scheme for dynamic XML updates*. 2013. Springer.
85. Li, Q. and B. Moon. *Indexing and querying XML data for regular path expressions*. 2001.
86. Kaplan, H., T. Milo, and R. Shabo, *A comparison of labeling schemes for ancestor queries*. 2002, ACM-SIAM. p. 954–963.
87. Mlynková, I. *An analysis of approaches to xml schema inference*. 2008. IEEE.
88. Zadjali, H. and S.D. North. *XML Labels Compression using Prefix-encodings*. 2016. SCITEPRESS, Science and Technology Publications.
89. Johnson, J.R., et al. *Extracting semantic information structures from free text law enforcement data*. 2012. IEEE.
90. Tahraoui, M.A., et al., *A survey on tree matching and XML retrieval.* Computer Science Review, 2013. **8**: p. 1-23.
91. Al-khazraji, S. and S. North. *A relevance comparison between interval and prefix labelling schemes*. 2017. IEEE.
92. Sans, V. and D. Laurent, *Prefix based numbering schemes for XML: techniques, applications and performances.* Proceedings of the VLDB Endowment, 2008. **1**(2): p. 1564-1573.
93. Weigel, F., K.U. Schulz, and H. Meuss. *The BIRD numbering scheme for XML and tree databases–deciding and reconstructing tree relations using efficient arithmetic operations*. 2005. Springer.
94. An, D. and S. Park. *Group-Based Prime Number Labeling Scheme for XML Data*. 2010. IEEE.
95. Jayanthi, P. *Vector based labeling method for dynamic XML documents*. 2013. IEEE.
96. Lee, Y.K., et al. *Index structures for structured documents*.
97. Lee, Y.K., et al. *Index structures for structured documents*. 1996.
98. Kha, D.D., M. Yoshikawa, and S. Uemura. *A structural numbering scheme for XML data*. 2002. Springer.
99. Al-Shaikh, R., et al. *A modulo-based labeling scheme for dynamically ordered XML trees*. 2010. IEEE.
100. Su-Cheng, H. and L. Chien-Sing, *Node labeling schemes in XML query optimization: a survey and trends.* IETE Technical Review, 2009. **26**(2): p. 88-100.
101. Liu, J., Z.M. Ma, and Q. Qv, *Dynamically querying possibilistic XML data.* Information Sciences, 2014. **261**: p. 70-88.
102. Duong, M. and Y. Zhang. *Dynamic labelling scheme for xml data processing*. 2008. Springer.
103. S, N.E.T. and P. Jayanthi. *Vector based labeling method for dynamic XML documents*. 2013. IEEE.
104. Ali Klaib, A., *Clustering-based Labelling Scheme-A Hybrid Approach for Efficient Querying and Updating XML Documents.* 2018.

105. Azzedin, F., et al., *Systematic partitioning and labeling XML subtrees for efficient processing of XML queries in IoT environments.* IEEE Access, 2020. **8**: p. 61817-61833.
106. Hsu, W.-C. and I.E. Liao, *UCIS-X: An Updatable Compact Indexing Scheme for Efficient Extensible Markup Language Document Updating and Query Evaluation.* IEEE Access, 2020. **8**: p. 176375-176392.
107. Liu, J., et al., *Enabling massive XML-based biological data management in HBase.* IEEE/ACM transactions on computational biology and bioinformatics, 2019.
108. Hao, W., K. Matsuzaki, and S. Sato. *A Dual-Index Based Representation for Processing XPath Queries on Very Large XML Documents*. 2021. Springer International Publishing.
109. Li, J., et al., *XML keyword search with promising result type recommendations.* World wide web, 2014. **17**(1): p. 127-159.
110. Zeng, Y., Z. Bao, and T.W. Ling. *Supporting range queries in XML keyword search*. 2013.
111. Yun, J.-H. and C.-W. Chung, *Dynamic interval-based labeling scheme for efficient XML query and update processing.* Journal of Systems and Software, 2008. **81**(1): p. 56-70.
112. Chiew, W.S., et al., *Labeling schemes for XML dynamic updates: A survey and open discussions.* E-Commerce, E-Business and E-Service, 2014: p. 79-83.
113. Assefa, B.G. and B. Ergenc. *OrderBased labeling scheme for dynamic XML query processing*. 2012. Springer.
114. Khanjari, E. and L. Gaeini, *A new effective method for labeling dynamic XML data.* Journal of Big Data, 2018. **5**(1): p. 1-17.
115. Miklau, G., *Xml Data Repository Http://Www.Cs.Washington. Edu/Research/Xmldatasets/ [Online]. [Accessed August 2018].* 2015.
116. Al-Badawi, M., *A Performance Evaluation of a New Bitmap-based XML Processing Approach.* 2010.
117. Schmidt, A., et al. *XMark: A benchmark for XML data management*. 2002. Elsevier.
118. Franceschet, M. *XPathMark: an XPath benchmark for the XMark generated data*. 2005. Springer.
119. Boag, S., et al., *XML path language (XPath) 2.0.* W3C, W3C Recommendation, Jan, 2007.
120. Arroyuelo, D., et al., *Fast in-memory XPath search using compressed indexes.* Software: Practice and Experience, 2015. **45**(3): p. 399-434.
121. Benedikt, M. and J. Cheney, *Schema-based independence analysis for XML updates.* Proceedings of the VLDB Endowment, 2009. **2**(1): p. 61-72.
122. Genevès, P. and N. Layaïda, *A system for the static analysis of XPath.* ACM Transactions on Information Systems (TOIS), 2006. **24**(4): p. 475-502.
123. Böttcher, S. and R. Steinmetz. *Evaluating xpath queries on XML data streams*. 2007. Springer.

124. Taktek, E., D. Thakker, and D. Neagu. *Comparison between Range-based and Prefix Dewey Encoding*. 2018.
125. Kobayashi, K., et al. *VLEI code: An efficient labeling method for handling XML documents in an RDB*. 2005. IEEE.
126. Zhang, C., et al. *On supporting containment queries in relational database management systems*. 2001. ACM.
127. Yergeau, F., *UTF-8, a transformation format of ISO 10646*. 2003.
128. Ali, S., et al., *A systematic review of the application and empirical investigation of search-based test case generation.* IEEE Transactions on Software Engineering, 2009. **36**(6): p. 742-762.
129. Wegener, J., A. Baresel, and H. Sthamer, *Evolutionary test environment for automatic structural testing.* Information and software technology, 2001. **43**(14): p. 841-854.
130. Andrews, G.E., *Euler's pentagonal number theorem.* Mathematics Magazine, 1983. **56**(5): p. 279-284.
131. Leung, H.-H., *On a generalization of the Pentagonal Number Theorem.* arXiv preprint arXiv:1809.00316, 2018.
132. Arcuri, A. and L. Briand, *A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering.* Software Testing, Verification and Reliability, 2014. **24**(3): p. 219-250.
133. Al-Khalifa, S., et al. *Structural joins: A primitive for efficient XML query pattern matching*. 2002. IEEE.
134. Chien, S.-Y., et al. *Efficient structural joins on indexed XML documents*. 2002. Elsevier.
135. Gottlob, G., C. Koch, and R. Pichler, *Efficient algorithms for processing XPath queries.* ACM Transactions on Database Systems (TODS), 2005. **30**(2): p. 444-491.
136. Min, J.-K., J. Lee, and C.-W. Chung, *An efficient XML encoding and labeling method for query processing and updating on dynamic XML data.* Journal of Systems and Software, 2009. **82**(3): p. 503-515.
137. Lu, J. and T.W. Ling. *Labeling and querying dynamic XML trees*. 2004. Springer.
138. Lu, J., et al. *From region encoding to extended dewey: On efficient processing of XML twig pattern matching*. 2005. VLDB Endowment.
139. Nachar, N., *The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution.* Tutorials in quantitative Methods for Psychology, 2008. **4**(1): p. 13-20.
140. Grün, C. *BaseX – The XML Database for Processing, Querying andVisualizing large XML data.*; Available from: http://basex.org.
141. Bourret, R., *XML and Databases.* 1999.
142. Meier, W. *eXist: An Open Source Native XML Database*. Available from: http://exist-db.org/exist/apps/homepage/index.html.
143. Grün, C., et al. *XQuery full text implementation in BaseX*. 2009. Springer.

144. Grün, C., A. Holupirek, and M.H. Scholl, *Visually exploring and querying XML with BaseX*. 2007.
145. Bosak, J. *XML markup of Shakespeare's plays*. January 1998.; Available from: http://ibiblio.org/pub/sun-info/standards/xml/eg/.
146. Ck, A.R. and J. Jayanthi. *A Generic Parser to parse and reconfigure XML files*. 2011. IEEE.
147. Lu, W., K. Chiu, and Y. Pan. *A parallel approach to XML parsing*. 2006. IEEE.
148. Shadura, R., *Input and Output with XQuery and XML Databases.* 2012.
149. Kolár, P. and P. Loupal, *Comparison of native XML databases and experimenting with INEX.* Paper in Electronic Proceedings (CD-ROM or web) in DATESO, 2006: p. 116-119.
150. Mabanza, N., J. Chadwick, and G. Rao. *Performance evaluation of open source native xml databases-a case study*. 2006. IEEE.
151. Li, C., T.W. Ling, and M. Hu. *Efficient processing of updates in dynamic XML data*. 2006a. IEEE.
152. MCHUGH, J., *https://www.nuwavesolutions.com/xpath-knime/*. 2019.
153. *W3C. XML Path Language (XPath) 2.0 (Second Edition)*. 2010; Updated October 2016; Available from: http: //www.w3.org/TR/xpath20/.