The University Of Sheffield.

Department of Computer Science

Compressing Labels of Dynamic XML Data using Base-9 Scheme and Fibonacci Encoding

Submitted for the degree of Doctor of Philosophy
(PhD Thesis)

By: Hanaa AbdulWahab Al-Zadjali

October 2017

Supervisor: Dr Siobhán North

بسم الله الرحمن الرحيم

Abstract

The flexibility and self-describing nature of XML has made it the most common mark-up language used for data representation over the Web. XML data is naturally modelled as a tree, where the structural tree information can be encoded into labels via XML labelling scheme in order to permit answers to queries without the need to access original XML files. As the transmission of XML data over the Internet has become vibrant, it has also become necessary to have an XML labelling scheme that supports dynamic XML data. For a large-scale and frequently updated XML document, existing dynamic XML labelling schemes still suffer from high growth rates in terms of their label size, which can result in overflow problems and/or ambiguous data/query retrievals.

This thesis considers the compression of XML labels. A novel XML labelling scheme, named “Base-9”, has been developed to generate labels that are as compact as possible and yet provide efficient support for queries to both static and dynamic XML data. A Fibonacci prefix-encoding method has been used for the first time to store Base-9’s XML labels in a compressed format, with the intention of minimising the storage space without degrading XML querying performance. The thesis also investigates the compression of XML labels using various existing prefix-encoding methods. This investigation has resulted in the proposal of a novel prefix-encoding method named “Elias-Fibonacci of order 3”, which has achieved the fastest encoding time of all prefix-encoding methods studied in this thesis, whereas Fibonacci encoding was found to require the minimum storage.

Unlike current XML labelling schemes, the new Base-9 labelling scheme ensures the generation of short labels even after large, frequent, skewed insertions. The advantages of such short labels as those generated by the combination of applying the Base-9 scheme and the use of Fibonacci encoding in terms of storing, updating, retrieving and querying XML data are supported by the experimental results reported herein.

Declaration

I hereby declare that the configuration of this thesis is entirely the result of my own independent work/investigation, except where clearly specified otherwise. I assert that this work has not been submitted for any other degree or professional qualification, except as stated.

Hanaa AbdulaWahab Al-Zadjali

Acknowledgments

At first, I praise Allah for endowing me with the opportunity to strive for a PhD degree, and giving me the perseverance, courage and strength to complete this research. I am very grateful to Allah, who guided these wonderful people to my PhD journey.

My supervisor, Dr Siobhan North, to whom I express my deepest gratitude for her time, extraordinary support, priceless advice, and boundless encouragement. She is truly an excellent supervisor and amazingly incomparable person.

I dedicate this thesis and my success to my parents, who have always motivated me to proceed further my education with a higher degree. I am honoured to grant my mother her wish by pursuing this degree. Her unconditional love and continuance prayers for me was what sustained me this far. Regretfully, I was not able to accomplish this work in my father lifetime.

I am extremely grateful to my beloved husband (Dr. Tariq) for his superlative understanding and great help with the kids. He has been a constant source of support throughout the challenges of this journey. I am fortunate for having such person in my life, who has encouraged me to work hard for what I aspire to achieve.

There is no word to express my gratitude to my lovely children (Ziyad, Sundus, and the twin Emran and Abdul Rahman) for being patience and considerate despite their young age. They have been and continues to be my inspiration.

Respecting my children desire for preferring to stay-at-home was the most difficult decision I ever encountered. And so, endless thanks to my fantastic sisters (Zahiya and Heba) and my nieces (Amira and Asmaa) for their unwavering support and incredible assistance with the kids in my absence over the last three years. Thanks a lot to my siblings (Nada and Mariam) and other family members for their prayers and reassurance during my difficult times. I am also indebted to Ghazala Husain who taught me how a stranger can be more than a sister. I am as well appreciative to my true friend Sally for the excellent discussant and for always being there for me.

Finally, I am obliged to Sultan Qaboos University, who offered me this chance to complete my academic studies.

Table of Content

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

Managing web-based information has become fundamental to keep up with the accelerating rate of expansion of the internet. As a result, the XML (eXtensible Mark-up Language) (Bray et al., 1998) has become a standard for data representation and exchange on the web (W3Schools., 2016b) (Abiteboul et al., 2000) (Ahn et al., 2017a) (Mathis et al., 2015) (Choi et al., 2014) (Thimma et al., 2013) (Assefa and Ergenc, 2012) (Luo et al., 2009) (He, 2015) (Tatarinov et al., 2002) (Ghaleb and Mohammed, 2013) (Qin et al., 2017). Extensive research has been carried out to improve the efficiency of storing, managing, updating and querying XML data (Liu and Zhang, 2016) (Agreste et al., 2014) (Assefa and Ergenc, 2012) (Tatarinov et al., 2001) (Ghaleb and Mohammed, 2013). Essential to querying XML data competently and rigorously is the use of an efficient XML labelling technique. This thesis investigates the limitations of existing XML labelling schemes and proposes a new labelling scheme to enhance the effectiveness of XML data management particularly in dynamic XML environments.

To emphasise the need for the current research on the improvement of XML performance, it is important to first highlight the significance of XML datasets and XML labelling schemes. These issues are discussed in Sections 1.2 and 1.3, respectively. Section 1.4 presents the research motivation and hypothesis. The structure of the thesis is outlined in Section 1.5. A list of published work is presented in Section 1.6, before concluding the chapter with Section 1.7.

1.2 Importance of XML

XML (eXtensible Mark-up Language) (Bray et al., 1998) has emerged as a standard for data representation and exchange on the web and in a wide variety of fields, such as technical data, science, finance, business, healthcare, manufacture and astronomical data (Beech, 2016) (Abiteboul et al., 2000) (Trippe and Waldt, 2008) (Chaudhri et al., 2003) (Connolly and Begg, 2005).

Initially, the development of XML was intended to assist web designers. Nowadays, database developers, document managers and publishers, scientists and other researchers employ XML to manage their data (St.Laurent, 1998). XML data

management has been applied in many contexts, from bioinformatics, geographical and engineering data to customer services and cash flow progress through distributed systems and inductive databases (Chaudhri et al., 2003). This is because XML data is self-describing. It simplifies the publication of electronic data by providing a simple format for data that is both human and machine understandable and legible (Khare and Rifkin, 1997) (Abiteboul et al., 2000) (Lloyd et al., 2004).

The dramatic increase in the popularity of XML is driven by its extensible, flexible, and standardised properties that makes it possible to address, and overcome, the restrictions of other mark-up languages such as HTML (Sonawane and Rao, 2015) (Chaudhri et al., 2003) (Seligman and Roenthal, 2001) (Barillot and Achard, 2000). XML offers developers the ability to set standards by defining the content of a document separately from its formatting, which makes it easy to reuse and share information in other applications and in different environments/organisations (St.Laurent, 1998) (W3C, 2016b). In general, there are many benefits to using XML (Mohr et al., 2000) (Trippe and Waldt, 2008) (Seligman and Roenthal, 2001) (Tidwell, 2002) (St.Laurent, 1998), including:

- **Simplicity**: the basic syntax of XML provides a friendly environment for programmers, database' developers and document authors. The information encoded within XML data is easy to read for both humans and machines.
- **Extensibility**: XML allows for the creation of extensible tag sets that can be employed in multiple applications. It also permits the fundamental XML set of capabilities to be extended by any standards to add styles, linking, and/or referencing ability.
- **Openness**: XML standards are open access and freely available on the web.
- **Individuality**: within XML, content is separated from presentation as XML tags describe content only. The data format can be handled by XSL stylesheets (W3C, 2016a). This supports multiple views of the same content, as well allowing the look of a document to be changed without affecting its content.
- **Interoperability**: as XML supports multilingual documents and Unicode, it can be interpreted though a wide range of tools and used on various platforms.
- **Ability to embed multiple data types:** XML data can comprise any possible type of data from multimedia data (e.g. image, video, and sound) to active

components (e.g., Java applets) and complex information (e.g., biological systems and living organisms (Chaudhri et al., 2003)).

1.3 Importance of XML Labelling Schemes

The increasing significance of XML data management has intensified research work focusing on XML storage, retrieval and querying (Liu and Zhang, 2016) (Agreste et al., 2014) (Assefa and Ergenc, 2012). To query XML data competently and accurately, several XML labelling schemes have recently been introduced.

Usually, the tree representation of XML documents and queries is used to process XML data implicitly or explicitly (Tahraoui et al., 2013) (Bressan et al., 2001) (Alghamdi et al., 2014) (Shnaiderman and Shmueli, 2015). XML labelling schemes basically facilitate XML query processing by assigning a unique label to identify XML tree nodes, as based on the structure of the XML document (O'Connor and Roantree, 2010a) (Li et al., 2008) (Ghaleb and Mohammed, 2015) (He, 2015) (Wang et al., 2003). The labelling approach is implemented such that the structural relationships between nodes can be efficiently determined by comparing their labels so that XML queries can be processed without the need to access the original data (Liu et al., 2013) (Zhuang et al., 2011) (Xu et al., 2009) (Fraigniaud and Korman, 2016). Hence, XML labelling scheme provides more flexibility and require less storage space in comparison to other XML querying techniques (Haw and Lee, 2011) (Li et al., 2006b) (Khaing and Ni Lar, 2006) (Duong and Zhang, 2005). Moreover, the XML data indexing process used for querying, keyword searching, and/or data retrieval purposes relies on XML labelling schemes (Johnson et al., 2012) (Lu et al., 2011b).

As XML warehouses over the web become more extensive, XML labelling schemes need to be dynamic such that an XML update can be permitted without causing re-labelling of existing nodes (Liu and Zhang, 2016) (Lizhen and Xiaofeng, 2013) (Härder et al., 2007). Labelling schemes should also generate compact labels, i.e., as small as possible. They should efficiently support all kinds of structural relationship queries. The identification of any structural relationship between any pair of nodes should be easy to establish through the direct examination of their labels.

1.4 Research Motivation and Hypothesis

Due to its flexibility and simplicity, the growing popularity of XML as a standard for data exchange has led to the extensive use of XML data updates (Liu and Zhang, 2016) (Tekli and Chbeir, 2012) (Tatarinov et al., 2001). Thus, it has become

necessary for an XML labelling scheme to support dynamic XML data (O'Connor and Roantree, 2010a) (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005).

Many researchers have studied dynamic XML labelling schemes (Xu et al., 2009) (Liu et al., 2014) (Duong and Zhang, 2008) (He, 2015) (Ghaleb and Mohammed, 2015) (Fraigniaud and Korman, 2016) (Subramaniam et al., 2014a) (Ren et al., 2006) (Liu and Zhang, 2016), but each of the existing schemes is limited in one way or another. For the most part, it is the update of XML data that remains a weakness in most of these XML labelling schemes due to their large label sizes (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005). As can be seen from the literature review presented in Chapter 3, almost all of the existing labelling schemes have ignored the issue of size when generating XML labels. Consequently, current XML labelling schemes still suffer from huge label sizes that can result in overflow problems in the case of frequent insertions. This is ultimately because of the limitations in the design of labelling algorithms when handling insertions (see Chapter 3) as well as how the labels are encoded within main memory (discussed in Chapter 4).

The current enormous growth in data has inspired the need for compression (Lohrey et al., 2012). Motivated by this, this thesis aims to improve the efficiency of XML labelling by introducing a new XML labelling scheme which will focus on the size of XML labels. These labels will be stored in compressed form using a prefix Fibonacci encoding method. As Fibonacci encoding allows for fast decoding and short labels, the efficiency of XML query processing can be dramatically increased.

In line with the above research motivation, the research hypothesis can be stated as follows:

"Providing compact XML labels based on lexicographical order using decimal strings may facilitate query performance and permit multiple insertions without causing any storage overhead. Storing such labels using a Fibonacci prefix-encoding techniques may reduce the storage capacity required and speed up the determination of structural relationships."

1.5 Thesis Structure

This section outlines the structure of the thesis. The discussion in this thesis is divided into three main parts. The first part, Chapters 1 to 4, introduces the research and

presents the related literature from which the research hypothesis emerged. The second part, Chapters 5 and 6, identifies the research problems and objectives and details the main concepts of this research work in both a theoretical and practical sense. The last part covers the experimental results, evaluation, and the thesis concludes in Chapters 7 to 9. The following gives a brief description of each chapter:

Chapter 1: Introduction: this chapter introduces the current research work, motivation and hypothesis, and the thesis structure.

Chapter 2: Background on XML data: this chapter provides a general overview of XML data and its basic concepts and related techniques, such as XML syntax, structures, parsers, and querying approaches.

Chapter 3: Literature on XML labelling schemes: this chapter illustrates the concepts, mechanisms, and the strengths and weaknesses of existing XML labelling schemes.

Chapter 4: Literature on encoding methods: this chapter presents the state-of-the-art in existing encoding techniques used to store XML labels. The chapter also discusses several available prefix-encoding methods that can be used to compress large XML labels.

Chapter 5: Base-9 labelling scheme for dynamic XML data: this chapter identifies the research problems and goals based on the literature presented in the previous three chapters. It introduces the research hypothesis as a possible solution. The chapter also describes the underlying principles, structure, and labelling algorithms of the proposed scheme (named “Base-9”) and describes how Fibonacci encoding is employed to compress/decompress and store the Base-9 XML labels generated.

Chapter 6: Experimental design and implementation: this chapter explains the design and implementation of the Base-9 scheme based on its description in Chapter 5. To justify the effectiveness of the scheme, seven different experiments are described in this chapter. The experimental objectives, setup, datasets used are also described.

Chapter 7: Experimental results and statistical analysis: this chapter shows the results of the experiments described in Chapter 6. The results are analysed statistically and presented graphically to assess the Base-9 scheme’s performance, reliability and scalability.

Chapter 8: Evaluation and further discussion: this chapter evaluates the reliability of the experimental designs and results. It provides further discussion to evaluate the proposed scheme's properties as a dynamic XML labelling scheme based on the results obtained. The chapter also highlights the main findings and limitations of this research.

Chapter 9: Conclusion and future work: this chapter summarises the thesis. It highlights the main findings of this research, its contributions to the literature, and suggestions for future work.

1.6 Publication

Some of the contents of this thesis were presented in a conference and published as follows:

AL-ZADJALI, H. & NORTH, S. 2016, "XML Labels Compression using Prefix-encodings". *In proceedings of the 12th International Conference on Web Information Systems and Technologies, (WEBIST2016)*, ISBN 978-989-758-186-1, volume 1, pages 69-75, Rome, Italy. 69-75.

1.7 Conclusion

This chapter has given an introduction to the thesis, including a short overview of XML as the main scope of the research. The motivation and hypothesis underlying this thesis are also stated in this chapter. In general, this thesis will focus on improving the efficiency of XML labelling in dynamic environments by providing compressed XML labels. Finally, the structure of the thesis was presented.

Chapter 2: Background on XML Data

2.1 Introduction

The importance of managing web-based information has become essential to keep up with the accelerating rate of expansion of the internet. This has resulted in the development of XML (eXtensible Mark-up Language) (Bray et al., 1998) as a standard for data representation and exchange on the Web (W3Schools., 2016b) (Tatarinov et al., 2002) (Ghaleb and Mohammed, 2013) (Qin et al., 2017). Consequently, there has been extensive research into improving the efficiency of storing, managing, updating and querying XML data (Liu and Zhang, 2016) (Agreste et al., 2014) (Assefa and Ergenc, 2012) (Tatarinov et al., 2001) (Ghaleb and Mohammed, 2013).

Matching structural queries within XML documents is the core of the information retrieval function for XML data. In many XML database management systems, an XML labelling scheme has been recommended for rapid query processing of massive XML documents (Ahn et al., 2017a) (Zhuang and Feng, 2012a) (Xu et al., 2009) (Li and Ling, 2005a). This is because an XML document is naturally modelled as a tree, and labelling schemes encode the structural tree information so as to permit answers to some queries without having to access the original XML file (Lin et al., 2013) (Zhuang et al., 2011) (Hye-Kyeong and SangKeun, 2010) (Yu et al., 2005) (Ghaleb and Mohammed, 2013) (Zhou et al., 2016).

This chapter provides a background to XML starting with a general overview of XML in Section 2.2, followed by a general review of XML structure and storage in Section 2.3. A description of XML syntax as a mark-up language is provided in Section 2.4, including its main components:- of XML elements and attributes. Since much of today's Web content is written in XML, either as well-formed or valid XML documents (Grijzenhout and Marx, 2013), this section also differentiates between the two types of XML data via a discussion of DTD and XML schema. Section 2.5 then presents the most common XML parser in the literature used to determine the validity of an XML document.

The usual representation of an XML document as a tree structure is explained in Section 2.6, along with the main structural relationships amongst XML elements, i.e., parent-child, ancestor-descendent, siblings, LCA, and document order. Determining

such structural relationships between nodes plays a fundamental role in querying XML (see Section 2.7). Several works have investigated means of improving XML query efficiency, either by using XML query languages as illustrated in Section 2.8, or by considering the hierarchical structure of XML data, such as in structural indexing (Section 2.9) and XML labelling schemes (Chapter 3). Finally, the chapter is summarised in Section 2.10.

2.2 XML: An Overview

In today's world, almost all information is moved online, mostly in unstructured text data format (Bertino et al., 2000) (Subramaniam et al., 2014a) (Sheng et al., 2011). As the size and complexity of web sites grows, the need to retrieve, display, manipulate, transfer and exchange information has similarly increased (Zisman, 2000) (Lu, 2013) (Subramaniam et al., 2014a). To accomplish this, HTML (Hyper Text Mark-up Language) (Graham, 1995) (W3Schools., 2016a) and XML have become standard representations of data delivered over the Web.

While HTML provides a standard to create, display and access web pages, it is merely a visual layer, and does not provide any mechanism for describing content, or managing remote data. Furthermore, there is no information in an HTML tag that enables other systems to recognise the structure and content of the data (Bertino et al., 2000) (Ciancarini et al., 1998) (Potok et al., 2002) (Reis et al., 2004) (Sonawane and Rao, 2015). To address this weaknesses in HTML, XML was developed in 1996 by the XML working group (previously known as SGML1 Editorial Review Board), which was formed through the sponsorship of the World Wide Web Consortium, W3C (W3C, 2016b).

XML is a self-describing language that separates content from formatting and describes the structure of the text within a document by the use of start and end tags. It also permits users to design their own tags, which makes it highly flexible (Tidwell, 2002). Because of its simplicity, flexibility and scalability, XML has become a commonly used technology for information representation, data transformation, data exchange, and retrieval over the Web (He, 2015) (Subramaniam and Haw, 2014b) (Klaib and Lu, 2014) (Xu et al., 2012) (Haw and Lee, 2011) (Xu et al., 2009) (Li et al., 2008).

2.3 XML Structure and Storage

XML structure may vary between a flat, regular, data-centric structure to a more complicated, irregular, document-centric structure (Fuhr et al., 2001) (Haw and Lee, 2011) (Chiew et al., 2014b) (Nambiar et al., 2002). This comprehensive range of structural variety makes XML the most commonly used representation for all types of data (Haw and Lee, 2011). There are two main approaches to storing XML data: either as an XML Enabled Database (XED) or as a Native XML Database (NXD). A hybrid has also been proposed (Hall and Strömbäck, 2010) (Haw and Lee, 2011) that stores XML data using a mapping-to-relations technique, as in XED, that allows for the storage of XML sub-trees in its native NXD format.

An XML-enabled database (XED): is normally used to store data-centric documents that involve well-structured information, and uses XML to transfer data into a traditional relational database (Nambiar et al., 2002) (Florescu and Kossmann, 1999), Object-Oriented database (Banerjee et al., 2000) or Object-Relational database (Klettke and Meyer, 2000). This type of database includes extensions for transferring data between XML documents and the data structures of their underlying relational database storage (Younas et al., 2008). Therefore, in XED querying an XML document relies on the query engine within the underlying storage.

A native XML database (NXD): is often used to store document-centric XML (i.e., a semi-structured XML document as a whole rather than separating out the data within) (Meier, 2002). Since data in NXD are stored and retrieved in their original hierarchical structure, the research study in this thesis will focus on facilitating an XML query process that relies on a native XML database only (the terms XML and native XML are used interchangeably in this thesis).

Data is stored for the purpose of being retrieved, and as the amount of data has expanded in the Internet, helping users to find the required data quickly from a large-scale XML document has become a particularly significant issue. To establish a clear understanding of the XML querying process, a description of XML syntax is given in the following section.

2.4 XML Basic Syntax

An XML database can be defined as an application profile that describes a class of data objects (named XML documents) and how computer programs can process them, such as with the behaviour of the XML processor (Microsoft, 2016). An XML document is a case-sensitive text file with a nested logical and physical structure (Bray et al., 2008) (Kurtev, 2001) (Sall, 2002) (Bray et al., 1998) (Bertino et al., 2000) (Qadah, 2016). Its physical structure consists of storage units called “entities”; each entity may itself refer to other entities within an XML document. Every XML document must start with a document entity, the “root”, that serves as the main storage unit (Sall, 2002) (Bray et al., 2008). The logical structure of an XML document comprises declarations, comments, elements, and attributes, collectively known as the mark-up. Figure 2.1 shows an XML sample of a “School” database.

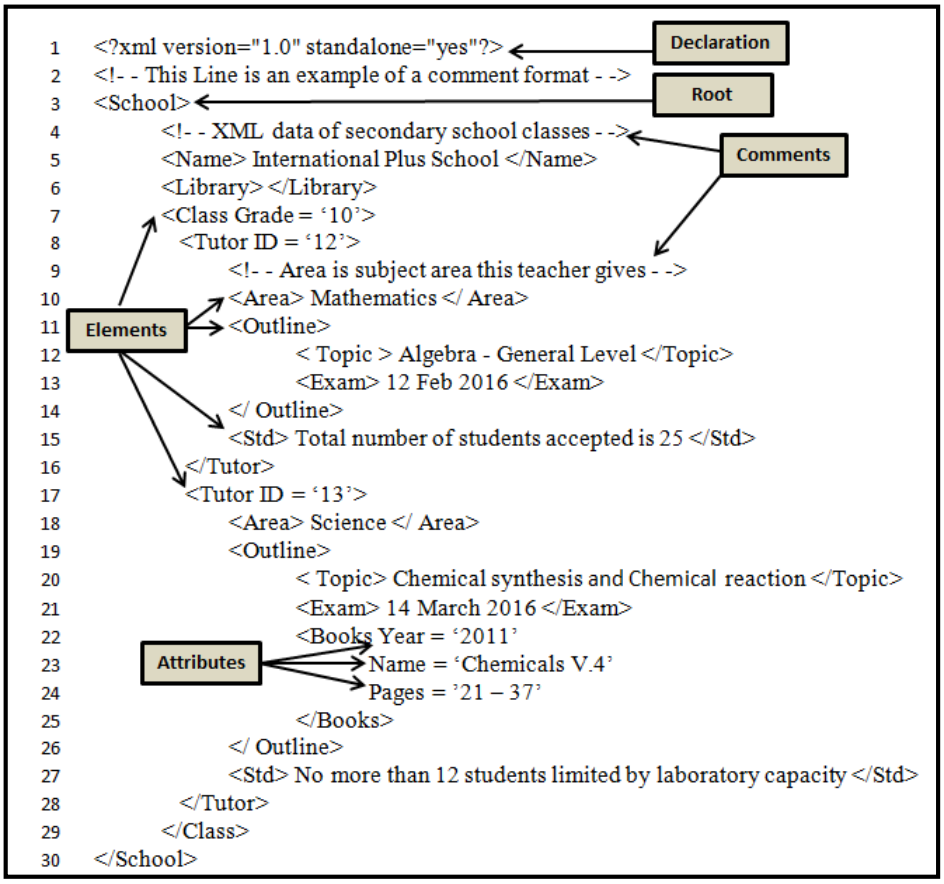


Figure 2.1 An XML sample - (School) example

2.4.1 XML Elements

XML represents data in a textual format in which its basic components are XML elements. Each element is a piece of text constrained by matching tags with case-sensitive names, such as <Tutor> (as a start-tag) and </Tutor> (an-end-tag) in Figure 2.1 (W3Schools., 2016b) (Abiteboul et al., 2000) (Bray et al., 2008) (Lee and Chu, 2001) (Luo, 2007). The start-tag and end-tag indicate the start and end of an element detail. An element can be empty (e.g., <Books> and <Library > in Figure 2.1) or may consist of text, other element(s) called sub-element(s) (e.g., <Outline> in Figure 2.1), or both. A root element represents the initial element in an XML document within which all other elements are nested (e.g., <School> </School> in Figure 2.1). As XML supports semi-structured data, repeated elements with the same tag names can be used to represent collections (e.g., element <Tutor> in Figure 2.1). Generally, the element tags must be balanced and nested properly in an XML document such that their closing tags should appear in a reverse order consistent with their opening tags (W3Schools., 2016b) (Abiteboul et al., 2000). Besides content, elements may have zero or more attributes that give additional specifications to the elements (Kurtev, 2001) (Bray et al., 2008) (Bray et al., 1998).

2.4.2 XML Attributes

An XML attribute consists of a name and a unique key value that is embedded within the start-tag of an element so as to provide extra information about that element (W3Schools, 2016c) (Evjen. B., 2007) (Tidwell, 2002) (Kurtev, 2001) (Zisman, 2000). For instance, the element <Books> in Figure 2.1 is composed of three attributes: “Year”, with value ‘2011’, “Name” with value ‘Chemicals V.4’, and “Pages” with value ’21 – 37’. An attribute value is of “string” type and must be delimited by single or double quotes (see Figure 2.1). Moreover, attributes can provide significant information for data management, for example by identifying each “Tutor” in Figure 2.1 by their ID numbers.

Unlike XML elements, each XML attribute value must be distinctive and cannot be repeated. Additionally, whilst elements can contain sub-elements, attributes cannot be expanded. Overall, there is a general preference for the use of elements over attributes to represent and maintain XML data (W3Schools, 2016c) (Whatley 2009) (Abiteboul et al., 2003) (Ray, 2003) (Tidwell, 2002). When an XML document follows the essential XML syntax that forms a tree hierarchy, it is referred to as being “well-formed” (Abiteboul et al., 2000) (Kurtev, 2001) (Bertino et al., 2000). A well-formed

document is limited in that it can only be parsed as a labelled tree (Abiteboul et al., 2000) (Bertino et al., 2000) (Bray et al., 2008) (Goldman et al., 1999), but no restriction is otherwise placed on its structure. However, if the semantics of an XML document have to be considered as well as the syntax, then some degree of restriction is necessary. A valid XML document is constrained as a well-formed document by an associated Document Type Definition (DTD) (Abiteboul et al., 2000) (Tidwell, 2002) (Walsh, 2016) (Bertino et al., 2000) (Jones et al., 2008).

2.4.3 XML Document Type Definition (DTD)

A Document Type Definition (DTD) comprises a schema for XML documents that encompasses a set of rules used to control the structure of XML documents (W3Schools, 2016d) (Lee and Chu, 2001) (Salminen and Tompa, 2012) (Mani and Sundaresan, 2003). It provides a set of element names used to define an XML document, along with their attribute types, if any. It also describes how these elements are related and the frequency of their occurrences within the XML document, as well as the order of their appearance (Lee and Chu, 2000) (W3Schools, 2016d) (Abiteboul et al., 2000). In other words, a DTD is a context-free grammar that underlies XML documentation, and can be stored either as an external file or internally within the XML document itself (Abiteboul et al., 2000) (Harold et al., 2004) (Lee and Chu, 2000) (Ray, 2003) (Zisman, 2000). An XML document is valid if it conforms to the rules stated in the DTD, such that the element sequences and nesting obeys the DTD specification, and required attributes are provided with the correct value types. Figure 2.2 shows an example of a DTD specification for the XML ‘School’ sample shown in Figure 2.1

DTD of School XML example:

```
<!ELEMENT School          (Name, Library, Class*)>
<!ELEMENT Name            (#PCDATA)>
<! ELEMENT Library        (#PCDATA| EMPTY)>
<!ELEMENT Class           (Tutor*)>
<!ATTLIST Class grade number #REQUIRED>
<!ELEMENT Tutor           (Area, Outline, Std, Books)>
<!ELEMENT Area            (#PCDATA)>
<!ELEMENT Outline         (Topic, Exam)>
<!ELEMENT Std             (#PCDATA)>
<!ELEMENT Topic           (#PCDATA)>
<!ELEMENT Exam            ((#PCDATA)>
<!ATTLIST Books  name, published year & pages #REQUIRED>
```

Figure 2.2 DTD for XML 'School' sample in Figure 2.1

Despite the ability of the DTD to provide considerable amount of control over XML structure, particularly in terms of vocabulary, it is limited (Shirrell, 2016) (Brandes et al., 2013). For instance, listing a set of acceptable values for the content of an element, and detailing the values of data types other than 'string' data is impossible. In order to address the shortcomings of DTDs, the W3C (W3C, 2016b) has recommended the implementation of the XML schema to deal with more complicated configurations only (Shirrell, 2016) (Brandes et al., 2013) (Roy and Ramanujan, 2001) (Bex et al., 2004).

2.4.4 XML Schema

An XML schema, unlike DTD, allows users to enforce proper syntax and semantics within XML documents rather than treating XML data as just plain text (W3C, 2016b) (W3Schools, 2016e) (Roy and Ramanujan, 2001). The W3C specification of the XML schema provides the capability to declare new data types to define elements values, as well as containing other built-in data types such as string, integer, Boolean, date and time. Moreover, an XML schema is essentially represented as an XML document in which the inherent elements and attributes are used to state the schema configurations (Abiteboul et al., 2000) (Harold et al., 2004) (Lee and Chu, 2000) (W3Schools, 2016e) (Radiya, 2000) (Waldt, 2010) - see Figure 2.3. An XML schema is more expressive than a DTD in term of supporting datatype definitions and values' domains, and it increases the flexibility of XML whilst overcoming the weaknesses of DTDs (Valentine; et al., 2001) (Roy and Ramanujan, 2001) (Bex et al., 2004).

XML schema of 'School' XML example

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.school.org/2016/XMLSchema">
<xs:element name="School">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="Name"    type="xs:string"/>
    <xs:element name="Library"   type="Empty"/>
    <xs: element name="Class"    type="xs: string"/>
     <xs:complexType>
        <xs: attribute name="Grade"   type="xs: string"/ use ="required">
        <xs: element  ref="Tutor"    maxOccurs="unbounded"/>
          <xs:complexType>
            <xs:sequencee>
              <xs:element name="Area" type="xs:string"/>
              <xs:element name="Outline" type="xs:string"/>
                  <xs:complexType>
                        <xs:element name="Topic" type="xs:string"/>
                        <xs:element name="Exam" type="xs:date"/>
                        <xs:element name="Books" type="Empty"/>
                          <xs:simpleType>
                             <xs: attribute name="Year"   type="xs: date"/>
                             <xs: attribute name="Name"   type="xs: string"/>
                             <xs: attribute name="Pages"   type="xs: integer"/>
                           </xs:simpleType>
                          </xs:element>
                    </xs:complexType>
              <xs:element name="Std" type="xs:string"/>
          </xs:sequencee>
       </xs:complexType>
      </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:element> </xs:schema>
```

Figure 2.3 XML Schema for 'School' example

2.5 XML Parsers

XML parsers can detect the validity and well-formedness of an XML document by reading its components via Application Programming Interfaces (APIs) (Su Cheng and Krishna Rao, 2007) (Haw and Rao, 2007) (Takase et al., 2005). An XML parser basically converts the underlying plain textual format of an XML document into its logical data representation by treating the XML document as an XML tree or stream (Haw and Rao, 2007) (Rangan and Jayanthi, 2011) (Wang et al., 2007) (Nicola and John, 2003). For the tree-based approach (e.g. DOM, JDOM, ElectricXML, and DOM4j XML parser), it loads the whole document into memory as a collection of objects representing the original document in tree structure. In terms of time and memory the loading process is inefficient, and as a result this approach is unsuitable for large-scale XML data since it can easily go beyond reasonable memory capacities (Kiselyov, 2002) (Tong et al., 2006) (Lam et al., 2008).

The stream-based approach (also called an event-based parser; such as SAX, StAX and XMLPull), reads the entirety of the XML document and applies user-defined actions whenever a new XML component appears (Lu et al., 2006) (Nicola and John,

2003). The actions can merge the received elements and XML data into custom data structures, including the XML tree, as in DOM (Kiselyov, 2002).

Of the above the two mostly commonly used XML parsers are DOM and SAX (Nicola and John, 2003) (Kiselyov, 2002) (Haw and Rao, 2007) (Lu et al., 2006), which are discussed in the following sections.

2.5.1 Document Object Model

Document Object Model (DOM) (W3C, 2005) (Wang et al., 2007) (Mani and Sundaresan, 2003) is a language and platform-independent definition purposed by W3C as a component of the Java API designed for XML processing (Oracle, 2014). Based on object technology, the DOM parser applies a tree-based approach that constructs the "Document Object Model" of an entire XML document as a structured tree (Haw and Rao, 2007) (Kiselyov, 2002) (Tong et al., 2006) (Lam et al., 2008). In the DOM tree model, each component is an object that contains values (textual content) and has its own methods to facilitate data access and modification.

Using DOM requires the entire XML tree to be built within main memory. This provides better performance for XML operations in terms of data access and navigation, data modification, and enabling XPath queries (Whitmer, 2004) (Wang et al., 2007). However, constructing an entire XML tree in memory is not suitable for large-scale XML documents, because the DOM tree could be up to 10 times larger than the original XML document (Wang et al., 2007) (Kiselyov, 2002).

2.5.2 Simple API for XML

The Simple API for XML (SAX) parser (Megginson, 2000) (Nicola and John, 2003) (Pan et al., 2008) (Matsuda, 2007) is a stream-based parser that invokes parsing events (such as the start and end of documents/elements) using call-backs. Unlike the DOM tree parser, SAX interacts with an application during the parsing process and does not store any information about XML components. This enables the XML parser to parse even large XML documents within a reasonable timeframe (Haw and Rao, 2007) (Takase et al., 2005) (Nicola and John, 2003) (Pan et al., 2008).

The SAX parser applies a depth-first traversal algorithm (Tahraoui et al., 2013) in which event-driven methods are triggered by the occurrence of an element's opening or closing tags, correlated attributes of an element, or comments and processing instructions. For instance, when an opening-tag is encountered, the start-element

event handler is triggered releasing the data from the previous element. Although this saves memory consumption, it makes it difficult to distinguish the structural relationships between nodes. In order to identify the logical structural relationships, the use of memory stacks (where the maximum size equals the maximum depth of an XML document) is required. Whenever a start-element event is invoked, the parsed element is stacked along with its associated nodes (siblings/children). When a closing tag is encountered, the element is removed from the stack(s). Therefore, controlling structural relationships in SAX is more complicated than in comparison to DOM.

2.6 XML Tree Structure

An important feature of XML is that the document itself describes the structure of the XML data, usually represented as a tree graph (W3Schools, 2016f) (Brandes et al., 2013) (Hachicha and Darmont, 2013) (Abiteboul et al., 2000) (Al-Khalifa et al., 2002) (Harold et al., 2004) (Ray, 2003). The terminology comes partly from a genealogical tree, where the root of the tree is some early ancestor, and the latest descendants located at the bottom of the tree. In other words, an XML document can be regarded as a tree whose nodes are the document items (elements, attributes, and/or values) and whose edges form the structural relationships between the nodes (Brandes et al., 2013) (Na and Guoqing, 2010) (Li et al., 2006a). The XML model tree can be either unordered (i.e. the order of the elements within the XML document is not important) or ordered (Deutsch et al., 1999) (Lohrey et al., 2015). Figure 2.4a and 2.4b demonstrate the XML tree representation of the XML ‘School’ example shown in Figure 2.1 (through only the first eight lines for simplicity) as unordered and ordered trees, respectively, where the oval-shaped boxes represent elements and the rectangular boxes represent attributes. Since document order is essential for querying XML (Tahraoui et al., 2013) (Hachicha and Darmont, 2013), here only the XML document representation as a rooted, ordered XML tree is considered.

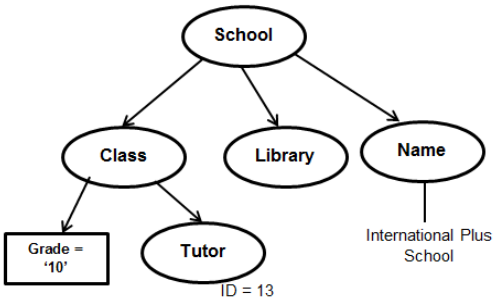


Figure 2.4a Unordered XML model tree of Figure 2.1

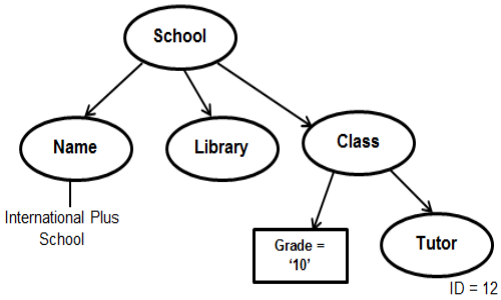


Figure 2.4b Ordered XML model tree of Figure 2.1

From the ordered XML tree shown in Figure 2.4b, it can be seen that the root "*School*" is the ancestor of the other tree nodes. The elements "*Name*", "*Library*", and "*Class*" have a sibling relationship and are all children of the element "*School*". Considering a left-to-right order among siblings in the XML tree node, since the node "*Name*" appears before "Library", it is known as the pre-order sibling to "Library" and vice versa. That is, "*Library*" is a post-order sibling to "*Name*". This family relationship representation between XML tree nodes makes it easier to understand an XML document (Teorey et al., 2011) (Tizag, 2003) (Bille, 2003). To expedite XML query processing, it is necessary to provide methods for determining the structural relationships between nodes (Li et al., 2006a) (Al-Khalifa et al., 2002) (Li and Moon, 2001) (Subramaniam et al., 2014a) (Sheng et al., 2011). The most common structural relationship queries among nodes are:- those of Parent-Child, Ancestor-Descendant, Siblings, Lowest Common Ancestor (LCA), and Document Order (Lizhen and Xiaofeng, 2013) (Xu et al., 2009) (Liu and Zhang, 2016). These relationships are discussed below based on the following definition of a rooted ordered XML tree and Figure 2.5, which represents the XML tree of the XML "School" sample in Figure 2.1.

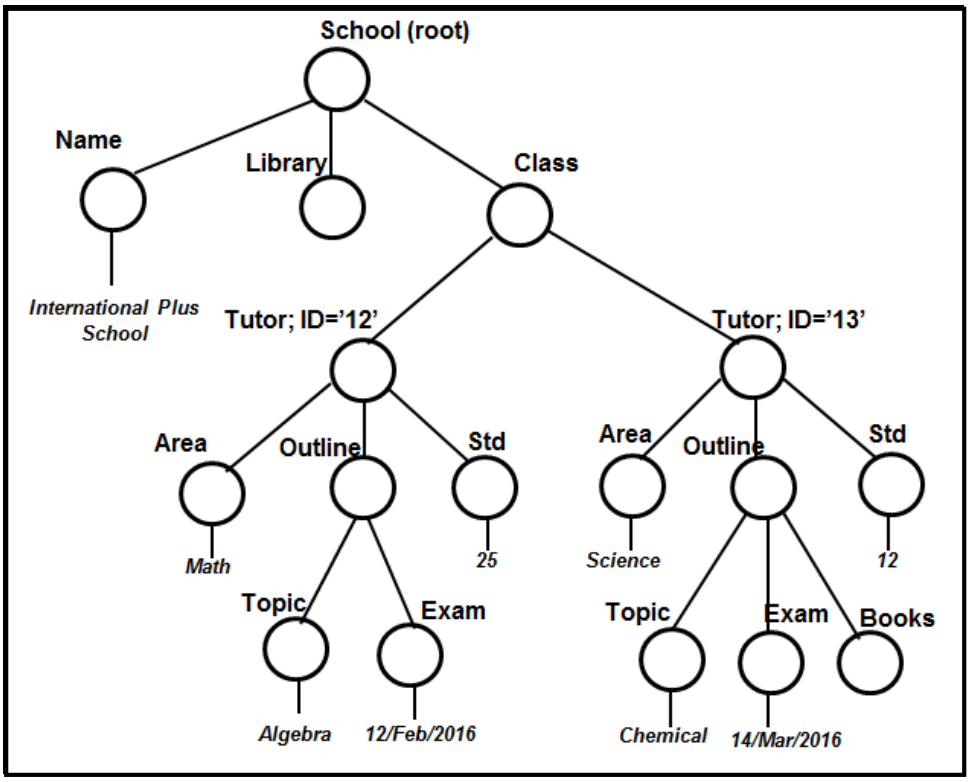


Figure 2.5 XML tree representation of the "School" example in Figure 2.1

2.6.1 Rooted Ordered XML Trees

XML documents are often illustrated graphically as rooted trees where vertices represent nodes and edges represent element, sub-element, element-value, and attribute-value relationships (Bille, 2003) (Chi et al., 2003) (Reis et al., 2004) (Yun et al., 2004) (Bousquet-Mélou et al., 2015). A rooted tree has one vertex singled out as the root, from which the rest of the vertices descend. A rooted tree is ordered if the set of children associated with each vertex in the tree has a predefined left-to-right order that reflects the appearance of their tags within an XML document. In other words, a rooted ordered tree can be defined as follows (Dalamagas et al., 2006) (Tahraoui et al., 2013):

A rooted ordered tree $T = (V, E, r)$ consists of a set of edges, E, a finite set of k vertices (nodes) $V = \{v_1, \dots, v_k\}$, and $r \in V$ is a distinguished vertex called the root, from which the rest of vertices descend. Each edge $e_{u \to v} \in E$ indicates an element-subelement or an element-attribute relationship between nodes u and v, where $(u, v) \in V$ (Luo, 2007) (Kaushik et al., 2002a) (Han et al., 2006). The level of a node v $(v \in V)$, denoted as $l(v)$, can be defined as the number of edges along the unique path between the root r and the node v. The depth , D, of a rooted tree, T, is the maximum level of any node in the tree. A rooted tree, T, is ordered if the children of

each node are ordered (Chi et al., 2003) (Bille, 2003) (Dalamagas et al., 2006) (Tahraoui et al., 2013). According to the definition of T, given two nodes u and v in a rooted ordered tree, T, with $l(u) < l(v)$ and u is on the path from the root to v, then v is a descendent of u and u is an ancestor of v (Yun et al., 2004). If u and v are adjacent nodes; i.e., there is a direct edge $e_{u \to v} \in E$ from node u to node v and $l(v) = l(u) + 1$, then u is called the parent of v, and v is a child of u (Lin et al., 2013) (Xu et al., 2007). If node v can be reached from node u through many directed edges where $l(v) > l(u) + 1$, then u is an ancestor of v (Luo, 2007). A set of nodes $S = \{s_1, s_2, \dots, s_n\}$ that share the same parent p are called siblings, where s_i is a pre-sibling to s_{i+1}, for $1 < i < n$ (Yanghua et al., 2012). A node x without children is called a leaf node, and other nodes are known as "internal nodes" (Tahraoui et al., 2013) (Mani and Sundaresan, 2003). For simplicity rooted ordered trees are referred to as trees or XML trees in this thesis. An example of such a tree can be seen in Figure 2.5.

2.6.2 Structural Relationships between XML Nodes

Parent-child relationship: can be identified between a node and any node immediately descending from it. A node u is a parent of node v if u and v are directly linked in an XML tree and u appears exactly one level above v. In Figure 2.5, for instance, the node "Tutor" is a parent of node "Outline", which itself a parent of node "Exam".

Ancestor-descendant relationship: A node u is an ancestor of a node v, and v is a descendent of u, if there is a linked path of nodes $n_1, \dots, n_y$ from the root to node v such that node $u = n_x$, and $v = n_y$, where $x < y$. Thus, u appears on the path from r to v but $v \neq u$. That is, node u is the root of a subtree containing node v. Referring to Figure 2.5, the node ""Class" is an ancestor of the nodes "Topic" and "Area", and all the nodes of the tree are descendants of the root node "School".

Lowest Common Ancestor (LCA): the lowest common ancestor node, L, exists between two nodes u and v if L is the deepest node in an XML tree, T, that has both u and v as descendants (Na and Guoqing, 2010). In other words, the LCA L is the shared ancestor of nodes u and v located farthest from the root in T. For example, in Figure 2.5, the LCA for both nodes "Area= science" and "Topic=chemical" is node "Tutor of ID=13"; even though nodes "Class" and "School" are also ancestors to both nodes.

Sibling relationship: nodes u and v are siblings if both nodes share the same parent and are at the same level in an XML tree, T. If u appears to the left of v in an ordered XML tree T, then u is called a pre-order sibling to node v, whereas v is a post-order sibling to node u. In Figure 2.5, the nodes "Name", "Library", and "Class" are siblings because they all share the same parent "School", where "Library" is a pre-order sibling to "Class" and a post-order sibling to "Name".

Document order: is preserved in the order of the elements as they appear in an XML tree, where the hierarchical structure is considered to be from top to bottom and the siblings from left to right (i.e., corresponding to depth-first left-most) (Härder et al., 2007). For instance, the nodes in the "School" XML tree shown in Figure 2.5 are ordered corresponding to the node order within the "School" XML document (see Figure 2.1).

Determining the structural relationships between XML tree nodes and locating all occurrences of these relationships lies at the very core of XML querying efficiency (Al-Khalifa et al., 2002) (Xu et al., 2012) (Hachicha and Darmont, 2013) (Chien et al., 2002) (Sheng et al., 2011).

2.7 Querying XML

As XML became a standard for data exchange and representation on the Internet, extensive research into the retrieval of XML and semi-structured data began to be carried out (Florescu and Kossmann, 1999) (Li and Moon, 2001) (Subramaniam and Haw, 2014b) (Sheng et al., 2011) (An and Park, 2011) (Schmidt et al., 2001b) (Sonawane and Rao, 2015) (Wang et al., 2003). Usually the tree representation of XML documents and queries are used to process XML data implicitly or explicitly (Tahraoui et al., 2013) (Al-Khalifa et al., 2002) (Bressan et al., 2001) (Alghamdi et al., 2014) (Shnaiderman and Shmueli, 2015), whereby retrieving XML data is achieved via a tree-pattern matching approach (Lu et al., 2011a) (Lu et al., 2005a) (Bruno et al., 2002) between the query tree and the document tree (Tahraoui et al., 2013) (Al-Khalifa et al., 2002) (Florescu et al., 2000) (Gheerbrant et al., 2013). Tree patterns are graphical representations of XML queries over an XML tree data model (Hachicha and Darmont, 2013) (Czerwinski et al., 2016). To answer a query, the semantics of tree pattern are obtained by mapping from the pattern nodes to the nodes in the XML document, such that the main structural relationships (Section 2.5) are satisfied (Lu et al., 2011a). Numerous XML query languages such as XPath (Robie, 2007), Lorel (Abiteboul et al., 1997), XIRQL (Fuhr and Großjohann, 2000), NEXI (Trotman and

Sigurbjörnsson, 2004), TeXQuery (Amer-Yahia et al., 2004), Quilt (Chamberlin et al., 2000), XQuery (Boag, 2007), and XQBE (Braga et al., 2005) primarily use tree patterns to identify the relevant data parts in an XML document. These are discussed below.

2.8 XML Query Languages

Due to the extensive use of XML (Bray et al., 1998) in today's applications, several XML query languages have been proposed to analyse the semantics (content) and syntax (structure) of XML queries (Li and Moon, 2001) (Bonifati and Ceri, 2000) (Al-Khalifa et al., 2002) (Kamps et al., 2006) (Sheng et al., 2011) (Wang et al., 2003) (Qadah, 2016) (Choi and Wong, 2015). Queries in these languages basically specify patterns to be matched to elements and specifies their structural relationships (Alghamdi et al., 2014) (Deutsch et al., 1999) (Fuhr et al., 2001) (Thonangi, 2006). In general, at each node in the query tree pattern, there is a node predicate that identifies a number of bases depending on the content of the user's request (Al-Khalifa et al., 2002). A major focus in designing XML query language is the ability to express complex structural queries (Wang et al., 2003) (Choi and Wong, 2015). According to Amer and Lalmas (Amer-Yahia and Lalmas, 2006) and Tahraoui (Tahraoui et al., 2013), XML query languages can be classified into three main categories, as based on their queries' structure :-

- ***Tag-based queries***: permit users to specify simple conditions about a tag that will contain the required content, such as in XSEarch (Cohen et al., 2003) query language. For example, the query "Topic: algebra" means that the user is searching for a "Topic" element about "algebra".
- ***Path-based queries***: refer to XML element nodes as a path-like syntax. This family of query languages includes XPath, XIRQL, and NEXI (CAS), and presents more sophisticated content conditions on structural XPath-based syntax. An example of XPath query is *"/School/*/Tutor[@ID="13"]",* which indicates the fact that the user needs to know all tutors in the school with identification number "13".
- ***Clause-based queries***: inspired by SQL syntax, these comprise nested clauses that allow users to express more complicated requirements. Examples of XML query languages built on clause-based queries are:- Lorel (Abiteboul et al., 1997), Quilt, TeXQuery, and XQuery, where XQuery supports path expressions similar to those of XPath (Connolly and Begg, 2005). For

instance, the XQuery path expression *"/School[Name .'Plus']//Exam[. 'Feb']"* matches the "*School*" element with child name that includes the word 'Plus' and has "*Exam*" as a descendent element with a content string value '*Feb'*.

Although most of the available query languages vary in the details of their grammatical representations, they typically use regular path expressions (or simply regular expressions) to evaluate XML queries and are capable of extracting and manipulating data directly from XML documents (Haw and Lee, 2011) (Lassila et al., 2015) (Huang et al., 2015) (Li and Moon, 2001). Regular path expressions allow users to navigate through arbitrarily long paths in semi-structured XML data by traversing the logical XML hierarchal tree structure (Ives et al., 2000) (Fernandez and Suciu, 1998) (Li and Moon, 2001). In the XML context, a regular expression is defined as a series of location steps in the XML tree linked by '/' or '//' to identify the location of a node starting from the root node (Robie, 2007) (Al-Badawi, 2010) (Almelibari, 2015) (Hidders and Paredaens, 2014). Every step includes an axis that defines the direction of navigation (Ramanan, 2003), and a 'node test' that selects nodes based on their type and name. Based on the document order, axes can be classified as forward axes (e.g. descendent, descendent-or-self, child, following, and following-siblings) and reverse axes (such as ancestor, parent, preceding, and preceding-sibling) (Robie, 2007).

Based on regular path expressions, XPath (Robie, 2007) and XQuery (Boag, 2007) are the standard XML query languages (Catania et al., 2005a) (Hsu and Liao, 2013), with the rest acting so similarly and that they can be eliminated from further discussion at this stage. Both the XPath and XQuery languages are defined by the W3C (World Wide Web Consortium) in terms of the manner in which they query XML data and follow the requirements for an XML query language (Chamberlin et al., 2001) (W3C, 2007) (Hidders and Paredaens, 2014). The W3C development teams have enhanced the XPath2.0/XQuery1.0 query languages to support the Full Text Search (FTS) functionality (W3C, 2011) (Al-Badawi, 2010).

2.8.1 XML Path language (XPath)

XPath models the XML document as a rooted-ordered tree that consists of seven distinct types of node: elements, attributes, text, comments, processing instructions, namespace nodes, and a root node. XPath navigates and selects the whole, or parts, of an XML document based on regular path expressions that are expressed using thirteen principle axes types identified by the appropriate structural relationship. A

description of an XPath axes is given in Table 2.1 (adapted from (Al-Badawi, 2010) (Almelibari, 2015) and (Robie, 2007)) along with XPath query examples (quoted from XPathMark (Franceschet, 2005)) that are based on the XMark benchmark (Schmidt et al., 2002). To express complex queries, XPath use predicates and other logical and arithmetic operators to facilitate the identification of specific nodes and values (Robie, 2007) (Connolly and Begg, 2005) (Elmasri, 2008) (Harold et al., 2004) (Gupta and Suciu, 2003).

Table 2.1 XPath Axes

Axis name	Objective	Example
child	Indicates all children of the current node	***/site/regions/*/item*** Means returns all the products items.
Parent	Refers to the parent of the current node	***/site/regions/*/item[parent::namerica or parent::samerica]*** *Find the American items*
descendent	Contains the descendants of the current node	***//keyword*** Means return all the keywords in the document
descendent-or-self	Contains the current node itself along with all its descendants	***/descendant-or-self::listitem//descendant-or-self::keyword*** Requests the return of certain keywords in a paragraph item
ancestor	Contains all the ancestors of the current node	***//keyword/ancestor::listitem*** Requests the return of the paragraph items containing a keyword
ancestor-or-self	Contains current node and all its ancestors	***//keyword/ancestor-or-self::mail*** Ask for a mail containing a keyword
following	Refers to all nodes following the current node in document order.	***/site/regions/*/item[@id='item0']/following::item*** Ask for the items that follow, in document order, for a given item
following-sibling	Contains the following siblings (i.e. post-order siblings) of the current node	***/site/open auctions/open auction[bidder[personref/@person= 'person0']/following-sibling::bidder[personref/@person='person1']]*** Find all open auctions in which bidder(s) issued a bid before a specified person with reference

		'person1'.
preceding	Refers to all nodes appearing in a document before the current node starting from the root.	***/site/open auctions/open auction/bidder[personref/ @person='person1']/preceding::bidder [personref/@person='person0']*** Find the bids issued (in document order) before a certain person with reference 'person0'.
Preceding-sibling	Refers to the preceding (i.e. pre-order) siblings of the current node.	***/site/open auctions/open auction[@id='open auction0'] /bidder/preceding-sibling::bidder*** Requests the past bidders of a given open auction
self	Represents the current node	***//*[self::open auction]*** Means select the current open auction.
attribute	Indicates the attributes of the current node	***id(/site/closed auctions/closed auction[buyer/@person='person4'] /itemref/@item)*** Means items bought by a given person
namespace	Contains the namespace of the current node	***[namespace-uri() = 'http://example.com']*** Returns namespace URIs matching 'example.com'

2.8.2 XML Query Language (XQuery)

XQuery (Boag, 2007) (W3C, 2014) is a clause-based declarative language that has some SQL-like semantic features. This is because XQuery is derived from Quilt XML query language (Chamberlin et al., 2000), which itself borrowed characteristics from XPath (Robie, 2007) and other languages supporting the SQL syntax, such as XML-QL (Deutsch et al., 1998), XQL (Robie et al., 1999), and SQL itself (Date and Darwen, 1993). Apart from the path expression adopted from XPath, XQuery has developed 'FLWOR' expressions to perform SQL-like transactions. 'FLWOR' is an abbreviation for "For, Let, Where, Order by, and Return" clauses. There are many advantages to using the 'FLWOR' syntax of XQuery over XPath syntax (W3C, 2014); for instance, "For, Let, and Where" clauses provide more expressiveness and flexibility in forming complex queries, which allows multiple XML documents to be joined during result construction (Boag, 2007). The "Order by" clause helps to organise results during the XML reconstruction (Al-Badawi, 2010). The "Return" clause controls the evaluation of the structure of the returned XML nodes by adding further meanings to the data or conditional statements (e.g. IF-THEN-ELSE).

Generally, the XPath and XQuery query languages provide a wide range of XML queries that cover almost all XML querying functionalities (Bressan et al., 2001) (Franceschet, 2005) (Jones et al., 2008). However, querying XML via query languages has a number of drawbacks, as illustrated in the next section.

2.8.3 XML Query Languages Weaknesses

As XML (Bray et al., 2008) has become the standard representation of data over the Internet, a wide range of users need to interact with XML documents to obtain their desired information. Despite the rich expressive power of XML query languages, their complexity has become a major difficulty for users in formulating appropriate queries, and for software applications to process the queries efficiently (Choi and Wong, 2015) (Wang et al., 2003). In order to design a query, the user must be familiar with the semantics of the query languages as well as the underlying concepts of XML data structure, such as DTD, XML schema, elements and attributes names (Choi and Wong, 2015) (Fuhr et al., 2001). Given that the hierarchical structure of XML data can be heterogeneous, any slight misinterpretation of the document structure whilst formulating a query would result in an incorrect or misleading answer (Liu and Yan, 2016). Such problems emerge when various types of documents with different structures are queried, as such queries often generate long, complicated path expressions. Moreover, the operating cost of traversing the hierarchy of XML data can be significant if the path lengths are very large, and so retrieving data directly from very large XML documents can be inefficient (Li and Moon, 2001) (Hakuta et al., 2014) (Choi and Wong, 2015) (Lassila et al., 2015) (Liu and Gawlick, 2015) (Fan et al., 2015), especially when the XML database is update-intensive (Qadah, 2016).

To overcome these limitations and improve the efficiency of XML querying, two main approaches have been proposed to facilitate query processing based on the hierarchical structure of XML data: ***structural indexing*** and ***labelling schemes*** (Li and Ling, 2005b) (Li et al., 2006b) (Khaing and Ni Lar, 2006) (Duong and Zhang, 2005). The next section gives a brief overview of structural indexing approaches. Since this thesis focuses on XML labelling schemes, the next chapter will present the state-of-the-art on XML labelling schemes particularly for dynamic XML data.

2.9 Structural Indexing

As discussed in the previous section, several XML query languages have been developed to evaluate path expressions that rely on XML tree traversal. Scanning and

extracting the user's required nodes directly from large-scale XML data is computationally expensive (Li et al., 2012) (Catania et al., 2005a) (Chen et al., 2003) (Kaushik et al., 2002a). This inefficiency has led to the development of structural indexing in order to speed up XML query processing (Hsu and Liao, 2013) (Li et al., 2012) (Catania et al., 2005a) (Alghamdi et al., 2014) (Han et al., 2006) (Shichuan et al., 2012). Structural indexing significantly reduces the portion of the XML data to be scanned during the query processing by constructing an index that summarises the structure (path information) of an XML data tree (Li et al., 2006b) (He and Yang, 2004) (Hsu and Liao, 2013). Here, the idea is to conserve an XML data tree in the form of a summarised tree, which is defined using a specific equivalence relation to the nodes of the original data tree. Thus, identical sub-structures in an XML document are merged to form the summarised tree, which is then used as the structural index to evaluate path expressions without the need to refer to the original data (Chen et al., 2003) (Hsu and Liao, 2013) (Li et al., 2012) (Zou, 2004).

Two common methods used to summarise an XML document into a structural index (Hsu and Liao, 2013) are:- those of path equivalence (Zou, 2004) (Chung et al., 2002) (Cooper et al., 2001) (Goldman and Widom, 1997) (Zhang et al., 2008) and bi-simulation (Kaushik et al., 2002a) (Chen et al., 2003) (Chen et al., 2008). In path equivalence, nodes with identical traversal paths are merged to construct a structural index. For instance, "the strong DataGuide" (Wu and Liu, 2008) holds all the direct edges representing parent-child relationships in an XML tree. As a result, the parent-child and ancestor-descendent relationships of a path expression can be evaluated directly from the structural index (Hsu and Liao, 2013). Index Cache (Li et al., 2012) and ToXin (Rizzolo and Mendelzon, 2001) indexing techniques have been proposed based on DataGuides (Wu and Liu, 2008) (Goldman and Widom, 1997) (Goldman and Widom, 1999).

A bi-simulation scheme (Henzinger et al., 1995) basically captures the local structures of an XML data tree and accordingly groups the nodes with the same set of incoming paths, forming collections of equivalence classes which are then stored as structural summaries (He and Yang, 2004) (Chen et al., 2003) (Alghamdi et al., 2014) (Hsu and Liao, 2013). In this scheme, XML data is partitioned into equivalence classes based on the backward path bi-similarity from the root to the indexed node (Milo and Suciu, 1999), forward path bi-similarity from the indexed node to the root (Kaushik et al., 2002a) (Chen et al., 2003), or in both directions, as in F&B-Index (Kaushik et al., 2002b).

In comparison to XML query languages, structural indexing reduces the query processing time by avoiding direct access to the original XML data whilst evaluating path expressions. However, as the size of an XML document increases, index sizes tend to rise dramatically (Alghamdi et al., 2014). In general, structural indices are large since each node of an XML database is referenced within the index along with its path summary from the root to that particular node (Haw and Lee, 2011). Thus, the traversal process applied to construct these indices is costly (Khaing and Ni Lar, 2006) (Li and Ling, 2005b). Furthermore, when XML data are updated it is necessary to re-build the structural indices (Duong and Zhang, 2005). Generally, structural indexing does not support complex queries (Alghamdi et al., 2014) and for long queries it requires a large part of the index (if not all) to exist in main memory in order to establish ancestor-descendent relationships between two nodes based on their structure summary (Na and Guoqing, 2010).

Unlike structural indexing, labelling schemes can efficiently establish the structural relationships between two nodes by using the nodes as the fundamental unit by which to query XML. Hence, labelling schemes provide greater flexibility and require reduced storage compared to structural indexing (Haw and Lee, 2011) (Li et al., 2006b). Therefore, this thesis focuses mainly on labelling scheme approaches, which are discussed in the next chapter.

2.10 Conclusion

This chapter has presented an overview of the basic concepts of XML data. Since XML is a vast subject, the main points covered in this chapter are limited, but are nevertheless sufficient to provide the necessary background to comprehend the research objectives. The main focus of this thesis is that of XML labelling schemes, and as such the focus of the next chapter will be to illustrate the state-of-the-art in XML labelling schemes.

Chapter 3: Literature on XML Labelling Schemes

3.1 Introduction

Due to the growing significance of managing XML data, a considerable amount of research has been dedicated to XML storage and querying (Liu and Zhang, 2016) (Agreste et al., 2014) (Ghaleb and Mohammed, 2013). To query XML data competently and rigorously, several XML labelling schemes have been introduced.

An XML labelling scheme can handle a rooted ordered XML tree data model such that structural information can be encoded into labels. The essential metrics for a labelling scheme are the speed at which these labels can be generated and used, as well as the compactness of the encoded labels. As the transmission of XML data over the Internet has become vibrant, it has also become necessary to have an XML labelling scheme that supports dynamic XML data (O'Connor and Roantree, 2010a) (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005). The challenge to developing such labelling schemes, which can handle dynamic updates to XML data without affecting the initial labels has become the main focus of many researchers (Xu et al., 2009) (Liu et al., 2014) (Duong and Zhang, 2008) (He, 2015) (Ghaleb and Mohammed, 2015) (Qin et al., 2017).

This chapter presents the state-of-the-art in research into XML labelling schemes. First, an overview on XML labelling schemes is given in Section 3.2; this includes XML labelling schemes' main principles, and their desirable properties, types and classifications. Generally, XML labelling schemes can be categorised into four groups: - interval-based, prefix-based, multiplicative, and hybrid, as discussed in Sections 3.3, 3.4, 3.5, and 3.6, respectively. Finally, Section 3.7 concludes the chapter with an illustration of the problems encountered with current research work on XML labelling schemes.

3.2 XML Labelling Schemes: An Overview

XML labelling schemes (also referred to as numbering schemes) can efficiently establish the structural relationships between two nodes by using the nodes as the

fundamental unit for querying XML, and so provide more flexibility, as well as requiring less storage space in comparison to other XML querying approaches (Haw and Lee, 2011) (Li et al., 2006b) (Khaing and Ni Lar, 2006) (Duong and Zhang, 2005).

XML Labelling schemes typically facilitate XML query processing by assigning a unique label to identify XML tree nodes according to the structure of the XML document (O'Connor and Roantree, 2010a) (Li et al., 2008) (Ghaleb and Mohammed, 2015) (He, 2015) (Wang et al., 2003). In this way, the structural relationships between nodes can be efficiently determined by comparing their labels so that XML queries can be processed without accessing the original data (Liu et al., 2013) (Zhuang et al., 2011) (Xu et al., 2009) (Fraigniaud and Korman, 2016). The existing labelling schemes can be categorised into two groups (Mirabi et al., 2012) (Thonangi, 2006) (Rusu et al., 2006) (Ghaleb and Mohammed, 2015): static labelling schemes (Dietz, 1982) (Li and Moon, 2001) (Tatarinov et al., 2002) that are adequate for non-updatable XML documents, and dynamic labelling schemes (Xu et al., 2009) (Liu et al., 2014) (Duong and Zhang, 2008), which are used to label XML documents that are frequently updated.

Much of today's Web content is written in well-formed or valid XML data format (Grijzenhout and Marx, 2013). The growing popularity of XML as a data exchange format due to its flexibility has led to an enormous amount of XML data update (Liu and Zhang, 2016) (Tekli and Chbeir, 2012) (Tatarinov et al., 2001). As the XML repositories over the web become more extensive as well changeable, the need for dynamic labelling schemes has become essential to support efficient XML queries and update (O'Connor and Roantree, 2010a) (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005). According to (Lizhen and Xiaofeng, 2013) (Härder et al., 2007) and (Wu et al., 2004) a good dynamic labelling scheme should have the following desirable characteristics:

- ***Deterministic***: the structural relationships between two nodes can be quickly established by examining their labels.
- ***Efficient***: it should support all kinds of structural relationship queries.
- ***Compact***: the labels should be sufficiently compact, as small as possible.
- ***Dynamic***: it should completely avoid the need to re-label XML trees nodes when XML files are updated.

Many works have studied dynamic XML labelling schemes, but each of the existing labelling schemes is limited through at least one of these characteristics. For the most part, updating XML data remains the weakness in the majority of these XML labelling

schemes (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005). The remainder of this chapter will focus on dynamic XML labelling schemes reported in the literature, and will present the strengths and weaknesses of these labelling approaches. In general, current XML labelling schemes can be classified into four categories (Su-Cheng and Chien-Sing, 2009) (Chiew et al., 2014a) (Liu and Zhang, 2016) (Subramaniam et al., 2014a) (Chen et al., 2011):- interval-based labelling schemes, prefix-based labelling schemes, multiplicative labelling schemes, and hybrid labelling schemes.

3.3 Interval-based Labelling Schemes

3.3.1 Structure and Concept

Interval-based labelling schemes (also known as Containment labelling schemes, Range-based labelling schemes, or Region Encoded labelling schemes) (Xu et al., 2012) (O'Connor and Roantree, 2010a) utilise the properties of tree traversal (Dietz, 1982) (Li and Moon, 2001) (Subramaniam and Haw, 2014b) to preserve document order and to establish structural relationships among nodes. Tree traversal (Tahraoui et al., 2013) is the process of sequentially visiting each node in a tree data structure, and can proceed in different directions (O'Connor and Roantree, 2010a) (Qadah, 2016). A pre-order (also called DFS: depth-first search) traversal of an ordered tree constitutes visiting a tree starting from the root (top-bottom) and processing each level from left to right, while post-order traversal start visiting the leaf nodes from left to right, and then processing their parent level (bottom-up) (Dietz, 1982).

(Kannan et al., 1992) introduced the idea of efficiently encoding the ancestry relation in a tree using interval-based schemes (Fraigniaud and Korman, 2016) (Peleg, 2000). This contains a mechanism of assigning the shortest possible labels to an XML tree nodes in such a way that information concerning any two nodes can be obtained directly from their labels (Fraigniaud and Korman, 2016). Accordingly, (Kannan et al., 1992) suggested an ancestry-labelling scheme, which is defined as follows: given a rooted tree, T, with n nodes, process depth-first traversal on T, starting from the root, and assign each node $v \in T$ a DFS number $DFS(v) \in [1, n]$ sequentially. Since in a depth-first traversal each node v is reached before all of its descendants, v has smaller DFS number than of any of its children. Then a node v is given an interval label $(v) = [DFS(v), DFS(v')]$, where v' is the last descendent of v.

Given two nodes u and v with interval labels $I(u)$ and $I(v)$, respectively, an ancestry query between u and v can be determined as follows: a node u is an ancestor of a node v if, and only if, both $DFS(u) \leq DFS(v)$ and $DFS(u') \geq DFS(v')$; where u' and v' are the last descendants of u and v, respectively. Using this scheme to label a tree with n nodes, each of the resulting interval labels is of size $2log(n)$ bits. Similar to work of (Kannan et al., 1992), considerable research has been carried out on interval-based labelling schemes to enhance the performance of XML querying where labels are even shorter than $2log(n)$ (Abiteboul et al., 2001) (Peleg, 2000) (Fraigniaud and Korman, 2016) (Alstrup and Rauhe, 2002).

3.3.2 Related Schemes

Dietz's numbering scheme (Dietz, 1982) was the first to use traversal to number tree nodes as a linked list to determine the order of elements within a tree in a constant time (Su-Cheng and Chien-Sing, 2009). Each node is labelled as an interval <pre-order, post-order>, where the pre-order value of a node u is the pre-order traversal rank position of u before its descendants are visited. Similarly, the post-order value of a node v is the post-order traversal rank position before the ancestors of node v are visited. According to (Dietz, 1982) the ancestor/descendant relationships between tree nodes can be maintained by exploring the pre-order and post-order values of tree nodes, where for any pair of nodes (say u and v) in a tree, T, u is an ancestor of v if, and only if, u appears before v in the pre-order traversal of T and after v in the post-order traversal of T. To determine the parent-child relationship between two nodes, the level value is also added to each node label (i.e. in as 3-tuple $< start, end, level >$). Although the main structural relationships can be determined efficiently, the insertion of a new node causes the re-labelling of all its ancestors. Figure 3.1 shows the XML tree of the 'School' example labelled using the interval-based labelling scheme and the re-labelling cost (indicated by the black nodes) when a new node (a) is inserted.

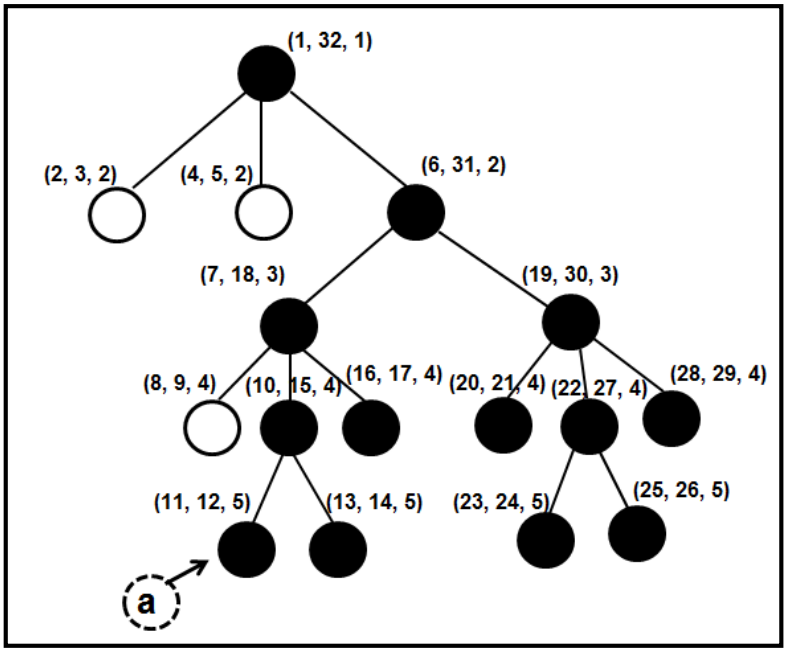


Figure 3.1 Interval-based labelling scheme

Many researchers have tried to solve the re-labelling problem: (Li and Moon, 2001) and (Zhang et al., 2001) have both assigned each node a pair of an extended pre-order and a range of descendants. (Li and Moon, 2001) alleviated the re-labelling problem by leaving gaps at the initial labels; the reserved space allocated within the range can either allow a limited number of insertions or result in wasted storage space if no insertions occur. (Amagasa et al., 2003) represented the start and end positions as floating-point values to extend the intervals. In a practical sense, the representation of floating-point numbers in a computer is limited to a fixed numbers of bits (Li and Ling, 2005b) (Li et al., 2008). Therefore, the approach of (Amagasa et al., 2003) does not eliminate re-labelling because a fixed place can execute up to 18 insertions when the initial labels are consecutive integers (Liu and Zhang, 2016) (Tatarinov et al., 2002).

(Yun and Chung, 2008) introduced the idea of adding a nested-tree structure to the interval-based labelling scheme, whereby the XML data insertion must be considered as adding a new sub-tree into the original XML tree. To insert a nested-tree structure into the original interval-based labelling scheme, each node is labelled as a 4-tuple $< document\ identifier, start\ list, end\ list, level\ of\ depth >$, leading to very long labels. Nevertheless, when the insertion size of a new sub-tree is larger than the available space, the entire post-order sibling sub-trees must be re-labelled.

To overcome these limitations, (Min et al., 2009) introduced the EXEL binary encoding algorithm to generate ordinal bit strings as start and end values. EXEL stores the

parent start value instead of the node level value to enhance the query processing time. However, including the parent-start value increases the label size (Al-Shaikh et al., 2010) and so, in the case of frequent insertions occurring repeatedly before or after a particular node (known as skewed insertions) the EXEL label size increases rapidly and leads to overflow problems (discussed in Section 4.3).

In (Lizhen and Xiaofeng, 2013), a Triple-code is generated by a depth first traversal in the form of 3-tuples $< start\ position, end\ position, parent\ id >$. In this approach, the level number within label intervals is replaced by the node's 'parent-id', making it straightforward to obtain parent/child and sibling relationships. Ancestor/descendant and document order can be determined, as in (Li and Moon, 2001). To identify the LCA between two nodes u and v in the Triple-code approach, all the ancestors of u and v must be traversed in order to create the paths from u and v to identify their LCA. So, for an XML tree of size, N, and depth, D,, the worst-case scenario for determining the LCA querying cost is $O(D)$. The authors reduced this cost by proposing the iTriple-code which creates ordered ancestor lists for leaf nodes that include the start value of all the ancestors from the root to the leaf's parent node. A pointer to the ancestor list is added as a fourth component of the Triple-code label value. Although this procedure reduces the LCA query performance cost to $O(\log D)$, when an XML tree is updated ancestor lists need to be updated too.

The Region-based Labelling scheme (ReLab) introduced by (Subramaniam et al., 2014a) uses $< level,\ ordinal,\ rID >$ to label XML tree nodes, where $level$ is level number of the node. The $ordinal$ value is the unique ID assigned to the node using a depth-first traversal, and rID is the ordinal of the rightmost sibling. In comparison to some other interval-based labelling schemes, such as Dietz's labelling scheme (Dietz, 1982) and the region-numbering scheme (Zhang et al., 2001), ReLab generates labels faster due to its greater simplicity in computing the intervals (Haw and Amin, 2015). However, ReLab (Subramaniam et al., 2014a) is a static labelling scheme, and does not support dynamic XML data (Haw and Amin, 2015) (Liu and Zhang, 2016).

3.3.3 Summary of Interval-based Labelling Schemes

In general, interval labelling schemes seek to determine structural relationships between nodes by using containment information, whereby node identifiers are represented as intervals (Liu and Zhang, 2016) (Zhuang et al., 2011) (Yu et al., 2005). Furthermore, these approaches generate very long labels, and so require and consume, large amounts of storage. It is difficult to decide the initial size of the

intervals that minimises storage cost whilst avoiding repetitive re-labelling in a dynamic XML environment (Sans and Laurent, 2008) (Ghaleb and Mohammed, 2015) (Li and Moon, 2001) (Liu and Zhang, 2016) (Yu et al., 2005). With the extent of available data on frequently updated XML applications, it is difficult to determine in advance either the actual data size or the number of possible updates. In addition, although interval-based labelling schemes establish ancestor/descendant, parent/child and document order more efficiently than other schemes, they cannot process sibling or LCA structural relationships (Lizhen and Xiaofeng, 2013) (Ghaleb and Mohammed, 2015) (Subramaniam et al., 2014a). Due to these limitations, an interval-based labelling scheme typically is not ideal for use with dynamic XML data (Liu and Zhang, 2016) (Ghaleb and Mohammed, 2015) (Kaplan et al., 2002) (Haw and Amin, 2015) (Su-Cheng and Chien-Sing, 2009).

3.4 Prefix-based Labelling Schemes

3.4.1 Structure and Concept

Prefix-based labelling schemes (Sans and Laurent, 2008) (Tatarinov et al., 2002) (Liu and Zhang, 2016) (Haw and Lee, 2011) (Ghaleb and Mohammed, 2015) (Xu et al., 2009) (O'Connor and Roantree, 2010a) (Liu et al., 2013) directly encode a node's parent label in an XML tree as the prefix of its label. Each node label in the tree comprises the parent's label concatenated with the node's identifier (self-label), and a delimiter, ".", is used to separate the label of the ancestor nodes at every level. Thus, determining the ancestor/descendant and parent/child relationships between two nodes is simply one of finding if one label is a prefix to the other. As containment of the path information within each prefix-based label facilitates the query processing, it has been the common choice for XML keyword querying (Sun et al., 2007) (Guo et al., 2003) (Sans and Laurent, 2008) (Li et al., 2014) (Zhang and Sun, 2011) (Bo et al., 2012) (Lu et al., 2011b). In particular, the Dewey Order labelling scheme (Tatarinov et al., 2002) has recently become common for research in XML query processing and indexing schemes (Zhou et al., 2016) (Li et al., 2014) (Liu and Chen, 2012) (Lou et al., 2012) (Zeng et al., 2013), due to its simplicity.

The Dewey Order labelling scheme was proposed by (Tatarinov et al., 2002) based on the Dewey decimal classification system for the organisation of library collections (Dewey, 1876). When using Dewey Order, each node is labelled as a vector denoting the path from the root node in an XML tree to the current node, whereby, the number of delimiters in a node's prefix-label is equal to the level number of that node. If the

root level is 0, then number of delimiters in a label is equivalent to the node's level number. Based on this mechanism, the structural relationships between nodes can be directly determined from their labels as follows: given two nodes u and v in a rooted ordered tree, T, if $label(u)$ is a prefix of $label(v)$, then u is an ancestor of v. If node u is an ancestor of v and $level(v) = level(u) + 1$, then u is a parent of node v. Nodes that share the same parent label as their prefixes and are in the same level of the XML tree are sibling nodes. Figure 3.2 below shows an XML tree labelled by the Dewey Order scheme.

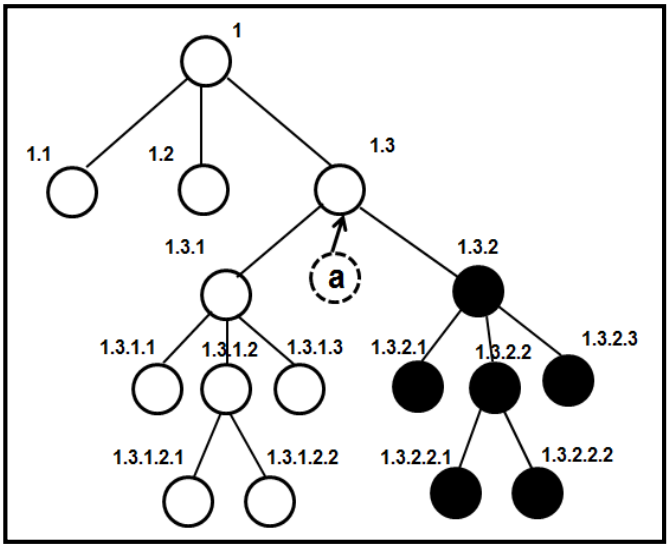


Figure 3.2 Prefix-based labelling scheme - Dewey Order

Despite the popularity of Dewey Order (Ghaleb and Mohammed, 2015) (Cohen et al., 2010) (Xu et al., 2009), it is not applicable for dynamic XML data. For example, inserting a new sibling node requires re-labelling all its right sibling nodes along with their descendants, as shown in Figure 3.2 (black circles indicate the re-labelled nodes after inserting node "a"). Several researchers (Li and Ling, 2005b) (Li and Ling, 2005a) (O'Connor and Roantree, 2012) (Mirabi et al., 2012) (Duong and Zhang, 2005) have proposed prefixed labelling schemes to support dynamic XML. Others (O'Neil et al., 2004) (Lu et al., 2005b) (Xu et al., 2009) (Liu et al., 2013) (Liu and Zhang, 2016) have investigated Dewey Order labelling properties to convert them into a dynamic XML labelling approach. The next section presents the strength and weakness of the existing dynamic prefix labelling schemes.

3.4.2 Related Schemes

With the intention of enhancing Dewey Order, (O'Neil et al., 2004) designed the ORDPATH labelling scheme that reserves negative-even integers for later nodes insertions to avoid re-labelling. Although this technique allows for a limited number of insertions, such gaps left between label values waste half of the storage (Liu et al., 2013) (Härder et al., 2007) (Li and Ling, 2005b) (Xu et al., 2009) (Haw and Lee, 2011). In addition, the complexity of the decoding mechanism in ORDPATH has a detrimental effect on XML query processing (Xu et al., 2009) (Li et al., 2006a) (Li et al., 2008) (Hye-Kyeong and SangKeun, 2010).

Later, (Lu et al., 2005b) proposed an extended Dewey labelling scheme that basically adds the elements tag names as a part of their Dewey labels. This feature speeds up twig pattern query matching (Hachicha and Darmont, 2013) (Lu et al., 2005a) by accessing only the leaf nodes that contain the labels of elements satisfying the query. In order to include elements tag names within node labels, the DTD of the XML data must be known. Otherwise, before assigning XML tree nodes labels, the whole XML document must be scanned at least once by a depth-first traversal in order to know the document's schema information called "*child names clue*". Such a process is time consuming (Yun and Chung, 2008).

The extended Dewey labelling scheme performs well in evaluating twig pattern query matching using the TJFast (which stands for "Twig Join Fast" algorithm) (Lu et al., 2005b) (Lu et al., 2005a) proposed by the same authors. However, the extended Dewey labelling has a number of disadvantages. The mapping process is a requirement to determine if an element name from its integer value makes the computation methods very expensive (Chiew et al., 2014a) (Haw and Lee, 2011). Apart from this, including XML tree elements names within their labels increases the label size even further, and most importantly this labelling scheme still does not support dynamic updates in XML trees since this requires the reconstruction of the "*child names clue*" data after insertions (Yun and Chung, 2008) (Liu and Zhang, 2016).

(Xu et al., 2009) revisited the notion of the Dewey Order encoding scheme, proposing two fully dynamic labelling schemes named DDE and CDDE (which stands for Compact DDE). DDE generalises the concept of vector order and vector equivalence over Dewey labels. Although the initial DDE labels are the same as those of the Dewey Order, the semantics of a DDE label are a sequence of vector codes

represented in the form $v_1.v_2 \ldots v_m$, where $v_1 = (x, y_1), v_2 = (x, y_2), \ldots, v_i = (x, y_i), \ldots v_m = (x, y_m)$, and whereby all the vector codes of a DDE label share a common x-axis component (Xu et al., 2012). In order to support dynamic updates, the authors (Xu, Ling et al. 2009) defined appropriate DDE labelling order properties (such as the pre-order relation and equivalence/in-equivalence relation) between DDE labels. Therefore, based on mathematical and logical equations, the determination of the structural relationships between nodes from their label values is preserved. DDE considers four cases of insertion, as shown in Figure 3.3:

Case A: Inserting before the leftmost sibling (e.g., node u): the new label is created by decrementing the local order value of the leftmost sibling by 1; here negative values are permitted.

Case B: Inserting after the rightmost sibling (e.g., node w): the new label is created by incrementing the local order value of the rightmost sibling by 1.

Case C: Inserting between two siblings (say X and Y; e.g., node v): the new label is assigned as the midpoint vector $X + Y$ (which is equal to $x_1 + y_1.x_2 + y_2.\ldots.x_m + y_m$); i.e., adding each m component in X to its corresponding component in Y.

Case D: Inserting a child into a leaf node (e.g., node z): where the new label is created by concatenating the parent label and the digit "1".

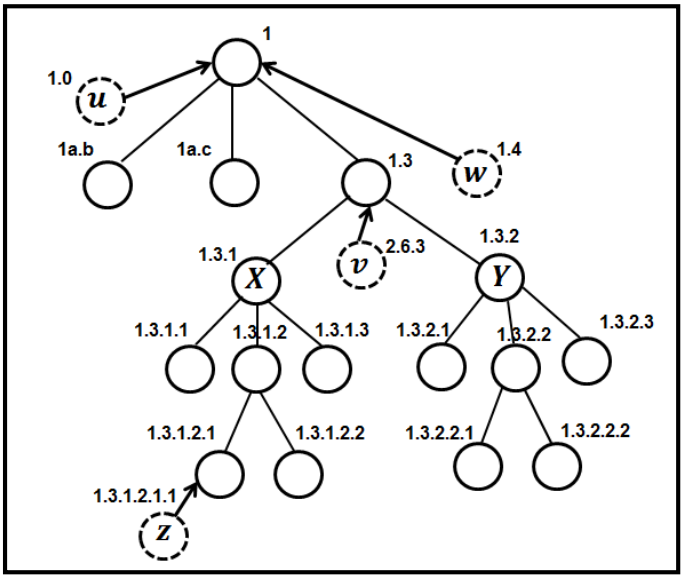


Figure 3.3 DDE labelling scheme

CDDE is a modified version of DDE introduced to improve the performance of DDE for insertions by allowing initial labels to be negative values. Nevertheless, as shown by (Xu et al., 2009), the enhancement of DDE performance via CDDE is insignificant in terms of updating time and label size. Overall, DDE and CDDE generate large labels at the cost of extra storage (Liu et al., 2013) (Yanghua et al., 2012) especially when frequent insertions occur between consecutive siblings due to the large gap generated by applying the midpoint vector $X + Y$ technique. Furthermore, DDE is not suitable for determining the structural relationships in multiple XML documents and requires an extra document identifier to distinguish labels within several XML documents (Liu et al., 2013) (Liu and Zhang, 2016) (Assefa and Ergenc, 2012).

More recent work by (Liu et al., 2013) has led to the proposal of the DFPD labelling scheme, which represents Dewey labels by float-point numbers. As in DDE, DFPD initially labels the XML tree based on Dewey labels and handles insertions before the leftmost sibling, after the rightmost sibling, and under a leaf node by applying the same techniques used in DDE (i.e. cases A, B, and D above). To insert a new sibling between two consecutive nodes X (labelled $x_1.x_2.\ldots.x_m$) and Y (labelled $y_1.y_2.\ldots.y_m$), the new label is computed by:

$$\left(a_1.a_2 \ldots\ a_{m-1}.\frac{(k_a * a_m) + (k_b * b_m)}{k_a * k_b}\right) \qquad \textit{Equation 1}$$

Where k_i is the smallest possible integer that makes the result of $(K_i * i_m)$ a float-point number such that its decimal part is 0. This makes the last digit of the new label to be stored a real number. For instant, assume a new node, Z, is inserted between node $X = (1.45.(302/3))$ and node $Y = (1.45.(503/5))$, then:

$$Label\,(Z) = (1.45.\frac{805}{8}) = (1.45.(3\,X\,\frac{302}{3} + 5\,X\,\frac{503}{5})\,/\,(3 + 5))$$

Recently, the authors enhanced the performance of DFPD by introducing the DPLS labelling scheme (Liu and Zhang, 2016) that re-uses deleted label values -if they exist-to lower the growth rate of label sizes when insertion and deletion take place alternatively. The DPLS basically computes the fractional part of the self-label component in the case of an insertion between two consecutive siblings through equation (2) instead of equation (1):

$$\left(a_1.a_2 \ldots\ a_{m-1}.\frac{(a_m + b_m)}{(k_a + k_b)}\right) \qquad \textit{Equation 2}$$

As in (Amagasa et al., 2003), the floating point numbers generated by DFPD and DPLS are of limited accuracy since the mantissa is actually represented by a fixed number of bits and can be extended by as many as 2 bits per insertion, leading to overflow problems (see Chapter 4, Section 4.3) (Xu et al., 2012). To address this limitation, when storing the fractional part DFPD and DPLS have adopted the ORDPATH (O'Neil et al., 2004) encoding technique, in which the labels can be assigned by a successive variable-length storage format (see Chapter 4, Section 4.4.4). However, as in ORDPATH, the complexity of decoding decelerates the XML querying process.

Similarly, (Mirabi et al., 2012) proposed a labelling scheme based on fractional numbers. Here, encoding n ordinal decimal numbers by recursively assigning the middle fractional number in the range $[0, 1]$ to the middle decimal number between 1 and n using the author's proposed algorithm, which is called fractional number generation (FNG). For example, for $n = 16$, the middle number is $(16/2)$= 8, which corresponds to the middle fraction number $((0 + 1)/2)$ = $(1/2)$. Then, the fractional number assigned to the decimal number 8 is $(1/2)$; to 4, it is $(1/4)$; to 6, it is $(3/8)$, and so on. After the fraction values are generated, they are mapped into bit-string codes, which are used to label an XML tree. Nonetheless, according to the authors, the bit string codes generated by their FNG algorithm differ depending on the value of n. For instance, bit string codes for a set of ordinal decimal numbers between 1 and 3 differ from those generated for a set of decimal numbers between 1 and 10. Determining the original value of n is not clear in the work of (Mirabi et al., 2012). Moreover, when it is applied over prefix labelling schemes (e.g., Dewey IDs) it has shown poor performance in comparison to other labelling schemes in terms of storage space, querying time and updating XML data. As the authors stated a possible reason of this is including the delimiter "." within the label values.

A more recent XML prefix-based labelling scheme that is also based on fractions, called DPESF Encoding, was proposed by (He, 2015), in which the mid-point of self-labels between two consecutive sibling nodes is stored in Numeric-Character format. To achieve this, (He, 2015) defined the rule to map each digit $n \in N = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ in the numerator to a matching character $c \in C = \{A, B, C, D, E, F, G, H, I, J\}$. For instance, $(125/14)$ is expressed as $BCF14$. The author indicated the need to adjust their DPESF labelling scheme in order to fully support dynamic updates in XML data (He, 2015). Furthermore, as with other alpha-numeric prefix-based labelling schemes (presented next), (Duong and Zhang, 2005) (Duong

and Zhang, 2008) (Khaing and Ni Lar, 2006) (Assefa and Ergenc, 2012), DPESF is prone to overflow problems due to repetitive long labels that can lead to collisions during XML querying (Subramaniam and Haw, 2014b) (Assefa and Ergenc, 2012) (Khaing and Ni Lar, 2006) (O'Connor and Roantree, 2010a) (Su-Cheng and Chien-Sing, 2009).

Alphanumeric prefix-based labelling schemes (Duong and Zhang, 2005) (Duong and Zhang, 2008) (Khaing and Ni Lar, 2006) utilise both integers and letters to construct XML tree node labels. Usually the number represents the level of an XML tree node followed by letter(s) representing the positional identifier of a parent node, concatenated with the separator "." before adding letter(s) signifying the current node's self-label. As the number of siblings within the same level increases, using the alphanumeric update technique may lead to redundancy in some label values during arbitrary insertions (Sans and Laurent, 2008) (Subramaniam and Haw, 2014b) (Khaing and Ni Lar, 2006). For example, consider the labelled XML tree shown in Figure 3.4; according to the LSDX (Labelling Scheme for Dynamic XML data) (Duong and Zhang, 2005) labelling algorithm, there are two possible values that can be assigned to the new node, z, but both possible values cause collisions. As shown in Figure 3.4, z can be assigned the label value "$2acb.cb$", which causes a collision with node y or z = "$2acb.b$" as this is a duplicate of the label of node m. Likewise, when node v is inserted before node x, a duplicate label value "$1a.cb$" is generated.

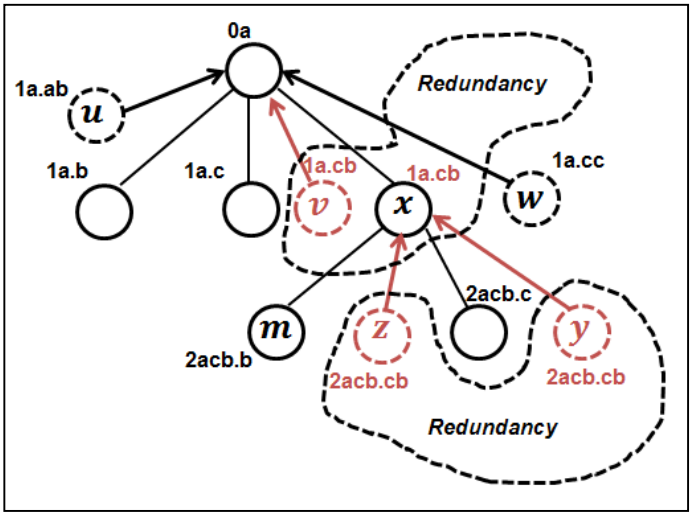


Figure 3.4 LSDX labelling scheme (possible collision cases)

To prevent such collisions, Assefa and Ergenc (Assefa and Ergenc, 2012) have proposed the OrderBased labelling scheme. The main concept of this scheme is to keep a global level based on horizontal order and parent order. OrderBased differs

from other prefix-based labelling schemes by initialising the node labels for the whole tree on a level-by-level basis, irrespective of ancestors. For each level of the XML tree starting from the most left node OrderBased starts assigning letters as node self-labels from "b","c", …,"z","zb" … etc. Unlike other alphanumeric prefix-based labelling schemes, this numbering technique ensures the uniqueness of label values. However, because letters representing nodes' self-labels are assigned according to the nodes' horizontal distance from the left-most node in the same level, finding the ancestors of a given node requires repeated visits to all parent nodes of the previous levels. Therefore, determination of ancestor-descendant and LCA relationships is costly in this labelling scheme (Silberstein et al., 2005) (Haw and Amin, 2015).

Instead of using alphanumeric values, various prefix labelling schemes have been proposed that use binary strings to represent label values lexicographically (see Section 5.4 for lexicographical order definition). In ImprovedBinary (Li and Ling, 2005a), the root is first labelled as an empty string. Then, the leftmost child and the rightmost child of the root are labelled as "01" and "011", respectively. This is to ensure positions for new node insertions as first sibling and last sibling (Duong and Zhang, 2008). Afterwards, the middle nodes are sequentially labelled based on two main rules:

- If $left\ selflabel\ size \leq right\ selflabel\ size$ then the label of the middle node is formed by changing the last digit of the right self-label to zero and is concatenated to the digit "1".
- Otherwise, the middle node label is the same as the left self-label, and concatenated to the digit "1".

When updating an XML tree, repeated insertions before the first rightmost child may cause duplicate node labels. Hence, ImprovedBinary cannot completely avoid re-labelling (O'Connor and Roantree, 2010a) (Duong and Zhang, 2008) (Li and Ling, 2005b).

To overcome the limitation of the ImprovedBinary scheme several prefix labelling schemes have been proposed based on lexicographical order (Section 5.4) over binary strings. These include:- Cohen's labelling scheme (Cohen et al., 2010), P-PBiTree (Yu et al., 2005), VLEI (Kobayashi et al., 2005), IBSL (Hye-Kyeong and SangKeun, 2010), EBSL (O'Connor and Roantree, 2010b), V-CDBS (Li et al., 2008), XDAS (Ghaleb and Mohammed, 2013), and dynamic XDAS (Ghaleb and Mohammed, 2015). However, as with the ImprovedBinary approach all these schemes suffer from

huge label sizes and require re-labelling after frequent insertions (Kobayashi et al., 2005) (O'Connor and Roantree, 2010a) (Duong and Zhang, 2008) (Li and Ling, 2005b) (Duong and Zhang, 2005).

Other prefix-based labelling schemes have used quaternary codes that are also based on the lexicographical order between node self-labels to support dynamic XML data. That of (Li and Ling, 2005b) is the first to use quaternary encoding (QED) to overcome the re-labelling problem in dynamic XML trees. QED encoding replaces the delimiter "." with the digit "0", and used only the digits "1", "2", and "3" to generate self-labels by applying a recursive division function. However, such quaternary labels increase in size dramatically in the case of skewed insertion; by 2-bits per insertion. To control the growth rate of quaternary labels, (O'Connor and Roantree, 2012) introduced the SCOOTER labelling scheme.

According to (Chiew et al., 2014a), SCOOTER (O'Connor and Roantree, 2012) is the most compact of dynamic labelling schemes, which controls the growth of label size when an XML database is updated via automatic reuse of the smallest deleted node label available. Based on this observation, this thesis is developed using the SCOOTER scheme in order to address the research question and objective (discussed in Chapter 5). Therefore, for a comprehensive understanding of this thesis, it is necessary to review the SCOOTER labelling scheme in some detail.

3.4.3 The SCOOTER Labelling Scheme

The SCOOTER scheme (O'Connor and Roantree, 2012) provides Scalable, Compact, Ordered, Orthogonal, Ternary Encoded, and Reusable labels (and hence the acronym SCOOTER). This section presents the initialisation and insertion mechanisms of the SCOOTER labelling scheme (here labels and self-labels are used interchangeably).

This scheme initialises labels by first obtaining the maximum possible label size (called $maxLabelSize$) based on the total number of child nodes (called $ChildCount$) using a logarithmic function in base-3. The first (left-most) child is assigned a label that consists of ($maxLabelSize - 1$) copies of digit 1, followed by the digit 2. Then, the labels of the remaining sibling nodes were determined lexicographically based on the label of the node to their immediate left. Figure 3.5 shows an example of initial self-labels generated by the SCOOTER scheme to represent the six child nodes of an element p (i.e. $ChildCount$ = 6 and so the $maxLabelSize$ is 2).

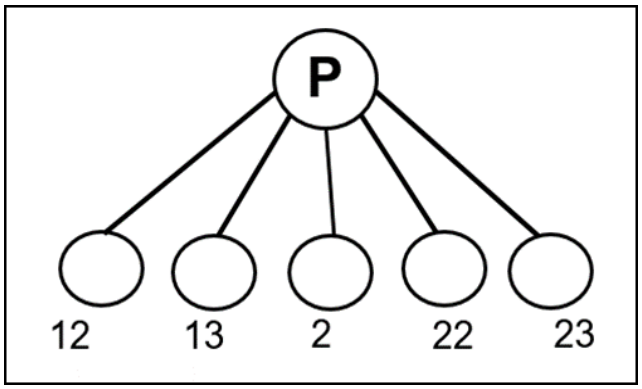


Figure 3.5 Example of initial SCOOTER self-labels

The SCOOTER labelling scheme controls the expansion of the quaternary labels by a compact adaptive growth mechanism to handle insertions. Three types of sibling node insertions are considered by this scheme: inserting after the right most node, inserting before the left most node, and inserting between two consecutive nodes.

When inserting after the right-most node, labelled $Nold$, the new node is assigned a $Nnew$ label based on the start digit value of $Nold$, as shown in Table 3.1:

Table 3.1 Inserted after the right-most-node, $Nold$ starts with d = 1, 2, or 3

Insert after (the right most node) $Nold$ that starts with $d \in \{1,2,3\} Inter$		
Condition	**Rule**	**Example**
If $d = 1$	$Nnew$ is "2"	$Nold = "112"$ then $Nnew = "2"$
If $d = 2$	$Nnew$ is "3"	$Nold = "22"$ then $Nnew = "3"$
If $d = 3$	Apply adaptive growth mechanism	$Nold = "3312"$ then $Nnew = "3313"$

Basically the SCOOTER's growth-adaptive mechanism treats a node's label (say $Nold$) as a combination of a prefix and a postfix string. The prefix string represents all consecutive '3's at the beginning of $Nold$, and the rest are the postfix string. Based on the value and length of the $Nold$ postfix string, $Nnew$ is allocated as described in Table 3.2:

Table 3.2 Inserted after the right-most-node, $Nold$ starts with 3

Insert after $Nold$ that starts with $d = 3$; use adaptive growth mechanism		
Condition	**Rule/Action**	**Example**
If $postfix$ is empty i.e. $Nold$ consists of all '3's	1 $Nnew$ postfix length = $Nold$ prefix length + 1 2 Compute $Nnew$ postfix by calling the SCOOTER's initialisation method passing $ChildCount$ equals to $Nnew$ postfix length. 3 $Nnew = Nold \oplus Nnew\ postfix$	$Nold =$ "33" then $Nnew =$ "3312"
If $postfix$ is NOT empty	*1* $Nnew$ postfix length = $Nold$ postfix length *2* Compute $Nnew$ postfix by calling the SCOOTER's initialisation method on the basis that $ChildCount$ equals to $Nold$ postfix length. *3* $Nnew =$ $Nold\ prefix \oplus Nnew\ postfix$	$Nold =$ "3312" then $Nnew =$ "3313"

Notice that after $(3_{postfix\ length} - 1)$ insertions the label size increases by $(postfix\ length + 1)$. This is because when $Nold$ consists completely of '3's, the new $maxLabelSize$ allocated for $Nnew$ is increased as follows:

- $Nnew$ prefix length = $Nold$ prefix length + $Nold$ postfix length; i.e., length of $Nold$
- $Nnew$ postfix length = $Nold$ prefix length + 1
- New maximum label size = $Nnew$ prefix length + $Nnew$ postfix length

The same adaptive growth mechanism is applied for insertion before the left-most node, but takes into account the number of consecutive '1's at the beginning of $Nold$ instead of consecutive '3's. Table 3.3 shows some examples of SCOOTER labels generated when 10 new nodes are inserted repeatedly before the left most node, n_{old}.

Table 3.3 Example of skewed insertions before the left most node in SCOOTER

	Node labels	Node labels	Node labels	Node labels
Insert after node	**1123**	**1112**	**22313**	**3333**
1	1122	11112	22312	3332
2	112	111112	223	333
3	1112	1111112	222	332
4	11112	11111112	22	33
5	111112	111111112	2	32
6	1111112	1111111112	12	3
7	11111112	11111111112	112	2
8	111111112	111111111112	1112	12
9	1111111112	1111111111112	11112	112
10	11111111112	11111111111112	111112	1112

The SCOOTER scheme has also provided an adaptive growth mechanism to handle insertions between the two nodes n_{left} and n_{right}, labelled as $Nleft$ and $Nright$, respectively (see Figure 3.6). In this case, generating a new label (say $Nnew$) relies on the length of $Nleft$ and $Nright$ as follows:

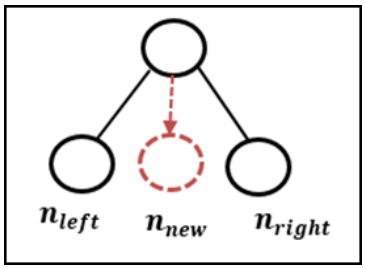


Figure 3.6 Insert between nodes

- **If $Nleft$ is shorter than $Nright$:**

In this case, $Nleft$ could be a prefix of $Nright$. If it is, then $Nnew$ is allocated based on an $Ntemp$ value which corresponds to the remains of $Nright$ after trimming the prefix part that matches that of $Nleft$. The algorithm then locates $Nnew$ based on the first digit (say d_f) in $Ntemp$ as follows (see Table 3.4):

Table 3.4 Inserted between nodes, $Nleft$ is a prefix of $Nright$

When $Nleft$ is a prefix of $Nright$; where $Nright = Nleft \oplus Ntemp$ and $Ntemp$ starts with a digit $d_f \in \{1, 2, 3\}$		
Condition	**Rule/Action**	**Example**
If $d_f = 3$	$Nnew$ is $Nleft \oplus$ '2'	$Nleft =$ "23" , and $Nright =$ "2333" then $Nnew =$ "232"
If $d_f = 2$	$Nnew$ is $Nleft \oplus$ '12'	$Nleft =$ "23" , and $Nright =$ "232" then $Nnew =$ "2312"
If $d_f = 1$	*1* $Nnew = Nleft$ *2* Locate the position p of the first not '1' digit D in $Ntemp$ *3* Add the '1's at the beginning of $Ntemp$ to end of $Nnew$. *4* If D is '3', $Nnew \oplus$ '2' *5* If D is '2', $Nnew \oplus$ '12'	$Nleft =$ "23" , and $Nright =$ "23112" then $Nnew =$ "231112"

However, if $Nleft$ is not a prefix of $Nright$ but is shorter than $Nright$, this indicates that there is at least one deleted node between $Nleft$ and $Nright$. To re-use such a deleted label for $Nnew$, first the position ($p \geq 1$) of first different digit between $Nleft$ and $Nright$ is detected, and accordingly the following actions are taken (see Table 3.5):

Table 3.5 $Nleft$ shorter that, but not a prefix of, $Nright$

When $Nleft$ is not a prefix of $Nright$ but is shorter than $Nright$, where the first different digit D between $Nleft$ and $Nright$ in position $p \geq 1$		
Condition	**Rule/Action**	**Example**
If $p = 1$; i.e. $Nleft$ and $Nright$ differ by their first digit.	$Nnew$ is the SCOOTER self-label after D; obtained by call SCOOTER next-sibling method used in the initialisation process assuming $maxLabelSize$ is 1.	$Nleft =$ "12" , and $Nright =$ "2112" ; so $D =$ "1" then $Nnew =$ "2"
If $p > 1$	*1* $Nnew$ is assigned the substring of $Nleft$ from its start up to $p - 1$; i.e. the similar digits at the beginning of $Nleft$ and $Nright$. *2* $S =$ is the SCOOTER self-label after D; obtained by call SCOOTER next-sibling method used in the initialisation process assuming $maxLabelSize$ is 1. *3* $Nnew = Nnew \oplus S$	$Nleft =$ "212" , and $Nright =$ "2232" ; so $D =$ "1", and $p = 2$ then $Nnew =$ "22"

- **If $Nleft$ is longer than $Nright$:**

In this case, the adaptive growth method applied for insertion after the right most node is used to allocate $Nnew$ as a new node inserted after $Nleft$. As an example, when $Nleft =$ "1333" and $Nright =$ "2", then $Nnew =$ "13332".

- **If both $Nleft$ and $Nright$ are the same size:**

When both $Nleft$ and $Nright$ are the same size, there are two possible scenarios, as illustrated in Table 3.6:

Table 3.6 $Nleft$ and $Nright$ are the same size

When $Nleft$ and $Nright$ are the same size, where the first different digit at position p		
Condition	**Rule/Action**	**Example**
If $Nleft$ and $Nright$ are lexicographically immediate neighbours (i.e. differ only in their last digit).	$Nnew$ = $Nleft \oplus$ "2"	$Nleft$ = "12" , and $Nright$ = "13" ; then $Nnew$ = "122"
Otherwise; i.e. $Nleft$ and $Nright$ are not lexicographically neighbours. This indicates there is a deleted label between $Nleft$ and $Nright$.	*1* $Ntemp$ = substring($Nleft, 0, p-1$). This makes $Ntemp$ shorter than $Nright$. *2* The SCOOTER's algorithm used for insertion between $Ntemp$ shorter than $Nright$ (illustrated earlier) is invoked to generate $Nnew$.	$Nleft$ = "122" , and $Nright$ = "133" ; → $Ntemp$ = "1", then $Nnew$ = "13"

In general, the adaptive growth-rate mechanism allows SCOOTER to generate more compact labels than the QED encoding method, but after ($3^{postfix\ length}-1$) insertions the label size increases by ($postfix\ length+1$) leading to very large labels. Thus, as with most of prefix-based labelling schemes, SCOOTER also suffers from overflow problems (Section 4.3), especially in skewed insertions where the label size grows rapidly (Ghaleb and Mohammed, 2015) (Chiew et al., 2014a) (Ghaleb and Mohammed, 2013). Figure 3.7 shows an XML tree labelled by SCOOTER with the new node labels (the black nodes) after 100 skewed insertions to the right of node n_1.

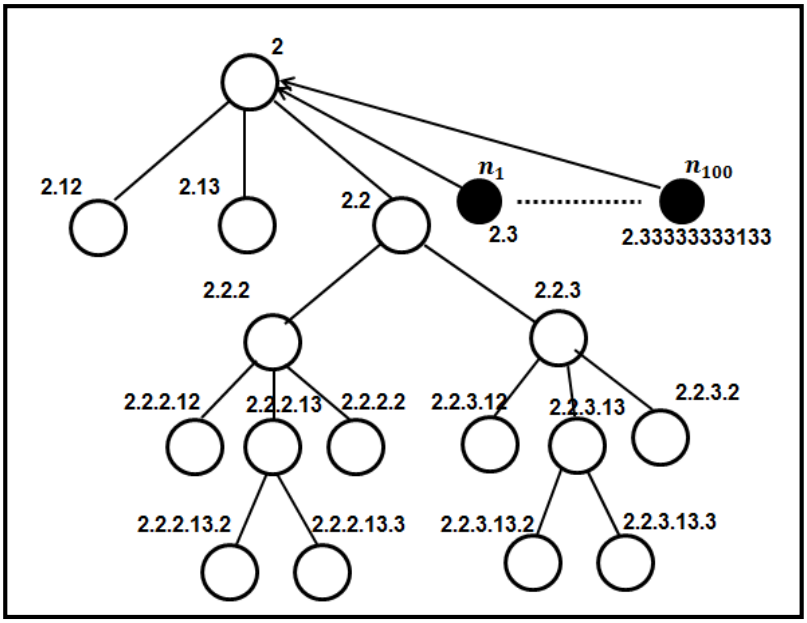


Figure 3.7 XML tree labelled by the SCOOTER scheme

3.4.4 Labelling Schemes for Re-using Deleted Labels

A substantial amount of research has focused on the development of dynamic labelling schemes that are capable of supporting XML updates. Most of this research has been confined to the impact of XML updates on label size and the computational complexity of the update cost while ignoring node deletion as part of XML updates. So, in these labelling schemes, when a node is deleted its label is just marked as deleted (O'Connor and Roantree, 2010b).

Due to the possible occurrence of the overflow problem that may result from large labels being generated when updating XML data, there are very few XML labelling schemes that consider reusing deleted nodes' labels to control the growth of label size during insertions (Hye-Kyeong and SangKeun, 2010) (O'Connor and Roantree, 2010b) (Li et al., 2006b) (Liu and Zhang, 2016) (O'Connor and Roantree, 2012). When a new node is inserted at the same position as a deleted node in an XML tree, the new label generated is usually larger than the deleted label. If a deleted label with a smaller size is used instead, then the increase of label size can be better controlled. Motivated by this concept, the IBSL (Hye-Kyeong and SangKeun, 2010) and EBSL (O'Connor and Roantree, 2010b) XML labelling schemes were implemented to re-use deleted nodes' labels of the type binary strings. Similarly, (Li et al., 2006b) introduced re-used QED code, which has been applied on quaternary strings as an enhancement to the QED labelling scheme (Li and Ling, 2005b). Nonetheless, these schemes have not only failed to generate the smallest available deleted label, but also were

unreliable and have produced duplicated labels that may cause ambiguity during the XML query process. For example, Figure 3.8 (adapted from (O'Connor and Roantree, 2010b)) shows how the re-used QED scheme (Li et al., 2006b) generates duplicated labels when nodes C and D are inserted in that order before node A.

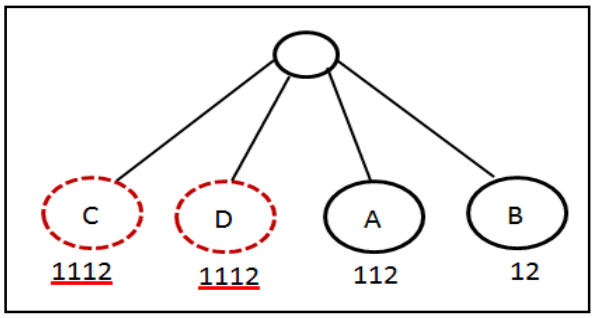


Figure 3.8 Reusing QED code example

Recently, the DPLS (Liu and Zhang, 2016) XML labelling scheme was developed to re-use deleted labels using their proposed technique, named "Reduction of a fraction operation" to minimise the storage space cost. This scheme focused on re-using deleted labels, in particular insertion cases, mainly when insertions and deletions take place alternatively between adjacent sibling nodes. Nonetheless, the complexity of decoding the fractional part of DPLS labels slows down the XML querying process, as discussed in Section 3.4.2.

On the other hand, the SCOOTER labelling scheme (O'Connor and Roantree, 2012) was successfully designed to support dynamic XML and reuses deleted labels. However, the insertion methods applied by SCOOTER attempt to re-use only the smallest quaternary codes "2" and "3" if available, and do not guarantee the re-use of every deleted labels. The adaptive growth mechanism used in the SCOOTER scheme treats node labels as a combination of a prefix and a postfix string. As a result, after $(3^{postfix\ length} - 1)$ insertions, the label size increases by $(postfix\ length + 1)$, leading to very large labels, particularly in the case of skewed insertions (Ghaleb and Mohammed, 2015) (Chiew et al., 2014a) (Ghaleb and Mohammed, 2013). For example, it might be noticed from Table 3.3 in Section 3.4.3, that after relatively few insertions before the first child node, a newly generated label always starts with consecutive '1's followed by a '2', and then it begins to grow rapidly (by at least one digit per insertion).

3.4.5 Summary of Prefix-based Labelling Schemes

Prefix-based labelling schemes directly encode the parent of a node in an XML tree as the prefix of its label (Sans and Laurent, 2008) (Almelibari, 2015). The containment of the path information within each prefix-based label facilitates query processing, but because it is verbose prefix label sizes increases rapidly as the XML tree goes deeper (Haw and Lee, 2011) (O'Connor and Roantree, 2010a) (Kaplan et al., 2002) (Qin et al., 2017). Unfortunately, prefix labels naturally extend when XML data is updated via frequent insertions, causing overflow problems (Section 4.3).

Many researchers have presented dynamic prefix-based labelling schemes based on several data types such as integers (Tatarinov et al., 2002) (O'Neil et al., 2004) (Xu et al., 2009) (Liu et al., 2013) (Liu and Zhang, 2016), alphanumeric (Duong and Zhang, 2005) (Duong and Zhang, 2008) (Khaing and Ni Lar, 2006) (Assefa and Ergenc, 2012), binary strings (Li and Ling, 2005a), (Hye-Kyeong and SangKeun, 2010) (O'Connor and Roantree, 2010b) (Ghaleb and Mohammed, 2015) and quaternary codes (Li and Ling, 2005b) (Li et al., 2006b) (O'Connor and Roantree, 2012). Amongst these approaches, quaternary codes produce the most compact labels, except in the case of multiple skewed insertions. However, decoding large labels in quaternary labelling schemes is costly and slows down the query processing (O'Connor and Roantree, 2013) (Härder et al., 2007).

In spite of the drawbacks of prefix-based labelling schemes, this class of labelling scheme appears to be more suitable for encoding large-scale dynamic XML data than other categories of labelling scheme (Sans and Laurent, 2008) (Li et al., 2006a) (Alkhatib and Scholl, 2009). Furthermore, prefix-based labelling schemes can support all structural relationship types, but they are less efficient in determining the ancestor/descendant and parent/child relationships than interval-based labelling schemes. Whereas interval-based schemes need extra information to support sibling relationships (Lin et al., 2013) (Yun and Chung, 2008).

3.5 Multiplicative Labelling Schemes

3.5.1 Structure and Concept

Multiplicative labelling schemes (Weigel et al., 2005) (Yanghua et al., 2012) (An and Park, 2010) (Noor Ea Thahasin and Jayanthi, 2013) (Almelibari, 2015) (Subramaniam and Haw, 2014b) (Lee et al., 1996) (Kha et al., 2002) (Al-Shaikh et al., 2010) label

XML tree nodes in such a way that the structural relationships between nodes can be determined based on arithmetic computations (Haw and Lee, 2011) (Chiew et al., 2014a) (Su-Cheng and Chien-Sing, 2009) (Assefa and Ergenc, 2012). The main concept in these schemes is to allocate node labels using atomic numbers defined by arithmetic properties based on their labels. Accordingly, determining the structural relationships between nodes is achieved by analysing the arithmetic properties of the numerical labels using mathematical principles (Almelibari, 2015) (Assefa and Ergenc, 2012).

In the prime numbering scheme (Wu et al., 2004) and the group-based prime number labelling scheme (An and Park, 2010), each node is given a unique (unrepeated) prime number as a self-label based on a top-down approach. A node label is the product of a node's self-label and its parent's label (see Figure 3.9). Structural relationships between nodes in these schemes are established by applying modular functions on node labels, as explained below. Alternatively, the Me-labelling scheme (Subramaniam and Haw, 2014b) uses odd numbers and the multiplication-division operation to interpret the structural relationships between XML tree nodes (Haw and Amin, 2015). On the other hand, Vector-based (Noor Ea Thahasin and Jayanthi, 2013), DDE (Xu et al., 2009), Vector-encoding (Xu et al., 2007), LSVP (Zhang and Dong, 2010), Order-Centric (Xu et al., 2012), and Vector Order-Based (Zhuang and Feng, 2012b) labelling schemes have been proposed based on the mathematical principles of vector order.

Usually, multiplicative labelling schemes have the ability to simultaneously determine structural relationships and facilitate query processing (Liu and Zhang, 2016). However, the main drawback of this labelling scheme class is that they carry a high computational cost (Härder et al., 2007) (Min et al., 2007) (Liu and Zhang, 2016) (Haw and Lee, 2011) (Zhuang and Feng, 2012b) (Li and Ling, 2005b) (O'Connor and Roantree, 2012). Therefore, such labelling schemes are inappropriate for labelling large-scale XML documents.

3.5.2 Related Schemes

The most common multiplicative labelling scheme is the prime number labelling scheme introduced by (Wu et al., 2004). A prime number is a positive integer greater than 1 that it can only be divided by itself and 1. Based on this feature the prime number labelling scheme applies a depth-first traversal on XML tree to give each node a unique (unrepeatable) prime number as its self-label. Figure 3.9 shows an XML tree

labelled by the prime number labelling scheme. Starting from the root labelled with number "1" each node is labelled as a product of the node's self-label and its parent's label.

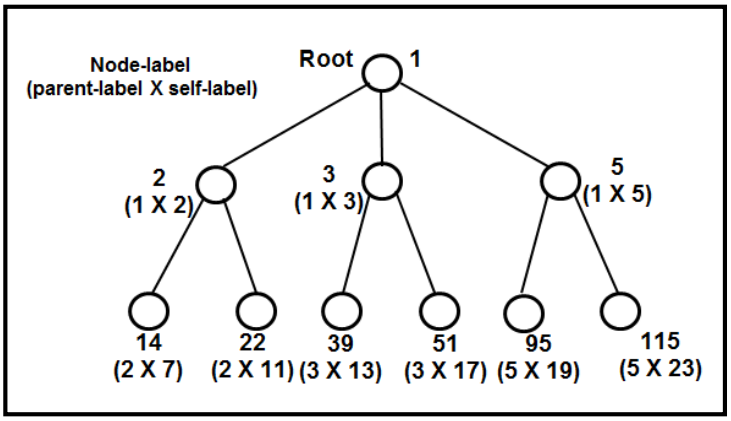


Figure 3.9 Prime number labelling scheme

In the prime-number labelling scheme, the ancestor-descendant relationship between two nodes can be determined using a modular function. For any two nodes u and v, u is an ancestor of v if, and only if, $Label(u)\ mod\ Label(v) = 0$. To preserve the document order when the XML document is updated, the prime number labelling scheme uses the Simultaneous Congruence (SC) values based on Chinese Remainder Theorem (Martins, 2009). The document order then can be obtained by:

$$SC\ mod\ prime\ number\ of\ selflabel\ =\ global\ document\ order \qquad \textit{Equation 3}$$

The SC value differs depending on the total number of nodes in the XML tree. Although the method used in the prime number labelling scheme (Wu et al., 2004) completely avoids re-labelling in case of XML updates, it requires the SC values to be re-computed when a node is inserted or deleted which consumes even more time than just re-labelling the appropriate nodes (Min et al., 2009) (Hye-Kyeong and SangKeun, 2010). In order to prevent SC values from growing overly large a list of SC values is indexed where each SC value represents the global document order for at least five nodes. However, for large-scale XML documents, the list of SC values is extremely large, which makes it costly in terms of storage and maintenance (Xu et al., 2009) (Mirabi et al., 2012) (Ahn et al., 2017b). Furthermore, to derive the whole sequence of a node's ancestors using the prime labelling scheme, additional SC index accesses are required which slows query processing (Härder et al., 2007). Nevertheless, the label's values are limited to prime numbers and so big gaps left between label values result in wasted of storage.

Another multiplicative labelling scheme also uses modular functions and multiplication operations to determine structural relationships is Branch code (Yanghua et al., 2012). Each node in the Branch code scheme is labelled as a quad-tuple $< b, g, h, d >$ that encodes information about its ancestors. For a node v, $b(v)$ is the b-code of the node v, which is used as its self-label; this is obtained as the summation of $g(v)$ and $h(v)$. The $g(v)$ value preserves the number of the siblings of node v's ancestors, whereas the $h(v)$ value holds the order of v's ancestor within its siblings. Both $g(v)$ and $h(v)$ values are computed using recursive functions tracing the node v's ancestor's siblings. Finally, $d(v)$ is the depth of node v within an XML tree, T.

Despite the fact that Branch code accelerates query processing by providing each node an informative label in terms of its ancestors, the recursive functions used to construct these labels are complicated and time consuming (Lizhen and Xiaofeng, 2013). The authors of Branch code (Yanghua et al., 2012) asserted that their proposed scheme has an inaccurate estimate of the probability of producing false positive results representing the structural relationships in very large or very deep XML trees. In addition, Branch code does not support document order and is not applicable for dynamic XML documents because it requires re-computing g and h values in the instance of new insertions (Lin et al., 2013).

Other multiplicative labelling schemes (Noor Ea Thahasin and Jayanthi, 2013) (Xu et al., 2009) (Xu et al., 2007) (Zhang and Dong, 2010) (Xu et al., 2012) (Zhuang and Feng, 2012b) have been introduced based on the arithmetical principles of vector order (Assefa and Ergenc, 2012). A vector, V, is an object with magnitude (weight) and direction (path) that can be represented as a binary tuple, $V = (x, y)$, where x and y are positive integers as illustrated in Figure 3.10; adapted from (Xu et al., 2007).

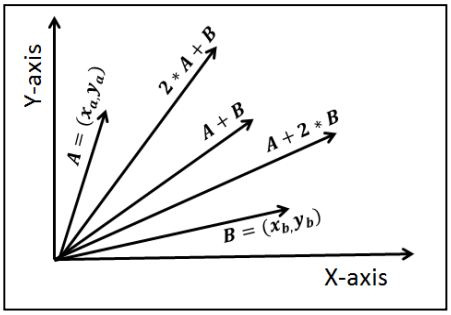


Figure 3.10 Graphical representations of vectors

In (Xu et al., 2007) Vector-encoding for labelling dynamic XML trees was designed using the interval-based labelling approach, as based on the following vectors properties:

1. A vector $V = (x, y)$ has an angle $\emptyset$ with respect to the x-axis, the gradient of V (denoted by $G(V)$) is computed as $G(V) = y/x$; where $G(V) \equiv \tan(\emptyset)$
2. For vectors $A = (x_a, y_a)$, $B = (x_b, y_b)$, and $C = A + B = (x_a + x_b, y_a + y_b)$, then $G(A) > G(C) > G(B)$, where $G(A) > G(B)$ if, and only if, $(y_a * x_b) > (x_a * y_b)$

Vector-encoding (Xu et al., 2007) represents interval-based labels in vector form. Each node is labelled as $< start\ vector, end\ vector, and\ level\ value >$, whereas the vectors' gradient values are used to preserve the order of the assigned vectors. When a new node C is inserted between nodes A and B, a vector value is allocated to C according to the second property above.

In (Xu et al., 2012), the authors have shown how Vector-encoding (Xu et al., 2007) can be applied to prefix-based labelling schemes as well. They have also stated that Vector-encoding performs better than QED (Li and Ling, 2005b) in skewed insertions. However, because each vector, V, is stored successively as start-vector and end-vector in UTF-8 encoding, the vector based labelling scheme suffers from overflow when the label size grows beyond the storage limit (O'Connor and Roantree, 2012).

To minimize the size of vector values so as to avoid overflow problems (Ni et al., 2012) introduced a numeric-based XML labelling scheme. In this approach, a positive pair value. (x, y). of a vector, V, is referred to as a radical sign value defined by $\sqrt[x]{y} = x$. For two nodes $A = (x_a, y_a)$ and $B = (x_b, y_b)$:

- $A < B$ if and only if $\sqrt[x_a]{y_a} < \sqrt[x_b]{y_b}$
- $A = B$ if and only if $\sqrt[x_a]{y_a} = \sqrt[x_b]{y_b}$

In a numeric-based labelling scheme, the insertion mechanism between two nodes depends on whether the two radical sign labels to be compared have the same root; i.e. $\sqrt[x]{y} = x$. Since there is no integer between two consecutive integers, the new node label must take a root value, x, greater than x_a and/or x_b to properly maintain the document order. However, this procedure leaves bigger gaps between vector values than in the Vector-encoding labelling scheme (Xu et al., 2007). Furthermore, when comparing two radical sign labels with different roots, the query time becomes expensive because of the "power (x, y)" operation, as shown in the experimental results of (Ni et al., 2012).

Many researchers (Noor Ea Thahasin and Jayanthi, 2013) (Xu et al., 2009) (Zhang and Dong, 2010) (Zhuang and Feng, 2012b) have minimised the size of vector labels as well as supporting XML updates. However, labelling schemes based on vector principles are not suitable for encoding extensive dynamic XML data (Sans and Laurent, 2008) (Noor Ea Thahasin and Jayanthi, 2013) (Xu et al., 2010) (O'Connor and Roantree, 2010a) for the reasons discussed below.

3.5.3 Summary of Multiplicative Labelling Schemes

Multiplicative labelling schemes can uniquely identify structural relationships directly from node labels using mathematical computations (Haw and Lee, 2011) (Chiew et al., 2014a) (Su-Cheng and Chien-Sing, 2009) (Assefa and Ergenc, 2012). However, such arithmetic computations are expensive and complex, which slows down XML query processing (Yanghua et al., 2012) (Lizhen and Xiaofeng, 2013) (Min et al., 2007) (Mirabi et al., 2012). This category of XML labelling scheme also suffers from large label sizes because it leaves big gaps between node label values that may lead to overflow problems (Ahn et al., 2017b) (Xu et al., 2009) (O'Connor and Roantree, 2012) (Haw and Amin, 2015) (Al-Shaikh et al., 2010). In general, this class of labelling schemes is inappropriate for dynamic XML data (Jiang et al., 2009) (Catania et al.,

2005b) (Li and Ling, 2005b) (Silberstein et al., 2005) since they usually require label values to be re-computed when new nodes are inserted.

3.6 Hybrid Labelling Schemes

Hybrid labelling schemes use combinations of existing labelling methods to balance the weaknesses of one labelling technique with the strengths of another in order to develop faster query processing (Qin et al., 2017) (Haw and Amin, 2015) (Haw and Lee, 2011) (Su-Cheng and Chien-Sing, 2009) (O'Connor and Roantree, 2010a).

Thonangi proposed the Sector-based labelling scheme (Thonangi, 2006) to minimise the label size of the interval-based labelling scheme (Li and Moon, 2001) by representing the labels intervals as sectors (Sans and Laurent, 2008). The sectors are assigned to node labels so that the angle created by the sector of a parent node at the origin totally encloses all of its descendants. In order to determine ancestor-descendant and document order relationships quickly, Sector-based labelling schemes use mathematical formulae similar to those modelled in multiplicative labelling schemes.

In the Sector-based labelling scheme a node A is labelled as an interval $< A_r, A_s >$, where 2^{A_r} is the radius of the sector assigned to the node A, and A_s is the smallest radial distance from the node A to the reference x-axis, as shown in Figure 3.11 (adapted from (Thonangi, 2006)). Such a representation reduces the interval label size by storing only the logarithm of a sector, A_r, rather than the sector's radius, 2^{A_r}. To label the descendants of node A through depth-first traversal, first the smallest value, k, such that 2^k is the minimum possible number of children of node A, is found. Then, each child of A (e.g., node D in Figure 3.11) is assigned a sector within an expanded sector of A (denoted A′ in Figure 3.11) to ensure the parent/child and ancestor/descendant relationships are represented.

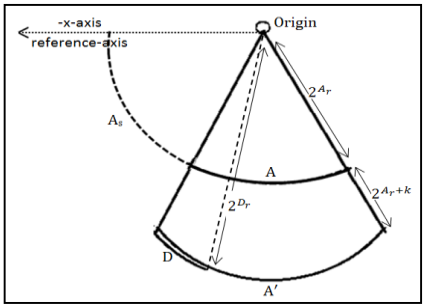


Figure 3.11 Graphical representation of node A and D as sectors

The Sector-based labelling scheme can recognise the ancestor-descendant relationship between node $A =< A_r, A_s >$ and node $D =< D_r, D_s >$ using arithmetic comparison, as follows:

$$\mathrm{A} \text{ is ancestor of } \mathrm{D} \text{ if, and only if, } \mathrm{A_s} \leq \mathrm{D_s} \ \mathrm{X} \ \frac{2^{\mathrm{Ar}}}{2^{\mathrm{Dr}}} \leq \mathrm{A_s} + 1$$

The Sector-based labelling scheme manages the parent/child relationship as ancestor/descendant relationship, but does not consider the sibling relationship. Furthermore, this scheme does not support dynamic XML documents and requires expensive arithmetic computations as in multiplicative labelling schemes (O'Connor and Roantree, 2010b) (O'Connor and Roantree, 2010a).

The VASLS -prime labelling scheme (VASLS stands for "Valid ASCII String labelling scheme") (Qin et al., 2013) is a hybrid labelling approach that adapts the prime number labelling scheme (Wu et al., 2004) to represent the structural information of XML tree nodes using VAS (Valid ASCII String). The VAS labelling scheme is applied to manage the node's document order. The valid ASCII strings are those from the 33^{rd} to 126^{th} described in VAS labelling scheme as $S = (s_1 s_2 \dots s_m | 33 \leq ASCII(s_i) \leq 126)$ in lexicographical order (Chapter 5, Section 5.4).

The intention in designing the VASLS-prime labelling scheme was to avoid re-labelling in a dynamic XML environment without sacrificing XML query performance. However, as the experimental results of (Qin et al., 2013) have shown, VASLS-prime labels are larger than the labels generated by Dewey (Tatarinov et al., 2002), DDE (Xu et al., 2009), QED (Li and Ling, 2005b) and ORDPATH (O'Neil et al., 2004). In terms of frequent updates at the leaf level retaining document order, VASLS-prime does not show any improvement in update time because of the need to re-calculate *SC* values in the prime number labelling scheme. Overall, the VASLS-prime labelling scheme suffers from large label sizes and does not support updating XML data.

A more recent hybrid labelling scheme is that of Dynamic XDAS (XML Documents Addressing and Sub-netting), as proposed by (Ghaleb and Mohammed, 2015). Dynamic XDAS generates binary labels using the masking technique of the XDAS labelling scheme (Ghaleb and Mohammed, 2013), whereas IBSL (Improved Binary String Labelling Scheme) (Hye-Kyeong and SangKeun, 2010) has been employed to avoid re-labelling when an XML document is updated. However, this scheme increases storage cost as storage required for dynamic XDAS labels increases rapidly in the case of frequent skewed insertions (Liu and Zhang, 2016) (Haw and Amin, 2015).

In summary, hybrid labelling schemes (Qin et al., 2017) (Kaplan et al., 2002) (He et al., 2005) (Chen et al., 2004) (Ghaleb and Mohammed, 2015) (Qin et al., 2013) (Thonangi, 2006) have the potential to support faster query processing by combining the advantages of two or more labelling schemes (Haw and Amin, 2015). However,

besides the fact that this class of labelling schemes do not support XML updates, constructing labels using hybrid approaches has so far proved to be computationally expensive (Haw and Lee, 2011) (Su-Cheng and Chien-Sing, 2009) (O'Connor and Roantree, 2010a) (Duong and Zhang, 2005) (Yun and Chung, 2008).

3.7 Summary and Limitations of XML Labelling Schemes

XML repositories available over the Internet have become more extensive and volatile. Consequently, dynamic labelling schemes have become essential to support efficient XML queries and updates (O'Connor and Roantree, 2010a) (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005). Several XML labelling schemes have been introduced to facilitate searching updatable XML data. In general, these schemes are categorised into four main classes based on their structure and the concepts used for generating node labels: interval-based, prefix-based, multiplicative, and hybrid labelling schemes.

Consistent with the fundamental properties required for a complete dynamic labelling scheme (identified in Section 3.2), each of the existing labelling schemes is limited in one regard or another. A complete dynamic labelling scheme must be simultaneously updatable and efficiently support XML querying by determining the main structural relationships directly from node labels (Lizhen and Xiaofeng, 2013) (Härder et al., 2007). Updating XML data remains the weakness in most of the current XML labelling schemes (Liu and Zhang, 2016) (Subramaniam and Haw, 2014b) (Yu et al., 2005) due to the natural conflict between the necessary requirements of update efficiency and those of query optimisation. Consequently, a labelling scheme must sacrifice one of the essential properties that make it a good dynamic labelling approach. As can be seen from the literature presented in this chapter, almost all of the existing labelling schemes suffer from large labels, which contribute to overflow problems; particularly after frequent skewed insertion.

Motivated by this observation, it is important to understand the main reasons behind the occurrence of overflow problems and how the encoding techniques may provide a solution. Therefore, the next chapter will consider some of the background to encoding methods applied to store XML labels, and from which the research hypothesis of this thesis is derived.

3.8 Conclusion

This chapter has presented the concept of XML labelling schemes and the state-of-the-art in research into XML labelling schemes, particularly within a dynamic XML environment. The chapter highlighted the limitations and strengths of the four main categories of XML labelling schemes: interval-based, prefix-based, multiplicative, and hybrid. The main disadvantage of the current dynamic XML labelling schemes is their large label sizes, which can contribute to overflow problems. The next chapter considers the literature available on existing label storage schemes.

Chapter 4: Literature on Encoding Methods

4.1 Introduction

As can be seen from the state-of-the-art of dynamic XML labelling schemes illustrated in Chapter 3, all of the existing labelling schemes encounter difficulties when XML data are updated. For instance, when re-labelling existing nodes, inefficient structural relationship determination, wasteful storage size, and/or overflow problems. This is mainly due to the design of the labelling algorithms or the limitations of the encoding techniques used to store the XML labels.

This chapter gives a degree of background to the encoding methods used for the storage of XML labels. The next section identifies the concept of the encoding scheme in the context of XML, followed by an explanation of the overflow problem in Section 4.3. Section 4.4 presents a comprehensive overview of different label storage schemes that describe the storage consumption of label values, namely length field (Section 4.4.1), control tokens (Section 4.4.2), separators (Section 4.4.3) and prefix-free codes (Section 4.4.4). In Section 4.5, several prefix-encoding methods are illustrated that have not, to date, been applied to XML labelling. Finally, Section 0 concludes this chapter.

4.2 Encoding methods

Usually in the context of XML, the terms “labelling scheme” and “encoding scheme” are used interchangeably. Therefore, it is important to clarify and differentiate between the meaning of the term “encoding” as a labelling approach or as a storage mechanism. In the concept of a labelling scheme, an encoding scheme is referred to as codifying the node structure within an XML tree (O'Connor and Roantree, 2010a), whereas in terms of storage mechanisms, an encoding method represents how data is physically stored on disk (O’Connor and Roantree, 2013). In this thesis, the term “encoding method” is used to indicate the notion of the storage mechanism.

A key factor for all XML dynamic labelling schemes is how their labels are physically encoded, decoded, and stored on a computer (O’Connor and Roantree, 2013) (Xu et al., 2012). In implementation, labels are actually stored on disk as binary numbers with either a fixed length (i.e., fixed number of bits for all label values) or variable

length format (depending on the size of the labels' binary representation) (Mirabi et al., 2012). However, the logical representation of a label value is quite different; for instance, in prefix labelling schemes the delimiter "." is encoded and stored separately from the label value (Li et al., 2008) (Tatarinov et al., 2002). Therefore, the logical interpretation of a label in the computer immediately affects the label size on disk and the computational cost of encoding/decoding between logical and physical representations (O'Connor and Roantree, 2013).

In a dynamic labelling scheme where the labels change, there are two main reasons that may cause the re-labelling of nodes when XML is updated (O'Connor and Roantree, 2013). The first is when arbitrary insertions are not enabled by the node insertion algorithms within a labelling scheme, such as in Dewey Order encoding (Tatarinov et al., 2002) and extended Dewey (Lu et al., 2005b); the second is the overflow problem produced by a labelling scheme that allows limited number of insertions, such as in QED (Li and Ling, 2005b), the Vector-order labelling scheme (Xu et al., 2007), SCOOTER (O'Connor and Roantree, 2012) and ImprovedBinary (Li and Ling, 2005a).

4.3 The Overflow Problem

The overflow problem is relevant to the label storage scheme used to encode and store the values of the labels in a computer system, where all nodes labels are stored either as fixed-length or variable length binary numbers at implementation. Fixed-length labels are subject to overflow and are not scalable (Li and Ling, 2005b) (O'Connor and Roantree, 2013), because re-labelling of all existing labels is required if all the assigned bits have been used up by frequent insert processes (O'Connor and Roantree, 2010a). On the other hand, using variable length labels necessitate storing the size of the label in addition to the label itself (O'Connor and Roantree, 2012). Therefore, as the label size increases due to insertions into an XML tree, the fixed length field (e.g., 4 bits) assigned to store the size of the label length becomes inadequate, and leads to the overflow problem (Mirabi et al., 2010). Even though this problem can be solved by increasing the size of the length field (e.g., from 4 bits to 6 bits), it cannot avoid re-labelling entirely and may waste storage space (Mirabi et al., 2010) (Li et al., 2008).

If there is insufficient storage space to accommodate a new node label, in practice a part of the new label might be missed and so it will appear as a duplicated label (Li and Ling, 2005a) (Liu et al., 2013). Otherwise, it might cause data corruption by

overwriting the content of an adjacent memory location. This is known as the overflow problem (Li and Ling, 2005b). Figure 4.1 illustrates an example of the overflow problem, assuming QED labels and a limited storage capacity of 2-bytes (for simplicity).

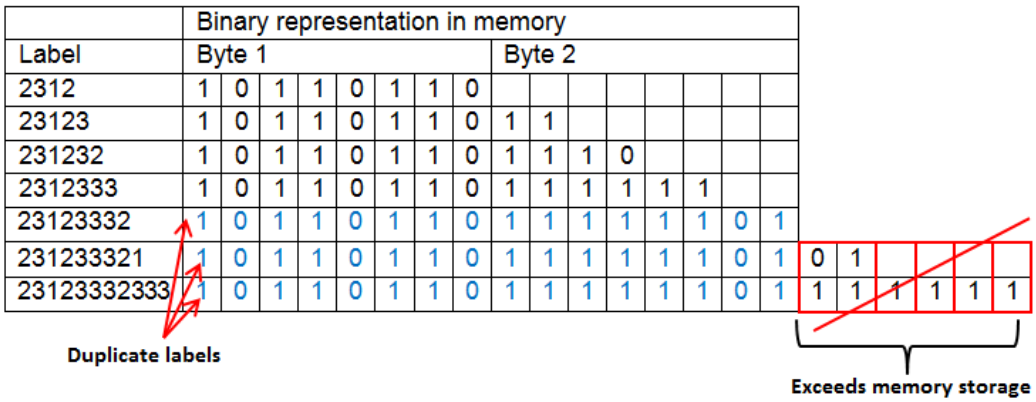


Figure 4.1 Example of overflow problem

Because prefix-based labelling schemes keep the ancestor labels attached to the node self-label for rapid determination of structural relationships, many prefix labelling schemes, such as ImprovedBinary (Li and Ling, 2005a), LSDX (Duong and Zhang, 2005), ORDPATH (O'Neil et al., 2004) and Dewey Order (Tatarinov et al., 2002), suffer from overflow problems due to the verbosity of their labels.

4.4 Label Storage Schemes

This section gives a comprehensive review on label storage schemes, and demonstrates the storage consumption of label values, i.e., how labels values are actually presented in the computer and their effects on the dynamic labelling mechanism.

As illustrated in Section 4.3, fixed-length labels are always subject to the overflow problem and are not scalable, therefore, this section considers only variable length representation. According to (Härder et al., 2007), all existing dynamic (variable length) label storage schemes can be categorised into four classes: length fields, control tokens, separators and prefix-free code.

4.4.1 The Length Field

The fundamental concept of length fields is to store the length of a node label, L_i, directly before the nodes' label value, O_i (Härder et al., 2007). The lengths of nodes'

labels can vary depending on the node position within the XML tree. A simple approach is to assign a fixed length bit number, L_i, to specify the length of the label (O'Connor and Roantree, 2013). However, in reality there is rarely advance knowledge of the possible number of node insertions that may subsequently occur. As a consequence, in dynamic XML the number of node insertions is limited to the capacity implied by the fixed-length field, leading to the overflow problem, such as in V-CDBS (Variable-length Compact Dynamic Binary String) labelling schemes (Li et al., 2006a).

4.4.2 Control Tokens

The key concept of control tokens is their use to indicate the position of a label value within a specific-level interval. These tokens are then used to determine how the subsequent bit sequence of a label value can be interpreted (by some form of metadata) (Härder et al., 2007) (O'Connor and Roantree, 2013).

An example of control tokens is UTF-8 (Yergeau, 2003), which is employed in Dewey Order (Tatarinov et al., 2002) to encode Dewey labels, where each component of the Dewey path is encoded in UTF-8 and then concatenated in the same path order (Tatarinov et al., 2002). In UTF-8, a binary number representing the Dewey ID is of variable length depending on the size of the Dewey ID integer value. For instance, an integer value between 0 and 27 is stored in a maximum of 8-bits, starting from *0xxxxxxx,* where *x* represents the bits used for the integer value (Li et al., 2008). Here, the first bit sequence in the label is the control token "0", denoting that the label length is 1 byte. If the first bit is the control token it starts with "1", then the number of bytes used to represent the label can be calculated by counting the total number of 1s before the control token "0" bit is encountered. Table 4.1 (adapted from (O'Connor and Roantree, 2013)) demonstrates the use of control tokens in the UTF-8 encoding method. However, as can be seen from Table 4.1, the UTF-8 used to encode Dewey IDs can only code up to 2^{31} labels (Li and Ling, 2005b) (O'Connor and Roantree, 2013) (O'Connor and Roantree, 2012). Similarly, the Vector-order labelling (Xu et al., 2007) and extended Dewey labelling schemes (Lu et al., 2005b) use UTF-8 encoding, and therefore only permit a limited number of insertions (O'Connor and Roantree, 2012).

Table 4.1 UTF-8 encoding method

Value range	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
0 – (2^7 -1)	0xxxxxxx					
2^7 – (2^{11} – 1)	110xxxxx	10xxxxxx				
2^{11} – (2^{16} -1)	1110xxxx	10xxxxxx	10xxxxxx			
⋮	⋮	⋮	⋮	⋮	⋮	
2^{26} – (2^{31} – 1)	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

4.4.3 Separators

In prefix-based labelling schemes, the (separator) delimiter "." is encoded and stored separately from the label itself (Li et al., 2008) (Tatarinov et al., 2002). A separator reserves a predefined bit sequence to indicate its interpretation as a delimiter, rather than a part of the label value.

An encoding approach to symbolise the separator is to reserve an m-bit code as a number in base k (Härder et al., 2007). For example, if $k = 3$, which has a maximum number of 2 bits, this represents the codes: "0" $=$ 00, "1" $=$ 01, "2" $=$ 10, and 3 $=$ "11", then "11" is a possible code to encode the separator. For example, a prefix label of value "1.5.11" is encoded as (01 11 01 01 11 01 00 10) (spaces added for clarity), where "11" indicates a separator.

Unlike the control token approach, separators do not preserve comparability (Härder et al., 2007). Suppose node N_1 has a value "1.5.11" $=$ (01 11 01 01 11 01 00 10), and node N_2 $=$ "1.5.7", encoded as (01 11 01 01 11 10 01). Then, bit-by-bit comparison implies $N_1 < N_2$ whereas "1.5.11" $>$ "1.5.7". Therefore, during decoding, the separators must be detected from the actual label value components. To remove such ambiguity, the quaternary encoding QED (Li and Ling, 2005b) and SCOOTER (O'Connor and Roantree, 2012) have employed their own separator storage scheme. For example, QED uses digit "0" for encoding the separators only, and therefore the separator code size remain constant regardless of label size. However, this approach decelerates a bit-by-bit or byte-by-byte comparison operation during decoding because of the process needed to recognize bit "0" or "00" as a separator, rather than the binary representation of the code itself (Härder et al., 2007). Consequently, identifying "0" as a separator slows down query performance due to the associated

expensive decoding time, particularly when an XML document has a deep tree representation (Ghaleb and Mohammed, 2013). Nevertheless in the case of frequent skewed insertions, the size of new nodes self-label codes will overflow (Ghaleb and Mohammed, 2015) (Liu and Zhang, 2016) (Chiew et al., 2014a) (Ghaleb and Mohammed, 2013).

4.4.4 Prefix-Free Codes

Prefix-free codes are based on the proposition of (Elias, 1975) that a prefix set, S, is said to be a prefix code if, and only if, no member of S is the beginning of another. In other words, a prefix set, S, is uniquely identifiable where no member in the set, S, is a prefix to any other member in S (O'Connor and Roantree, 2013). For example, set $S = \{0, 1, 2, 3, 4, 5, 6, 7\}$ is a prefix set, while the set $X = \{1, 2, 3, 4, 22\}$ is not a prefix set since "2" is a prefix of "22". Therefore, a prefix-free code approach often requires fewer bits to represent a label than a control token scheme. This is because the prefix-free codes can be adjusted according to the number of members within a prefix set if a suitable assignment of codes and value ranges are defined (Härder et al., 2007).

An example of a dynamic labelling scheme that uses prefix-free codes is ORDPATH (O'Neil et al., 2004). The compressed binary string representing an ORDPATH label are consecutive L_i/Q_i bit-strings stored with variable length (O'Neil et al., 2004). L_i/Q_i represents the i^{th}-component of an ORDPATH label, where the $\mathrm{L_i}$ sub-string identifies the length of bits in which the binary number representing the i^{th}-component of an ORDPATH label exists within an $\mathrm{O_i}$ range value. For example, consider the ORDPATH label value "1.5.3", a compressed binary string representing the second component "5" (where $\mathrm{i} = 1$, note that component count starts from 0) is $L_1/Q_1 = 01/101$ (= 3/5, i.e., the binary number equivalent to integer number 5 is "101", which is of 3-bits length). This encoding method was further compressed by omitting further unnecessary bit-spaces, considering that the $\mathrm{L_i}$ bit-string "01" requires $\mathrm{O_i}$ of length 0 to represent the binary digit 1. However, this technique makes the decoding process in ORDPATH more time consuming (Mirabi et al., 2012).

4.4.5 Limitation of Label Storage Schemes

Variable length field and separator storage scheme work properly if the XML document is rarely updated. However, when these storage schemes are used in dynamic labelling schemes such as ImprovedBinary (Li and Ling, 2005a), and CDBS

(Li et al., 2006a), the overflow problem occurs because of the bit sequence reserved for a separator as well for the label length value. Consequently, all length field label storage schemes are exposed to re-labelling when frequent node insertions occur.

On the other hand, control tokens and prefix-free code storage schemes are widely applied in dynamic XML labelling schemes for encoding numerical and alphanumeric labels such as: LSDX (Duong and Zhang, 2005), Dewey Order (Tatarinov et al., 2002), extended Dewey (Lu et al., 2005b), Vector-order labelling scheme (Xu et al., 2007), and ORDPATH (O'Neil et al., 2004). Such labelling schemes are not scalable due to the long labels generated by control token and prefix-free code schemes (O'Connor and Roantree, 2013) (O'Connor and Roantree, 2012) (Chiew et al., 2014a) (Ghaleb and Mohammed, 2015).

To overcome with such limitations, there have been many prefix-encoding methods that can be used to store XML labels. These are presented in the next section.

4.5 Prefix-encoding Methods

Currently, one of the most common data compression techniques are prefix codings (Gagie et al., 2015) (Karpinski, 2009). A prefix code is a variable-length code suitable for coding a set of text or integers whose size is unknown beforehand. Research has shown that prefix-encoding methods give a higher compression ratio than other encoding schemes (Walder et al., 2012) (Klein and Ben-Nissan, 2010) (Bača et al., 2010) (Fredriksson and Nikitin, 2007) (Somasundaram and Domnic, 2007). Although many prefix-encoding methods exist in the literature, they have never been applied to code XML labels. Motivated by this, some of these encoding methods have been tested to compress XML labels in this thesis. This section presents such encoding schemes.

4.5.1 Fibonacci of Order $m \geq 2$

Fibonacci code, as a well-known representative of prefix code, was introduced by (Fraenkel and Klein, 1985), and is based on Fibonacci numbers (Knott, 1998). Generalised Fibonacci code of order $m \geq 2$ was introduced by (Apostolico and Fraenkel, 1987) as follows:

Definition 1: **Fibonacci numbers of order $m \geq 2$** :

$$F_i^{(m)} = F_{i-1}^{(m)} + F_{i-2}^{(m)} + \cdots + F_{i-m}^{(m)}, \quad \text{for } i \geq 1, \qquad \textit{Equation 4.1}$$

$$\text{where } F_j^{(m)} = 0, \text{ for } j \leq -2 \text{ and } F_{-1}^{(m)} = F_0^{(m)} = 1$$

Examples of Fibonacci numbers of order $m = 2$ and $m = 3$ are presented in Table 4.2 below (adapted from (Walder et al., 2012)).

Table 4.2 Sample of Fibonacci numbers of order 2 and 3

i	-2	-1	0	1	2	3	4	5	6	7	8	9	10
$F_i^{(2)}$	0	1	1	2	3	5	8	13	21	34	55	89	144
$F_i^{(3)}$	0	1	1	2	4	7	13	24	44	81	149	274	504

Definition 2: Binary representation of Fibonacci code:

Generalised Fibonacci code of order $m \geq 2$ (Apostolico and Fraenkel, 1987) states that for each non-negative integer value, N, there is an exact unique binary encoding of the form:

$$N = \sum_{i=0}^{k} d_i F_i, \qquad d_i \in \{0,1\}, 0 \leq i \leq k \qquad \textit{Equation 4.2}$$

Such that there are no m consecutive 1-bits within the summation result of Fibonacci numbers of order m, whereas each Fibonacci code ends up with exactly m consecutive 1-bits. This is called the $F(m)$ numeration system (Fraenkel, 1985). Table 4.3 below shows some examples of Fibonacci code of order 2 and 3 for various values (spaces are added for clarity).

Table 4.3 Some Fibonacci codes of order 2 and 3

x	$F^{(2)}(x)$	$F^{(3)}(x)$
1	11	111
2	011	0111
3	0011	00111
4	1011	10111
5	00011	000111
6	10011	010111
7	01011	100111
⋮		
100	00101000011	00000110111
112	01000010011	00100100111

Many researchers have used Fibonacci code for data compression. (Bača et al., 2010) applied Fibonacci code of order 2 and order 3 to code for the compression of XML node stream arrays. (Gog, 2009) has shown that the use of Fibonacci code for Compressed Suffix Arrays (CSAs) can provide fast access times and minimal space

for text of low compressibility. Whereas, in (Fischer, 2009) the author applied Fibonacci coding and ternary coding to examine if prefix-free code could lead to shorter labels for Lowest Common Ancestors in trees. (Lelewer and Hirschberg, 1987) has proved that Fibonacci coding is a good choice for compressing small integers and for fast decoding. Fibonacci codes can not only be used as a simple alternative to Huffman codes as studied in (Przywarski et al., 2006), but also to dense codes for large text-based compression systems (Klein and Ben-Nissan, 2010). (Apostolico and Fraenkel, 1987) have suggested Fibonacci codes as compression codes for the unbounded transmission of strings.

The following two sections describe encoding integers $x > 0$ in Fibonacci codes of order 2 and order 3, respectively.

4.5.1.1 Fibonacci coding of order $m = 2$

A Fibonacci encoding algorithm of order 2 utilises a stack for bit storage since it stores the bits in reverse order. For an integer, x, the Fibonacci code of order $m = 2$ (referred to as $F^{(2)}(x)$) algorithm is described as follows:

1 Initialise $F^{(2)}(x)$ to empty.
2 Find the i^{th} index of the largest Fibonacci number, such that $F_i^{(2)} \leq x$.
3 If $F_i^{(2)} \leq x$, compute $x = x - F_i^{(2)}$ and push the 1-bit to the stack. Otherwise, push the 0-bit to the stack.
4 Set $i = i - 1$, if $i \geq 0$ repeat step 3.
5 While the stack is not empty, remove a bit from the stack and place it at the end of $F^{(2)}(x)$.
6 Add the 1-bit at the end of $F^{(2)}(x)$.

Example 1: Suppose integer $\mathrm{x} = 112$. Because $F_9^{(2)} = 89 \leq 112 < 144 = F_{10}^{(2)}$, i is set to 9. So, bit 1 is pushed to the stack (i.e., stack $= 1$). Then, a new x is computed such that $\mathrm{x} = 112 - 89 = 23$, and $i = i - 1 = 8$. Since $F_8^{(2)} = 55 > 23$, a 0-bit is pushed to the stack (i.e., stack $= 10$) and $\mathrm{i} = 7$. Going back to step 3, again a 0-bit is pushed to the stack (i.e., stack $= 100$) since $F_7^{(2)} = 34 > 23$ and $\mathrm{i} = 6$. Since $F_7^{(2)} = 21 < 23$, 1-bit is pushed to the stack (i.e., stack = 1001) and the new $\mathrm{x} = 23 - 21 = 2$, and $\mathrm{i} = 5$. Since $F_5^{(2)}, F_4^{(2)}, F_3^{(2)},$ and $F_2^{(2)}$ are < 2, 0-bits are pushed to the stack (stack = 10010000) until $\mathrm{i} = 1$, where $F_1^{(2)} = 2 \leq x = 2$. Then, push 1-bit to the stack (i.e. stack $= 100100001$) and $\mathrm{x} = 2 - 2 = 0$. However, i still > 0, so 0-bit(s) are

pushed to the stack until $i = 0$; i.e., stack $= 1001000010$. Then, the bits are popped out of the stack in reverse order and assigned to $F^{(2)}(x) = 0100001001$. Finally, 1-bit is added to the end of $F^{(2)}(x)$, thus the result of $F^{(2)}(112) = 01000010011$. Table 4.3 above shows some examples of the Fibonacci code of order 2 for various integers.

4.5.1.2 Fibonacci coding of order $m > 2$

To generate Fibonacci code of an order greater than 2, the Fibonacci sum $S_x^{(m)}$ introduced by (Apostolico and Fraenkel, 1987) must be used.

Definition 3: Fibonacci sum

$$S_x^{(m)} = \begin{cases} 0, & for\ x < -1 \\ \sum_{i=-1}^{x} F_i^{(m)}, & for\ x \geq -1 \end{cases} \qquad \textit{Equation 4.3}$$

Subsequently, the Fibonacci code of order $m > 2$ encoding algorithm is as follows:

1 If $x = 1$, then $F^{(m)}(x) = m$ consecutive 1-bits → *END*.

2 If $x = 2$, then $F^{(m)}(x) = 0$ followed by m consecutive 1-bits → *END*.

3 For $x > 2$, find k such that $S_{k-2}^{(m)} < x \leq S_{k-1}^{(m)}$, then let $Q = x - S_{k-2}^{(m)} - 1$.

4 Compute $F^{(m)}(Q)$.

5 Reverse the bit ordering in $F^{(m)}(Q)$, then append $01m$ as a suffix to the reversed $F^{(m)}(Q)$. If the length of $F^{(m)}(Q) < m + k$, append 0-bits to the beginning of the $F^{(m)}(Q)$ code to make an $F^{(m)}(x)$ of length $m + k$.

Table 4.4 demonstrates the first 10 values of $S_{10}^{(3)}$.

Table 4.4 Fibonacci sum for the first 10 values of Fibonacci of order 3

i	-1	0	1	2	3	4	5	6	7	8	9	10
$F_i^{(3)}$	1	1	2	4	7	13	24	44	81	149	274	504
$S_i^{(3)}$	2	4	8	15	28	52	96	177	326	600	1104	2031

Example 2: Assume integer $x = 112$. Since $S_5^{(3)} = 96 < 112 \leq 177 = S_6^{(3)}$, thus $k = 7$. Consequently, $Q = 112 - 96 - 1 = 15$. As $15 = 2 + 13 = F_1^{(3)} + F_4^{(3)}$; therefore $F^{(3)}(15) = 01001$. Finally, to compute $F^{(3)}(112)$, first reverse $F^{(3)}(15)$ bits $= 10010$. Then append 0111 to the end. That is; $F^{(3)}(112) = 100100111$. Since length of (100100111) is $9 < m + k = 3 + 7 = 11$, then append 0-bits to the beginning of

the code. Thus, $F^{(3)}(112)$ in binary $= 00100100111$. More examples of Fibonacci code of order 3 for various integers can be seen in Table 4.3.

4.5.1.3 Fibonacci Label Storage Scheme

The Fibonacci label storage approach proposed by (O'Connor and Roantree, 2013) represents the middle ground between the length field and the control token scheme. It computes the variable length size of an XML label value based on the Fibonacci sequence (Chandra, 1999) and the *Zeckendorf* representation (Weisstein, 1999a). The main principle is that any positive integer n can be represented as the sum of one or more distinct discrete Fibonacci numbers that satisfies the *Zeckendorf* representation of n. For example, integer number 112 can be represented as summation of Fibonacci numbers as $(112 = 89 + 13 + 5 + 3 + 2)$ or $(112 = 89 + 21 + 2)$. However, only the second representation is *Zeckendorf* since the first one has three consecutive Fibonacci numbers (i.e., 2, 3, 5).

(O'Connor and Roantree, 2013) used the Fibonacci-Zeckendorf principle for the encoding and decoding of the length field of a label value; which is stored on the disc directly before the label value itself. For example, to encode the length of a node label value "101101" (length $= 6\ bits$), the *Zeckendorf* representation of the label length is first found ($e.g., 6 = 5 + 1$), after which each number in the Fibonacci sequence is compared and assigned bit "1" for those that match a member of the *Zeckendorf* representation; otherwise, value "0" is assigned. Therefore, for a length value 6, the binary string of Fibonacci encoding is "1001", as explained in Figure 4.2. The decoding is simply the reverse procedure.

Example; Length = 6	satisfies Zeckendorf representation								
Fibonacci sequence	0	1	1	2	3	5	8	13	21
Length value encoding	0	0	1	0	0	1	0	0	0

Figure 4.2 Example of a Fibonacci label storage scheme

4.5.2 Lucas Coding

Lucas numbers were introduced by Edouard Lucas (MacTutor, 1996) based on Fibonacci sequence properties. The Lucas numbers, $\{L_n\}_0^\infty$ are defined by

$$L_{n+2} = L_{n+1} + L_n \; ; for \; n \geq 0, where \; L_0 = 2, and \; L_1 = 1. \qquad \textit{Equation 4.4}$$

The first ten Lucas numbers are shown in Table 4.5.

Table 4.5 Examples of Lucas numbers

n	0	1	2	3	4	5	6	7	8	9
L_n	2	1	3	4	7	11	18	29	47	76

Each positive integer can be uniquely represented in binary as sums of distinct Lucas numbers (Brown Jr, 1969). In (Association, 2011), the authors approved that coding theorems for Lucas numbers correspond to Fibonacci code of order 2 coding theorems (i.e., using Zeckendorf states) (Keller, 1972). (Chergui, 2015) also provided the proof for using Zeckendorf theorems for Lucas numbers to generate unique binary representations for any positive integer.

Definition 4: Zeckendorf theorem for Lucas numbers

Every natural number, x, satisfying $0 \leq x \leq L_k$, for $k \geq 1$, has a unique binary representation in the form:

$$x = \sum_{i=0}^{k-1} \alpha_i L_i \, , where \; \alpha_i \in \{0,1\} \, ,$$
$$\text{Such that } \begin{cases} \alpha_i \alpha_{i+1} = 0, for \; i \geq 0 \\ \alpha_0 \alpha_2 = 0 \end{cases} \qquad \textit{Equation 4.5}$$

Like Fibonacci (order 2), Lucas coding stores the bits in reverse order and so it uses stacks. Before starting the coding algorithm, the position of the first two Lucas numbers are swapped after computing all the necessary Lucas sequences (Association, 2011), as follows:

L_1	L_0	L_2	L_3	L_4	L_5	L_6	L_7
1	*2*	*3*	*4*	*7*	*11*	*18*	*29*

For each integer, x, the Lucas code (referred to as $L(x)$) is described as follows:

1 Initialise $L(x)$ to empty.
2 Find the i^{th} index of the largest Lucas number such that $L_i(x) \leq x$.
3 If $L_i(x) \leq x$, compute $x = x - L_i(x)$ and push the 1-bit to the stack. Otherwise push the 0-bit to the stack.
4 Set $i = i - 1$, if $i \geq 0$ repeat step 3.
5 While the stack is not empty remove a bit from the stack and place it at the end of $L(x)$.

6 Add the 1-bit at the end of $L(x)$.

For example, let $x = 24$. Since $24 < 29$, $i = 6$. So bit 1 is pushed to the stack (i.e., stack = 1). Then new x is computed such that $x = 24 - 18 = 6$, and $i = i - 1 = 5$. Since $L_5 = 11 > 6$, push 0 to the stack and decrement i. Since $L_4 = 7 > 6$, push 0 to the stack (now stack $= 100$) and $i = i - 1 = 3$. For $L_3 = 4 < 6$, push 1 to the stack (i.e., stack $= 1001$) and thus the new $x = 6 - 4 = 2$, $i = 2$. Since $L_2 = 3 > 2$, push 1 to the stack (i.e., stack $= 10010$), and $i = 1$. Since $L_1 = 2 \leq 2$, push 1 to the stack (now stack $= 100101$) and $x = 2 - 2 = 0$. Since $i > 0$ keep decreasing i and pushing 0-bit to the stack until $i = 0$. That is, the stack $= 1001010$. Finally, $L(x)$ equals the reverse of the stack and add 1-bit at the end. As a result, $L(24) = 01010011$. For more clarity, Table 4.6 below shows the performance of Lucas code for various values $18 < x < 29$:

Table 4.6 Examples of Lucas codes

Lucas sequence	L_1	L_0	L_2	L_3	L_4	L_5	L_6	Extra 1-bit
	1	2	3	4	7	11	18	
$L(24)$	0	1	0	1	0	0	1	1
$L(21)$	0	0	1	0	0	0	1	1
$L(19)$	1	0	0	0	0	0	1	1

4.5.3 Elias-Delta Coding

Introduced by Peter Elias (Elias, 1975), the Elias-delta code is one of the most commonly used prefix code. It is defined as follows. For each integer value, x, the Elias-delta code $E(x)$ can be obtained by these steps:

1 Let $B(x)$ be the binary representation of x excluding insignificant 0-bits (at the left of the binary number). Let $B'(x)$ be $B(x)$ without the foremost 1-bit (most-left 1-bit).
2 Let $LN(x)$ be the length of $B(x)$; i.e., number of bits of $B(x)$.
3 Let $L(x)$ be the binary representation of $LN(x)$.
4 Let $S(x)$ be a sequence of 0-bits of size equals to the (length of $L(x)) - 1$.
5 The Elias-delta code is then generated as $E(x) = S(x) \oplus L(x) \oplus B'(x)$, where $\oplus$ means concatenating.

Table 4.7 shows some examples of Elias-delta codes $E(x)$ for various values (spaces are added for clarity)

Table 4.7 Examples of Elias-delta codes

Integer x	$B(x)$	$S(x)$	$L(x)$	$B'(x)$	$E(x) = S(x) \oplus L(x) \oplus B'(x)$
1	1	-	1	-	1
2	10	0	10	0	0 10 0
3	11	0	10	1	0 10 1
4	100	0	11	00	0 11 00
10	1010	00	100	010	00 100 010
19	10011	00	101	0011	00 101 0011
50	110010	00	110	10010	00 110 10010
100	1100100	00	111	100100	00 111 100100

Williams (Williams and Zobel, 1999) applied Elias-delta and Elias-gamma codes to store integers in compressed form in order to improve the performance of disk access and data retrieval. Elias-delta code was also utilised by (Scholer et al., 2002) to compress inverted indices to speed up the query performance and query evaluation. Moreover, (Walder et al., 2009) used Elias-delta code to generate a compression scheme for an R-tree (Guttman, 1984) data structure in order to minimise the index file and reduce the query processing time.

4.5.4 Elias-Fibonacci of Order $m \geq 2$

Elias-Fibonacci code was introduced by (Walder et al., 2012) as a combination of Elias-delta code and Fibonacci code of order 2, and is defined as follows:

$$EF(x) = F^{(2)}\big(L(x)\big) \oplus B(x) \qquad \textit{Equation 4.6}$$

Where $B(x)$ is the binary representation of x, $L(x)$ is the length of $B(x)$, and $F^{(2)}\big(L(x)\big)$ is the Fibonacci of order 2 of $L(x)$. Elias-delta, Fibonacci of order 2 and order 3, and Elias-Fibonacci codes have been applied for the compression of XML node stream arrays by (Bača et al., 2010) and for the compression of the R-tree in (Chovanec et al., 2010).

Table 4.8 Examples of Elias Fibonacci (of order $m = 2$ and $m = 3$)

Integer (x)	Elias Fib 2 = $F^{(2)}(L(x)) \oplus B(x)$:	Elias Fib 3 = $F^{(3)}(L(x)) \oplus B(x)$:
1	1 1	11 1
2	01 10	011 10
3	01 11	011 11
4	001 100	0011 100
5	001 101	0011 101
10	101 1010	1011 1010
19	0001 10011	00011 10011
50	1001 110010	01011 110010
100	0101 1100100	10011 1100100
500	10001 111110100	000011 111110100

In this thesis, a new Elias-Fibonacci (of order 3) code is purposed. The purposed algorithm basically uses of Fibonacci of order 3 instead of order 2; i.e., $EF(x) = F^{(3)}(L(x)) \oplus B(x)$. The Fibonacci code of order 3 here adds 0111 at the end, instead of 011. Examples of Elias-Fibonacci of order 2 and order 3 is given in Table 4.8 (spaces added for clarity). A generalised method of Elias-Fibonacci (order $m > 2$) is illustrated below.

Definition 5: Elias-Fibonacci ($m > 2$):

Similar to the Elias-Fibonacci code presented by (Walder et al., 2012), the new Elias-Fibonacci ($m > 2$) is a universal code for positive integers. Each Elias-Fibonacci ($m > 2$) code consists of two main parts. The second part is the binary representation of the integer, x, denoted as $B(x)$. The first part represents the length of $B(x)$ (referred to as $L(x)$) coded using Fibonacci code of order $m > 2$, where the Fibonacci code of order $m > 2$ is ends with '0'$\oplus$ '1'$_{m-1}$ bits instead of '01'. The purposed Elias-Fibonacci of order $m > 2$ is described as follows:

1. Compute $B(x)$, and let $L(x)$ be the length of $B(x)$.
2. Compute $F^{(m)}(L(x))$; Fibonacci of order m code of $L(x)$, where $F^{(m)}$ is ended by 01_{m-1}.
3. The Elias-Fibonacci code of order $m > 2$ is $EF^{(m)}(x) = F^{(m)}(L(x)) \oplus B(x)$, ($\oplus$ means concatenation).

In this thesis, Elias-Fibonacci (of order $m \geq 2$) have been applied to compress the XML labels. The study has also covered the behaviour of the resulting codes in relation to the increment of the order value m. Chapters 6 and 7 present the details of the implimentation and discussion of this study, which has already been published at the WEBIST 2016 conference (Al-Zadjali and North, 2016).

4.6 Conclusion

Whilst XML material has became more abundant, its heterogeneity and structural irregularity limit the management of the querying and update process within large-scale XML databases. Although many XML labelling schemes have been proposed in the literture to facilitate XML querying, most of these have ignored the compactness of the labels generated. Consequently, current XML labelling schemes still suffer from huge label sizes that may lead to overflow problems in the case of frequent insertions. This is due to the design of the labelling algorithm and how it handles insertions as illustrated in detail in Chapter 3.

An overview of the existing labelling storage schemes and their limitations have been presented in this chapter. Several perfix encoding methods have also been illustarted in this chapter. Although research (Williams and Zobel, 1999) (Chovanec et al., 2010) (Bača et al., 2010) (Scholer et al., 2002) (Walder et al., 2009) (Guttman, 1984) has shown the usefulness of some of these prefix-encoding methods for compression systems, they have never been applied to XML label compression. Motivated by this observation and the literture review represented in this chapter and Chapter 3, the research hypothesis of this thesis has thus been derived. The following chapter presents the research problem in detail, and gives a possible solution to the maniuplation and management of dynamic XML data, namely via the Base-9 XML labelling scheme proposed in this thesis.

Chapter 5: Base-9 Labelling Scheme for Dynamic XML Data

5.1 Introduction

The importance of dynamic XML labelling schemes has been established for accommodating the increasing significance of XML data management (Ghaleb and Mohammed, 2015) (Almelibari, 2015). Several XML labelling schemes have been introduced to process queries efficiently with minimum label size as well as to address the ability to process order-sensitve updates effecitively. Therefore, designing a dynamic labelling scheme that can preserve such properties whilst handling as many insertions as possible without re-labelling the existing labels is a challenging task, as demonstrated in the literature (Chapters 3 and 4).

This chapter starts by specifing the research problem in the following section. Section 5.3 presents the motivation and research hypothesis derived from the literture review presented earlier (Chapters 2, 3, and 4). This is followed by an overview of the proposed dynamic XML labelling scheme, named “Base-9”, in Section 5.4. Based on the lexicographical order defined in Section 5.4, the Base-9 labels initialisation algorithm and the insertion techniques are proposed in Sections 5.5 and 5.6, respectively. In Section 5.7, the use of Fibonacci coding for compressing and storing Base-9 labels is described. This chapter also describes the ability of the Base-9 scheme to determine the structural relationships in Section 5.8. Finally, the chapter ends with a general conclusion in Section 5.9.

5.2 Problem Identification

During the lifecycle of an XML document there can be arbitrary insertions of new nodes. Various XML labelling schemes have been proposed to improve the storage and retrieval of XML data in a dynamic XML environment. According to (Härder et al., 2007) and (Wu et al., 2004), a good dynamic XML labelling scheme must be compact, updatable and at the same time support the main operations of XML query processing by determining the common structural relationships efficiently and directly from the label values. However, there is a natural conflict between the requirements of update efficiency and those of query optimisation and, consequently, a labelling scheme usually sacrifices one of the essential properties that otherwise would have made it a

good dynamic labelling approach. As can be seen from the above-mentioned research literture in Chapter 3, all the existing labelling schemes suffer from the large labels that contribute to the overflow problem, particularly under frequent skewed node insertions. This is either due to a failure to consider the size of the labels generated whilst designing the labelling algorithms or due to the inadequacies of the encoding techniques used to store the XML labels.

The current encoding mechanisims used to store XML labels are illustrated in Chapter 4. These encoding methods have limited storage capacity and do not support frequent insertions in large-scale XML data, particularly in prefix-based labelling schemes. Therefore, there is a need to develop an efficient dynamic XML labelling approach that generates compressed XML labels not only during intialisation but also when the XML is subsequently updated. In order to achieve this, it is essential to investigate the possibility of storing XML labels in a compressed format, such as via prefix-encodings.

5.3 Research Objective, Motivation and Hypothesis

The main aim of this thesis is to develop an efficient XML labelling scheme focusing on the size of XML labels. A possible way to achieve this is using Fibonacci encoding to store XML labels in a form that is compact but will still support XML data updates. The efficiency of query processing can be dramatically increased because this allows for both fast decoding and short labels.

In Chapter 4, several prefix-encoding methods were described such as Fibonacci encoding, Elias-delta, and Elias-Fibonacci coding (Walder et al., 2012). In spite of the existence of these methods and the research supporting their usefulness for data compression (see Chapter 4), prefix encodings have never been applied as an alternative encoding technique for XML labels. Motivated by this, all the presented prefix-encoding techniques described in Chapter 4 have been studied. The experimental implementation and results of this study are presented later in Chapter 6 and Chapter 7, respectively. The comparison between these prefix-encoding methods in terms of the compression performance has been published at the WEBIST-2016 conference (Al-Zadjali and North, 2016).

As stated in Chapter 4, one of the properties of Fibonacci codes that there are no m consecutive 1-bits within the summation result of Fibonacci numbers of order m, whereas each Fibonacci code ends up with exactly m consecutive 1-bits. Thus, for an integer, x, the appearance of m-consecutive 1-bits in Fibonacci codes $F^{(m)}(x)$ indicates the ends of the binary representation of x. Motivated by this, Fibonacci

encoding is considered a viable alternative storage mechanism for XML labels. The criteria of the existence of exactly m-consecutive 1-bits only at the end of each Fibonacci code can be used to indicate the separators in prefix-based labels, and so avoiding the need to store the delimiters separately. Nevertheless, the Fibonacci decoding process has the capability to facilitate the query processing as examined in this thesis.

Seeing that Fibonacci coding represents non-negative integers $x > 0$ into binary-string (see Section 4.5.1), it is fundamental to consider an XML labelling scheme that generates integer labels. However, at present, such schemes do not fully support node insertion, especially between two consecutive sibling nodes such as in Dewey (Tatarinov et al., 2002), DDE (Xu et al., 2009), DFPD (Liu et al., 2013), Vector-order (Xu et al., 2007), and ORDPATH (O'Neil et al., 2004). This is because the computation of a new integer label value between two consecutive siblings depends on the remaining integers that satisfy the XML labelling scheme properties. Similar to interval-based labelling schemes, the problem is obviously that there is a finite number of integers between two consecutive integer values (Ren et al., 2006) (Sans and Laurent, 2008) (Amagasa et al., 2003).

On the other hand, XML labelling schemes that consider node order lexicographically (see Section 5.4) rather than numerically are more capable of handling skewed insertions (O'Connor and Roantree, 2013) (Chiew et al., 2014a), such as in QED (Li and Ling, 2005b), SCOOTER (O'Connor and Roantree, 2012), and ImprovedBinary (Li and Ling, 2005a). Both QED and SCOOTER use quaternary strings to label XML data, whereas ImprovedBinary employs binary strings. Recently, DPLS (Liu and Zhang, 2016) have enhanced the DFPD scheme (Liu et al., 2013) to handle node insertions based on lexicographical order. The DPLS labels initialisation algorithm concurs with the Dewey order labelling scheme (Tatarinov et al., 2002). In case of insertions, DPLS represents self-label values of the new nodes as fractions, which are then encoded in a similar manner to ORDPATH labels. Consequently, DPLS labels are subject to overflow problems, as with other floating-point labelling schemes (Liu et al., 2013) (Amagasa et al., 2003). Furthermore, the decoding process in DPLS inherits the complexity of ORDPATH coding, so the XML querying is relatively slow as stated in Section 4.4.4.

According to (Chiew et al., 2014a), the SCOOTER labelling scheme is the most compact dynamic labelling scheme, which controls the growth of label size when XML is updated by automatically reusing the smallest deleted node label if available. As discussed in Section 3.4, in spite of the advantages of the SCOOTER labelling

scheme, as with other prefix-based labelling schemes it suffers from the overflow problem during skewed insertions, where the label size grows rapidly (Chiew et al., 2014a) (Ghaleb and Mohammed, 2013).

To enhance the performance of XML updates and simultaneously reduce label size via Fibonacci coding, a new XML labelling scheme is proposed in this thesis called "Base-9". The main objective of the Base-9 scheme is to allocate as many integer labels as possible between any two consecutive nodes. Motivated by the SCOOTER labelling scheme (O'Connor and Roantree, 2012), the Base-9 scheme considers node order lexicographically using decimal strings rather than quaternary strings in order to enable a larger range of integers. The next section presents in detail the main principles of the new proposed Base-9 labelling scheme.

5.4 The Principles of the Base-9 Labelling Scheme

The aim of proposing the Base-9 scheme in this thesis is to facilitate the querying and updating process of dynamic XML trees without sacrificing storage overhead, especially in the case of recurrent node insertions. Therefore, whilst designing the Base-9 scheme, it is important to consider the desirable properties of a good dynamic XML labelling scheme (Chapter 3, Section 3.2): i.e., deterministic, efficient, compact, and dynamic. Accordingly, the principles of the Base-9 scheme are stated as follows:

- The Base-9 labelling scheme uses decimal strings to represents XML labels as with a prefix-based labelling scheme. The initialisation process, as well as the structural relationship determination, are based on a lexicographical order (Hye-Kyeong and SangKeun, 2010) (Li et al., 2008) rather than a numerical one.

- The initialisation process verifies the compactness of Base-9 labels by considering the maximum number of child per node; this mechanisim is adapted from the SCOOTER labelling scheme (O'Connor and Roantree, 2012) and enhanced to use all the decimal strings including '0' as part of the label values (explained in detail in the next section).

- To preserve the node order lexicographically when XML is updated, the insertion algortihms generate labels by obtaining the nearest possible lexicographical number to the current label, where the new node is inserted immediately before or after the current label. This principle not only maintains the efficiency of structural relationship determination after XML updates, but also provides for the re-use of deleted nodes labels if they exist.

- Base-9 labels are encoded using Fibonacci coding, which is updated to include integer '0' (see Section 5.7). The Fibonacci encoding conserves minimum memory space by omitting the storage of delimaters, ".". Nevertheless, the Fibonacci decoding process facilitates the XML query processing.

In order to present a comprehensive understanding of the Base-9 initialisation, insertion and determination process, it is important to first present the definition of lexicographical order (Li et al., 2008) (Li and Ling, 2005b) (Li et al., 2006a) (Hye-Kyeong and SangKeun, 2010) (Min et al., 2007) between Base-9 labels.

Definition 5.1: Lexicographical Order

Let $\prec$ denote a lexicographical ordering relation on the set $\{1,\ 2, \dots, n\}$ such that $i \prec j$ implies $i < j$. The main principle of generating labels in the Base-9 scheme is basically that of generating all permutations $a_1 a_2 \dots a_n$ of the decimal number set $\{0,1,\ 2, \dots, 9\}$ such that $i < j$ implies $a_i < a_j$. Given two Base-9 labels B_L= $a_1 a_2 \dots a_n$ (as the left node label) and B_R= $b_1 b_2 \dots b_n$ (as the right node label), then $B_L \preccurlyeq B_R$ if, and only if, B_L and B_R both satisfy the following conditions:

- B_L is lexicographically equal to B_R if they are exactly the same.
- B_L is lexicographically smaller than B_R (i.e., $B_L \prec B_R$) if:
    - During the lexicographical comparison from left to right of B_L and B_R, the current digit of B_L is smaller than the current digit of B_R.
    - B_L is a prefix of B_R.

5.5 Base-9 Labels Initialisation

Similar to the SCOOTER labelling scheme (O'Connor and Roantree, 2012), the Base-9 labelling scheme generates labels based on the combinatorial number system (Knuth, 1979). The combinatorial number system of degree $k > 0$ is a correlation between positive natural numbers N and k-combinations represented as strings in strictly increasing sequences $a_0 < a_1 < \cdots < a_{k-1} < a_k$ whereby, distinct numbers corresponding to distinct k-combinations are produced in lexicographic order (Knuth, 1979). More formally, if a set $S \subset N$, such that $S = \{0, 1, \dots, n\}$, then a k-combination is a subset of k distinct elements of S. Unlike SCOOTER, which uses a set of quaternary numbers $S = \{0, 1, 2, 3\}$, Base-9 uses the decimal number set $S = \{0, 1, \dots, 9\}$ to generate distinct labels in lexicographical order.

The algorithm used to assign the initial labels of the SCOOTER labelling scheme was adapted and enhanced to use the set of decimal numbers including '0' rather than

quaternary numbers excluding '0'. In SCOOTER, the maximum number of labels with length k is $(3^k - 1)$ labels. If $(3^k - 1)$ corresponds to the minimum number of digits needed to represent child nodes (referred to as $\mathrm{childCount}$) in base 3, then a maximum label length k (referred to as $maxLabelSize$) can be computed by the formula $[log_3(childCount + 1)]$. Each child node is then assigned a SCOOTER label of a size no greater than $maxLabelSize$.

Similarly, in Base-9 the maximum label size k was determined based on the minimum number of digits required to represent x child nodes using the formula $k = [log_9(x + 1)]$. Mathematically, there is a unique number k such that $b^k = x > 1$ can be denoted as the logarithm of x to base b; i.e., $k = log_b(x)$. Base-9 was chosen here because during label initialisation only the set of 9 elements $\{1, \ldots, 9\}$ were used. The digit '0' was reserved for later node insertions except for the root node, which is the only node labelled as "0", due to the Fibonacci code properties (discussed in Section 5.7). To verify the compactness of the Base-9 labels, each label must be no longer than the permissible maximum label size (refered to as $maxLabelSize$). Starting with the first child node (the left most), this node will always have a Base-9 label of length equal to $maxLabelSize$. Figure 5.1 shows the XML tree of the XML "School" sample in Figure 2.1 labelled here using the Base-9 scheme as a prefix-based labelling scheme. The root node (School) is assigned a label value "0".

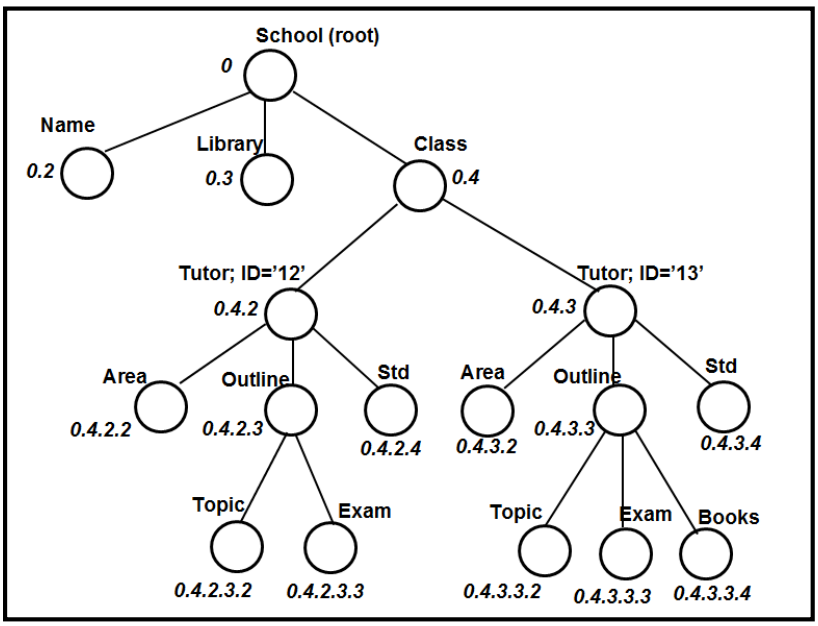


Figure 5.1 XML tree labelled by the Base-9 scheme

As with the SCOOTER labelling scheme, in the Base-9 scheme the first child node's label must not end with a digit less than '2'. Consequently, a node label must not consist entirely of consecutive '1' digits. This is in order to maintain the lexicographical

order between sibling nodes and support XML updates, particularly in the case of skewed insertions before the first child. To justify this rule, let us assume that the left most child node's label can end with digit '0' or '1', as shown in the following example.

Example: Suppose that $maxLabelSize = 3$, and the first child node, n_1, has the self-label "112". When a new node, n_{new}, is added before n_1, a possible Base-9 self-label (say $Nnew$) is generated that corresponds to the immediate preceding lexicographical value to the n_1 self-label; i.e., $Nnew =$ "111". Accordingly, if $j > 1$, new nodes are inserted before n_1, then each n_j is assigned a Base-9 self-label $Nnew$ as follows: $Nnew_1 =$ "111", $Nnew_2 =$ "110", $Nnew_3 =$ "11", $Nnew_4 =$ "10", $Nnew_5 =$ "1", $Nnew_6 =$ "0". Since only the root node can be labelled as "0", the number of new sibling node insertions before the left most node is limited, in this example to $j = 5$.

In general, there are a total of n^k k-combinations representing the lexicographically ordered permutations of a set S (with n elements) and of length k; i.e., 9^k in the Base-9 scheme. Since there are restrictions on how often the digit '1' can appear, in which the total number of k-combinations has to be adjusted to $(9^k - 1)$ in order to exclude the possible label value of all consecutive '1's within a set of k-combinations. Therefore, the actual total number of x nodes that fit within maximum label size k in base 9 is $(9^k - 1)$. Accordingly, the computation of the maximum label length k (= $maxLabelSize$) in SCOOTER is modified to $maxLabelSize = \lceil log_9(childCount + 1) \rceil$, as illustrated in line 1 of the assigning Base-9 initials algorithm (refered to "**AssignBase9Initials**") in Figure 5.2.

Base-9: AssignBase9Initials Algorithm

Input: a parent node (P) label and number of child nodes of P ($childCount$)
Output: Base-9 self-labels of each child node of P.
Comment: complete Base-9 label of a node = a parent node label ⊕ selfLabel [child]

```
1  maxLabelSize ← ⌈log9(childCount + 1)⌉   //find maximum label size
2  selfLabel[1] ← null                      //self-label value of first child (most left)
3  for (i = 1; i < maxLabelSize; i++) do
4  |    selfLabel[1] ← selfLabel[1] ⊕'1'    // ⊕ denotes concatenation
5  end
6  selfLabel[1] ← selfLabel[1] ⊕'2'
7  //based on self-label value of first child continue computing the self-label values of the next siblings
8  for (i = 2; i <= childCount; i++) do
9  |    selfLabel[i] ← nextSiblingLabel(selfLabel[i − 1], maxLabelSize)
10 end
11 return selfLabel[childCount]
```

Figure 5.2 Assigning Base-9 Initials algorithm

Like the SCOOTER labelling scheme, in the Base-9 scheme the XML data has to be parsed to determine the number of child nodes (i.e. $childCount$) for each XML element before assigning the initial Base-9 labels. Then, by applying the

"**AssignBase9Initials"** algorithm, the $maxLabelSize$ of possible label values to be assigned to the child nodes of an element are computed in line 1. Notice that for each element $maxLabelSize$ value differ depending on its $childCount$ number. Based on this $maxLabelSize$, the label value of the first child (left-most) is allocated (lines 2-6) by generating a sequence of $(maxLabelSize - 1)$ of '1's followed by '2'. As the node order between sibling nodes is lexicographic, the rest of the child nodes from left to right (as represented in the XML tree) are then labelled, as based on the label of the adjacent left sibling, by calling the "**nextSiblingLabel**" algorithm in line 9. Notice that $maxLabelSize$ can either be stored with parent node label or computed based on first child node as in first line of "**AssignBase9Initials"** algorithm.

Figure 5.3 presents the algorithm used to compute the next sibling label (refered to as "**nextSiblingLabel**") given the label value of the immediate left child node. The "**nextSiblingLabel"** algorithm is adapted from SCOOTER labelling scheme and enhanced to consider the set of 9 elements $\{1, \dots, 9\}$ instead of the quaternary string set $\{1, 2, 3\}$.

Base-9: nextSiblingLabel Algorithm

Input: previous sibling node label ($Nleft$), and $maxLabelSize$
Output: $Nnew$ - current node self-label such that $Nleft \prec Nnew$.

/* Note: $maxLabelSize$ is computed based on parent's total number of child nodes and sent as input to this algorithm (see AssignBase9Initials algorithm)*/

```
1  Ntemp ← Nleft
2  if (length(Ntemp) == maxLabelSize) then    //in case Nleft label equals to maximum label size
3  |    remove all occurrences of '9' from end of Ntemp
4  |    Nnew ← Ntemp with last symbol incremented by 1
5  else if (length(Ntemp) < maxLabelSize) then // in case Nleft label less than maximum label size
6  |    Nnew ← Ntemp
7  |    for (i = length(Ntemp) + 1; i ≤ maxLabelSize; i++) do
8  |    |      Nnew ← Nnew ⊕ '1'
9  |    end
10 end
11 return Nnew
```

Figure 5.3 Computing next sibling label algorithm

Table 5.1 presents an example of Base-9 and SCOOTER initial self-labels generated assuming the number of child nodes is 50. The algorithms to assign initial labels of Base-9 and SCOOTER start by obtaining a $maxLabelSize$ of 50 nodes' labels (in line-1); which matches with 2 in Base-9 and 4 in SCOOTER. According to the first two rules, the first child label is assigned the digit '2' preceded by a sequence of $(maxLabelSize - 1)$ of digit '1'. This corresponds to the label "12" in Base-9 and "1112" in SCOOTER (as shown in Table 5.1). The labels of the remaining child nodes are then each generated by incrementing the label of their immediate preceding node lexicographically using the set $\{1, \dots, 9\}$ in Base-9 and $\{1, 2, 3\}$ in SCOOTER.

Table 5.1 Examples of Base9 and SCOOTER labels

Child number	Base-9	SCOOTER	Child number	Base-9	SCOOTER
1 (1st from left)	12	1112	**26**	37	1333
2	13	1113	**27**	38	2
3	14	112	**28**	39	2112
4	15	1122	**29**	4	2113
5	16	1123	**30**	41	212
6	17	113	**31**	42	2122
7	18	1132	**32**	43	2123
8	19	1133	**33**	44	213
9	2	12	**34**	45	2132
10	21	1212	**35**	46	2133
11	22	1213	**36**	47	22
12	23	122	**37**	48	2212
13	24	1222	**38**	49	2213
14	25	1223	**39**	5	222
15	26	123	**40**	51	2222
16	27	1232	**41**	52	2223
17	28	1233	**42**	53	223
18	29	13	**43**	54	2232
19	3	1312	**44**	55	2233
20	31	1313	**45**	56	23
21	32	132	**46**	57	2312
22	33	1322	**47**	58	2313
23	34	1323	**48**	59	232
24	35	133	**49**	6	2322
25	36	1332	**50**	61	2323
Maximum Label Size: in Base-9 is ***2*** and in SCOOTER is ***4***					

The node order between any two sibling nodes is determined lexicographically. For instance, assume nodes u and v share the same parent prefix-label and have self-labels in Base-9 of “43” and “2193”, respectively. Since in digit-by-digit comparison the first digit in $v = '2' < '4'$, thus $v \prec u$ implies that node v is a precedent sibling to node u in the XML tree.

5.6 Handling Insertions

This section describes how the Base-9 labelling scheme supports XML updates by controlling insertions based on the lexicographical relation between node labels, similar to the initialisation principles. In most cases, three distinct insertion scenarios are considered (see Figure 5.4):

- Insert a new node after the right most child.
- Insert a new node before the left most child.
- Insert a new node between two consecutive sibling nodes.

All these types of insertions focus on adding a sibling node in an XML tree, whereas, a child (i.e., leaf) node in the Base-9 labelling scheme is treated by

AssignBase9Initials algorithm. This thesis does not cover the case of inserting a node p between a parent node A and a child node B, where nodes A and B are linked directly in an XML tree (e.g., as in Figure 5.4).This process is a very expensive as it requires the re-labelling all of the new node descendants. (Liu et al., 2014) has investigated the possibility of minimising the number of re-labelling events in such a case by using a dynamic state transducer (DST) to decode node names during the on-line processing step. Nevertheless, the results of (Liu et al., 2014) study has shown that this case is still time consuming and requires the re-labelling of all the descendants of the new node p. In addition, (Mirabi et al., 2012) have shown that when a new parent node is inserted into an XML tree, all the prefix labelling schemes tested (O'Neil et al., 2004) (Mirabi et al., 2012) (Li and Ling, 2005b) required the re-labelling of all the descendants of the newly inserted parent node.

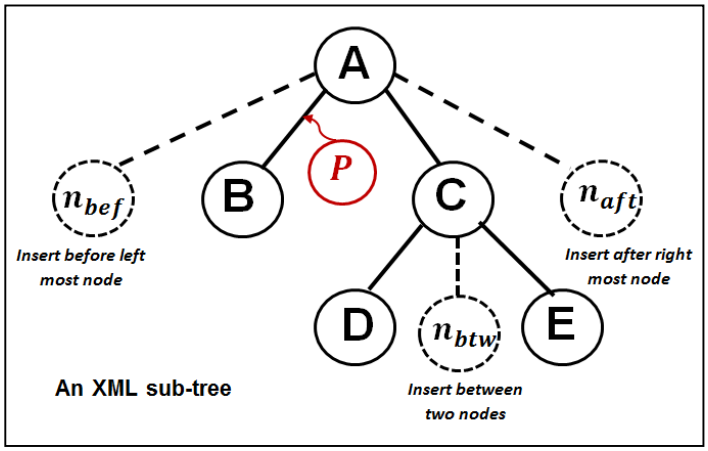


Figure 5.4 Types of insertions

Unlike the SCOOTER scheme, the Base-9 labelling scheme handles insertions by finding the most immediate obtainable lexicographical value in comparison to the label of the node, n_{old}, where n_{old} is the adjacent sibling to the new node, n_{new}. This approach also allows Base-9 to re-use the deleted labels (if any) automatically, as illustrated in the following sections. Notice here the terms “label” and “self-label” are used interchangeably.

5.6.1 Insertion After the Right-most Node

When a new node n_{new} is inserted after the right most node, n_{old}, within an XML sub-tree (see Figure 5.5), the Base-9 labelling scheme computes the n_{new} self-label (say $Nnew$) by incrementing the self-label of n_{old} (say $Nold$) lexicographically using the "**InsertAfterRightMost**" algorithm presented in Figure 5.6. The algorithm allocates a $Nnew$ value following a similar notion to the "**nextSiblingLabel**" algorithm. That is, $Nnew$ is generated by first comparing the length of $Nold$ with $maxLabelSize$, as follows:

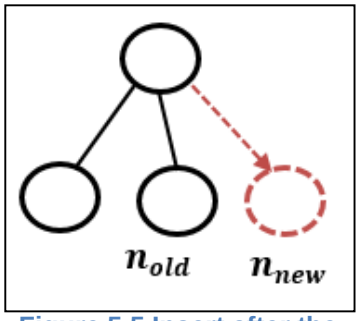


Figure 5.5 Insert after the right-most node

Base-9: InsertAfterRightMost Algorithm

Input: $maxLabelSize, Nleft$ – self-label of the last child (right most); where the new node is inserted after
Output: $Nnew$ – current node self-label newly inserted after last child node such that $Nleft \prec Nnew$

```
1  Nnew ← null
2  if (length(Nleft) < maxLabelSize) then
3  |    Nnew ← Nleft ⊕'1'
4  else
5  |    if (last symbol of Nleft < 9) then
6  |    |       Nnew ← Nleft with (last symbol + 1)
7  |    else if (last symbol of Nleft == 9) then
8  |    |       p ← position of first non '9' digit from end of Nleft
9  |    |       if (p does not exists) then //(p < 0): all digits of Nleft = '9'; i.e. Nleft consists of all consecutives '9'
10 |    |       |   Nnew ← Nleft ⊕'1'
11 |    |       else                    //there is none '9' digit in Nleft located in position p
12 |    |       |   Symbol = the first non '9' digit from end of Nleft (reverse tracing: right to left)
13 |    |       |   Nnew = substring (Nleft, 0, p − 1) ⊕(Symbol + 1)
14 |    |       end
15 |    end
16 end
17 return Nnew
```

Figure 5.6 Base-9 Insert after right-most algorithm

If $Nold$ is less than $maxLabelSize$, the algorithm generates the $Nnew$ value simply by extending $Nold$ with the digit '1'. However, if the $Nold$ size equals $maxLabelSize$, then $Nnew$ is obtained by first tracing $Nold$ in reverse order (i.e., starting from the end) to locate the position (p) of the first non '9' (if any) from the end of $Nold$, and accordingly $Nnew$ is allocated as described in Table 5.2:

Table 5.2 Insert after $Nold$: find $Nnew$ when $Nold$ is $maxLabelSize$

When $Nold$ is of size $maxLabelSize$.		
Condition	**Rule**	**Example**
If the last digit d_L in $Nold$ is less than '9'	$Nnew$ is $Nold$ with d_L replaced by $d_L + 1$	If $maxLabelSize = 3$ and $Nold = "254"$ then $Nnew = "255"$
If the last digit d_L in $Nold$ is '9' and $p < 0$; i.e., $Nold$ consists of all consecutive '9's	Increase $maxLabelSize$ by 1 and $Nnew$ is $Nold$ appended by digit "1"	If $maxLabelSize = 3$ and $Nold = "999"$ then $Nnew = "9991"$ and new $maxLabelSize = 4$
if the last digit d_L in $Nold$ is '9' and $p \geq 0$. Notice that $p = 0$ indicates the position of first non '9' digit. D is the first digit in $Nold$	$Nnew$ is substring of $Nold$ up to digit D; whereas D value is incremented by 1	If $maxLabelSize = 5$ and $Nold = "49699"$ then $Nnew = "497"$

In contrast, the "**nextSiblingLabel**" algorithm does not deal with the case when $Nold$ consists of all consecutive '9's, since initially the nodes are labelled based on the premise that they all fit within $maxLabelSize$. Moreover, when $Nold$ is smaller than $maxLabelSize$, the "**nextSiblingLabel**" algorithm allocates an initial Base-9 label by adding a sequence of '1' digits at the end of $Nold$ up to $maxLabelSize$.

To demonstrate an example of how the Base-9 labelling scheme handles insertions Figure 5.7 shows labels generated by the the "**InsertAfterRightMost"** algorithm when new nodes $n_a, n_b,$ and n_c are inserted, respectively, after the element "Class" in the XML tree of the "School" sample shown in Figure 5.1.

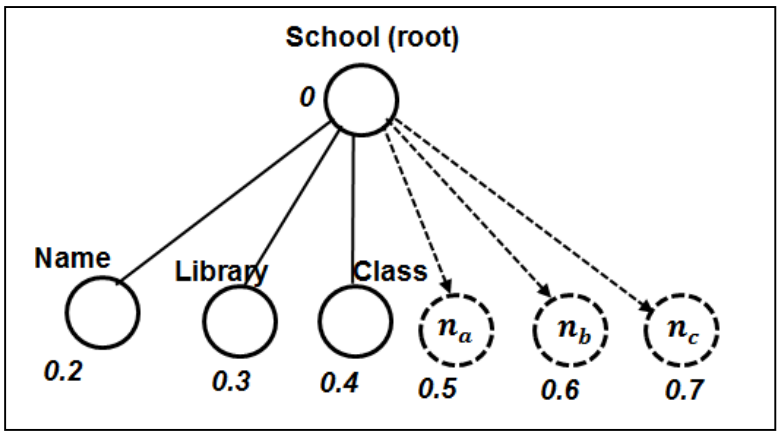


Figure 5.7 Example of handling insertions after the right-most node

The "**InsertAfterRightMost**" algorithm also supports skewed insertions after the right most node. For simplicity, Table 5.3 shows up to 10 new node insertions after the right most node, n_{old}, labelled as $Nold = "56", "1254", "9999"\ or\ "4969";$ assuming $maxLabelSize = length(Nold)$.

Table 5.3 Examples of skewed insertions after right-most node in Base-9

	Node labels	Node labels	Node labels	Node labels
Label of the right most node	**56**	**254**	**9999**	**49699**
1	57	255	99991	497
2	58	256	99992	4971
3	59	257	99993	49711
4	6	258	99994	49712
5	61	259	99995	49713
6	62	26	99996	49714
7	63	261	99997	49715
8	64	262	99998	49716
9	65	263	99999	49717
10	66	264	999991	49718

Notice that Base-9 controls the growth of the label size during insertions by considering the availability of the nearest lexicographical value to $Nold$ whilst generating the new labels. This leads to smaller increments in label size compared to the SCOOTER scheme (see Section 3.4.3). In SCOOTER, for any $Nold$ value the new labels generated are controlled via the growth-adaptive mechanism by which new label values after at most two insertions will always start with consecutive "3" digits, and so after $(3^{postfix\ length} - 1)$ insertions the label size increases by $(postfix\ length + 1)$.

5.6.2 Insertion Before the Left-most Node

The algorithm for insertion before the left most node in Base-9 is named "**InsertBeforeLeftMost**" and is presented in figure 5.9. This algorithm is designed to generate Base-9 self-labels that are lexicographically smaller than $Nold$, where $Nold$ is the self-label value of the current left most node n_{old} within an XML sub-tree (see Figure 5.8). When a new node, n_{new}, is inserted before n_{old}, there are two main factors that affect the newly generated label (say $Nnew$): the current $maxLabelSize$ (must be ≥ 1) and the start value of $Nold$.

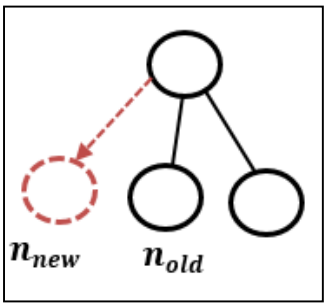


Figure 5.8 Insert before the left-most node

Base-9: InsertBeforeLeftMost Algorithm

Input: $maxLabelSize \geq 1$ and $Nright$ – self-label of the first child (left most); where the new node is inserted before
Output: $Nnew$ – current node self-label newly inserted before left most node such that $Nnew \prec Nright$

```
1  Nnew ← null
2  if (first symbol of Nright > '1') then      // possible case of available deleted node
3      if (length(Nright) > 1) then            // Nright has more than one digit
4          if ((last symbol of Nright == '2') and (length(Nright) < maxlabelSize)) then
5              Nnew ← Nright with last symbol replaced by '19'
6              CONTROL SKEWED INSERTION STATMENT
8          else if (last symbol of Nright > 2) then
9              Nnew ← Nright with (last symbol – 1)
10             CONTROL SKEWED INSERTION STATMENT
11         else
12             Nnew ← Nright with last symbol removed
13         end
14     else                                    // case of Nright of minimum length equals 1
15         if (Nright > 2) then Nnew ← Nright − 1
16         else if (Nright == 2) then
17             Nnew ← '19'
18             CONTROL SKEWED INSERTION STATEMENT
19         end
20     end
21     return Nnew
22 end
23 p ← position of last '1' digit from start of Nright  /* case no deleted nodes OR first child starts with '1' */
24 firstNon1 ← value of first non '1' digit from start of Nright
25 if (p < length(Nright)) then                //case of first non '1' digit is not last symbol of Nright
26     if (last symbol of Nright > 2) then
27         Nnew ← Nright with (last symbol – 1)
28         CONTROL SKEWED INSERTION STATEMENT
29     else if (firstNon1 is not '0') then
30         if (last symbol of Nright == 2) then
31             Nnew ← Nright with (last symbol – 1)
32             CONTROL SKEWED INSERTION STATEMENT
33         else if (last symbol == 1) then
34             Nnew ← Nright with last symbol removed
35         end
36     else if (firstNon1 == '0') then
37         if (p == 1) then        // case of '0' is second digit in Nright
38             D ← first none '0' digit from start of Nright
39             if (D == 1) then Nnew ← Nright with D replaced by '0'
40             else      //if D== 2 then replace '2' by '19' and trim the rest of Nright
41                 Nnew ← substring(Nright, 0, p) ⊕ "19"
42             end
43         else                    //case of '0' is NOT the second digit inNright and preceded by '1'; i.e. p>2
44             Nnew ← substring(Nright, 0, p − 1) ⊕ '09'  //(substring of Nright up to digit '1' before '0') ⊕ '09'
46         end
47     end
48 else if (p == length(Nright)) then  // if first non '1' is last symbol
49     if (last symbol of Nright > 2) then Nnew ← Nright with (last symbol – 1)
50     else if (last symbol of Nright == 2) then
51         if (length(Nright) ≤ maxlabelSize) then
52             Nnew ← Nright with last symbol replaced by '19'
53             CONTROL SKEWED INSERTION STATEMENT
54         else
55             Nnew ← Nright with last two digits replaced by '09'
56         end
57     end
58 end
59 return Nnew
```

CONTROL SKEWED INSERTION STATEMENT:
While $(length(Nnew) < maxLabelSize)$ **do**
    $Nnew \leftarrow Nnew \oplus$ '9'
end

Figure 5.9 Base-9 insert before left-most node algorithm

According to the Base-9 initialisation process, a minimum self-label value assigned to the left-most node, n_{old}, is $Nold =$ "2" if $maxLabelSize$ is 1. If not, then for $maxLabelSize$ greater than 1 $Nold$ must start with the digit '1' (see Section 5.5). If n_{new} is added before $Nold =$ "2", then $Nnew$ is given the value "19", which is the immediate preceding lexicographical value to "2". However, if $Nold$ starts with a digit greater than '2', this indicates there is a deleted node's label available that can be re-used for $Nnew$ as follows (Table 5.4):

Table 5.4 Insert before $Nold$, find $Nnew$ if $Nold$ starts with *digit >2*

When $Nold$ starts with a digit greater than '2'		
Condition	**Rule**	**Example**
If the last digit d_L in $Nold$ is greater than '2'	$Nnew$ is $Nold$ with d_L replaced by $d_L - 1$	if $Nold =$ "456" then $Nnew =$ "455"
If the last digit d_L in $Nold$ is '2' and $Nold$ size is less than $maxLabelSize$	$Nnew$ is $Nold$ with d_L replaced by "19"	If $maxLabelSize = 4$ and $Nold =$ "232" then $Nnew =$ "2319"
Otherwise; i.e., if the last digit d_L in $Nold$ is '2' and $Nold$ size is the $maxLabelSize$. Or the last digit d_L in $Nold$ is less than '2'	$Nnew$ is $Nold$ with d_L is trimmed	If $maxLabelSize = 3$ and $Nold =$ "232" then $Nnew =$ "23"

When initialising Base-9 labels where $maxLabelSize$ is greater than 1, $Nold$ starts initially with consecutive '1's followed by the digit '2'. In this case, to avoid the scenario of generating a self-label comprised entirely of consecutive '1's that might limit the number of insertions before the first child node (explained in Section 5.5), $Nnew$ is allocated based on the $Nold$ size as follows (Table 5.5):

Table 5.5 Insert before $Nold$, find $Nnew$ if $Nold$ starts with consecutive '1's⊕'2'

When $Nold$ starts with consecutive '1's followed by the last digit d_L = '2'		
Condition	**Rule**	**Example**
If $Nold$ size $\leq$ $maxLabelSize$	$Nnew$ is $Nold$ with d_L replaced by "19". This might make the $Nnew$ size greater than $maxLabelSize$ (next condition)	If $maxLabelSize = 2$ and $Nold$ = "12" then $Nnew$ = "119"
If $Nold$ size $>$ $maxLabelSize$	$Nnew$ is $Nold$ with d_L replaced by "09"	If $maxLabelSize = 2$ and $Nold$ = "112" then $Nnew$ = "109"

Another possible situation to consider is when $Nold$ starts with the digit '1' but the first non '1' digit is not the last digit, d_L, of $Nold = ('1's \oplus d_f \ldots \oplus d_L)$. In this case, different scenarios are considered based firstly on the d_L value and then on d_f, as shown in the following steps (Table 5.6) that are triggered to allocate $Nnew$:

Table 5.6 Insert before $Nold$, find $Nnew$ if $Nold$ starts with $'1's \oplus d_f \dots \oplus d_L$

<table>
<tr><td colspan="3">When inserting before $Nold$ that starts with the digit ‘1’ but there is a non ‘1’ digit d_f in position $p \geq 2$ of $Nold$ where p is not the last digit d_L of $Nold$; notice that d_L must be ≥ ‘2’</td></tr>
<tr><th>Condition</th><th>Rule</th><th>Example</th></tr>
<tr><td>If $d_L >$ ‘2’</td><td>$Nnew$ is $Nold$ with d_L reduced by 1</td><td>$Nold = \text{“1124”}$ then $Nnew = \text{“1123”}$</td></tr>
<tr><td>If $d_L =$ ‘2’</td><td colspan="2">The following cases are considered relying on d_f:</td></tr>
<tr><td>If $d_f = 1$</td><td>$Nnew$ is $Nold$ with d_L is trimmed</td><td>If $Nold = \text{“11341”}$ then $Nnew = \text{“1134”}$</td></tr>
<tr><td>If $d_f \geq 2$</td><td>$Nnew$ is $Nold$ with d_L is reduced by 1</td><td>If $Nold = \text{“1134”}$ then $Nnew = \text{“1133”}$</td></tr>
<tr><td rowspan="2">If $Nold$ starts with “10”. In this case, there must be a non ‘0’ digit (say D) from the start of $Nold$, where $D \in \{1, 2\}$. Notice here $d_f = 0$.</td><td>Trace the first non ‘0’ digit D. For $D =$ ‘1’; $Nnew$ is $Nold$ with D replaced by “0”</td><td>If $Nold = \text{“10112”}$ then $Nnew = \text{“10012”}$</td></tr>
<tr><td>For $D =$ ‘2’; $Nnew$ is $Nold$ with D replaced by “19” and the rest of $Nold$ is trimmed</td><td>If $Nold = \text{“10022”}$ then $Nnew = \text{“10019”}$</td></tr>
<tr><td>Otherwise; if d_f is ‘0’ and $p > 2$</td><td>$Nnew$ is assigned a substring of one less consecutive ‘1’s at the start of $Nold$ concatenated by “09”</td><td>If $Nold = \text{“11102”}$ then $Nnew = \text{“1109”}$</td></tr>
</table>

Figure 5.10 illustrates an example of how the Base-9 labelling scheme handles the insertion of new nodes $n_a, n_b,$ and n_c, respectively, before the element “Name” (the left-most child) in the XML tree of the “School” sample (Figure 5.1).

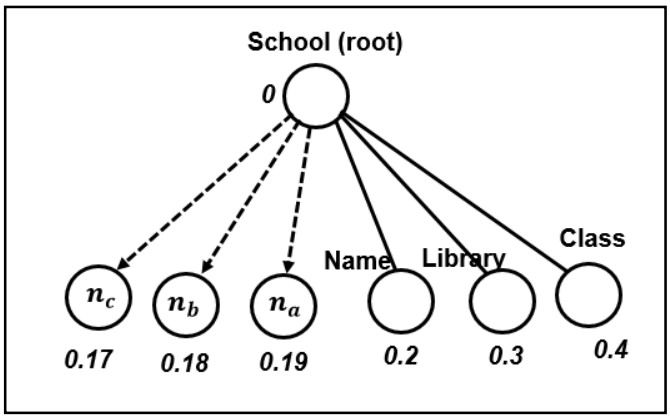


Figure 5.10 Example of handling insertions before the left most node

To maximise the availability of Base-9 self-labels for more skewed insertions before the left-most node, n_{old}, the control skewed insertion statement is added assuming $maxLabelSize$ exceeds the length of $Nold$. Table 5.7 displays new Base-9 self-labels generated when 10 new nodes are inserted repeatedly before the left-most node, n_{old}, labelled as $Nold = 232", "112", "100112", "1002" \; or \; "19"$, assuming $maxLabelSize = length(Nold)$.

Table 5.7 Examples of skewed insertions before the left-most node in Base-9

	Node labels	Node labels	Node labels	Node labels	Node labels
Insert before node	**232**	**112**	**100112**	**1002**	**19**
1	23	1119	100012	10019	18
2	229	1118	100002	10018	17
3	228	1117	1000019	10017	16
4	227	1116	1000018	10016	15
5	226	1115	1000017	10015	14
6	225	1114	1000016	10014	13
7	224	1113	1000015	10013	12
8	223	1112	1000014	10012	119
9	222	1109	1000013	10002	118
10	22	1108	1000012	100019	117

As can be seen from Table 5.7, the length of a newly generated label grows by at most one digit per 10 repeated insertions before the left-most node. On the contrary, in the SCOOTER labelling scheme (see Section 3.4.3), after a few insertions before the left-most node, the new labels generated by the adaptive growth mechanism start to form a pattern of consecutive ‘1’s followed by a ‘2’. This leads to rapid increases in the label size up to at least 1 digit per insertion (e.g., see Table 3.3 in Section 3.4.3).

5.6.3 Insertion Between Two Nodes

When inserting a new node, $\mathrm{n_{new}}$, between two consecutive sibling nodes, $\mathrm{n_{left}}$ and $\mathrm{n_{right}}$, with self-labels $Nleft$ and $Nright$, respectively (see Figure 5.11), the Base-9 labelling scheme considers three different cases depending on the label size of $\mathrm{n_{left}}$ and $\mathrm{n_{right}}$, similar to the SCOOTER scheme (see Section 3.4.3):

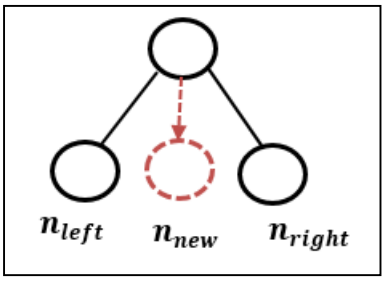


Figure 5.11 Insert between two sibling nodes

1. $Nleft$ is shorter than $Nright$.
2. $Nleft$ is longer than $Nright$.
3. Both $Nleft$ and $Nright$ have the same size.

In all these cases, the Base-9 labelling scheme provides an insertion mechanism that generates the shortest label $Nnew$ such that $Nleft \prec Nnew \prec Nright$. Each case is described below.

- ***Case $Nleft$ Shorter Than $Nright$***

The "**InsertBetweenLessThan**" algorithm (Figure 5.12) is designed to obtain a new node label (say $Nnew$) inserted between two nodes, where $Nleft$ is shorter than $Nright$. When generating the initial Base-9 labels, n_{left} has a shorter label than the adjacent next sibling node n_{right} if, and only if, the n_{left} label (say $Nleft$) is a prefix string of the n_{right} label (say $Nright$) (for examples, see Table 5.1). In order to preserve the lexicographical relation between sibling nodes, $Nleft$ also has to be a prefix string to $Nnew$ (i.e., $Nnew = Nleft \oplus new\,postfix$). Determining the postfix value of $Nnew$, relies on the $Ntemp$ value which corresponds to $Nright$ after excluding the prefix part matching $Nleft$. If $Ntemp$ starts with a digit $d > 1$, then $Nnew$ is assigned $Nleft \oplus$ "1". For instance, let $Nleft =$ "3" and $Nright =$ "364". Since $Nleft$ is a prefix of $Nright$ and $Ntemp =$ "64" starts with the digit $6 > 1$, thus $Nnew$ = "31". However, if $Ntemp$ starts with a digit $d \leq 1$, then $Ntemp$ is updated by removing all '1' digits from the end of $Ntemp$ (if any). Based on the remaining value in $Ntemp$, $Nnew$ is allocated as follows (see Table 5.8):

Table 5.8 Insert between two nodes (less than), find $Nnew$ if $Nleft$ is prefix of $Nright$

When $Nleft$ is prefix of $Nright$; where $Nright = Nleft \oplus Ntemp$ and $Ntemp$ starts with a digit $d \leq 1$. $Ntemp$ here is generated by removing all '1's from its end (if any); so $Ntemp$ cannot end with the digit d ='1'		
Condition	**Rule**	**Example**
If $Ntemp$ is empty; (means that $Ntemp$ originally consists of consecutive '1's only)	$Nnew$ is assigned $Nleft \oplus$ "01"	If $Nleft =$ "3" and $Nright =$ "3111" → $Ntemp$ = "111" → updated $Ntemp$ is empty then $Nnew =$ "301"
If $Ntemp$ ends with a digit $d =$ '0'	$Nnew$ is $Nleft \oplus Ntemp \oplus$ "01"	If $Nleft =$ "3" and $Nright =$ "3101" → $Ntemp$ = "101" → updated $Ntemp$="10" then $Nnew =$ "31001"
If $Ntemp$ ends with a digit $d >$ '1'	Decrease the last digit d in $Ntemp$ by 1 and then $Nnew$ is located as $Nleft \oplus Ntemp$.	If $Nleft =$ "3" and $Nright =$ "3151" → $Ntemp$ = "151" → updated $Ntemp$="15" then $Nnew =$ "314"

Base-9: InsertBetweenLessThan Algorithm

Comment: Insert new node between 2 consecutive nodes such that $length(Nleft) < length(Nright)$
Input: $Nleft$ – self-label of left node to new node, $Nright$ – self-label of the right node to new node
Output: $Nnew$ – current node self-label newly inserted such that $Nleft \prec Nnew \prec Nright$

```
1  Nnew ← null
2  if (Nleft is prefix of Nright) then
3      Ntemp ← substring (Nright, length(Nleft))  // Nright after excluding prefix = Nleft
4      if (first symbol in Ntemp > 1) then
5          Nnew ← Nleft ⊕'1'
6      else
7          Ntemp ← Ntemp without any '1' at the end  // Remove all '1' digit(s) from end of Ntemp if any
8          if (Ntemp is empty) then  // i.e. Ntemp was sequence of '1's only
9              Nnew ← Nleft ⊕ '01'
10         else if ( last symbol of Ntemp == 0) then  //case last symbol in Ntemp is '0'
11             Nnew ← Nleft ⊕ Ntemp ⊕ '01'
12         else //case last symbol in Ntemp > 1 ; possible of deleted nodes exist
13             Nnew ← Nleft ⊕ Ntemp with (last symbol – 1)
14         end
15     end
16 else  // if Nleft is not prefix of Nright ; possible of deleted nodes exist
17     Nnew ← InsertAfterRightMost(Nleft)
18 end
19 return Nnew
```

Figure 5.12 Insert between two nodes less than algorithm in Base-9 scheme

There is also a possibility that $Nleft$ could be shorter than $Nright$ but not a prefix of $Nright$. This will happen if, and only if, there are is at least one deleted sibling node between n_{left} and n_{right}. Because the Base-9 labelling scheme attempts to re-use deleted labels if available, in this case the "**InsertAfterRightMost"** method (see Section 5.6.1) is invoked assuming the new node is inserted after the right-most node labelled as $Nleft$. For instance, if $Nleft = "254"$ and $Nright = "5711"$, then $Nnew$ is obtained by inserting after the right-most node labelled as $Nleft = "254"$, resulting in $Nnew = "255"$ (see example in Table 5.3).

- ***Case $Nleft$ Greater Than $Nright$***

Considering the initialisation process of the Base-9 labelling scheme, $Nleft$ can be lexicographically smaller than $Nright$ but have a longer label than $Nright$ if, and only if, the last possible combinations of $maxLabelSize$ that start with digit d have been used for $Nleft$. In other words, this occurs when $Nleft$ starts with a digit $d < 9$ followed by a sequence of $(maxLabelSize - 1)$ '9' digits, and $Nright$ is labelled as the next digit (i.e., $(d + 1) \leq 9$). The "**InsertBetweenGreaterThan"** algorithm (see Figure 5.13) was developed to generate a new self-label for a node inserted between two consecutive nodes n_{left} and n_{right}, where n_{left} has a self-label, $Nleft$, that is longer than the n_{right} self-label, $Nright$.

Base-9: InsertBetweenGreaterThan Algorithm

Comment: Insert new node between 2 consecutive nodes such that $length(Nleft) > length(Nright)$
Input: $Nleft$ – self-label of left node to new node, $Nright$ – self-label of the right node to new node
Output: $Nnew$ – current node self-label newly inserted such that $Nleft \prec Nnew \prec Nright$

```
1  Nnew ← null
2  p ← position of first different digits between Nleft and Nright
3  if (last symbol of Nleft < 9) then
4  |    Nnew ← Nleft with (last symbol + 1)
5  else          // last symbol in Nleft is '9'
6  |    Ntemp ← Nleft with all sequence of '9' removed from end
7  |    if (p < length(Nright)) then  // Nleft and Nright are not lexicographically immediate neighbours
8  |    |    Nnew ← Ntemp with (last symbol + 1)
9  |    else
10 |    |    diff ← last symbol(Nright) – last symbol(Ntemp)
11 |    |    if (diff == 1) then  // Ntemp and Nright same length and lexicographically immediate neighbours
12 |    |    |    Nnew ← Nleft ⊕ '1'
13 |    |    else              // deleted nodes between Nleft and Nright might exist
14 |    |    |    Nnew ← Ntemp with (last symbol + 1)
15 |    |    end
16 |    end
17 end
18 return Nnew
```

Figure 5.13 Insert between two nodes greater than algorithm in Base-9

This is different to the "**InsertBetweenLessThan**" algorithm, as here $Ntemp$ is created as $Nleft$ excluding the sequence of '9' digits from the end. Let us assume $diff$ is the difference between the last digit in $Ntemp$ and $Nright$, then $Nnew$ is allocated as follows (see Table 5.9):

Table 5.9 Insert between two nodes (greater than), find $Nnew$ if $Nleft$ ends with '9's

When $length(Nleft) > length(Nright)$, and $Nleft$ ends with '9's. Then $Ntemp$ is $Nleft$ after removing all '9's from the end and $diff = last\ digit\ of\ Nright - last\ digit\ of\ Ntemp$		
Condition	**Rule**	**Example**
If $diff = 1$; indicates that $Ntemp$ and $Nright$ are lexicographically immediate neighbours	$Nnew = Nleft \oplus$"1"	If $Nright =$ "65" and $Nleft =$ "6499" → $Ntemp$ = "64" → $diff$ = 5 - 4 = 1 then $Nnew =$ "64991"
If $diff > 1$	$Nnew$ is $Ntemp$ after incrementing its last digit by 1	If $Nright =$ "67" and $Nleft =$ "6499" → $Ntemp$ = "64" → $diff$ = 7 - 4 = 3 then $Nnew =$ "65"

In further consideration of the existence of a deleted node between n_{left} and n_{right}, the "**InsertBetweenGreaterThan"** algorithm presents the codes to re-use deleted labels when $Nleft$ is longer than $Nright$. These cases are treated by giving $Nnew$ the value of $Nleft$ after incrementing its last digit by 1. For example, let $Nleft =$ "496" and $Nright =$ "5", then $Nnew$ is given the value "497", which is lexicographically $Nleft = 496 \prec Nnew = 497 \prec Nright = 5$.

- ***Case $Nleft$ Same Length of $Nright$***

Once more, to analyse the possible cases where $Nleft$ could have the same length of $Nright$, the Base-9 initialisation process has to be examined. Accordingly, $Nleft$ and $Nright$ can both have the same size if, and only if, they are immediate sibling nodes where $Nleft \prec Nright$. Either $Nleft$ and $Nright$ differ by their last digit only, or there are some deleted nodes between n_{left} and n_{right}. The algorithm "**InsertBetweenSameLength"** illustrated in Figure 5.14 shows how the Base-9 labelling scheme handles inserting a new node n_{new} between two consecutive nodes with the same label size.

Base-9: InsertBetweenSameLength Algorithm

Comment: Insert new node between 2 consecutive nodes such that $length(Nleft)\, equals\, length(Nright)$
Input: $Nleft$ – self-label of left node to new node, $Nright$ – self-label of the right node to new node
Output: $Nnew$ – current node self-label newly inserted such that $Nleft \prec Nnew \prec Nright$

```
Nnew ← null
p ← position of first different digit between Nleft and Nright
if (p == length(Nleft)) then     //case Nleft and Nright differ by the last symbol
    diff ← last symbol (Nright) – last symbol (Nleft)
    if (diff == 1) then   // Nleft and Nright are lexicographically immediate neighbours
        Nnew ← Nleft ⊕ '1'
    else     //there possible deleted nodes between Nleft and Nright
        Nnew ← Nleft with (last symbol + 1)
    end
else          //case position of different digit between Nleft and Nright is not last symbol
    if (last symbol of Nleft < 9) then
        Nnew ← Nleft with (last symbol + 1)
    else     //if last symbol of Nleft == 9
        Ntemp ← Nleft with all sequence of '9' removed from end
        Nnew ← Ntemp with (last symbol + 1)
    end
end
return Nnew
```

Figure 5.14 Insert between two nodes same length algorithm in Base-9

Suppose that there are no deleted nodes between n_{left} and n_{right} with the self-labels $Nleft$ and $Nright$, respectively. In this case, $Nleft$ and $Nright$ differ by their last digit, whereby the difference between their last digit is 1. Therefore, to compute the $Nnew$ value such that $Nleft \prec Nnew \prec Nright$, $Nnew$ is assigned the $Nleft$ value after

concatenating it with the digit "1", which consequently increases the $maxLabelSize$ by 1. For instance, if $Nleft =$ "535" and $Nright =$ "536" then $Nnew =$ "5351".

However, if $Nleft$ and $Nright$ differ in their last digit, and the difference between their last digits is greater than 1, this indicates there is at least one available deleted label to be re-used. Thus, $Nnew$ is $Nleft$ after incrementing its last digit by 1. For example, if $Nleft =$ "535" and $Nright =$ "538" then $Nnew =$ "536"

The "**InsertBetweenSameLength"** algorithm also recognises the existence of a deleted node label by identifying whether the position (p) of the first different digit between $Nleft$ and $Nright$ is before the last digit. That is, $0 \leq p < length(Nleft)$, and accordingly $Nnew$ is assigned based on the last digit value of $Nleft$, as explained in Table 5.10:

Table 5.10 Insert between two nodes (same size L), find $Nnew$ if $p < L$

When $length(Nleft) = length(Nright)$, and p is position of the first different digit between $Nleft$ and $Nright$ is not the last digit		
Condition	**Rule**	**Example**
If the last digit of $Nleft$ is $d < 9$	$Nnew$ is $Nleft$ after incrementing its last digit by 1	$Nleft =$ "4256" and $Nright =$ "4395", then $Nnew =$ "4257".
If the last digit of $Nleft$ is $d = 9$	Allocate $Ntemp$ = $Nleft$ with all '9's digits removed from the end (i.e. $Ntemp$ ends with digit $D < 9$) $Nnew$ is $Ntemp$ after incrementing its last digit D by 1	if $Nleft =$ "2199" and $Nright =$ "4123" then $Ntemp =$ "21" and so $Nnew =$ "22".

The Base-9 labelling scheme supports skewed insertions between any two nodes by repeatedly calling the "**InsertBetweenNodes**" algorithm (see Figure 5.15). This algorithm invokes the appropriate method to generate a new node label based on a comparison between the length of $Nleft$ and $Nright$, as discussed previously.

Base-9: InsertBetweenNodes Algorithm

Input: $Nleft$ – self-label of left node to new node, $Nright$ – self-label of the right node to new node
Output: $Nnew$ – current node self-label newly inserted such that $Nleft \prec Nnew \prec Nright$

```
1 Nnew ← null
2 if (length(Nleft) < length(Nright)) then
3 |     Nnew ← InsertBetweenLessThan(Nleft, Nright)
4 else if (length(Nleft) > length(Nright)) then
5 |     Nnew ← InsertBetweenGreaterThan(Nleft, Nright)
6 else if (length(Nleft) == length(Nright)) then
7 |     Nnew ← InsertBetweenLessThan(Nleft, Nright)
8 end
9 return Nnew
```

Figure 5.15 Insert between two consecutive nodes in the Base-9 labelling scheme

For example, Figure 5.16 shows how the Base-9 labelling scheme handles the insertion of new nodes $n_a, n_b,$ and $n_c,$ in that order, between the elements "Library" and "Class" in the XML tree representing the "School" XML sample in Figure 5.1. Since both self-labels of "Library" and "Class" are of same size, the "**InsertBetweenSameLength"** algorithm is called, and consequently n_a is given a label value of "0.31". When n_b is inserted between "Library" and n_a, the "**InsertBetweenLessThan"** algorithm is triggered as $label(\text{"}Library\text{"})$ is shorter than $label(n_a)$, then n_b is labelled as "0.301". Alternatively, the "**InsertBetweenGreaterThan"** method is called when n_c is inserted between n_a and node "Class" due to their label sizes. Thus, n_c is assigned as "0.32".

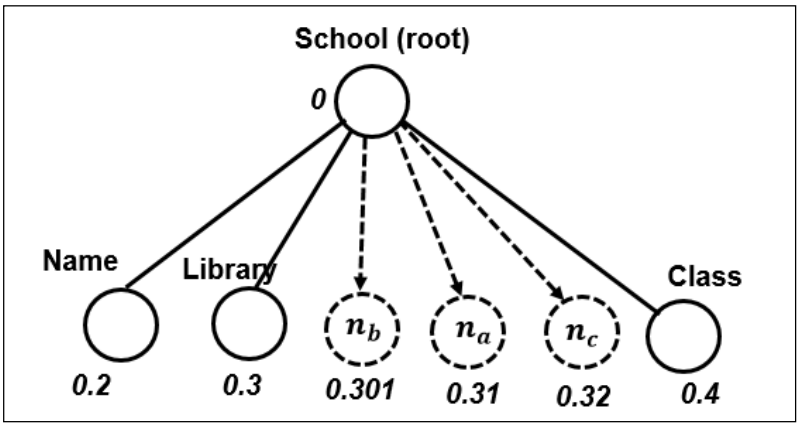


Figure 5.16 Example of handling insertions between two nodes in Base-9

Table 5.11 shows further examples of Base-9 labels generated when 10 nodes are inserted successively between two nodes considering $Nleft$ and $Nright$ to be of the same or different sizes.

Table 5.11 Examples of skewed insertions between two nodes in Base-9

	Node labels	Node labels	Node labels
$Nleft$	**64 (same length of $Nright$)**	**496 (greater size than $Nright$)**	**3 (less size than $Nright$)**
$Nnew$ **1**	641	497	301
$Nnew$ **2**	642	498	302
$Nnew$ **3**	643	499	303
$Nnew$ **4**	644	4991	304
$Nnew$ **5**	645	4992	305
$Nnew$ **6**	646	4993	306
$Nnew$ **7**	647	4994	307
$Nnew$ **8**	648	4995	308
$Nnew$ **9**	649	4996	309
$Nnew$ **10**	6491	4997	31
$Nright$	**65**	**5**	**311**

5.6.4 Re-using Deleted Node Labels

As mentioned previously, the Base-9 labelling scheme handles insertion by finding the nearest available lexicographical value to the label of the current node, which is the adjacent sibling to the new node. The insertion algorithms of Base-9 are designed to automatically re-use the deleted labels, if any, as explained in previous section.

When inserting after the right-most child node, the insertion mechanism of Base-9 allocates a new label considering the obtainability of the immediate next lexicographical value to the current right-most node's label (see Section 5.6.1). This consequently allows automatic re-use of any available deleted labels. For insertion before the left-most child node, the algorithm starts by finding the availability of deleted labels to be re-used for new nodes, see lines 2-22 in figure 5.9.

Similarly, when inserting between two consecutive sibling nodes in the Base-9 scheme (section 5.6.3), the insertion algorithms deliberate the re-use of deleted node's labels as follows:

- In case $Nleft$ is shorter than $Nright$: see lines 16-18 of "**InsertBetweenLessThan**" algorithm (figure 5.12).
- In case $Nleft$ is longer than $Nright$: see lines 13-15 of "**InsertBetweenGreaterThan**" algorithm (figure 5.13)

- In case $Nleft$ is the same length as $Nright$: see lines 7-9 of "**InsertBetweenSameLength**" algorithm (figure 5.14)

An experimental test has been carried out investigating the ability to re-use the deleted node labels in the Base-9 scheme as well as in the SCOOTER scheme. The implementation details of the test and the results obtained are provided later in Chapters 6 and 7. The following section explains how the Fibonacci coding is used to encode Base-9 labels to minimise the storage space cost.

5.7 Fibonacci coding

The generalised Fibonacci code of order $m \geq 2$ was introduced by (Apostolico and Fraenkel, 1987), but to date has never been used for XML labels. In this thesis, Fibonacci coding is applied for the first time to encode Base-9 XML labels. The Fibonacci encoding of orders $m = 2$ and $m = 3$ are illustrated in detail in Section 4.5.1. Both methods were applied separately to encode the Base-9 labels in order to study the effect of different order values, m, over the resulting codes.

The Base-9 scheme is a prefix-based labelling scheme in which a node's self-label is preceded by its parent's label where a delimiter "." is used to separate the label of the ancestor nodes at every level. Each component of the Base-9 prefix-label is encoded separately by Fibonacci coding and then concatenated, though with the separators omitted. Since there is no Fibonacci code for integer '0' in the Base-9 labelling scheme, the label "0" representing the root node is encoded as the bit '0'.

One of the criteria of Fibonacci codes is that there are no m consecutive 1-bits within the summation result of Fibonacci numbers of order $m \geq 2$, but each Fibonacci code ends up with exactly m consecutive 1-bits. Thus, the appearance of m-consecutive 1-bits in Fibonacci codes plays the role of a separator by indicating the ends of the binary code representing a component self-label in the Base-9 scheme, so avoiding the need to store the delimiters.

To demonstrate the encoding mechanism on Base-9 labels, let us consider encoding the label of the element "Outline" (in Figure 5.1), which is "0.4.2.3", as follows:

- **Using Fibonacci code of order $\boldsymbol{m = 2}$**: "0.4.2.3" is encoded as "0 1011 011 0011" (the spaces are added for clarification – see Table 4.3). The first bit '0' represents the root label. Then, starting from the second bit until two consecutive '1's signifies the next component of the Base-9 label (i.e., $F^{(2)}(4) =$ "1011") , and so on.

- **Using Fibonacci code of order $m = 3$**: "0.4.2.3" is represented as "0 10111 0111 00111" (the spaces are added for clarification – see Table 4.3). As in the Fibonacci encoding of order 2, the first bit '0' here also stands for the root label. The following bits, up to the appearance of three consecutive '1's, indicate the next component of the Base-9 label (i.e., $F^{(3)}(4) =$ "10111"), and so on.

The Fibonacci decoding mechanism is simply the reverse of the encoding process (Walder et al., 2012). The algorithm for decoding the Fibonacci binary code of order $m \geq 2$ (say $B9Code$) into a Base-9 label is given in Figure 5.17.

Base-9: Decode Base-9 Labels Algorithm

Input: $B9Code$ – Fibonacci binary code of Base-9 label, and order value m
Output: $label$ – Base-9 prefix-based label

```
1  label ← "0"                    //start with the root node label
2  B9Code ← substring (Code, 1)   //remove first '0' bit representing the root label
3  while (B9Code is not empty) do //there are more components in Base-9 label
4      position ← m − 1
       //allocate first m consecutive bits of B9Code into array bit[m] (of size m)
5      for (i = 0; i < m; i++) do bit[i] = B9Code.charAt(i) end for
6      while (NOT endOfFibCode(bit[m]) ) do  // call function endOfCode
           /* endOfFibCode function return true if for all 0 ≤ i < m → bit[i] == 1; otherwise return false */
           //This loop to find the end position of current component code within Fibonacci Code
7              position ← position + 1
8              for (i = 0; i < m − 1; i++) do bit[i] = bit[i + 1] end for   //shift m bits by 1
9              bit[m − 1] ← B9Code.charAt(position)
10     end
11     // Extract next component's code from the start of Fibonacci Code
12     component ← substring (B9Code, 0, position)
13     B9Code ← substring (CB9ode, position)
14     label ← label ⊕ "." ⊕ stringOf(Decode(component)) // Call function Decode
15 end
16 return label
```

Figure 5.17 The algorithm for decoding a Base-9 label

Knowing that the possible minimum $B9Code$ is "0", corresponding to the root node label, the algorithm works as shown in the following flowchart (Figure 5.18). To simplify the demonstration, these abbreviations are used:

- $B9Code$ is Fibonacci code input to be decoded into a Base-9 label (referred to as $label$).
- $FibCode$ is the current component's Fibonacci code.
- $position$ indicates the position of the last bit of $FibCode$.
- Array bit of size m to trace the end $(position)$ of a $FibCode$ within $B9Code$.
- The integer, x, matches the Fibonacci number representing the Base-9 self-label of the current component by calling the "**FibDecode**" method (Figure 5.19).

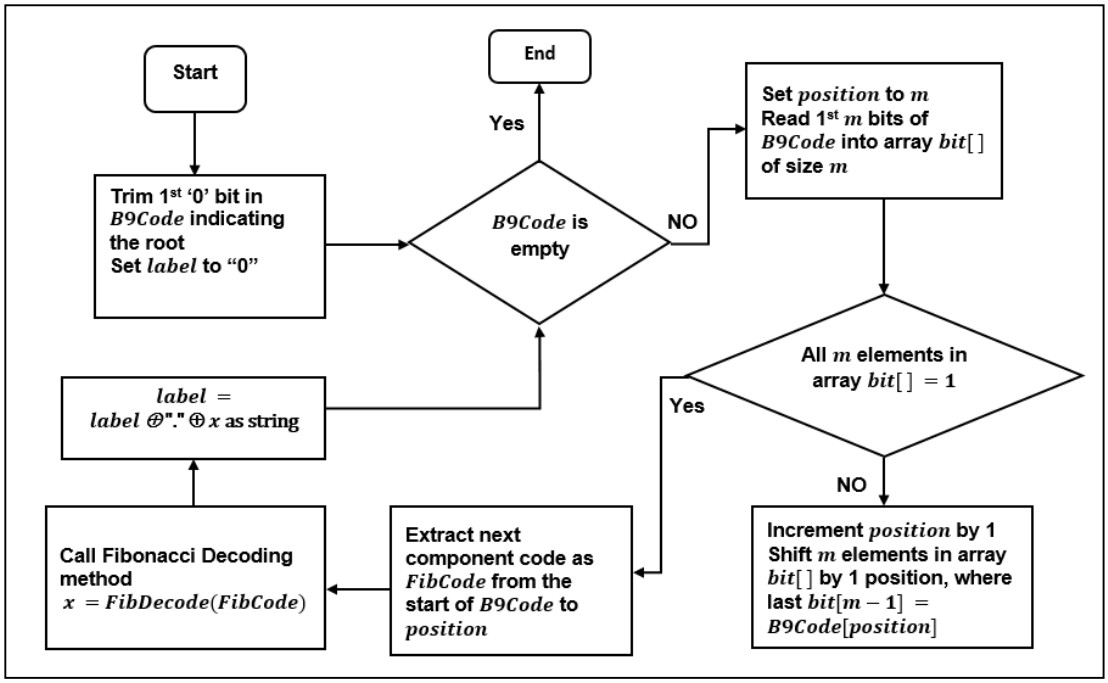


Figure 5.18 Flowchart to decode a Base-9 label

A Fibonacci code $B9Code$ representing the Base-9 label of a node, u, is basically a concatenated sequence of Fibonacci codes starting from the root code "0" up to the code of the node, u. A $B9Code$ must consist of at least one code indicating the root node, i.e., "0". Thus, the first '0' bit in the $B9Code$ is trimmed and the Base-9 $label$ is set to "0". After that, if the $B9Code$ is not empty, this implies there are more descendant codes to be decoded. Thus, the decoding process allocates the end of each component's code $FibCode$ by the appearance of m consecutive '1's. Then, the $FibCode$ value is truncated from the start of $B9Code$. Next, the summation of $F^{(m)}$ represented by $FibCode$ is found by calling the "**FibDecode**" method and assigned it to x. Afterwards, to obtain the full prefix-based Base-9 label of the node u, $label$ is appended by a delimiter "." followed by x as a decimal string. The whole process is repeated until the $B9Code$ is empty which means all components within the Base-9 label are attained.

The "**FibDecode**" method (see Figure 5.19) presents the generalised decoding algorithm for any Fibonacci code (here referred to as $FibCode$) of order $m \geq 2$ for any positive integer $x > 0$. In the Fibonacci encoding process, the bit '1' is appended to a Fibonacci code of order $m = 2$ to indicate the end of the code, whereas, a Fibonacci code of order $m > 2$ is concatenated with '0' followed by m consecutive '1's as its suffix; that is, "0111" for order $m = 3$. Therefore, the first step in the "**FibDecode**" method is the removal of the suffix of m consecutive '1's from the end of $FibCode$ if

the Fibonacci code is of order $m > 2$. After that, if $m > 2$, the last bit in $FibCode$ will be either '0' for any integer $x > 1$ or empty for $x = 1$, since $F^{(m)}(1)$ is exactly m-consecutive '1's, whereas for order $m = 2$, the last bit in $FibCode$ is always '1'. According to the remaining value of the $FibCode$, the "**FibDecode**" method processes by these steps to obtain the integer, x:

1. Set x to 0.
2. If $FibCode$ is of size 0, this indicates x is 1.
3. If $FibCode$ is of size 1, then x is 2.
4. If $FibCode$ is of size greater than 1, parse $FibCode$ bit by bit (except for the last bit) and if the i^{th} bit in $FibCode$ is '1' then keep incrementing x by the i^{th} Fibonacci number of order m; i.e., $F_i^{(m)}$ (see Section 4.5.1).
5. If $x > 2$, then consider the Fibonacci sum (Section 4.5.1) and add $S_{pos-1}^{(m)}$ to x, where pos is the position of the last '1' bit in $FibCode$.

Base-9: Fibonacci decoding ("FibDecode") Method

Input: $m, and\ FibCode$ – Fibonacci code of order m representing a component code of Base-9 label
Output: x – integer value corresponds to $FibCode$

```
1  if (m > 2) then      //remove suffix m-consecutive '1' bits from end of FibCode of order m>2
2      FibCode ← substring(FibCode, 0, length(FibCode) − m)
3  end
4  if (FibCode is empty) then x ← 1
5  else if (length(FibCode) is 1) then x ← 2
6  else if (length(FibCode) > 1) then
7      x ← 0
8      pos ← 0              //pos is the position of the last '1' bit in FibCode
9      for (i = 0; i < length(FibCode) − 1; i++) do
10         bit ← FibCode.charAt(i)
11         if bit then
12             x ← x + F_i^(m)   //if the i^th bit is '1' add the Fibonacci number F_i^(m) to x
13             pos ← i
14     end
15     if (m > 2) then x ← x + S_(pos−1)^(m) + 1 end
16 end
17 return x
```

Figure 5.19 The algorithm of the "$FibDecode$" method

The Fibonacci decoding process has the capability to facilitate query processing as examined in this thesis (see Chapters 6 and 7). As discussed in Chapter 2, the core of the query processing is the determination of the structural relationships between any

two nodes directly through their XML labels. The next section shows how the Base-9 labelling scheme handles the determination of the main structural relationships.

5.8 Relationship Determination

Being a prefix-based labelling scheme, Base-9 follows the same technique for identifying the structural relationships as described in Section 3.4. The containment of the path information within each Base-9 label facilitates query processing. The determination of the main structural relationships works as follows:

Level order: assuming the first level is the root level, which is counted as level 0, then the number of delimiters "." within the Base-9 label of a node corresponds to the level of that node. For example, the element "Outline" (in the XML tree in Figure 5.1) labelled as "0.4.2.3" is located in level 3 because the label contains a total of three delimiters.

Parent-child relationship: a node u is the parent of a node v if, and only if, $label(u)$ is the prefix of $label(v)$ and $level(v) = level(u) + 1$. For example, in Figure 5.1 the element "Outline" with label "0.4.2.3" is the father of the element "Topic" labelled "0.4.2.3.2".

Ancestor-descendant relationship: a node u is an ancestor of a node v and the node v is descendant of the node u if, and only if, $u \neq v$, $label(u)$ is the prefix of $label(v)$, and $level(v) > level(u) + 1$. For example, in Figure 5.1 the element "Class" labelled "0.4" is an ancestor of the elements "Outline" and "Topic", labelled "0.4.2.3" and "0.4.2.3.2", respectively.

Lowest common ancestor (LCA): the LCA node w is the shared ancestor of nodes u and v located farthest from the root. The LCA relationship between nodes u and v is only determined when neither node is an ancestor of another, and both $label(u)$ and $label(v)$ start with the same longest prefix, which is equal to $label(w)$. For instance, in Figure 5.1 the element "Class", labelled "0.4", is the LCA between the element "Outline", labelled "0.4.2.3" and the element "Books", labelled "0.4.3.3.4".

Sibling relationship: nodes u and v are siblings if both nodes share the same parent and are at the same level in an XML tree. Determining the pre-order sibling among nodes u and v is based on the lexicographical order of their self-labels. As an example, in Figure 5.1 the elements "Area" and "Outlines" are siblings because both are in the same level and share the same parent "Tutor". However, "Area" is a pre-

order sibling to "Outline" since $selflabel(\text{“}Area\text{”}) = \text{“}2\text{”}$ is lexicographically less than $selflabel(\text{“}Outline\text{”}) = \text{“}3\text{”}$.

Document Order: there are three scenarios where a node u appears before a node v in an XML tree: if the node u is either a parent, an ancestor or a pre-order sibling to the node v. If none of these relationships exist between u and v, then determining which node appears before the other depends on their labels, as follows:

1. First, remove any shared prefix from $label(u)$ and $label(v)$.
2. Then, compare the self-label of the first remaining components lexicographically. The smaller value indicates that the first node appears in document order.

For instance, let us consider the elements "Outline" (labelled "0.4.2.3") and "Books" (labelled "0.4.3.3.4"). After removing the longest shared prefix "0.4", the remaining labels of "Outline" and "Books" are "2.3" and "3.3.4", respectively. Now, the first component's self-label in "Outline" is "2" and in "Books" is "3". As "2" is lexicographically smaller than "3", "Outline" appears before "Books" in the XML tree.

5.9 Conclusion

This chapter has presented a newly proposed dynamic XML labelling scheme named the Base-9 scheme, referring to the set of 9 decimal digits $\{1, 2, \dots, 9\}$ used to initialise XML labels. The initialisation and the insertion mechanisms are designed based on lexicographical order, whereby digit '0' is used for generating new labels when the XML is updated. This chapter also covered how the insertions methods of the Base-9 scheme support re-using any available deleted node labels. Then, it was demonstrated how the Fibonacci coding can be applied for encoding and decoding the Base-9 labels. Finally, the determination of different structural relationships in the Base-9 scheme was described. The next chapter discusses the implementation of the proposed Base-9 labelling scheme as regards the various aspects presented in this chapter.

Chapter 6: Experimental Design and Implementation

6.1 Introduction

The previous chapter introduced a new XML labelling scheme based on the principles of the SCOOTER labelling scheme (discussed in Chapter 3) but with an emphasis on enhancing the compact and dynamic aspects. In order to evaluate the performance and reliability of the proposed scheme, several experiments were carried out over various XML datasets. Each experiment was run on both the proposed Base-9 scheme and the SCOOTER scheme for optimal comparison of their performances. This chapter describes the design details and objectives of these experiments as well as the datasets used for the evaluation process.

As mentioned in Chapters 1 and 4, this thesis also studies the possibility of compressing XML labels using the prefix-encoding methods presented in Chapter 4. Hence, further experiments were conducted to test these methods on some of the existing labelling schemes, as explained in this chapter.

The remainder of this chapter is structured as follows: the next section presents a general description of the strategy behaind the experimental design. Section 6.3 provides an overview of the current evaluation system of XML labelling schemes. Section 6.4 gives a review on the existing experimental XML datasets used for assessment purposes. In Section 6.5, the evaluation framework is set out, and then presented in detail as follows: the selection of datasets in Section 6.5.1, and the experimental design and objectives in Section 6.5.2. The implementation of the XML label compression study is explained in Section 6.6. The implementation platform setup is given in Section 6.7. Finally, this chapter is concluded with Section 6.8.

6.2 The Overall Experimental Design

The main aim to running the experiments explained in this chapter was to assess the research hypothesis stated in Chapters 1 and 5. These experiments were designed to test whether the Base-9 scheme introduced in Chapter-5 met the standards required to be classified as a fully dynamic XML labelling scheme (see Chapter-3). The setup of the experiments was constructed based on the details illustrated in Chapter-5 to

confirm that the proposed scheme's design and implementation met the objectives and requirements discussed in Chapter-1. Six main experiments were conducted to evaluate the Base-9 scheme's validity, functionality, scalability, efficiency, and performance:

- Label Initialisation
- Handling insertions
- Re-using deleted node's labels
- Label encoding
- Relationship determination
- Querying performance

For the performance comparison between other XML dynamic labelling schemes, the SCOOTER labelling scheme (O'Connor and Roantree, 2012) was chosen, as it is the foundation on which the Base-9 scheme is formed (see Chapter-5). Thus, the SCOOTER labelling scheme was also implemented from scratch following its design details, as presented in the work of (O'Connor and Roantree, 2012) and discussed in Section 3.4.3. All the six experiments were run using each scheme to test their characteristics and performance against each other. Section 6.8 presents the design details of these experiments, as based on their objectives.

To gain a fair assessment of the Base-9 scheme, it must be compared with other existing labelling schemes over the same test-bed. Therefore, it is important to first investigate the existing evaluation framework.

6.3 Evaluation of XML Labelling Schemes: An Overview

Obviously, from the XML labelling schemes discussion in Chapter-3 it can be seen that no two labelling methods share exactly the same characteristics. According to the fundamental properties required for a complete dynamic labelling scheme (see Section 3.2), each labelling scheme available in the literature is limited in one aspect or another. Unfortunately, there is no existing standard evaluation framework that can be used by XML labelling scheme designers to provide a comprehensive analysis of their labelling approaches' properties.

In (O'Connor and Roantree, 2010a), the authors outlined the general advantages and disadvantages of a number of XML labelling schemes. The aim of O'Connor and Roantree's work (O'Connor and Roantree, 2010a) was to generate a list of essential properties for XML update mechanisms. Based on these properties, those authors

presented a possible evaluation framework to compare a number of the existing XML labelling schemes. However, the feature classification of the schemes was based on the experimental results presented by the designers of these labelling schemes. As there is no standard evaluation method in the literature; researchers evaluated their own labelling schemes according to the main objectives underlying the design of their particular labelling method. Consequently, many researchers ignored the evaluation of all the desirable properties required for an effective labelling scheme (see Section 3.2). For example, LSDX's authors (Duong and Zhang, 2005) only illustrated the query performance in a theoretical sense, and did not evaluate it experimentally. Based on their theoretical concept, (O'Connor and Roantree, 2010a) indicated in their evaluation framework that LSDX fully supports XPath evaluation properties. However, later (Sans and Laurent, 2008) and (Khaing and Ni Lar, 2006) have shown that LSDX generates redundant labels leading to ambiguous XPath query expressions. Furthermore, the properties listed by (O'Connor and Roantree, 2010a) are very general and do not state the means by which each labelling scheme should be evaluated to unify the experimental mechanism and ensure a reliable comparison between the existing labelling schemes in terms of their required dynamic properties.

There is no existing standard framework for evaluating dynamic XML labelling schemes. Since all the available labelling schemes are built to enhance one or more aspects of other labelling schemes, the evaluation process is basically one of comparing the performance of these aspects only between the proposed scheme and other labelling schemes. For example, in (Assefa and Ergenc, 2012), the authors compared the label sizes and labelling time of their proposed OrderBased labelling method with the LSDX (Duong and Zhang, 2005) and Com-D (Duong and Zhang, 2008) schemes purely because all these labelling schemes are of an alphanumeric data type (see Section 3.4), whereas, (O'Connor and Roantree, 2012) compared the performance of their proposed SCOOTER labelling scheme with QED (Li and Ling, 2005b), Vector-based (Xu et al., 2007), and V-CDBS (Li et al., 2008) because they all have the ability to process frequent skewed insertions.

As mentioned in Chapter 3, a dynamic XML labelling scheme is considered effective if it is deterministic, efficient, compact and dynamic. The following sections illustrate the state-of-the-art of different evaluation approaches applied to dynamic XML labelling schemes.

6.3.1 Determining Relationships and Query Efficiency

Determining the common structural relationships between nodes is a core process in facilitating XML queries. Most of the XML labelling schemes have described how each structural relationship can be established from their labels from a theoretical perspective, but not all of them have had the performance of such a determination actually tested. ORDPATHs (O'Neil et al., 2004), DPESP (He, 2015), Persistent (Khaing and Ni Lar, 2006), Cohen's (Cohen et al., 2010), VLEI (Kobayashi et al., 2005), Dynamic XDAS (Ghaleb and Mohammed, 2015), LSDX (Duong and Zhang, 2005), QED (Li and Ling, 2005b), EBSL (O'Connor and Roantree, 2010b), and SCOOTER (O'Connor and Roantree, 2012) did not demonstrate any testing of query performance, whereas, the OrderBased labelling scheme (Assefa and Ergenc, 2012) examined only the determination of ancestor/descendant relationships between nodes. Other labelling schemes such as DDE (Xu et al., 2009), (Liu et al., 2013), and DPLS (Liu and Zhang, 2016) evaluated the XML query process by determining the relationships over thousands of randomly-selected label pairs.

The most common evaluation approach used to test the performance of XML querying is by studying the ordered and unordered queries over the Shakespeare's plays data set using XPath expressions, as proposed by (Tatarinov et al., 2002). Where (Wu et al., 2004) and (Li and Ling, 2005a) applied all nine queries from (Tatarinov et al., 2002). Whilst (Hye-Kyeong and SangKeun, 2010), (Yun and Chung, 2008), and (Li et al., 2008, Li et al., 2006a) processed only some of Shakespeare's queries. These queries mainly represent parent/child, ancestor/descendant, following and preceding siblings (Tatarinov et al., 2002).

In reality, retrieving and decoding XML labels usually affect the overall XML querying process. Therefore, it is important to consider the decoding time as part of the structural relationship determination between nodes. Unfortunately, up to now the decoding mechanism has been neglected completely when evaluating the relationship determination and querying the efficiency of XML labelling schemes.

Recently, most of the XML labelling schemes proposed have focused on supporting a dynamic XML environment and simultaneously controlling the label size to overcome the overflow problem (Chapter 4, Section 4.3). Therefore, their evaluation experiments were mainly designed to test the effects of XML updates (particularly insertions) over the generated labels in terms of the storage size, as discussed in the following section.

6.3.2 Compact and Dynamic Labels

The main disadvantage of the labelling schemes available in the literature is the growth of label sizes as the XML tree depth and/or the fan-out of the XML tree increases (see Chapter 3). Therefore, most research has considered supporting new node insertions whilst maintaining the labels sizes. The evaluation experiments in such labelling schemes focused more on measuring label size before and after insertions, such as in DDE (Xu et al., 2009), (Liu et al., 2013), and DPLS (Liu and Zhang, 2016), Fractional (Mirabi et al., 2012), V-CDBS (Li et al., 2008), SCOOTER (O'Connor and Roantree, 2012), and Modulo-based labelling (Al-Shaikh et al., 2010).

In terms of assessing the compactness property, almost all the existing XML labelling schemes have measured the initialisation/insertion time and the generated labels' sizes. Although some of these schemes have defined the encoding methods used for storing their labels in memory, they all neglected the computing of the actual label sizes occupied after encoding. Thus, the efficiency of the encoding methods used to store XML labels has never been evaluated.

In terms of XML updates, there is also no existing universal evaluation technique to measure the scalability of XML labelling schemes. Some labelling schemes have run uniform and skewed insertions, but mostly between two sibling nodes, such as in DDE (Xu et al., 2009), V-CDBS (Li et al., 2008), QED (Li and Ling, 2005b). Since the main drawback of current XML labelling schemes is the lack of support for skewed insertions, some researchers have tested the XML update performance only over skewed insertions, as in SCOOTER (O'Connor and Roantree, 2012), Fractional (Mirabi et al., 2012) and DPLS (Liu and Zhang, 2016). Other labelling schemes proposed their own evaluation systems (see Appendix A). For example, OrderBased (Assefa and Ergenc, 2012) computed the time required to insert sub-trees immediately under the root node. IBSL (Hye-Kyeong and SangKeun, 2010) investigated leaf node insertions only, whereas, many XML labelling schemes only illustrated the XML update in a theoretical sense, e.g., ORDPATHs (O'Neil et al., 2004), Com-D (Duong and Zhang, 2008), Persistent (Khaing and Ni Lar, 2006), Cohen's (Cohen et al., 2010), VLEI (Kobayashi et al., 2005), and EBSL (O'Connor and Roantree, 2010b).

Furthermore, the existing XML labelling schemes have been evaluated based on one or more of the available experimental XML datasets (illustrated in Section 6.4). The selection of the dataset in each case was made by the schemes' designers, and was

based on the aims of their research. Some labelling schemes specified their own characteristics over datasets via the generator tools provided by XML benchmarks. Others carried out their assessments on at least one of the experimental real-life datasets available on the XML Data Repository website (Suciu, 2002). All of the existing experimental XML datasets represent various features of XML trees such as file sizes, total number of available nodes, maximum depth, the degree of fan-out, and number of files per benchmark dataset (see Table 6.1 and Table 6.2). The following section presents an overview of the existing experimental XML datasets.

6.4 A Review of Current Experimental XML Datasets

In XML database management, XML datasets are tools for evaluating the performance of new XML systems (Schmidt et al., 2001a). There are, generally speaking, two types of experimental XML datasets: real-life XML datasets and XML benchmarks. A real-life dataset is a single validated XML document whose design is based on real public data. An XML benchmark is a tool used to generate synthetic datasets in XML format with different sizes as required by the experimental evaluation criteria. Each XML benchmark provides a range of query-sets simulating real-world scenarios to assess new XML systems. Therefore, the XML benchmark supports a comparison between the XML approach developed against the existing XML technology in terms of the query processing and storage techniques (Schmidt et al., 2001a). In order to assess the selection of appropriate datasets for the experimental implementation in this thesis, it is necessary to first provide an overview of existing synthetic XML benchmarks and real-life XML datasets.

6.4.1 Existing XML Benchmarks: An Overview

Existing XML benchmarks were designed with XML data storage and XML querying as their main considerations (Schmidt et al., 2001a). In general, XML benchmarks are categorised into application benchmarks (Schmidt et al., 2002) (Yao et al., 2004) and micro benchmarks (Runapongsa et al., 2006a). The application benchmarks were developed to evaluate the overall functioning of an XML database. By comparison, micro benchmarks focus on the assessment of the query-able aspects within an XML database. This section briefly describes the most widely used XML benchmarks, and Table 6.1 summarises the basic properties of these benchmarks.

- ***XMark Benchmark:***

The XMark benchmark was designed by (Schmidt et al., 2002) and it is commonly used for the assessment of XML applications by the XML development community (Davis et al., 2003) (Wang et al., 2003) (Arion et al., 2004) (Lawrence, 2004) (Chen et al., 2006) (Li et al., 2007) (Lee et al., 2010). This benchmark can generate various sizes of XML dataset along with the query-set that comprehends most of the XML query-able aspects using the generator tool XMLGen (Kochmer and Frandsen, 2002). Each XMark dataset is generated as a single file simulating the data of an online auction website in such a way that it makes the contents of the XMark data self-explanatory. The size of the XMark database generated is controlled via a scaling factor to allow developers the flexibility to regulate their datasets according to their needs. The XMark data generator is available via the XMark project website (Schmidt, 2003). Regardless of the size of an XMark database, it is always presented in an XML tree of depth twelve that has a repetitive structure with a reasonable number of recursions (Chen et al., 2005) (Zhang et al., 2005). In terms of scalability assessment, XMark datasets can effectively evaluate different performance aspects of an XML system. Although the XMark query-set does not consider update transactions, there are twenty queries carefully designed by (Schmidt et al., 2001a) to address search transactions (Schmidt, 2003).

- ***XBench Benchmark:***

XBench (Yao et al., 2004) is a template-based XML benchmark that generates a wide range of XML databases. Four types of XML database can be created by the “toXgen” tool (Yao et al., 2004) as follows: data-centric/single-document database (DC/SD), text-centric/single-document database (TC/SD), data-centric/multiple-document database (DC/MD), and text-centric/multiple-document database (TC/MD). This benchmark can provide a variety of XML databases of fixed sizes: small (10 MB), normal (100 MB), large (1 GB) and huge (10 GB) (Yao et al., 2004). The depth of the generated XML database is restricted by a parameter that takes only a limited range. Like XMark, this benchmark also provides twenty queries considering only search transactions.

- ***XOO7 Benchmark:***

The Object Oriented RDBMS benchmark OO7 (Carey et al., 1993) was adapted and enhanced into the XOO7 XML benchmark by Li et al (Li et al., 2001) to support the XML environment. The data and the query-set of the OO7 were also adjusted so as to be employed by the XOO7 XML benchmark, which is available on the XOO7

Benchmark website (Bressan et al., 2003). XOO7 creates XML data as a single XML file of small, medium, or large size. Regardless of the size of the XML database generated, the depth of the XML tree is always five; such restrictions on XOO7 dataset features (i.e., size and depth) limits the assessment of scalability. The query-set of XOO7 contains twenty-three queries supporting only search transactions.

- ***XMach-1 Benchmark:***

Unlike the benchmarks mentioned earlier, the XMach-1 benchmark (Böhme and Rahm, 2003) was developed as a multi-user XML database management system based on a Web-based application scenario. The structure of this benchmark contains four main parts: the XML document, servers, loaders and browser clients. The XMach-1 database is a collection of a large number (between 10^4 and 10^7) of small XML files whose maximum possible depth is six. Depending on the number of XML files, the overall size of the generated XMarch-1 database varies from 2 KB to 100 KB. Such small XML datasets plus the depth variation restriction, makes this benchmark unsuitable for assessing large-scale implementations and/or scalability testing. The XMach-1 benchmark provides eleven queries, eight of which cover search transactions whilst the remaining three focus on update transactions (Böhme and Rahm, 2000) (Böhme and Rahm, 2001). However, this query-set does not support the majority of essential XML transactions identified by (Schmidt et al., 2001a), such as path traversal, joins, and aggregation. This benchmark can be obtained from the website of XMach-1: A benchmark for XML Data Management (Böhme and Rahm, 2000).

- ***The Michigan Benchmark:***

The Michigan Benchmark (Runapongsa et al., 2006a) was developed as a micro benchmark to examine specific system properties (Yao et al., 2004). Hence, unlike the others, this benchmark helps designers focus on the parts of their systems that need enhancement (Al-Zadjali and North, 2016). The dataset generated by the Michigan benchmark is a single file with at least 728,000 total nodes, which can be multiplied up to 100 times. Regardless of the total number of nodes, the depth of this dataset is sixteen, whereas the breadth varies at each level from two to thirteen nodes based on the fan-out parameter set by the designer. In reality, the regularity of XML data is unpredictable and so the distribution of nodes' fan-out within each level avoids the consideration of a number of essential XML database features. The Michigan benchmark source can be accessed via its project website (Runapongsa, 2006b). The

source also includes the thirty-one queries on the Michigan's query-set, of which twenty-eight are search queries and three cover update queries.

- ***TPoX Benchmark:***

(Nicola et al., 2007b) introduced the Transaction Processing over XML (TPoX) benchmark as an application benchmark that aims to assess the entire XML system. In the TPoX benchmark, the XML generation tool, "ToXGene" (Nicola et al., 2007c), is used to create XML datasets. ToXGene employs templates to define the features of the XML database produced. The database is generated as a collection of small XML files with sizes ranging from 2 KB to 20 KB based on the schema used (Nicola et al., 2007b). Generally, the TPoX benchmark provides three XML schemas to control the size of the XML files by stating the depth and the breadth required of these files. In contrast to other benchmarks, the TPoX query-set consists of seventeen queries that focus mainly on update transactions rather than search transactions. The benchmark can be found on its project website (Nicola et al., 2007b).

Table 6.1 Features of the most common XML benchmarks

Benchmark Name	XMark	XBench	XOO7	XMach-1	Michigan	TPoX
Type	Application Level	Application Level	Application Level	Application Level	Micro	Application Level
Number of XML files in Dataset	1	Mixed: (1 or more)	1	Multiple ($10^4, 10^5, 10^6,$ or 10^7) files	1	Multiple: range from $3.6\ X\ 10^6$ to $3.6\ X\ 10^{11}$
Dataset Size	Varies from small (KB) to huge (GB)	Small (10MB) Medium (199MB) Large (1GB) Huge (10GB)	Small (500B) Medium (1000B) Large (1000B) but with more nodes	Varies from 2KB to 100KB per XML file	Min (default): 728,000 nodes Max: 100 times default	Varies from 2KB to 20KB per XML file
Schema of XML file	DTD of an internet auction database	DTD/XSD	DTD derived from OO7 relational schema	DTD of Data with Chapters, paragraphs and sections	DTD/XSD of the recursive element	XSD
Average/Max Depth	6/12	Limited by depth parameter	5/7	3/6	5/16	Controlled by the application's template
Number of search queries	20	20	23	8	28	7
Number of update queries	0	0	0	3	3	10

6.4.2 Existing XML Real-Life Datasets: An Overview

In contrast to synthetic datasets (see Section 6.4.1), real-life datasets contain real data and structures that facilitate the evaluation process. This section provides an overview of the most widely used real-life datasets for XML system assessments. The real-life datasets presented in this section can be obtained from the XML Data Repository website (Suciu, 2002); Table 6.2 outlines their main features.

- ***Protein Sequence Database:***

This database was developed by Georgetown University as a resource for integrated bioinformatics, comprising information on protein sequences. This dataset is a large-scale XML file of size 683 MB, which creates a broad and regular XML tree structure of shallow depth and expands up to seven levels (Wong et al., 2007). The protein sequence database has been employed by several applications to evaluate the performance of XML systems, such as for XML storage (Wong et al., 2007), processing XML streams (Green et al., 2003) (Jittrawong and Wong, 2007) (Wong et al., 2007), and filtering (Suciu, 2002) (Silvasti et al., 2009).

- ***DBLP Database:***

The Digital Bibliography Library Project (DBLP) database (DBLP, 2013) is a large-scale XML document comprising actual bibliographic data about computer science publications. The information stored includes major conferences papers (e.g., PODS, VLDB, ICDE), journals (e.g., CACM, TODS, TOIS), series (e.g., LNCS/LNAI, IFIP), and books pertaining to the topic of computer science (Suciu, 2002) (DBLP, 2013). Several XML database applications (Al-Badawi, 2010) (Liefke and Suciu, 2000) (Wang et al., 2003) (Lawrence, 2004) (Xu and Papakonstantinou, 2005) (Chen et al., 2006) (Li et al., 2007) have used the DBLP database to evaluate the development of their XML systems. Like the protein sequence dataset, the DBLP has a simple, shallow and broad XML tree structure (Lee et al., 2010) (Chen et al., 2006). The size of the DBLP database reached 1.1 GB in March 2013 (DBLP, 2013), whereas a smaller version of the same dataset, with a size of 127 MB, is available on (Suciu, 2002).

- ***NASA Database:***

As a part of the GSFC/NASA XML project, the NASA dataset (NASA, 2001) is designed from a flat file format and contains actual astronomical data. This XML dataset is 23 MB (Suciu, 2002) (NASA, 2001) with a shallow XML tree structure that presents only 18 recursive elements (Onizuka, 2003). It is widely used for evaluating XML applications in terms of XPath and XML query processing (Green et al., 2003) (Jittrawong and Wong, 2007) (Wong et al., 2007) (Onizuka, 2003) (Zhang et al., 2005), filtering (Silvasti et al., 2009), searches (Lee et al., 2010), indexing methods (He and Yang, 2004), and XML labelling (Xu et al., 2009) (Liu et al., 2013) (Liu and Zhang, 2016) (He, 2015).

- ***Treebank Database:***

The Treebank database was implemented by the Computer and Information Science Department at the University of Pennsylvania. It contains English sentences that are interpreted for linguistic structure and has a file size of 82MB (Suciu, 2002) (Treebank, 1999). In order to maintain the copyright of the text nodes, this dataset has been partially encrypted in such a way as to leave the XML structure unaffected. The deep recursive structure of the Treebank database makes it both an interesting and challenging case for XML experimental evaluation (Chen et al., 2006) (Onizuka, 2003) (Wong et al., 2007) (Chen et al., 2005). Moreover, this XML database is considered a complicated dataset because its XML tree covers a huge number of 386,614 nested structures (Onizuka, 2003). Consequently, the Treebank dataset is frequently used for assessment of XML applications (Liefke and Suciu, 2000) (Onizuka, 2003) (Green et al., 2003) (Chen et al., 2006) (Steedman et al., 2003) (Wong et al., 2007) (Chen et al., 2005) (Li et al., 2007).

- ***Sigmod Record Database:***

This dataset contains actual data about a number of articles published on the ACM SIGMOD website. Unlike the other real-life datasets, Sigmod record (Merialdo., 1999) is a comparatively small database with an XML file size of about 0.5 MB (Suciu, 2002). Therefore, it is preferentially used to evaluate XML system performance over small XML databases (Mirabi et al., 2012) (Lee et al., 2010) (Hye-Kyeong and SangKeun, 2010) (Li et al., 2007) (Rafiei et al., 2006) (Lawrence, 2004) (Li and Moon, 2001).

Table 6.2 Features of the most common XML real-life databases

Database Name	*Protein Sequence*	DBLP	NASA	Treebank	*Sigmod Record*
Dataset Size	683MB	127MB	23MB	82MB	467 KB
Number of Elements	21,305,818	3,332,130	476,646	2,437,666	11,526
Number of Attributes	1,290,647	404,276	56,317	1	3,737
Avg/Max Depth	5.15/7	2.90/6	5.5/8	7.8/36	5.14/6

6.5 The Guideline for Experimental Assessment

As discussed in Section 6.3, there is no standard framework to evaluate the functionality of an XML labelling scheme. This has led to a particular challenge in verifying the proposed scheme's reliability as a fully dynamic labelling scheme. Therefore, it became essential to setup the evaluation criteria whilst designing the experiments so that the research objectives of this thesis (stated in Chapters 1 and 5) can be achieved. The criteria below were identified with the intention of providing a comprehensive assessment framework that covers the main aspects of a dynamic XML labelling scheme. Hence, the need for the following experimental evaluation standards are discussed next.

6.5.1 The Selection of Experimental Datasets and Queries

Due to the lack of a standard evaluation framework (see Section 6.3), the choice of experimental XML datasets from those presented in Section 6.4 was made based on the objectives of these experiments (stated in Section 6.5.2).

To ensure the scalability of the evaluation, it is necessary to take into consideration the fact that the shape of the XML tree representing an XML document may be reflected in the results. Therefore, the selection of the experimental databases is based on the diversity of their XML tree features (i.e., size, depth, and breadth). Table 6.3 reports the properties of the experimental datasets used in this thesis.

Table 6.3 The properties of the experimental datasets selected

XML dataset	File size	Max depth	Max breadth	Total elements
NASA	23MB	8	80,396	47,664,6
Treebank	82 MB	36	144493	2,437,666
DBLP	127MB	6	328858	3,332,130
XMark	111MB	12	25500	1,666,315

Due to the simple and realistic data contained within the real-life XML datasets, these were selected in preference to synthetic datasets formed by XML benchmarks (see Section 6.4.1). Among the real-life datasets reviewed in Section 6.4.2, DBLP, Treebank and NASA were used for all the experiments in this thesis. These datasets provide a collection of different XML tree structure specifications (see Table 6.3). The DBLP dataset has a very large XML file size with a shallow and wide XML tree structure. In contrast, the Treebank dataset was selected due to its complex recursive structure that is represented as an XML tree with high depth and low breadth. The NASA dataset provides an XML tree of relatively average depth and average width. Since size variation is an essential criterion for the evaluation's scalability, the varying size of these three genuine datasets makes the evaluation system more reliable.

XMark is the most common benchmark used for XML data management (Franceschet, 2005) and XML labelling scheme evaluation (Lu et al., 2005b) (Xu et al., 2009) (Liu et al., 2013) (Liu and Zhang, 2016) (Mirabi et al., 2012). Thus, XMark was selected for the experiment's implementation in this thesis. XMark was developed whilst taking into consideration the standardisation issues around XML, particularly in terms of storing and querying (Schmidt et al., 2002). It provides a framework for evaluating XML databases via different query types (Runapongsa et al., 2006a).

Due to its scalability, the XMark benchmark provides a comprehensive set of queries; each was designed to highlight intuitive semantics (Schmidt et al., 2002). XPathMark queries (Franceschet, 2005) have been designed for the XMark Benchmark to include the main aspects of the XPath 1.0 language (see Section 2.8.1). These queries are widely used in XML research such as that of (Arroyuelo et al., 2015), (Benedikt and Cheney, 2009), (Genevès and Layaïda, 2006), and (Böttcher and Steinmetz, 2007). Four XPathMark queries were selected to evaluate the query performance in this thesis. Table 6.4 reports the details of these queries and their descriptions. These queries were chosen

because the axis containment in each query represents the essential structural relationships: - of Q1-parent/child, Q2-anscestor/descendent, Q3-following sibling, and Q4 following/preceding (document order). Other axes can be handled in a similar way to these axes (Min et al., 2009), so their evaluation was omitted.

Table 6.4 The experimental queries set (adopted from (Franceschet, 2005a))

Query number	Axis Type	XPath and Description
Q1	Parent/child	Return the American items: */site/regions/*/item [parent:: namerica or parent:: samerica]*
Q2	Ancestor/descendent	Find the mails containing a keyword: *//keyword/ancestor:: mail*
Q3	Following/preceding siblings	Allocate the preceding bidder of each open auction: */site/open_auctions/open_auction/bidder [preceding-sibling:: bidder]*
Q4	Following/preceding elements (document order)	Return items of the document (per region) except the last one: */site/regions/*/item[following::item; item(level+2) of region]*

In order to generate the XML labels that represent the XML tree structure of the XML files used, the SAX parser was applied (see Section 2.5.2). This is due to the better performance of the SAX parser over the DOM parser in terms of manipulating large-scale XML documents. The DOM parser consumes memory space and so restricts the XML file size used in such implementations (see Section 2.5.1). Since the datasets selected for the experimental evaluation are mostly large, as described in Table 6.3 above, the SAX parser was selected in preference to the DOM parser.

6.5.2 Experimental Objectives

As mentioned earlier, the aim of the experimental implementation is to evaluate the proposed Base-9 scheme as a fully dynamic XML labelling scheme. Hence, the experiments were designed to test the functionality of the main features of a good dynamic labelling scheme (see Section 3.2) over the proposed scheme. Accordingly, each experiment was implemented with the intention of testing some of these features on the Base-9 scheme. The experiments, along with their objectives, are described individually below:

6.5.2.1 Label Initialisation

This experiment was designed to test the initial labelling process focusing on two main aspects: the time required to generate the initial labels of an XML document and the growth in the size of the labels. The experiment was conducted over several XML datasets (see Section 6.5.1) to study how the document features (i.e. size, and its XML tree depth and breadth) affect the time and/or the size. This experiment was applied separately for both the SCOOTER and the Base-9 labelling schemes. The results will be analysed in Chapter 7. Since the initialisation mechanisms in both schemes are very similar, it should be expected that the difference in the labelling time will be insignificant. As generating sufficiently compact labels to fit in the main memory is one of the major properties of a good labelling scheme, the growth rate of the labels size in each scheme is of particular interest as a comparison factor. Since the proposed scheme uses more digits than that of SCOOTER (see Section 5.5), it should be expected that Base-9 will produce shorter labels and so consume less storage capacity than SCOOTER.

6.5.2.2 Handling Insertions

This experiment was designed to test the extent to which the proposed scheme can support XML updates, i.e., it focuses on the dynamic characteristic. This is achieved by measuring the scheme's scalability, particularly with regards to handling insertions. Two types of insertions were applied: uniform insertions and skewed insertions. Uniform insertion is basically the random insertion of new sibling nodes within an XML tree (Liu et al., 2013) (Xu et al., 2009) (Xu et al., 2007), in this instance with up to 50,000 nodes, whereas skewed insertions refers to the repeated insertion of nodes at a fixed place within an XML tree (Liu et al., 2013). The latter scenario was performed by inserting a large number of nodes (up to 10,000 nodes) at a randomly selected position, with this process being repeated 10 times. According to the random position selected, the insertion algorithm (section 5.6) is applied as follows:

- If the selected node is the first child; then the "**insertBeforeLeftMost**" method is invoked.
- If the selected node is the last child; then the "**insertAfterRightMost**" method is employed.

- If the selected node is a middle child; then the "**insertBetweenNodes**" method is called.

For any type of insertions (uniform/skewed), two main factors were considered: the growth in label size after insertion and the overall insertion time. This study also considered how the number of nodes inserted could further affect these two factors. While the proposed scheme handles insertions based on lexicographical comparison rather than enforcing the adaptive growth mechanism of SCOOTER, the expected result was that Base-9 would perform better in terms of time and size.

6.5.2.3 Re-using Deleted Nodes' Labels

The objective of this experiment is to test the ability to re-use deleted labels, if any, in order to further limit the growth of the label's size. For this experiment, the DBLP dataset was used because it provides a wider range of sibling nodes. The test was also carried out on 1,500 self-labels that were generated to represent 1,500 sibling nodes. The experiment was performed by first selecting a sample label set of adjacent siblings' nodes from the Base-9 labels and its corresponding label set from the SCOOTER labels. From each label set, a group of n sibling nodes were deleted. The same n number of nodes were then inserted in the same positions as the n deleted nodes. Three types of updates were tested: delete and insert n siblings before the first child, delete and insert n siblings after the last child, and delete and insert n siblings between two nodes. For each type of update, the original n deleted labels and the new n inserted labels were recorded separately in two lists (e.g., by using two files or array-lists). Then the percentage of the existing of the same labels in both lists was computed. Since the proposed scheme allocates a new label that is lexicographically closest to the label of the current node (see Section 5.6), it can be expected that the Base-9 scheme will re-use all the deleted labels.

6.5.2.4 Label Encoding

The aim of this experiment is to measure the storage capacity required to store the labels generated before and after XML updates. The Base-9 labels were encoded using Fibonacci coding (see Section 5.7), whereas the SCOOTER labels were encoded by the QED encoding method (see Section 3.4). The effect of the coding method applied by each scheme on storage size was studied for

the initial labels. The time required to encode the initial labels was also recorded. To examine whether the growth rate of the labels has any influence on the encoding process, this experiment measured the encoded label size and time after extensive skewed insertion (Section 6.5.2.2). As mentioned in Section 4.5, the Fibonacci coding has proven to be a good choice for data compression, whilst the QED codes may increase in size rapidly at about 2-bits per insertion (see Section 3.4). Therefore, the expected result was that the Fibonacci code used for the Base-9 labels would occupy less storage space than SCOOTER's encoded labels. On the other hand, due to the simplicity of the QED encoding mechanism, it might be expected that the SCOOTER labels will be encoded faster than the Base-9 labels.

6.5.2.5 Relationships Determination

As mentioned in Chapters 2 and 3, the main purpose of any XML labelling scheme is to support query processing by determining the structural relationship between any two nodes. This experiment was designed to measure how fast the main structural relationships discussed in Chapter 2 can be established directly from the labels before and after XML updates.

Out of the initial labels, 200,000 pairs of labels were chosen at random from the first 15,000 labels in an XML dataset and the execution time for computing the relationships between each pair was recorded. To test the determination after insertions, the first 5,000 labels of an XML document were selected and then 10,000 nodes were inserted randomly over the selected set of labels. Using this set, the execution time for computing the relationships between each of the 400,000 pairs chosen was again calculated. Researchers such as (Xu et al., 2009) and (Liu et al., 2013) have performed similar studies over many datasets, and have noted that the results were consistent for all XML datasets used. Thus, this part of the experiment was performed using just the Treebank XML dataset since it has the deepest recursive structure with a maximum depth of 36 (i.e., more ancestor/decedent nodes) and an average fan-out of 1623 (siblings), providing a sufficient variety of structural relationships. The experiment was run to test the determination of each relationship separately. Furthermore, the time taken to determine the relationships between any two nodes were computed for all the experimental datasets (see Section 6.5.1). The relationships tested were those of parent/child, ancestor/descendent, sibling, LCA, and document order.

This experiment also investigates whether the decoding process affects the query processing both after and before insertions. Therefore, the same experiment as above was conducted on all the experimental datasets (see Section 6.5.1), but by randomly selecting 200,000 of the encoded labels (instead of labels) from the first 15,000 encoded labels of an XML dataset. The measurement of the decoding time was included as a part of the determination process. All these experiments were run on the Base-9 and the SCOOTER schemes. The results are presented in Chapter 7.

6.5.2.6 Query Performance

The objective of this experiment is to evaluate the performance of the main types of XPath query on an XML-labelled dataset both before and after insertions. These queries along with their purposes, were described in detail in Section 6.5.1. To process these queries in both the proposed scheme and the SCOOTER scheme, the structural joins algorithm (Al-Khalifa et al., 2002) was applied. Structural joins provides a special stack-tree algorithm for evaluating XPath axes that work more efficiently in practice (Gottlob et al., 2005), and leads to optimal join performance (Chien et al., 2002). Thus, efficient support for structural joins is the key to the efficient implementation of XML queries (Chien et al., 2002). Other XML labelling schemes also used structural joins for XML querying such as:- those of (Min et al., 2009), (Lu et al., 2005b), (Lu and Ling, 2004) and (Mirabi et al., 2012). The expected result was that the difference between the queries' response time in both schemes would be insignificant.

The results obtained from the experiments are discussed and analysed statistically in Chapter 7 in order to evaluate the Base-9 scheme's performance. Based on the analysis of these results, Chapter 8 presents further discussion on the reliability of the specified criteria above as a standard evaluation guideline.

Besides the experiments described in this section to evaluate the proposed scheme, further tests have been carried out to study the ability of prefix-encoding techniques to compress XML labels. The experimental design and objectives of the study are detailed in the following section.

6.6 XML Label Compression Using Prefix Encoding

The aim of this experiment is to study the possibility of compressing XML labels via prefix encoding as presented in Section 4.5, in order to reduce the storage

space and minimise the chances of overflow (see Section 4.3). As mentioned in Chapter 4, several encoding methods have been applied by the existing XML labelling schemes to store XML labels, but up to now, prefix encoding has not been amongst them. The methodology of six of the most common prefix-encoding methods were presented in Section 4.5. In this experiment, the performances of these prefix-encoding techniques were tested in terms of compressing and storing XML labels:

- Fibonacci coding of order 2.
- Fibonacci coding of order 3.
- Lucas coding.
- Elias-delta coding.
- Elias-Fibonacci coding of order 2.
- Elias-Fibonacci coding of order 3.

For this experiment, the three real-life datasets described in Section 6.5.1 were used. To setup the XML labels model for testing proposes, two XML labelling schemes were used: the Dewey order (Tatarinov et al., 2002) and the SCOOTER scheme (O'Connor and Roantree, 2012). The selection of these two schemes was based on the popularity of the Dewey scheme and the efficiency of the SCOOTER scheme, as discussed in Chapter 3. For each dataset, the Dewey and the SCOOTER labels were generated and then compressed and decompressed separately by each of the six prefix-encoding methods. The prefix-encoding methods were applied to encode the Dewey/SCOOTER XML prefix-based labels in a similar manner to the mechanism used to encode the Base-9 labels by Fibonacci codes (see Section 5.7), where the labels' components were coded as long integers. To study the difference on the prefix encodings' performance over these labels, the original encoding methods proposed by the designers of the Dewey and SCOOTER schemes were also implemented for a better comparison. That is, UTF-8 for Dewey labels, and QED for the SCOOTER labels (see Chapter 4).

In general, this experiment consisted of two main experiments to test the performance of encoding and decoding for each prefix-encoding method. In terms of encoding, the execution time and the code size were the two main factors considered in the comparison of results. For the decoding process, the

test focused on measuring the run time performance in order to assess the fastest decoding method. To study the effect of the XML dataset size on the compression process, Treebank and DBLP file sizes were reduced to 23 MB (equal to the NASA file size) but their XML tree features were preserved, as described in Table 6.3. The encoding and decoding experiments were then repeated over these re-sized datasets and the results were compared with the original ones.

The results of this experiment were published at the WEBIST 2016 conference (Al-Zadjali and North, 2016), and are also presented in more detail in Chapter 7.

6.7 The Experimental Platform Setup

All experiments were performed on a laptop with a 2.40 GHz Intel Core™ i7-4500 CPU, 8.0 GB main memory and a Windows 10 64-bit operating system with a x64-based processor. Both the proposed Base-9 scheme and the SCOOTER scheme were implemented using Eclipse Java EE IDE version Luna 4.4 and Java language JDK 1.7.

6.8 Conclusion

This chapter has illustrated the specifications and guidelines for the experimental evaluation of the proposed scheme. It outlined the lack of a standard evaluation framework for XML labelling schemes. Accordingly, the experimental settings were designed to meet the objectives of the research hypothesis. Six sets of experiments were applied to evaluate the scalability and the functionality of the proposed scheme. Additionally, this chapter discussed the experimental details of XML label compression using a prefix encoding study. The implementation platform setup was stated and the selection of the datasets and queries were determined based on the experimental XML datasets available. The results obtained from these experiments are analysed in the next chapter.

Chapter 7: Experimental Results and Statistical Analysis

7.1 Introduction

Chapter 6 described the six experiments that were used to evaluate the proposed Base-9 labelling scheme. They were designed to examine various aspects of the scheme's functionality, scalabitily and performance. The first experiment evaluated the label initialisation process in terms of time and size. The second and third experiments assessed the scheme's ability to handle XML updates. The fourth experiment focused on the compactness of the labels using the Fibonacci coding method. The remaining two experiments were designed to evaluate the scheme's efficiency in terms of determining relationships and querying performance. In addition, a further experiment was described in Chapter 6 to assess the performance of several prefix-encoding methods in terms of label compression.

This chapter presents an analysis of the results obtained from these experiments. The next section provides an overview as to how appropriate, statistically significant results were obtained. Then, the statistical interpretations of the results of each experiment are discussed individually. Finally, the chapter concludes with Section 7.5.

7.2 Statistical Significance Analysis: An Overview

The concept of statistical significance was introduced by Ronald Fisher (Fisher, 1925). Statistically significant results are observed results which are unlikely to have occurred purely by coincidence (Ali et al., 2010). There is always a finite probability that the results obtained by the statistical tests could have occurred by chance (Motulsky and Searle, 2003). This probability is referred to as the p –value, and can be calculated as a minimum threshold of statistical significance. If the p –value obtained is less than the significance level (e.g., $p < 0.05$), then it can be concluded that the results reflect the characteristics of the method(s) applied rather than sampling error (Sirkin, 2005).

The term “null hypothesis” is usually used in relation to the p −value. The null hypothesis basically states that there is no difference between the methods applied (Motulsky and Searle, 2003). The concept of statistical significance is the minimum level of p −value at which the null hypothesis can be rejected.

Statistically significant results are required in various areas of computer science research, especially in the area of software verification and validation in which randomised algorithms are widely used (Arcuri and Briand, 2014). A randomised algorithm (Motwani and Raghavan, 1996) can be strongly affected by chance since it will have at least one component based on randomness. For example, when applying uniform insertions in an XML labelling scheme, the positions of new nodes inserted are selected at random. Another example is when evaluating the performance of algorithms based on execution time, which is itself affected by many factors such as hardware configuration, loop transformation and the number of threads (Li et al., 2005c). As randomness might affect the evaluation of the efficiency of randomised algorithms, many researchers such as (Arcuri and Briand, 2014) (Dybå et al., 2006) and (Grissom and Kim, 2005) developed techniques to ensure reliability when analysing the performance of randomised algorithms.

When an algorithm is developed to address a computer science problem, it is common to compare it to existing alternative techniques, but the comparative criteria must be first decided, such as label sizes and execution time. Based on the research objectives, different measures (M) can be chosen when attempting to determine the efficiency or the cost of the algorithms. To enable statistical analysis by M, the algorithms compared should be run independently a large number of times to gather data on the probability distribution of M for each algorithm. The data being tested affects the probability distribution of M, which in turn affects the statistical test used for the evaluation.

A statistical test can help to assess if there is enough experimental evidence to assert the null hypothesis, i.e., if there is a difference between the algorithms compared. Thus, the statistical test is intended to verify the acceptance or the rejection of the null hypothesis. The selection of the appropriate statistical test depends on two principle factors: the normality of the data distribution and the number of algorithms being compared (i.e., two or more). In this thesis, the normality of the execution time was tested using the graphical estimation of normality by the Kolmogorov-Smirnov (K-S) test (Oztuna D, 2006) (Ghasemi and

Zahediasl, 2012). For all the experiments, the $K-S$ test results have shown that the assumptions of residuals were not normally distributed, and so the non-parametric statistical tests were carried out. Thus, the selection of non-parametric statistical tests that are relevantly valid for each experiment in this thesis was based on the number of algorithms to be evaluated.

- ***Non-parametric Statistical Tests for Pairwise Comparison:***

The two most common non-parametric statistical tests applicable for evaluation comparison between two algorithms are (LaMorte, 2016) the Wilcoxon rank sum test (Wilcoxon, 1945) and Mann-Whitney U-test (Nachar, 2008). For pairwise comparison these two tests allow the derivation of p-values when experimenting with randomised algorithms. A low p-value (e.g., p < 0.05) indicates the rejection of the null hypothesis, which implies with a high level of confidence, that there is a difference between the two algorithms. If so, then further factors have to be considered to assess which algorithm performs better, such as the effect size measure (Grissom and Kim, 2005).

A non-parametric standardised effect size measure is the Vargha and Delaney's $\hat{A}_{12}$ statistic (Vargha and Delaney, 2000). $\hat{A}_{12}$ determines the probability that a running algorithm (say A) yields higher performance measures than running another algorithm (say B). In comparison to other standardised effect size measures, that of Vargha and Delaney is easier to interpret (Grissom and Kim, 2005); such that:

- $\hat{A}_{12}\ (A,B) = 0.5$ indicates that the two algorithms are equivalent.
- $\hat{A}_{12}\ (A,B) = x\ < 0.5$ entails that $x\%$ of the time, algorithm A performs better than B, and vice versa for $x\ > 0.5$.

As reported by (Vargha and Delaney, 2000) the formula derived by Vargha and Delaney is applied in this research as follows:

$$\hat{A}_{12}\ (A,B) = \left(\frac{R_1}{m} - \frac{(m+1)}{2}\right)/n$$

R_1 is the rank sum of the algorithm A under comparison. The rank sum (Gibbons and Chakraborti, 2011) is an essential component in the non-parametric statistical tests, such as in the Wilcoxon rank sum test (Wilcoxon, 1945) and Mann-Whitney U-test (Nachar, 2008), where m and n are the number of observations in algorithms A and B, respectively. To achieve more accurate

statistical results, the two randomised algorithms should be executed the same number of times; i.e., $m = n$.

When the effectiveness of randomised algorithms is addressed, the choice of artefacts (e.g., number of nodes inserted and the position of the insertion) is important as it usually affects the assessment of results (Arcuri and Briand, 2014) (McPherson et al., 2004). Analysing randomised algorithms over empirical data raises the challenge of ensuring the credibility of the results. This consequently questions the validity of the proposed algorithm, so making it difficult to generalise the results to other untested systems or data. To achieve realism, a large number of artefacts should be chosen to improve the validity of the evaluation. For more reliable statistical results, there should be a balance between the number of artefacts applied and how many times each artefact is run (Arcuri and Briand, 2011). These numbers should be large enough to maintain a statistically significant difference for each artefact when comparing two randomised algorithms. (Arcuri and Briand, 2014) recommended that more artefacts should be used since it is more important to address the target of the problem and to execute fewer runs, even as low as 10 runs. For each artefact, the non-parametric pairwise statistical test has to be performed and the overall result indicates the effectiveness of the algorithms compared.

- ***Non-parametric Statistical Tests for Multiple Comparisons:***

It is possible to deal with the comparison of multiple techniques by using a non-parametric statistical test called the Kruskal–Wallis test (Vargha and Delaney, 1998), which is equivalent to the well-known parametric test ANOVA (Cuevas et al., 2004). The Kruskal-Wallis test compares the mean rank for each technique in relation to the comparison parameters (e.g., the number of runs for each encoding method). In multiple comparisons, obtaining a low p-value means there is very strong evidence to suggest a difference between at least one pair of the techniques applied. In order to find out which technique gives the best performance, the Manny-Whitney tests (Nachar, 2008) can be carried out on every pairwise comparison between the individual algorithms. The results of the "pairwise comparisons" show that there is very strong evidence of a difference between the two methods if, and only if, $p < 0.001$ (p-value adjusted using the Bonferroni correction (Armstrong, 2014)). As non-parametric tests are more sensitive to medians than to means (Howell, 2012), comparison of the median values gives a further indication of the methods' effectiveness.

The following sections provide a results analysis and discussion of the experiments described in Chapter 6. For each experiment, an appropriate non-parametric statistical test was selected as described above. In addition, the box plot (McGill et al., 1978) charts were drawn to represent the statistical significance between the methods graphically.

7.3 Experimental Results for the Base-9 Scheme

As mentioned previously, to evaluate a new scheme, it is necessary to compare it against an existing scheme based on the chosen comparison criteria (Arcuri and Briand, 2014). To compare the proposed Base-9 labelling scheme to the SCOOTER labelling scheme, several experiments were performed in an attempt to capture either the effectiveness or the cost of each scheme.

Many aspects have been considered in this comparison. These are: initialisation, insertion, label re-usability, label encoding, structural relationship determination, and query performance. To facilitate statistical analysis for each of these aspects, both the Base-9 and SCOOTER algorithms were executed independently a large enough number of times to assemble information on the probability distribution of the required measurements, M, (i.e., code size and/or execution time). A statistical test was then performed with the aim of deciding whether the null hypothesis should be rejected or not. In this section, the null hypothesis was defined as there being no difference between the Base-9 scheme and the SCOOTER scheme. For the time comparison, the first five runs were excluded before counting the n execution times to avoid cache memory effects and to verify the accuracy and reliability of the results. As the normality of the probability distribution of the randomised algorithms' results had been found to be negative using the K-S test (Oztuna D, 2006), a non-parametric statistical test was selected for each experiment, as discussed below.

The next section describes how the evaluation of the initialisation process was handled.

7.3.1 Label Initialisation

As explained in Chapter 5, the initialisation algorithm for the Base-9 labelling scheme was adopted from SCOOTER. However, these two labelling schemes differ in the number of digits available to generate labels, i.e., 9-digits in Base-9

including 0, and only 3-digits in SCOOTER excluding 0. To investigate if this adjustment has affected the initialisation time and the initial label sizes, the label initialisation experiment was conducted as discussed in Chapter 6. The significance of the results is discussed in the next section.

7.3.1.1 Analytical Strategy

Two main factors were considered when evaluating this experiment: the initialisation time (in milliseconds) and label size (in Kbytes). As mentioned in Section 7.2, the computation of the execution time falls into the category of randomness. Therefore, it is important to identify the number of runs needed to obtain a statistically significant difference between the initialisation algorithms of both schemes. According to (Ali et al., 2010) and (Wegener et al., 2001), in order to enable the analysis of a statistical hypothesis with minimal statistical power (Dybå et al., 2006), the number of runs had to be at least 10. However, (Arcuri and Briand, 2014) recommended using more runs (at least 30 runs) to improve the accuracy of the statistical test. Hence, in every XML experimental dataset, each initialisation algorithm was executed 100 times to achieve a more reliable data analysis. As the comparison is based on two schemes, the non-parametric Mann-Whitney U-test was selected to obtain the p-value. For further statistical analysis, the effect size using the Vargha and Delaney $\hat{A}_{12}$ measurement was also computed.

In terms of evaluating the growth in label sizes, for each experimental dataset the total label size was computed for both labelling schemes separately. As the results were constant and unrelated to the number of runs, the total size for each dataset was analysed graphically.

7.3.1.2 Results Analysis

- ***Initialisation Time:***

Figure 7.1 shows the bar chart representing the median initialisation time (of 100 runs) required to label each XML experimental dataset individually by the Base-9 and SCOOTER schemes. As can be seen from the chart, there is a correlation between time and number of elements in the dataset rather than the file size, where the time increases as the number of elements increases. Notice from the dataset properties (Table 6.3) that the XMark dataset is a larger file and has

more elements than the Treebank dataset, but takes less initialisation time on average compared to Treebank. This could be due to the deep complex and recursive structure of the Treebank dataset.

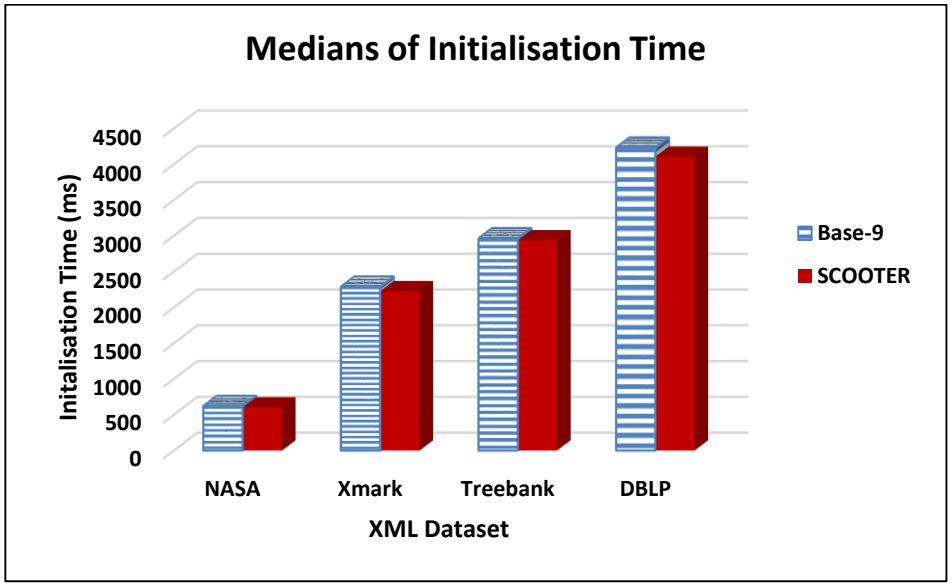


Figure 7.1 Initialisation time comparison (Base-9 vs SCOOTER)

For both schemes, the results were consistent for every XML dataset used. The SCOOTER scheme performed slightly better than the Base-9 scheme in terms of time (see Figure 7.1). The average difference between the execution time in the two schemes was in the range 1.10% to 4.87%. To study the significance of the results, the Mann- Whitney U-test was carried on for each dataset. The p- value obtained was always as low as $0.01\text{x}10^{-7}$ (for all datasets), which implies the rejection of the null hypothesis statement. This indicates there is a significant difference between the two schemes in terms of initialisation time. In order to investigate the preference in performance between the two schemes, the effective size, $\hat{A}_{12}\,(A,B)$, was measured for each XML experimental dataset. Knowing that group A and group B were presented as the SCOOTER and the Base-9, respectively, the effective size $\hat{A}_{12}$ value obtained was in the range $[0, 0.03]$. This confirms that the SCOOTER's initialisation algorithm operates faster than the Base-9's initialisation algorithm.

Figure 7.2 shows the box plot distribution of the Base-9 and the SCOOTER initialisation times for the XMark dataset. As the median line of the Base-9 box appears higher than the median line in the SCOOTER's box, this suggests that the SCOOTER scheme generates initial labels faster than the Base-9 scheme.

Due to the consistency of the results, the box plot charts for the remaining datasets can be found in Appendix B.

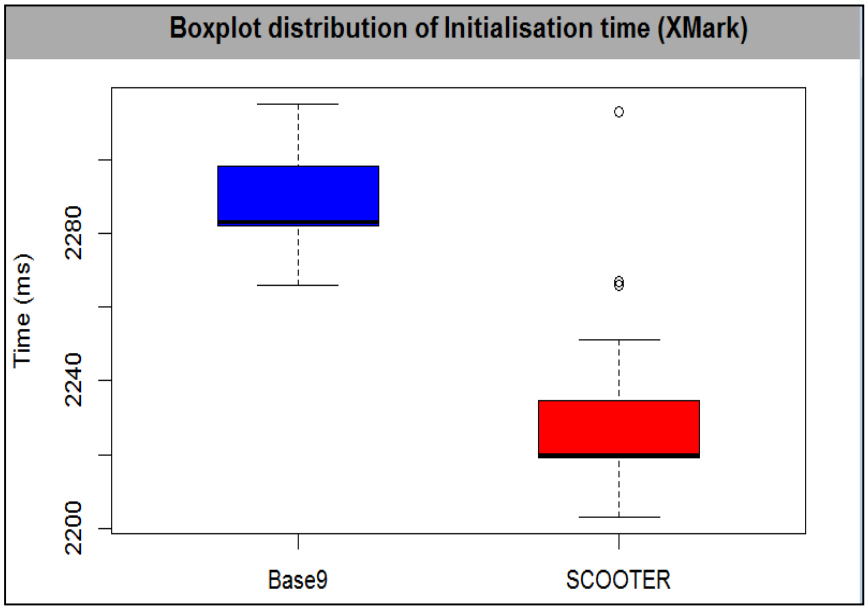


Figure 7.2 boxplots of the Base-9 and the SCOOTER initialisation times for the XMark dataset

- ***Label Size:***

The bar chart in Figure 7.3 shows an overall comparison between the Base-9 scheme and the SCOOTER scheme in terms of total label size generated for each XML dataset used. It shows that the results are consistent in all datasets; Base-9 labels are smaller than SCOOTER labels. Since the Base-9 scheme uses logarithm of base '9' rather than base '3' as in SCOOTER, it is expected that the total labels lengths will be reduced by about 30%. Table 7.1 shows the percentage decrease in total label size for each dataset. Considering the two different bases values used during initialisation process (i.e. 9 for Base-9 and 3 for SCOOTER), the total label lengths were converted to the same base $x = 2$ and the results are presented in Table 7.1.

Table 7.1 Total Label lengths comparison (Base-9 vs SCOOTER)

	Total labels size (no. of digits)			Total label lengths in same base (2)		
Dataset	Base-9	SCOOTER	different	Base-9	SCOOTER	difference
NASA	6882731	9467686	27.3%	21817741.1	15005927.3	31.2%
Treebank	45472288	59582496	23.7%	144143742.6	94436021.9	34.5%
DBLP	33872079	54869481	38.3%	107371950.1	86966069.8	19.0%
XMark	23937356	34251920	30.1%	75879623.3	54288008.8	28.5%

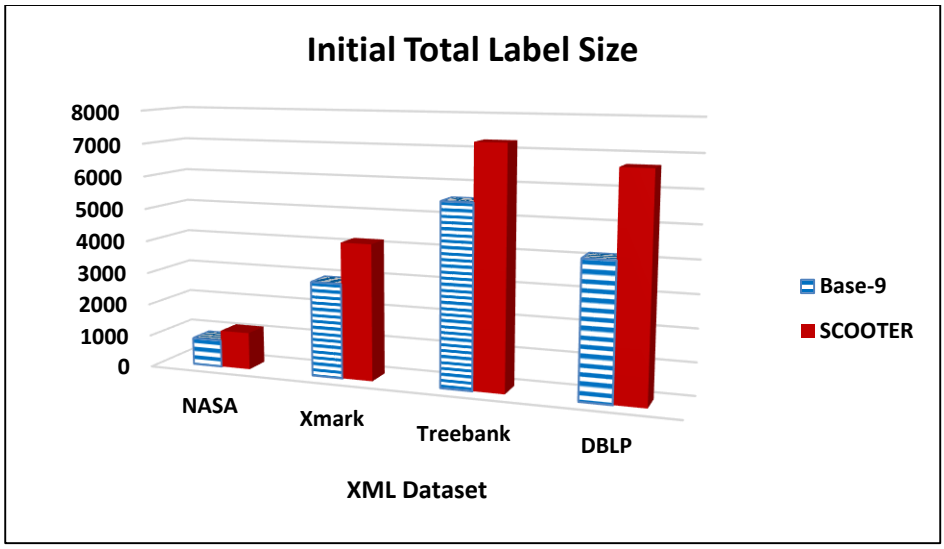


Figure 7.3 Initial label size comparison (Base-9 vs SCOOTER)

To show the difference in label size using the Base-9 scheme in preference to the SCOOTER scheme, Figure 7.4 represents the overall percentage distribution of label sizes generated for the XMark dataset. The percentage decrease in total label size for each dataset was calculated, the results of which were: 27.30% for NASA, 30.11% for XMark, 23.68% for Treebank, and 38.27% for DBLP. For an XML dataset with wider tree (i.e. with more $childCount$ per a node), such as in DBLP, takes extra advantage of using more digits (say y) to produce labels. This is because the $maxLabelSize$ computed by $log_y\ (ChildCount + 1)$ decreases as the value of y increase. In SCOOTER as number of nodes increases the label size increases by at least 1 digit per insertion, and gradually the algorithm generates larger labels in comparison to the Base-9.

The results confirm the achievement of the research objective in terms of generating more compressed labels via the proposed scheme. The percentage distribution charts for the label sizes for the other datasets can be found in appendix B.1 as their results were similarly consistent.

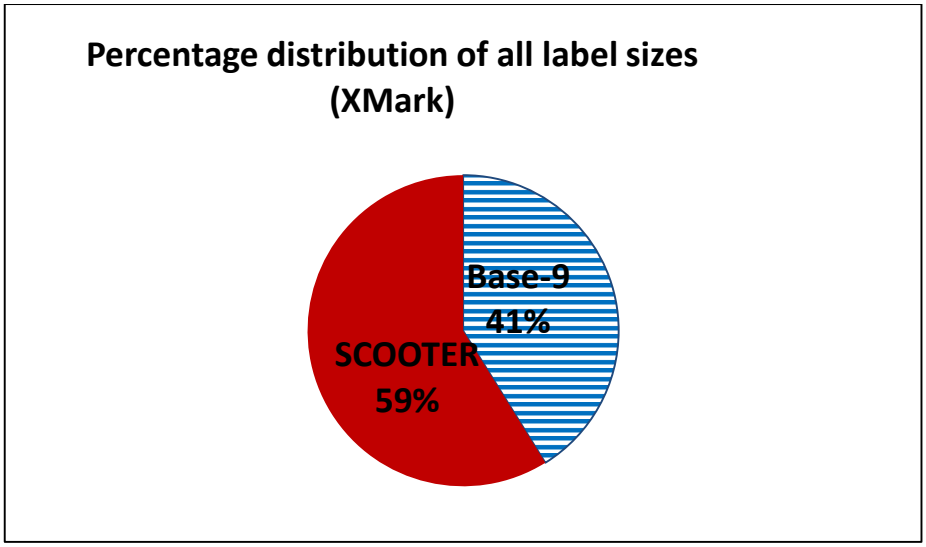


Figure 7.4 Difference in label size between Base-9 and SCOOTER

7.3.1.3 Conclusion

To conclude, the label initialisation experiment compared the performance of the Base-9 scheme against the SCOOTER scheme in terms of initialisation time and label sizes. The analytical results have shown that SCOOTER generates initial labels faster than the Base-9 scheme. However, the Base-9 scheme outperformed the SCOOTER scheme in terms of the compactness of the initial labels by an average of 29.84% for the four datasets used.

7.3.2 Handling Insertions

As discussed in Chapters 3 and 5, the insertion mechanism of the Base-9 labelling scheme was developed based on lexicographical comparison rather than using an adaptive growth technique as in the SCOOTER scheme. To study whether such an enhancement to the insertion methodology has affected the insertion time and label size, the handling insertions experiment discussed in Chapter 6 was performed. The significance of the results is analysed in the next section.

7.3.2.1 Analytical Strategy

Unlike the initialisation process, the new labels generated for nodes inserted later are related to their randomly selected positions. Hence, the total label size of a dataset varies after each insertion. It is thus essential to consider the randomness when evaluating label size and insertion time. The performance of

the insertion mechanisms may also vary depending on the number of nodes inserted, so it is important to consider the choice of the number of new nodes inserted as the “artefacts” factor of this evaluation.

The main goal to using multiple artefacts is to improve the external validity when evaluating the insertion methods of the Base-9 and SCOOTER labelling schemes. That is, which of the two labelling schemes performs better when a small or a large number of nodes are inserted, and whether the type of insertion (i.e., uniform and/or skewed) effects the performance. To address such questions, it is important to use statistical tests to assess which insertion algorithm is significantly better than the other for all the artefacts. As mentioned earlier, for more reliable statistical results, there should be a balance between the number of artefacts and how many times each artefact is run. It is recommended (Arcuri and Briand, 2014) that it is better to have more artefacts but the number of runs can be relatively low, perhaps as low as 10 runs. For the insertion time assessment, each artefact is executed 20 times. Because the label size falls under the randomisation category in this experiment, each artefact was run 10 times to measure the difference in label size after insertion.

The uniform and skewed insertions were tested in all the experimental datasets separately. To study the effectiveness of the insertion algorithms of both Base-9 and SCOOTER labelling schemes, 500, $1{,}000$, $5{,}000$, $10{,}000$, and $50{,}000$ nodes inserted were considered as the artefacts for testing the uniform insertion. To evaluate the skewed insertion, the artefacts were selected as 100, $5{,}000$, and $10{,}000$ nodes inserted, and each was performed repeatedly at 10 randomly selected positions.

The Mann-Whitney U-test was applied for each artefact to find the p-value in order to justify the difference in significance between the two labelling schemes in terms of time and size. Box plots were also generated to clarify the findings graphically.

7.3.2.2 Analysis of the Results

- **Uniform Insertion**

The uniform insertion performance was tested by inserting 500, $1{,}000$, $5{,}000$, $10{,}000$, and $50{,}000$ nodes individually in every experimental dataset using the Base-9 scheme. The same process was repeated using the SCOOTER scheme

for statistical comparison. To assess the insertion time, 20 runs (after excluding the first 5 runs) were recorded for every test and analysed below. To measure the increase in label size after insertion, for each test the total label size increase of the initial size was recorded for 10 runs. The medians of the sizes obtained were then compared statistically and graphically.

- ***Time comparison:***

Figure 7.5 illustrates the comparison of median time taken to insert 50,000 nodes at random positions in each experimental dataset. It can be seen in Figure 7.5 that both the Base-9 and SCOOTER schemes have almost the same insertion time. Apart from the Treebank dataset, there is generally a positive correlational relationship between the insertion time and the dataset size. This could be because the deep recursive structure of the Treebank dataset necessitates more separators within the prefix-based labels generated. Such an observation was consistent with the results for the other tests with fewer nodes. The results for $500,\ 1{,}000,\ 5{,}000,$ and $10{,}000$ node insertions are shown in appendix B.2.

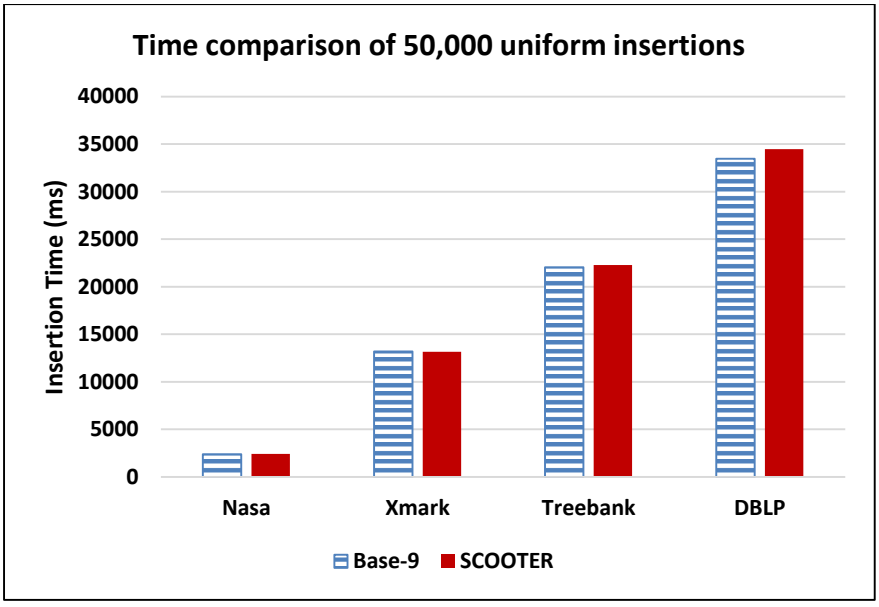


Figure 7.5 Uniform insertion time comparison (Base-9 vs SCOOTER)

To investigate the statistical significance of the difference in insertion time distribution between the two schemes, the Mann-Whitney U-test was applied for each artefact in every dataset. The p-values obtained were more than the significance level, 0.05, in all cases except when inserting 50,000 nodes in the Treebank and DBLP datasets (see Table 7.2). For the tests where $p > 0.05$ the null hypothesis is retained, implying that there is no difference between the two

schemes in terms of insertion time. For the exceptional results of $p < 0.05$ when inserting 50,000 nodes in Treebank and DBLP, further statistical analysis was undertaken to study the difference in performance between the two schemes.

Table 7.2 p-values of uniform insertion time distribution

XML Dataset	500 insertions	1,000 insertions	5,000 insertions	10,000 insertions	50,000 insertions
NASA	0.529	0.779	0.883	0.056	0.056
XMark	0.183	0.820	0.678	0.211	0.904
Treebank	0.495	0.620	0.165	0.265	**0.026**
DBLP	0.277	0.841	0.512	0.149	**0.001X10^{-4}**

The effective size, $\hat{A}_{12}\,(SCOOTER, Base9)$, was measured for 50,000 uniform insertion tests on Treebank and DBLP. The results obtained were 0.704 and 0.835, respectively. This confirms that the Base-9 scheme processes very large uniform insertions up to 50,000 nodes in relatively large datasets, such as in Treebank and DBLP, faster than the SCOOTER scheme by at least 70.4%. The box plot charts in Figure 7.6 demonstrate the difference in the performance between the two schemes in terms of insertion time.

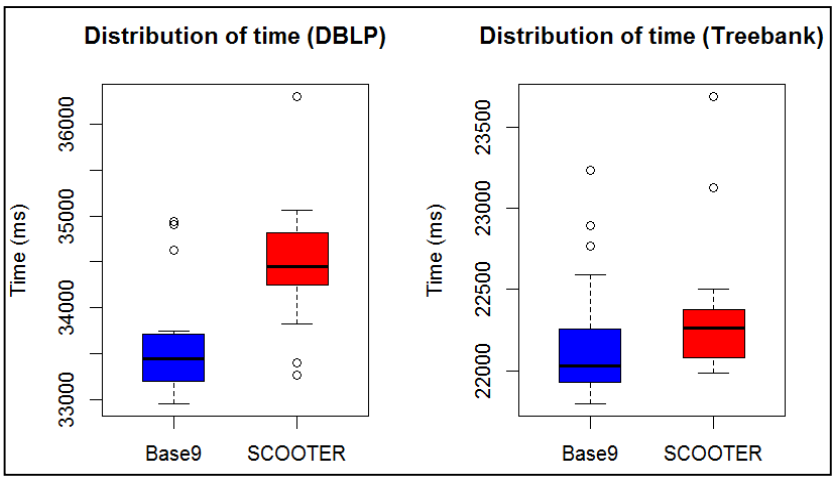


Figure 7.6 Box plot distribution of 50,000 uniform insertion times

- ***Size comparison:***

Figure 7.7 demonstrates the difference in the growth rate of the label sizes between the Base-9 and SCOOTER schemes after 50,000 uniform insertions in each XML dataset. The size here represents the 50,000 labels (total lengths in digits) added to the initial labels of that dataset. In every run, different random positions were chosen for the new 50,000 nodes. The median of the total size of 10 runs (after excluding the first 5 runs as usual) were recorded and compared in

Figure 7.7. This figure shows that, in general, the Base-9 scheme generates shorter labels than SCOOTER. Due to the stability of the results after 500, $1{,}000$, $5{,}000$, and $10{,}000$ insertions, their discussions are omitted but appendix B.2 presents similar graphical comparisons.

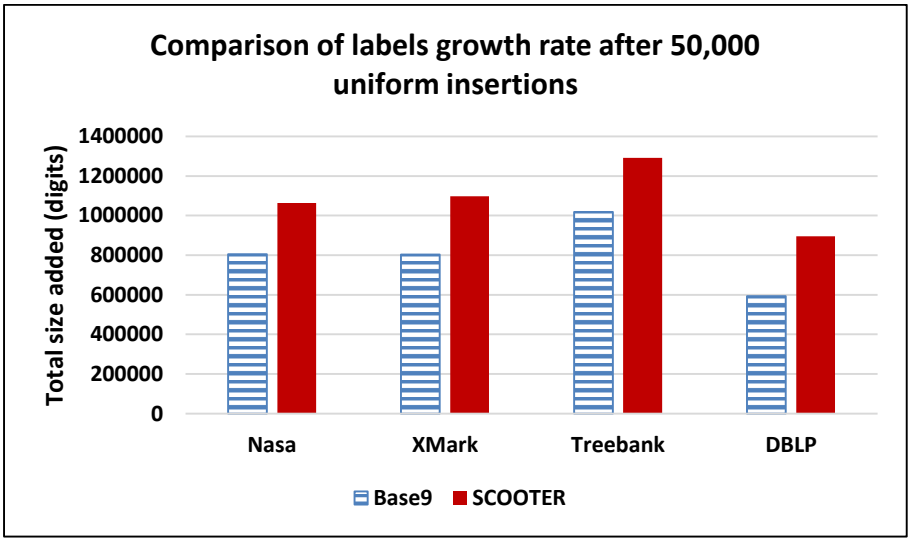


Figure 7.7 Difference in the growth of label size (Base-9 vs SCOOTER)

The statistical significance of the results was measured using the Mann-Whitney U-test for each test, and the p-value obtained was always between $1.083\text{x}10^{-5}$ and $1.085\text{x}10^{-5}$. This confirms that there is difference in the performance of the two schemes in terms of label sizes after uniform insertions. The box plot chart in Figure 7.8 shows that, in contrast to the SCOOTER scheme, Base-9 controls the growth of the label sizes better when 500 nodes were inserted at different random positions. Appendix B.2 provides similar box plots for all the uniform insertion tests. The effect size, $\hat{A}_{12}\ (SCOOTER, Base9)$, in all the tests were found to exactly equal 1, which means that 100% of the time the Base-9 scheme generated shorter labels than SCOOTER during uniform insertions.

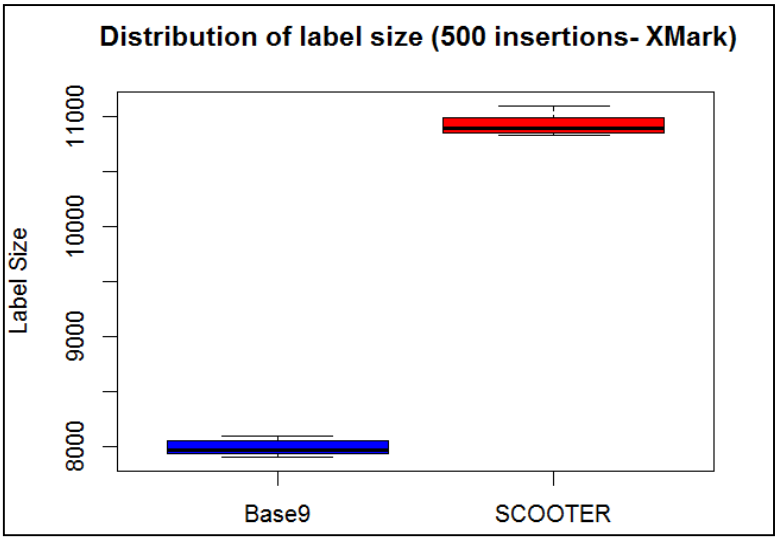


Figure 7.8 box plot distribution label sizes of 500 insertions in XMark

Table 7.3 shows the average reduction percentage in the label sizes of the new 500, $1{,}000$, $5{,}000$, $10{,}000$, and $50{,}000$ nodes added during uniform insertions by using the Base-9 scheme rather than the SCOOTER scheme.

Table 7.3 Average reduction (percentage) in label size after uniform insertions

XML Dataset	500 insertions	1,000 insertions	5,000 insertions	10,000 insertions	50,000 insertions
NASA	24.33%	24.36%	24.35%	24.29%	24.23%
XMark	26.87%	26.73%	26.78%	26.72%	26.81%
Treebank	20.47%	20.91%	21.13%	21.14%	21.14%
DBLP	33.58%	33.83%	33.86%	33.83%	33.81%

- **Skewed Insertion**

Skewed insertion has recently become one of the main focuses in developing XML labelling schemes (see Chapter 3); relatively large numbers of 100, $5{,}000$, and $10{,}000$ skewed insertions were carried out with all the experimental datasets. Each set of skewed insertions was repeated at 10 random positions. Hence, the total number of nodes inserted were $100X10 = 1000$, $5{,}000X10 = 50{,}000$, and $10{,}000X10 = 100{,}000$ (referred to as "artefacts"), which were labelled by the Base-9 scheme and the SCOOTER scheme. For insertion time assessment, each test was executed 20 times (after excluding the first 5 runs). To study the growth of label sizes after insertion, the total size of the new labels was computed for each test. Since the label values vary based on their randomly selected insertion positions, the sizes obtained for 10 runs were recorded and analysed.

- ***Time comparison:***

As in the uniform insertion test, the median values of the skewed insertion times were compared graphically. For example, Figure 7.9 presents the time comparison for $10{,}000X10$ insertions in all datasets. The correlation between the insertion time and the dataset size is the same as in the uniform insertion. For the lower number of skewed insertions, $100X10$ and $5{,}000X10$, the same observation was made; see appendix B.2.

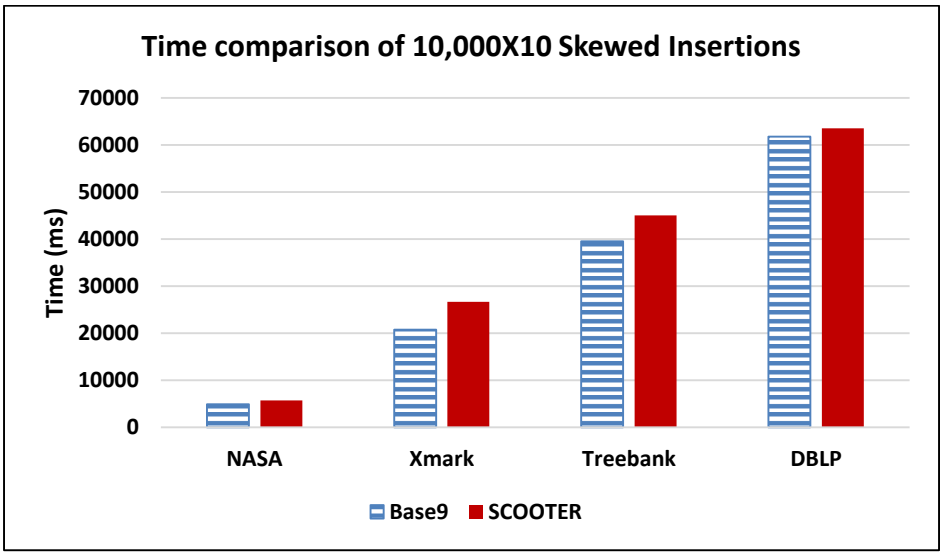


Figure 7.9 Time comparison for skewed insertion (Base-9 vs SCOOTER)

The Mann-Whitney U-test was applied for each artefact in every dataset to obtain the statistical difference of the insertion time distribution between the Base-9 scheme and SCOOTER. Table 7.4 shows the p-values obtained for each test. The majority of these tests show there is no difference between the two schemes in terms of insertion time as their p-values were higher than 0.05. However, the cases highlighted in Table 7.4 provided $p < 0.05$, which indicates the rejection of the null hypothesis. To examine the difference in the performance of the two schemes, the time distribution in these cases was illustrated by the use of box plots.

Table 7.4 p-values of skewed insertion time distribution

XML Dataset	100X10 insertions	5,000X10 insertions	10,000X10 insertions
NASA	0.052	0.060	0.009×10^{-4}
XMark	0.004	0.002×10^{-5}	0.001×10^{-5}
Treebank	0.841	0.014	0.583
DBLP	0.108	0.659	0.327

The box plots in Figure 7.10 illustrate the distribution of times taken to insert 100 nodes repeatedly in 10 different random positions in the XMark dataset. As the level of the median line in the SCOOTER box appears higher in the box plot, this suggests that Base-9 performs this particular number of skewed insertions faster than SCOOTER. For the other cases highlighted in Table 7.4, similar observations were found from their box plots, which are available in appendix B.2.

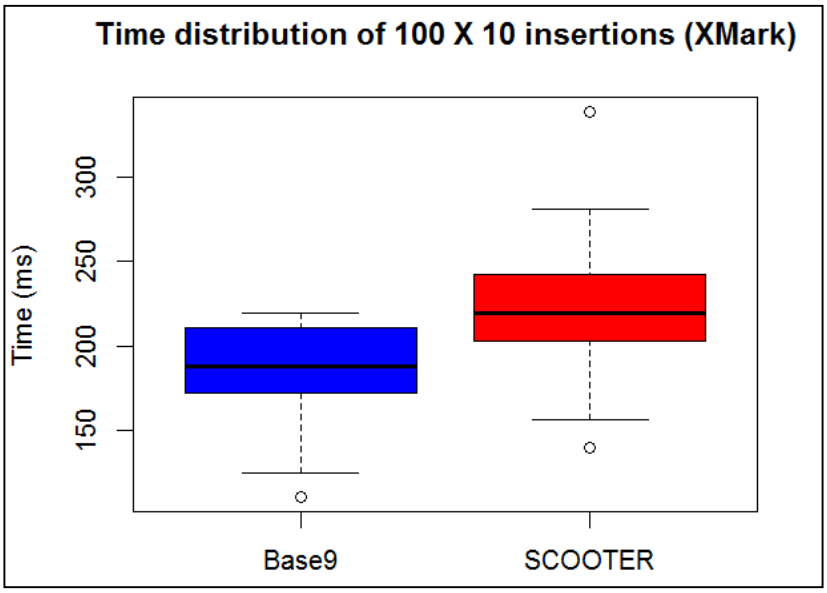


Figure 7.10 box plot distribution of 100X10 skewed insertion times in XMark

- ***Size comparison:***

Figure 7.11 and Figure 7.12 demonstrate the difference between the Base-9 and SCOOTER schemes in terms of the growth rate of the new $100X10$ and $5{,}000X10$ labels added by skewed insertions in each XML dataset. Due to the randomness of the insertion mechanisms, each test was run 10 times. In each run the total size of the labels generated was computed. The sizes in Figure 7.11 and Figure 7.12 are the medians of the total sizes (in Kbytes) obtained. See appendix B.2 for the bar chart representation of the 10,000X10 skewed insertion results, which are consistent with Figure 7.11 and Figure 7.12.

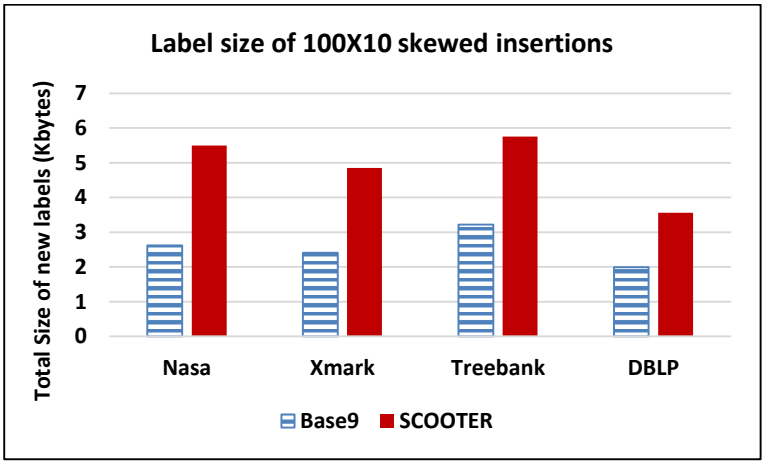


Figure 7.11 Size comparison in 100X10 insertions (Base-9 vs SCOOTER)

It can easily be seen from these figures that there is a big difference between the two schemes in terms of the increase in label sizes, especially when very large numbers of skewed insertions (e.g., 5,000 X 10) occurred. Base-9 generates very compressed labels in comparison to the SCOOTER scheme. The relative percentage change in label sizes generated using Base-9 instead of SCOOTER were computed as follows (MathGoodies, 2015): $((size_{SCOOTER} - Size_{Base9})/ Size_{SCOOTER})$. The percentages obtained for $100X10$, $5{,}000X10$ and $10{,}000X10$ insertions are presented in Table 7.5.

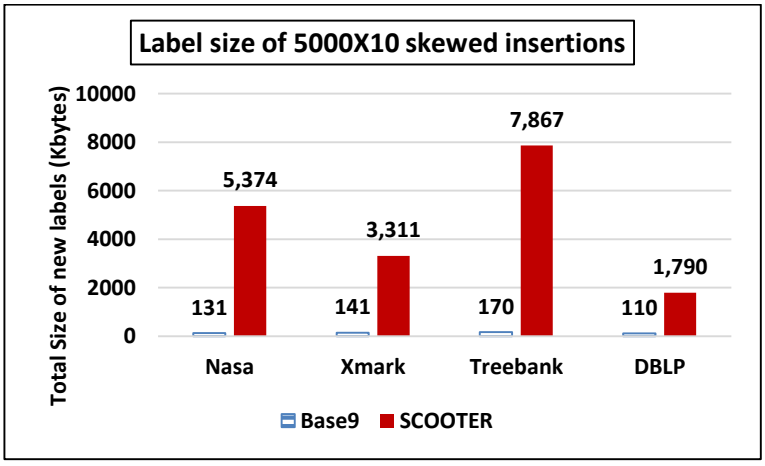


Figure 7.12 Size comparison in 5000X10 insertions (Base-9 vs SCOOTER)

Table 7.5 Average decrease percentage of the size's growth rate in skewed insertions

XML Dataset	100X10 insertions	5,000X10 insertions	10,000X10 insertions
NASA	52.33%	97.56%	98.69%
XMark	50.37%	95.73%	98.44%
Treebank	44.09%	97.85%	98.81%
DBLP	44.00%	93.85%	95.94%

As mentioned in Chapters 3 and 5, the SCOOTER scheme is currently the most compact dynamic labelling scheme that supports skewed insertion (O'Connor and Roantree, 2013) (Chiew et al., 2014a). In view of the reduction in size shown here (see Table 7.5), it can be concluded that the Base-9 scheme improves skewed insertion performance in terms of compressing XML labels by at least 44%. The p-values obtained by the Mann-Whitney U-test applied in all the artefacts was always ($p = 1.083\text{x}10^{-5}$) < 0.05 for every dataset. This implies the rejection of the null hypothesis indicating there is a difference in performance between the two schemes in terms of label sizes, as observed earlier. The effect size, $\hat{A}_{12}$ $(SCOOTER, Base9)$, was measured for all the artefacts in every dataset and the result was always equal to 1. This confirms the conclusion that the Base-9 scheme always generates more compressed labels than SCOOTER when dealing with skewed insertions. The box pots in Figure 7.13 illustrate this observation graphically for the XMark dataset (see appendix B.2 for box plots of the remaining datasets).

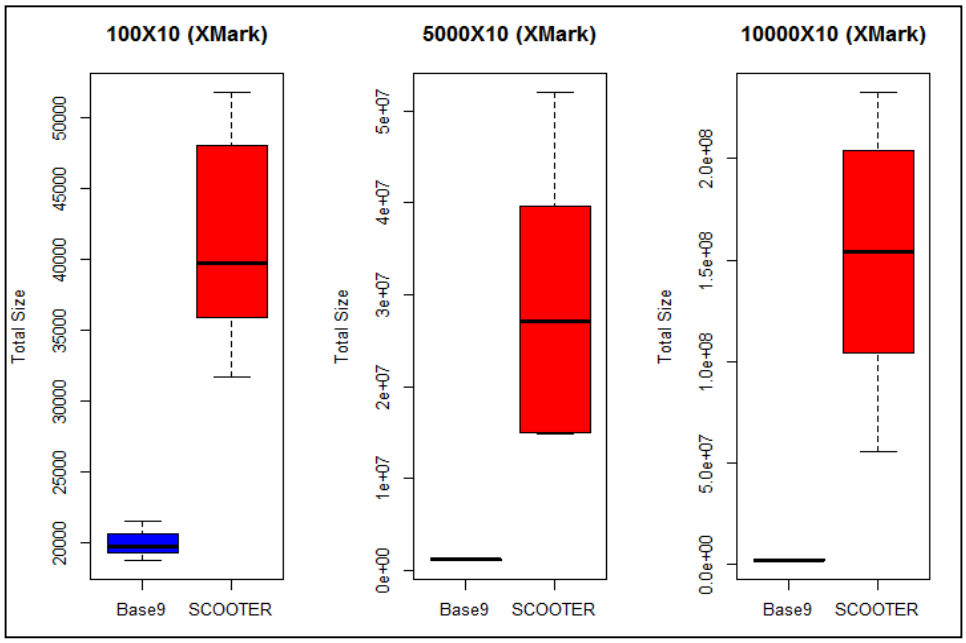


Figure 7.13 Box plot distribution of total label sizes (Kbytes) in XMark (Base9 vs SCOOTER)

7.3.2.3 Conclusion

This experiment compared the ability of the Base-9 scheme and the SCOOTER scheme to handle insertions. In all the experimental datasets, two types of insertions were tested: uniform insertion and skewed insertion. The tests have covered inserting small and large numbers of nodes. The results showed that both schemes require almost the same insertion time in most cases, and where there is a difference then the Base-9 scheme labels new nodes faster. In terms of size, it has been proven that every time, and in all insertion tests, the Base-9 scheme consistently generates more compressed labels than the SCOOTER scheme.

7.3.3 Re-using Deleted Nodes' Labels

The SCOOTER scheme has shown its capability for re-using the smallest available deleted label (O'Connor and Roantree, 2012). The Base-9 scheme generates labels based on the lexicographical comparison technique (see Chapter 5) during initialisation and insertion. This approach should allow re-use of almost all of the deleted labels. This experiment was designed to test whether the proposed scheme enables re-use of deleted nodes' labels. It also compares the ability of both schemes in terms of re-using deleted labels.

7.3.3.1 Analytical Strategy

As described in Chapter 6, to test the re-usability of deleted labels, the first n adjacent sibling nodes were deleted from a sample set of labels. The deleted label values were stored in an array-list (say $list\_A$). Then, the same number n of new nodes were inserted at the same positions as those of the n deleted nodes. Three types of updates were tested:

- Delete and insert n siblings after the last child.
- Delete and insert n siblings before the first child.
- Delete and insert n siblings between two nodes.

The new n labels inserted were recorded in another array-list called $list\_B$. Then, the two lists were compared and the percentage of identical labels in both lists was computed. The results were analysed based on this percentage.

This experiment was performed in the DBLP dataset as it provides the widest range of sibling nodes due to its extensive fan-out (see Section 6.5.1), with $n = 10$. The labels tested were displayed in tabular format to show the difference between the original deleted labels and the new labels. The experiment was also carried out on 1,500 self-labels generated to represent 1,500 sibling nodes, with $n = 1000$. For a fair test, the sample label set selected from the Base-9 labels corresponded to the set selected from the SCOOTER labels.

7.3.3.2 Analysis of the Results

Table 7.6 displays a sample set of Base-9 labels and their corresponding SCOOTER labels representing 12 adjacent sibling nodes in the DBLP dataset. Index denotes the nodes' order of appearance.

Table 7.6 Label set sample from DBLP

Index	Base-9 Labels	SCOOTER Labels
1	0.111119.12	2.111111111133.112
2	0.111119.13	2.111111111133.113
3	0.111119.14	2.111111111133.12
4	0.111119.15	2.111111111133.122
5	0.111119.16	2.111111111133.123
6	0.111119.17	2.111111111133.13
7	0.111119.18	2.111111111133.132
8	0.111119.19	2.111111111133.133
9	0.111119.2	2.111111111133.2
10	0.111119.21	2.111111111133.212
11	0.111119.22	2.111111111133.213
12	0.111119.23	2.111111111133.22

To test the re-usability when inserting after the right-most child node, the last 10 nodes were deleted and then 10 new nodes were inserted after the remaining last child. Table 7.7 shows the original and new labels generated. It is obvious from the Base-9 comparison in Table 7.7 that Base-9 re-created the deleted label values. On the other hand, as the SCOOTER insertion mechanism ensures the generation of the smallest available label value at first, only one deleted label was re-used (i.e., self-label = “2”).

Table 7.7 Testing re-usability when inserting after last child

Insertion order	Original Base-9	New Base-9	Original SCOOTER	New SCOOTER
Start node:	0.111119.13	0.111119.13	2.111111111133.113	2.111111111133.113
1	0.111119.14	0.111119.14	2.111111111133.12	***2.111111111133.2***
2	0.111119.15	0.111119.15	2.111111111133.122	2.111111111133.3
3	0.111119.16	0.111119.16	2.111111111133.123	2.111111111133.32
4	0.111119.17	0.111119.17	2.111111111133.13	2.111111111133.33
5	0.111119.18	0.111119.18	2.111111111133.132	2.111111111133.3312
6	0.111119.19	0.111119.19	2.111111111133.133	2.111111111133.3313
7	0.111119.2	0.111119.2	***2.111111111133.2***	2.111111111133.332
8	0.111119.21	0.111119.21	2.111111111133.212	2.111111111133.3322
9	0.111119.22	0.111119.22	2.111111111133.213	2.111111111133.3323
10	0.111119.23	0.111119.23	2.111111111133.22	2.111111111133.333

To examine the re-usability when inserting before the left-most child node, the first 10 nodes were deleted and then 10 new nodes were inserted before the remaining first child. Table 7.8 illustrates the original and the new label comparison. Again, the Base-9 scheme re-produced all the deleted labels, whereas, SCOOTER generated 40%.

Table 7.8 Testing re-usability when inserting before the first child

Insertion order	Original Base-9	New Base-9	Original SCOOTER	New SCOOTER
10	0.111119.12	0.111119.12	***2.111111111133.112***	2.111111111133.111111112
9	0.111119.13	0.111119.13	2.111111111133.113	2.111111111133.11111112
8	0.111119.14	0.111119.14	***2.111111111133.12***	2.111111111133.1111112
7	0.111119.15	0.111119.15	2.111111111133.122	2.111111111133.111112
6	0.111119.16	0.111119.16	2.111111111133.123	2.111111111133.11112
5	0.111119.17	0.111119.17	2.111111111133.13	2.111111111133.1112
4	0.111119.18	0.111119.18	2.111111111133.132	***2.111111111133.112***
3	0.111119.19	0.111119.19	2.111111111133.133	***2.111111111133.12***
2	0.111119.2	0.111119.2	***2.111111111133.2***	***2.111111111133.2***
1	0.111119.21	0.111119.21	***2.111111111133.212***	***2.111111111133.212***
Start node	**0.111119.22**	**0.111119.22**	**2.111111111133.213**	**2.111111111133.213**

In order to test the re-usability when inserting between two adjacent siblings, the 10 middle nodes in Table 7.6 were deleted and replaced with 10 new nodes, as shown in Table 7.9. Consistent with the previous results, the Base-9 scheme re-used all the deleted labels in this scenario, whereas the SCOOTER scheme re-used 3 out of 10 deleted labels (those highlighted in Table 7.9).

Table 7.9 Testing re-usability when inserting between two sibling nodes

Insertion order	Original Base-9	new Base-9	Original SCOOTER	new SCOOTER
Node1:	0.111119.12	0.111119.12	2.111111111133.112	2.111111111133.112
1	0.111119.13	0.111119.13	2.111111111133.113	***2.111111111133.2***
2	0.111119.14	0.111119.14	2.111111111133.12	***2.111111111133.212***
3	0.111119.15	0.111119.15	2.111111111133.122	***2.111111111133.213***
4	0.111119.16	0.111119.16	2.111111111133.123	2.111111111133.2133
5	0.111119.17	0.111119.17	2.111111111133.13	2.111111111133.2123312
6	0.111119.18	0.111119.18	2.111111111133.132	2.111111111133.2123313
7	0.111119.19	0.111119.19	2.111111111133.133	2.111111111133.212332
8	0.111119.2	0.111119.2	***2.111111111133.2***	2.111111111133.2123322
9	0.111119.21	0.111119.21	***2.111111111133.212***	2.111111111133.2123323
10	0.111119.22	0.111119.22	***2.111111111133.213***	2.111111111133.212333
Node2:	0.111119.23	0.111119.23	2.111111111133.22	2.111111111133.22

To generalise the results, 1,500 self-labels were generated using the initialisation algorithms of the Base-9 and SCOOTER schemes separately. In order to study the effect of the insertion algorithms over a wider range of labels, these 1,500 self-labels were used as the testing sample. The same experiment was repeated but with $n = 1000$ instead of 10 nodes. The percentage of re-used labels for each insertion type were computed for both schemes individually and are presented in Table 7.10. These percentages were found by counting the number of new labels (after insertion) that existed in the original label set (before deletion).

Table 7.10 Percentage of re-used deleted labels (Base-9 vs SCOOTER)

Insertion type	Base-9	SCOOTER
Inserting after last node	98.7% used (987/1000 re-used)	0.2% used (only 2 labels re-used)
Inserting between nodes	99.1% used (991/1000 re-used)	0.9% used (only 9 out of 1000 re-used)
Inserting before first node	98.7% used (987/1000 re-used)	0.7% used (only 7 labels re-used)

As shown in Table 7.10, Base-9 outperforms the SCOOTER in terms of re-use of deleted labels. This is because the insertion algorithm in the Base-9 labelling scheme was established based on the same principle as the initialisation

process; that is, by finding next sibling label value that is lexicographically closest to the label value of the current node. The new label size was always controlled by $maxLabelSize,$ which is computed based on the maximum number of children per node (see Chapter 5). This mechanism guaranteed reusing all the deleted label values, but as seen from the experimental results above, about 1% or 2% missing deleted nodes. To further investigate this observation, extra nodes were inserted and then the comparison between the original and re-used labels was repeated. The results show that after almost 15 nodes insertions before the first child and/or after the last child, all the deleted nodes were re-used. For insertion between two consecutive siblings, after inserting only 8 extra nodes the algorithm regenerated all the deleted nodes successfully.

These results were obtained due to more restriction on using digit '1' and '0' during the initialisation process (see Section 5.5) in comparison to the insertion mechanism, which allows more lexicographical combination including '1' and '0'. Table 7.11 illustrates examples of labels generated initially and then re-used later after deletion. The highlighted labels represent the new labels generated by insertion algorithms without restriction of number of '1's at the end of a label value. Because of this, more insertions were necessary to establish 100% of deleted labels in the experiment.

Table 7.11 Examples of generated labels (initial vs updated)

Insert before first node		Insert between two nodes		Insert after last node	
Initial	**Insertion**	**Initial**	**Insertion**	**Initial**	**Insertion**
1998	1998	1398	1398	1698	1698
1999	1999	1399	1399	1699	1699
2	2	14	14	17	17
2111	21	1411	141	1711	171
2112	211	1412	1411	1712	1711
2113	2111	1413	1412	1713	1712

The insertion algorithm in the SCOOTER labelling scheme was applied based on the compact growth mechanism introduced by (O'Connor and Roantree, 2012) to control the label growth rate regardless of the extent of skewed node insertions and deletions. The experiments illustrated above have shown that the SCOOTER insertion algorithm at first reused the smallest deleted label values before

generating the rest of the new labels, which were derived based on the growth mechanism which always led to larger labels.

If all the child nodes of an element are deleted, then the initialisation algorithm is applied to label the new leaf nodes. Therefore, in both labelling schemes the same deleted values would always be generated.

7.3.3.3 Conclusion

This experiment was executed to test the Base-9 scheme's ability to re-use deleted nodes in comparison to SCOOTER. The results, as shown above, confirm the insertion technique applied by the Base-9 scheme based on the lexicographical comparison, enables the re-use of at least 98.7% of the deleted labels (if any). Although the SCOOTER guarantees re-use of the smallest quaternary string(s) (i.e., "12", "2", and/or "3"), the adaptive growth mechanism naturally produces larger new labels following the maximum quaternary digit '3' (see Table 7.7 and Table 7.9) or preceding the smallest quaternary string "12" (see Table 7.8).

7.3.4 Label Encoding

As mentioned in Section 6.5, this experiment focused on examining the storage capacity required to store the Base-9 labels encoded using the Fibonacci encoding of order 2 and order 3 (see Chapter 5) whereas, the SCOOTER labels were encoded by the QED encoding method (see Section 3.4). The study also presented a comparison between the two schemes in terms of encoding time and size of both the initial labels and the updated labels.

7.3.4.1 Analytical Strategy

Two main factors were considered when evaluating this experiment: the encoding time (in milliseconds) and total encoded label size (in Kbytes). As the time computation falls into the randomness classification, the number of runs was identified to obtain a statistically significant difference between the encoding methods of the two schemes. For each experimental dataset, the encoding time of 100 runs over the initial labels were recorded. To analyse the encoding time on the inserted labels, 20 runs were measured for each artefact of the skewed insertion for each dataset. Fewer runs were taken into consideration for the

updated labels as in each insertion run per artefact, different labels were generated based on the new randomly selected node positions (see Section 7.3.2).

In terms of evaluating the encoded label size, for each experimental dataset the total size of the encoded labels was calculated for each encoding method for both labelling schemes separately. As for the initial labels, the results were constant and irrelevant to the number of runs, the total size of each encoding method for each dataset was analysed graphically. The inserted labels differ for each run, so the encoded size measurement in this case was categorised as a randomised assessment (see Section 7.3.2). Thus, the total size of 20 runs was taken for each encoding method and applied over each dataset.

As the comparison is based on multiple techniques (Fibonacci of order 2 and Fibonacci of order 3 for Base-9, and QED for SCOOTER), the non-parametric Kruskal–Wallis test (Vargha and Delaney, 1998) was selected to obtain the p-value (see Section 7.2).

7.3.4.2 Analysis of the Results

➢ **Initial Labels**

- ***Time comparison:***

As Figure 7.14 shows the difference on encoding time for the XMark dataset, it can be seen that, in general, QED generates label codes faster than Fibonacci coding. This observation is consistent with the remaining XML experimental dataset, as reported in appendix B.2.

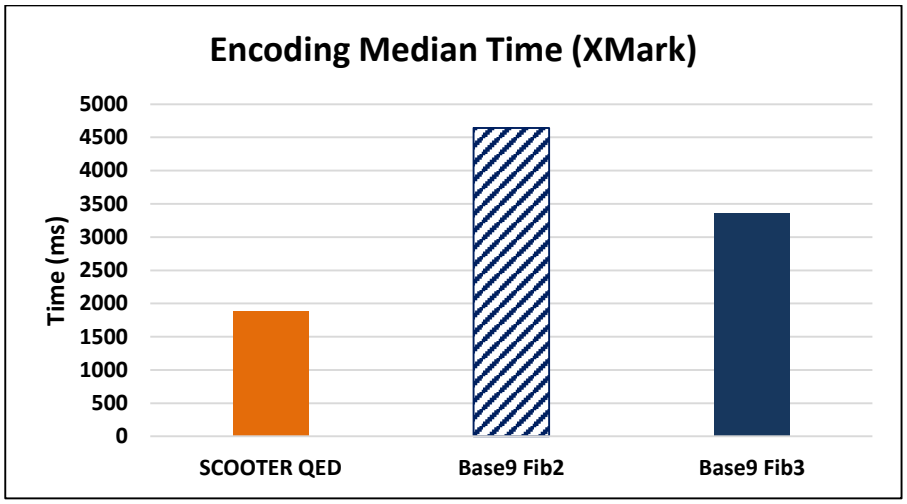


Figure 7.14 Encoding time comparison (XMark)

To justify the statistical significance of the results, the non-parametric Kruskal–Wallis test was carried out on the encoding time. The p-value obtained was less than $0.001\text{x}10^{-6}$ for all datasets, indicating the rejection of the null hypothesis. This implies there is very strong evidence to suggest a difference between at least one pair of the applied encoding methods. Thus, further analysis was conducted using the Manny-Whitney U-tests to study the difference between each pair of methods for every dataset in terms of encoding time. The results of the "pairwise comparisons" have shown that there was very strong evidence (p<0.001, adjusted using the Bonferroni correction) for a difference between every pair on all the datasets.

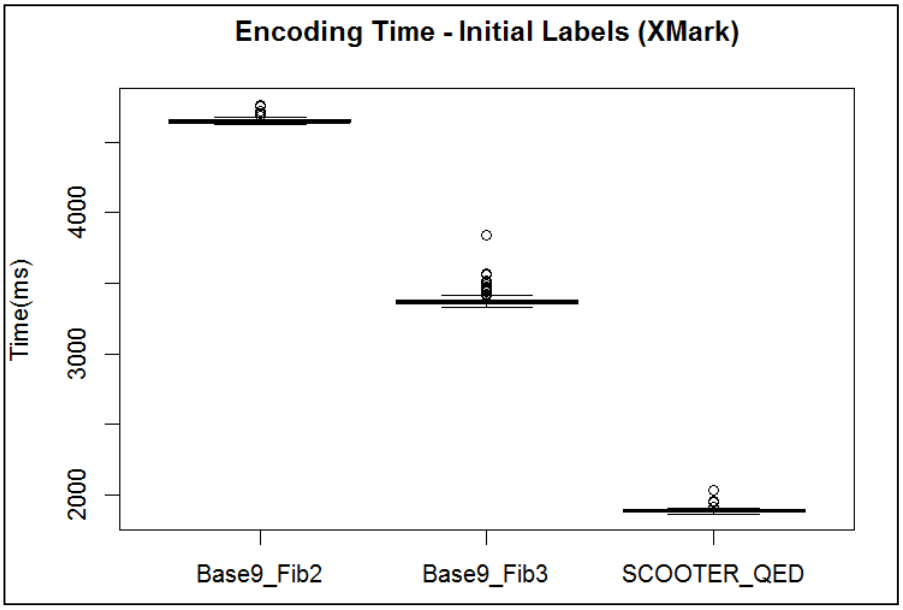


Figure 7.15 Box plot of encoding time (initial labels) distribution for XMark dataset

The box plots in Figure 7.15 present the distribution of time taken (for 100 runs after excluding the first 5 runs) to encode the initial labels of the XMark dataset using QED for the SCOOTER labels and Fibonacci of order 2 and order 3 for the Base-9 labels. In the box plots, the level of the median line in the SCOOTER_QED box is the lowest, which confirms the results observed from Figure 7.14 that the QED method is the fastest among the various methods compared. Fibonacci of order 3 performed faster than the Fibonacci of order 2. The same observation was established for all the experimental datasets used; see appendix B.2.

- ***Size comparison:***

The bar charts in Figure 7.16 present the comparison between encoding methods in terms of encoded label size for each dataset. As can be seen from this figure, the shape of a dataset affects the results obtained. The XMark and NASA datasets are each represented by an XML tree with less extreme depth and/or width in comparison to Treebank and DBLP (see Section 6.5). Both datasets show similar results, in which the total size of the encoded Base-9 labels generated by any Fibonacci coding is always smaller than the QED codes representing the SCOOTER labels. The Fibonacci of order 2 produced the smallest codes in total.

Alternatively, the Fibonacci of order 2 generated the largest codes for the DBLP dataset, which has the shallowest and widest XML tree (see Section 6.5). For this dataset, the Fibonacci coding of order 3 gave the most compressed codes in comparison to the other encoding methods.

By comparison, for Treebank, which has the deepest XML tree with narrowest width, the Fibonacci of order 2 produced the smallest codes whilst the Fibonacci of order 3 formed the largest codes.

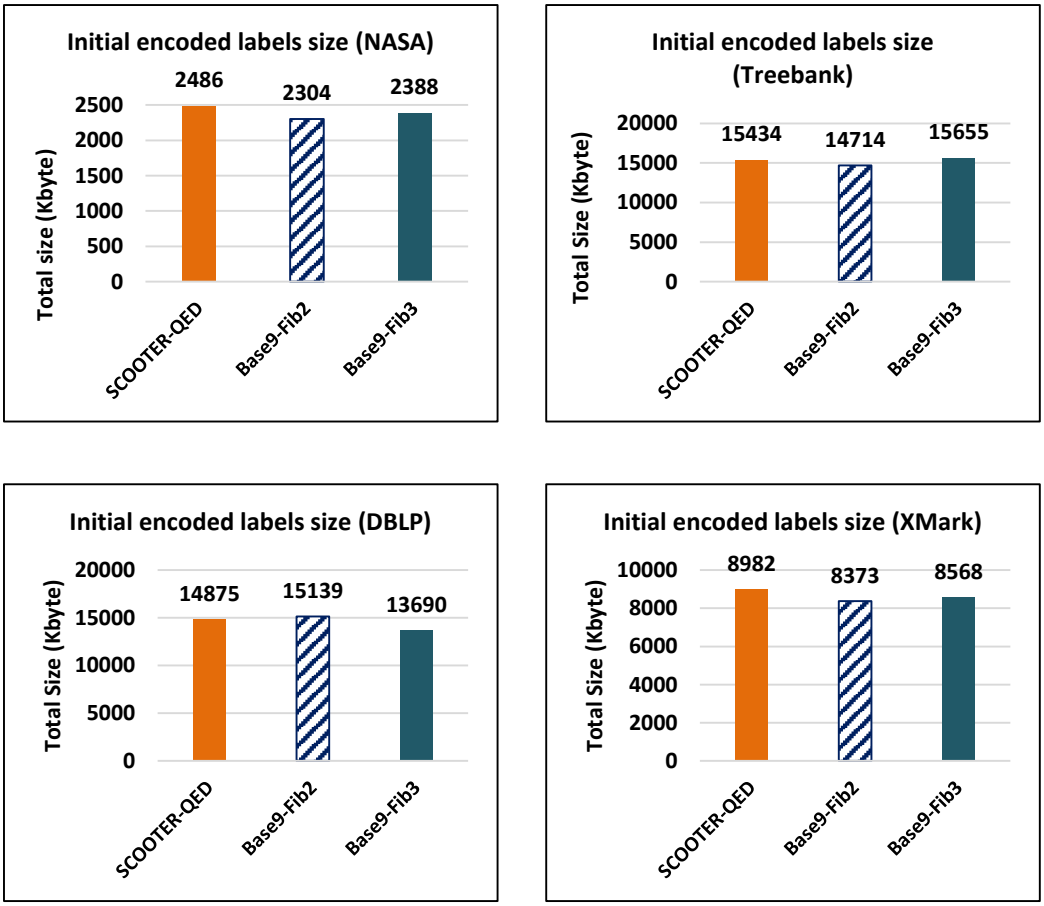


Figure 7.16 Encoding size comparison of initial labels

In terms of code size, order m Fibonacci encoding is relative to the shape of the tree representing an XML dataset. Section 7.4 discusses this observation in more detail as the performance of the encoding methods applied here are consistent with the results achieved in the XML label compression experiment. Overall, Fibonacci coding always generated the smallest codes in total, either using Fibonacci of order 2 or of order 3 if not both. Table 7.12 shows the percentage difference between QED and both Fibonacci encoding methods separately. The negative values indicate that QED performs better than Fibonacci of order 2 for

the DBLP dataset by 1.74% and also better than Fibonacci of order 3 for the Treebank dataset. The comparison between the two Fibonacci methods confirms that Fibonacci of order 3 produces shorter codes for larger labels by about 10.58% in DBLP. Otherwise, Fibonacci generates shorter codes.

Table 7.12 Percentage different on total code size between encoding methods

Dataset	QED vs Fib2	QED vs Fib3	Fib2 vs Fib3
NASA	7.90%	4.10%	-3.52%
Treebank	4.89%	-1.41%	-6.01%
DBLP	-1.74%	8.66%	10.58%
XMark	7.27%	4.83%	-2.28%

➢ **Inserted Labels**

- ***Time comparison:***

When inserting a small number of nodes (e.g., 100 nodes repeated at 10 different positions) QED encodes SCOOTER labels faster than the Fibonacci coding used for the Base-9 labels (see figure 7.17). However, when encoding a large number of nodes inserted in any XML experimental dataset, the QED consumed at least 98% more time than both Fibonacci coding (see Figure 7.18). Figures 7.17 and 7.18 present the encoding time comparison for a small (100X10) and large (5,000X10) number of skewed insertions. For an even larger number of insertions up to 10,000X10, the encoding methods behaved similarly to 5,000X10 insertions, as per the bar chart presented in appendix B.2.

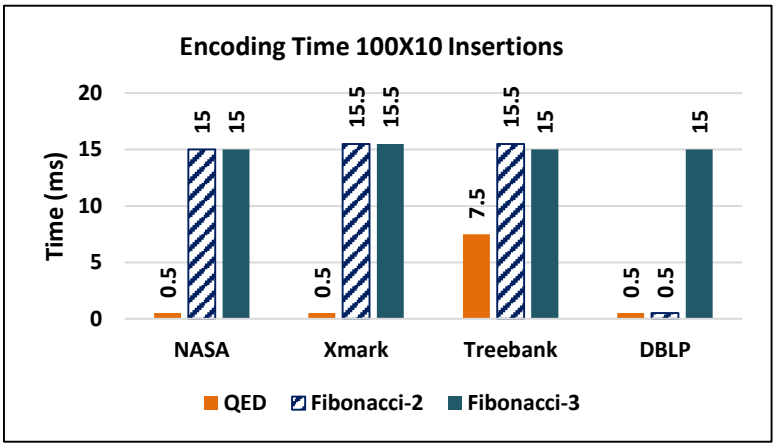


Figure 7.17 Encoding time comparison after 100 X 10 insertion

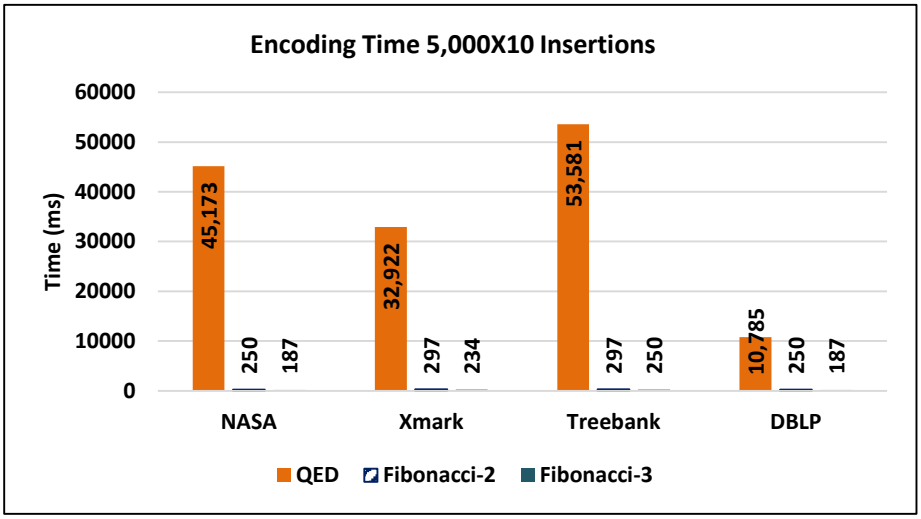


Figure 7.18 Encoding time comparison after 5,000 X 10 insertion

The Kruskal–Wallis test was applied to find the statistical significance of the results. For 100X10 insertions, the p-values obtained for the NASA and Treebank datasets were 0.090 and 0.194, respectively. This suggests there is no significant difference between any of the encoding methods in terms of encoding time. However, for the XMark and DBLP datasets, the p-values were 0.003 and 0.014, respectively, showing that there is a difference between at least two of the methods. To identify the differences, a pairwise comparison via the Mann-Whitney U-test was carried out on the encoding time. For the DBLP dataset, the results show a difference between QED and Fibonacci of order 3 only, whilst for XMark, the results indicated there is no difference between either Fibonacci coding method but there is a difference between QED and each of the Fibonacci codings. The box plot in Figure 7.19 shows the distribution of encoding time after 100X10 insertions. As can be seen from this figure, in the XMark dataset the QED performed better. In the DBLP dataset, both QED and Fibonacci of order 2 have low median lines of almost zero. Thus, the effect size for this case was computed and the result was $\hat{A}_{12}\ (QED, Fib2)$ = 0.45, which implied that 45% of the time QED encodes SCOOTER labels faster than the Fibonacci of order 2 for the Base-9 labels.

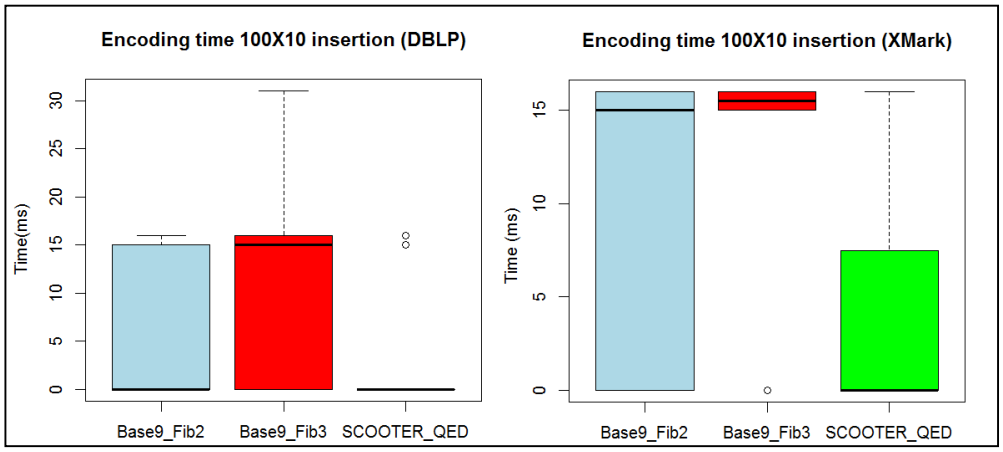


Figure 7.19 Box plot of the encoding time distribution after 100X10 insertions (DBLP and XMark)

To analyse the results of time taken to encode a large number (5000X10) of node insertions, the Kruskal–Wallis test was applied. The p-value obtained was less than $0.001\text{x}10^{-4}$ for all the datasets, indicating there is a difference between at least two encoding methods. A pairwise comparison via Mann-Whitney always produced very low p-values ($p < 0.001\text{x}10^{-4}$), showing there is a difference between any pair of encoding methods. The box plots in Figure 7.20 show that the Fibonacci of order 3 outperforms the other encoding methods when encoding a large number of new nodes. Although QED can encode a small number of new nodes faster, it had the slowest performance when encoding a large number of inserted nodes.

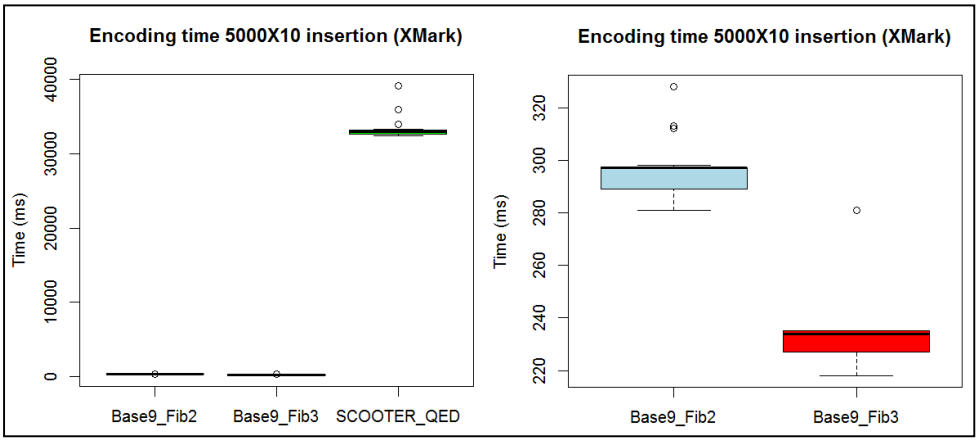


Figure 7.20 Box plot distribution of encoding time after 5000X10 insertions (XMark)

The same results were obtained for all the experimental XML datasets when inserting 5000X10 or an even larger number up to 10,000X10, as shown by the box plot charts in appendix B.2. In general, both Fibonacci encodings were about 99% faster than QED in terms of encoding new labels after large skewed

insertions. Fibonacci encoding of order 3 performed better than Fibonacci of order 2 by an average about of 20.71% in all the datasets tested. Appendix B.2 presents the percentage comparison in detail between QED and Fibonacci encodings for each dataset.

- ***Size comparison:***

Figures 7.21 and 7.22 present encoded labels size comparison of a small and a large number of nodes inserted in all the experimental XML datasets used. When a large number of nodes are inserted, the SCOOTER's new labels grow rapidly (see Section 7.3.2) and so their QED code sizes also increase rapidly. In this case, the QED codes representing SCOOTER's labels are larger than the Fibonacci codes of the Base-9 labels. Particularly in the Treebank dataset (see figure 7.22), which has the deepest XML tree that requires more separators each represented by 2 bits in QED. Whereas, in Fibonacci encoding the separators are not stored for saving more space.

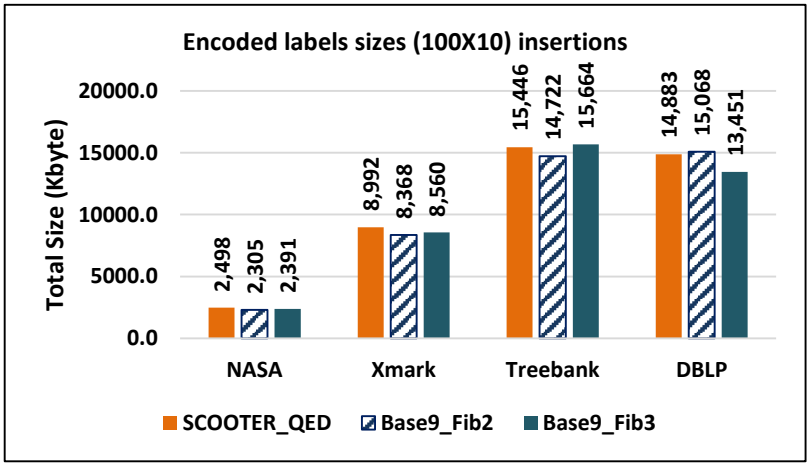


Figure 7.21 Encoded label size comparison after 100 X 10 insertion

The Kruskal–Wallis test was used to test the statistical significance of the results for all types of skewed insertion $100X10$, $5000X10$, and $10{,}000X10$ in terms of encoded label sizes. The p-value obtained was less than 0.05 in all cases for every dataset, indicating there is a difference between at least two encoding methods. The majority of the pairwise comparison via the Mann-Whitney U-test gave a p-value in the range $[0.033, 0.001\text{x}10^{-2}]$, implying there was very strong evidence ($p < 0.001$, adjusted using the Bonferroni correction) of a significant difference between the encoding methods. The three exceptional cases were:

- 5000X10 insertions in DBLP dataset with $p = 0.928 > 0.05$ between QED and Fibonacci of order 2
- 10,000X10 insertions in DBLP dataset with $p = 0.791 > 0.05$ also between QED and Fibonacci of order 2
- 10,000X10 insertions in XMark dataset with $p = 0.058 > 0.05$ between Fibonacci of order 2 and of order 3

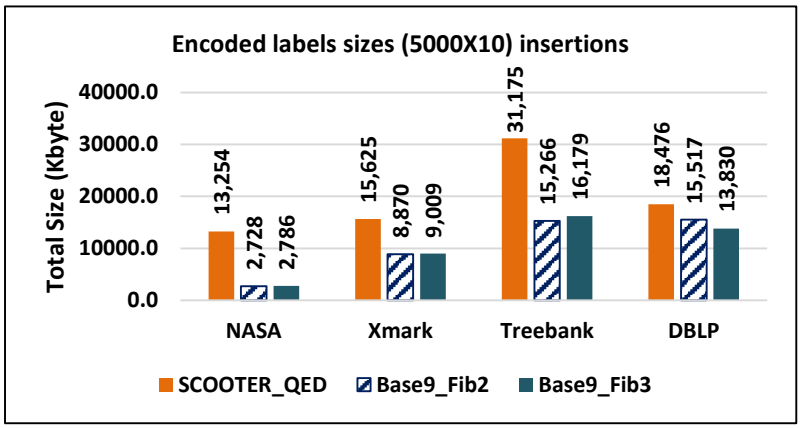


Figure 7.22 Encoded label size comparison after 5,000 X 10 insertions

The box plots for each dataset presenting the size distribution after 100X10 insertions are displayed in Figure 7.23. The results are consistent with the size of the initial label encoding illustrated earlier, where the overall size is affected by the XML tree shape of each dataset.

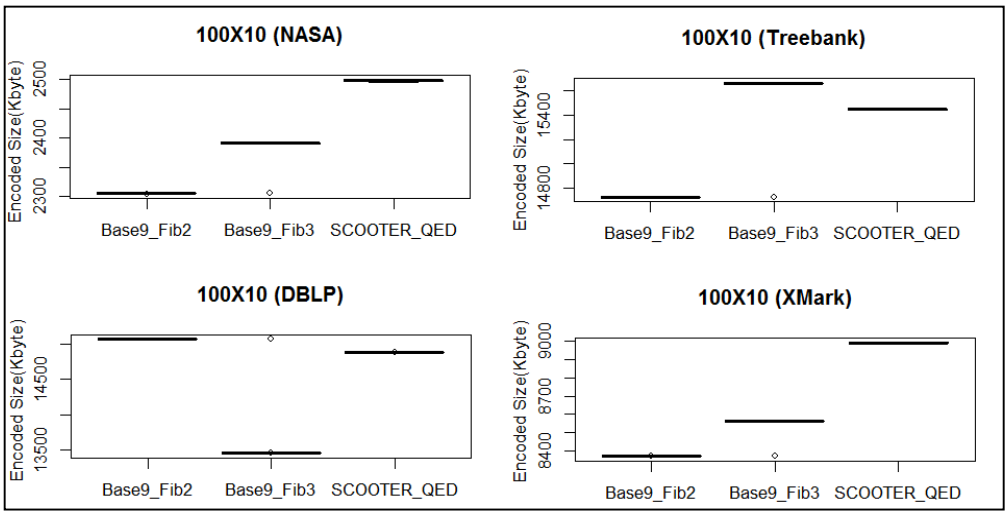


Figure 7.23 Box plot distribution of encoded labels size after 100X10 insertion

However, when dealing with a large number of insertions the Fibonacci codes of the Base-9 labels are always smaller than the QED codes representing the

SCOOTER labels for all datasets (see Figure 7.24). Appendix B.2. illustrates the results after 10,000X10 insertions, which are consistent with the results of 5,000X10 insertions presented in this section.

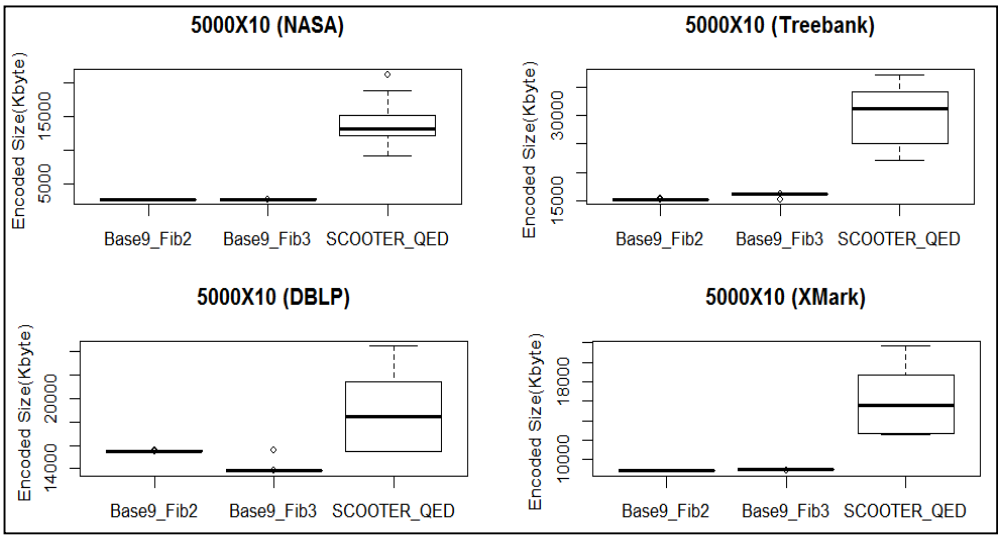


Figure 7.24 Box plot distribution of encoded labels size after 5000X10 insertion

The relative percentage change in encoded label sizes generated using Fibonacci coding and QED to represent Base-9 and SCOOTER labels, respectively, were measured as follows (MathGoodies, 2015): $((\text{Code}size_{SCOOTER} - CodeSize_{Base9})/CodeSize_{SCOOTER})$. The percentages obtained for encoded labels after large ($5{,}000X10$ and $10{,}000X10$) insertions are presented in Table 7.13.

Table 7.13 percentage difference between sizes of QED and Fibonacci codes

Insertion type	10000X10		5000X10	
Dataset	Percentage difference QED vs Fib2	Percentage difference QED vs Fib3	Percentage difference QED vs Fib2	Percentage difference QED vs Fib3
NASA	93.32%	93.42%	79.42%	78.98%
XMark	79.82%	79.64%	43.24%	42.35%
Treebank	79.44%	78.31%	51.03%	48.10%
DBLP	42.65%	49.05%	16.01%	25.15%

7.3.4.3 Conclusion

The encoding mechanisms used to store Base-9 labels and SCOOTER labels were examined in this experiment. The performance of these mechanisms in terms of encoding time and size were compared in initial labels and updated labels for all experimental XML datasets. In terms of time, the QED method applied to the SCOOTER labels was the fastest when it was used on initial nodes or when a small number of nodes was inserted. Alternatively, the QED method was the slowest when it was applied to a large number of new nodes as the SCOOTER label grew rapidly (see Section 7.3.2). The Fibonacci coding outperformed the QED encoding in generating codes after a large number of insertions, particularly Fibonacci of order 3.

Considering the sizes of the encoded labels, the tree shape of the XML dataset used affected the performance of the encoding methods. In general, Fibonacci coding has always produced smaller codes than QED. This leads to the conclusion that Fibonacci coding enables the storage of the Base-9 labels in more compressed form than QED encoding for the SCOOTER scheme. Furthermore, the Base-9 scheme provides faster encoding and consumes less storage than SCOOTER in case of large skewed insertions.

7.3.5 Relationship Determination

This experiment was designed to measure how quickly the main structural relationships can be established directly from the labels before and after XML updates (see Section 6.5). Both schemes (the Base-9 and SCOOTER) determine structural relationships based on lexicographical comparison (see Chapters 3 and 5). Hence, this experiment investigated the effect of the compressed Base-9 labels against the SCOOTER labels on the determination process. The study also examined the influence of the scheme's decoding mechanisms on the speed of the determination process.

7.3.5.1 Analytical Strategy

Two aspects were measured when evaluating this experiment: the determination time and the decoding time (in milliseconds). In view of the randomness of the time computation, the number of runs required to obtain a statistically significant

difference between the two schemes in terms of determination, including and/or excluding the decoding process, was identified.

To test the determination of each relationship type individually, the Treebank dataset was selected (for the reasons given in Section 6.5). The time taken to determine each the relationship between any two labels was computed before and after insertion. The determination time for 100 runs (after excluding the first 5 runs) over 200,000 initial pairs of labels was recorded. Similarly, the determination time of 100 runs over 400,000 pairs of labels after insertion was also computed. The relationships tested were: parent/child (P/C), ancestor/descendent (A/C), sibling, lower common ancestor (LCA), and document order (DO).

To study the effect of the decoding process on the determination, the measurement of the decoding time was included as a part of the determination process. The determination time here represents finding all relationships between any 200,000 pairs of encoded labels. All tests were performed on the Base-9 and SCOOTER schemes. The time taken for 100 runs (after excluding the first 5 as usual) were recorded and then statistically analysed by the Mann-Whitney U-test. This part of the experiment was conducted on all the experimental datasets to achieve more reliable results.

7.3.5.2 Analysis of the Results

- **Individual relationship**

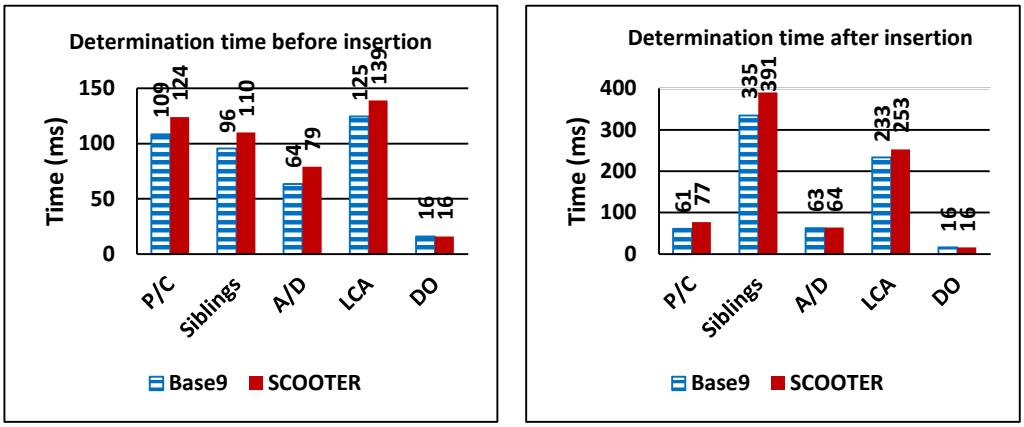


Figure 7.25 determination time comparison before and after insertion

The determination time comparison between the Base-9 and SCOOTER schemes before and after insertion is presented graphically in figure 7.25. These data were collected by running the test on the Treebank dataset. For each relationship determination over the initial labels, this figure shows that Base-9 outperformed SCOOTER in establishing every relationship except the document order relationship, where the difference found in this instance was insignificant. It seemed that after insertion, the difference in determination time between the two schemes became smaller (see figure 7.26).

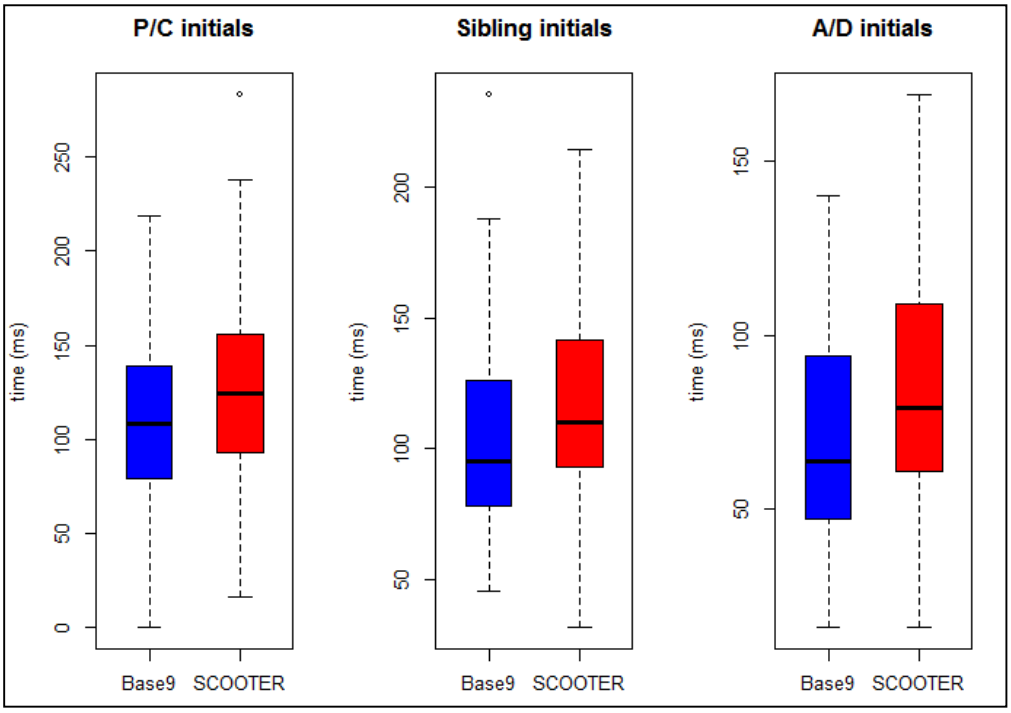


Figure 7.26 Box plot distribution of determination time over initial labels

Further tests were carried out using the Mann-Whitney U-test to obtain the statistical significance of the results. For the initial labels, the tests gave a p-value less than the significance level of 0.05 for the parent/child ($p = 0.01$), sibling ($p = 0.015$), and ancestor/descendant ($p = 0.042$) relationships. The null hypothesis was retained for the other two types of relationship: LCA ($p = 0.092$) and document order ($p = 0.793$).This indicates that there is a difference between the two schemes when determining parent/child, sibling, and ancestor/descendant relationships over the initial labels. The effect sizes for these three type of relationship were measured and the results were always $\hat{A}_{12}\ (SCOOTER, Base9) \approx 0.60$, which implies that 60% of the time Base-9 determines these relationships faster than SCOOTER. The box plots in figure 7.26 confirm this observation.

When considering the determination time after insertion, the p-value obtained via the Mann-Whitney U-test was in the range $[0.01\text{x}10^{-3}, 0.043] < 0.05$ for all the relationships except for the document order ($p = 0.948$). To investigate the difference between the two schemes in the four relationships for which the null hypothesis was rejected, the effect sizes were measured. The $\hat{A}_{12}\ (SCOOTER, Base9)$ values found were between 0.60 and 0.72. This indicates that at least 60% of the time Base-9 outperformed SCOOTER in determining

these four relationships. Figure 7.27 illustrates the box plot distribution of the determination time for parent/child, sibling, ancestor/descendant, and LCA relationships. Statistical descriptions of the details of this experiment data are presented in appendix B.4.

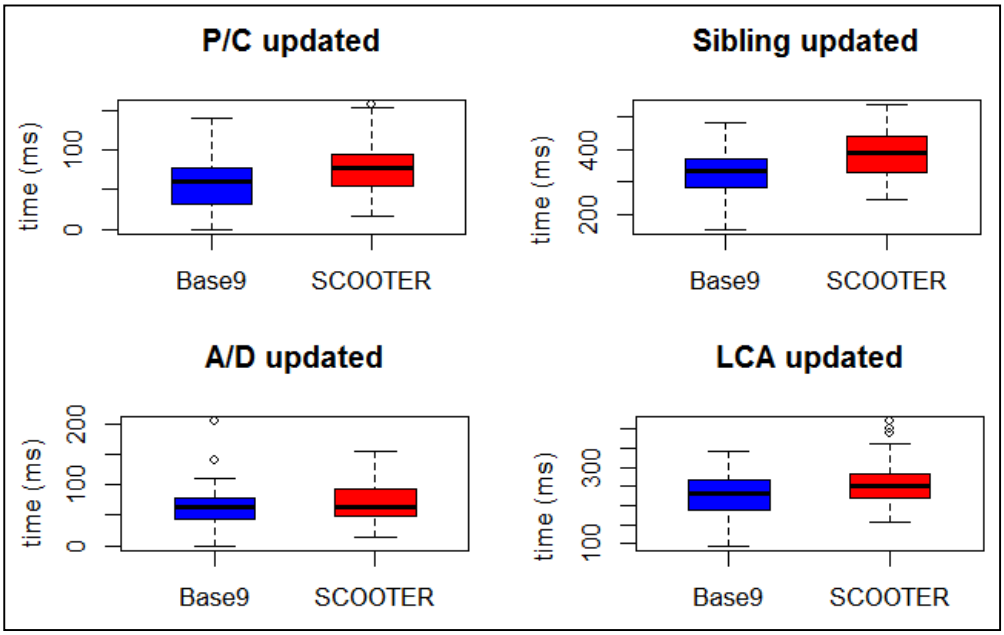


Figure 7.27 Box plot distribution of determination time over updated labels

- **All relationships with decoding**

This experiment was run on all the experimental datasets. The determination time in this part of the experiment represents the time taken to determine all five relationships combined (parent/child, sibling, ancestor/descendant, LCA, and document order) between any 200,000 pairs of labels before and after insertion. At first, each pair of encoded labels was selected and then decoded into XML prefix-based labels (Base-9 and SCOOTER separately) before establishing the relationships. The Fibonacci codes of the Base-9 labels were decoded using Fibonacci decoding, whilst the QED codes were decoded into the SCOOTER labels. Figures 7.28 and 7.29Figure 7.29 show the decoding time comparison between the schemes in terms of decoding the initial and updated labels, respectively. In both scenarios, the Fibonacci decoding process was faster than the QED decoding.

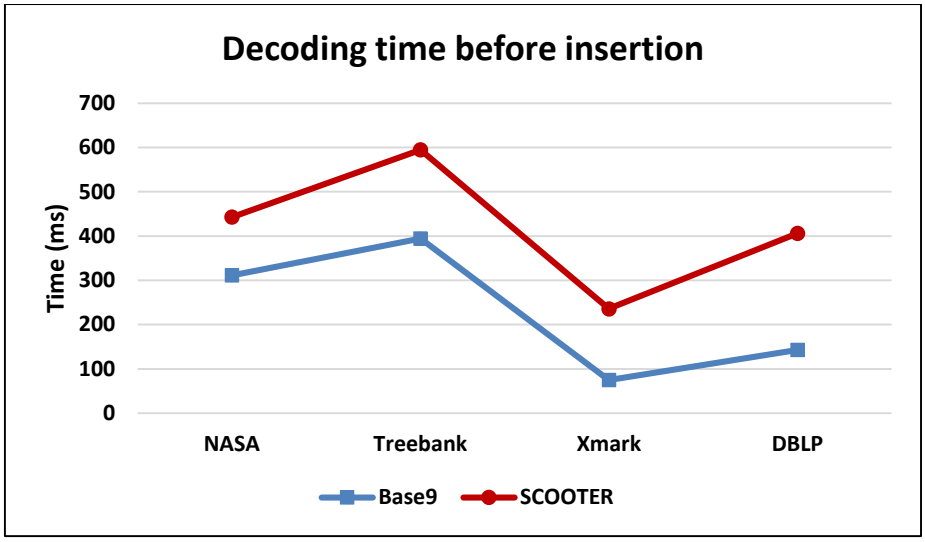


Figure 7.28 Decoding time comparison of initial labels

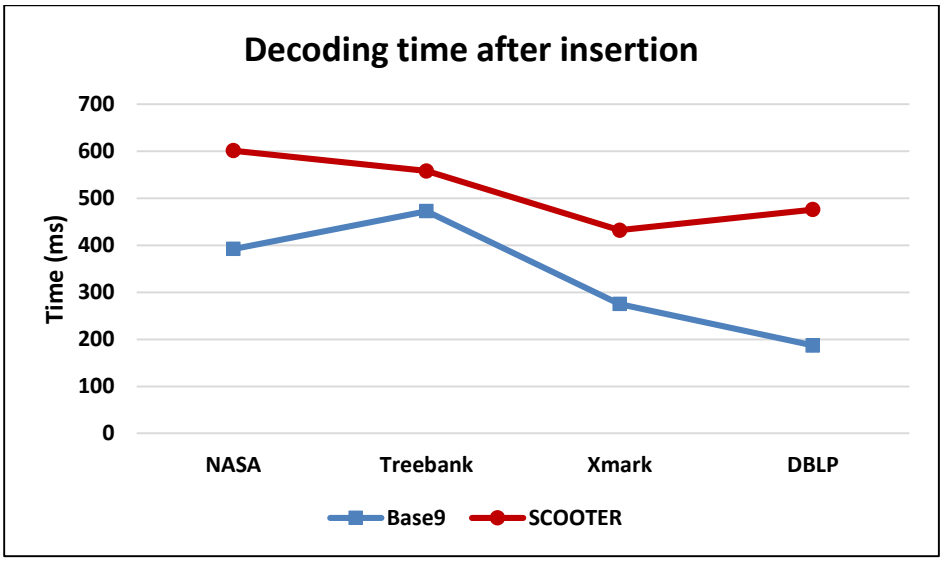


Figure 7.29 Decoding time comparison of updated labels

Considering the determination time only, Figures 7.30 and 7.31 illustrate the comparison between the Base-9 and SCOOTER schemes before and after insertion. Although both schemes apply lexicographical comparison when determining the structural relationships, the Base-9 compressed labels gave a better performance than the SCOOTER labels. The Mann-Whitney U-test provided p-values as low as $0.01\text{x}10^{-6}$ for all the experimental datasets, confirming there was a difference between the determination time between the two schemes.

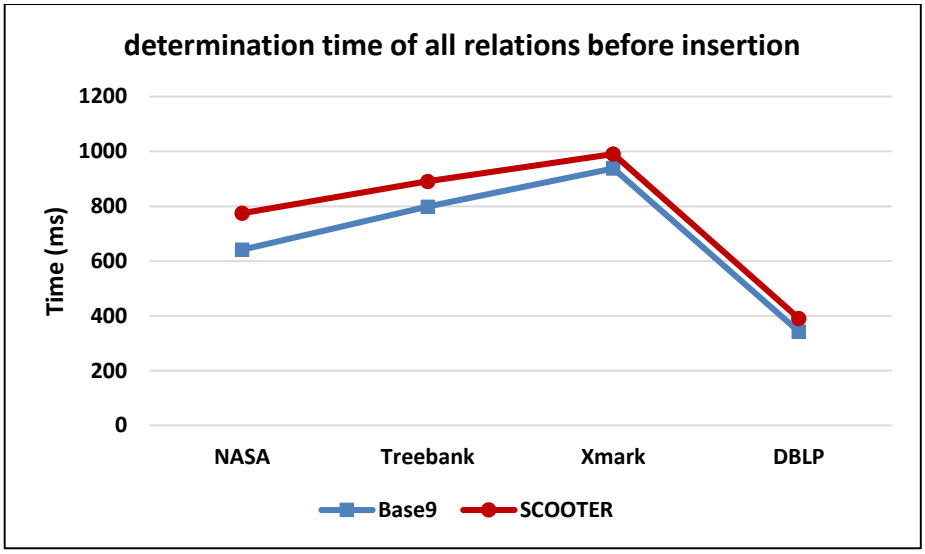


Figure 7.30 Determination time (all relations) comparison on the initial labels

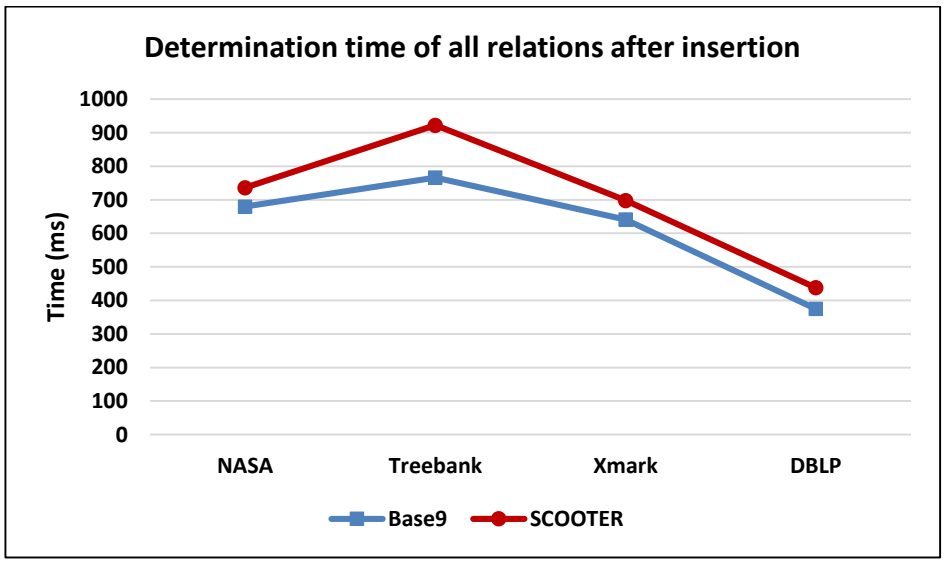


Figure 7.31 Determination time (all relations) comparison on updated labels

The effect sizes for all datasets were computed and the results were always $\hat{A}_{12}\ (SCOOTER, Base9) \geq 0.95$. This implied that 95% (or more) of the time, Base-9 determines relationships faster than SCOOTER. The box plot distribution for the NASA dataset is shown in figure 7.32, verifying the results obtained. As the results were consistent for all the datasets; appendix B.4 shows box plot distributions for the other experimental datasets.

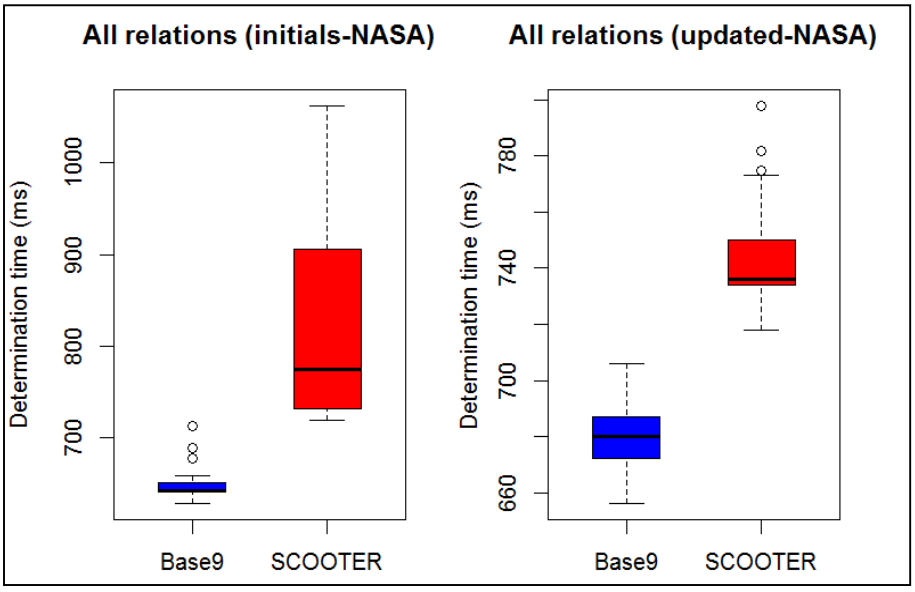


Figure 7.32 Box plot distribution of determination time (all relations) before and after insertion

Similar results were found when combining the decoding time and determination time. See figure 7.33 for initial encoded labels and figure 7.34Figure 7.34 for encoded labels after insertion. The compressed Base-9 labels reduced the determination performance (including decoding process) by at least 16.32%. Table 7.14 displays the percentage decrease $((Time_{SCOOTER} - Time_{Base9})/Time_{SCOOTER})$ of the median time taken for decoding and then determining relationships when using Base-9 in preference to SCOOTER. The median values of 100 runs before and after insertions on both schemes were used due to the non-normal distribution of the collected times.

Table 7.14 Percentage decrease of median time (decoding and determination)

Datasets	NASA	Treebank	XMark	DBLP
Before insertion	21.72%	19.69%	17.44%	39.02%
After insertion	19.82%	16.32%	18.94%	38.51%

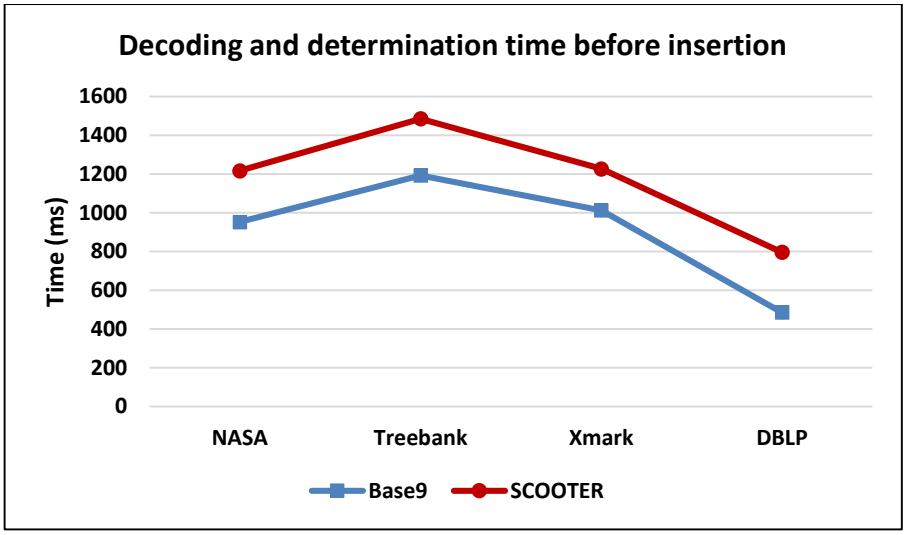


Figure 7.33 Decoding and determining time comparison on initial labels

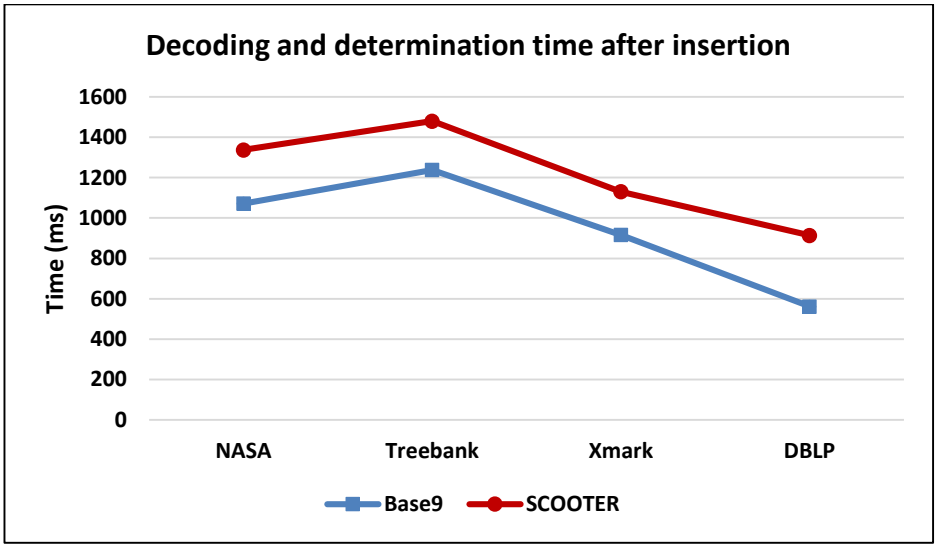


Figure 7.34 Decoding and determining time comparison on updated labels

To check the statistical significance of the results, the Mann-Whitney U-test was carried out and the p-value was found as $0.01\text{x}10^{-6}$ for all datasets. All the effect sizes measured indicate that at all times Base-9 was faster than SCOOTER in terms of decoding and then determining the structural relationships. The comparison via a box plot of the distribution of decoding and determination time for the NASA dataset is presented in figure 7.35Figure 7.35. The box plots for the other datasets can be found in appendix B.4.

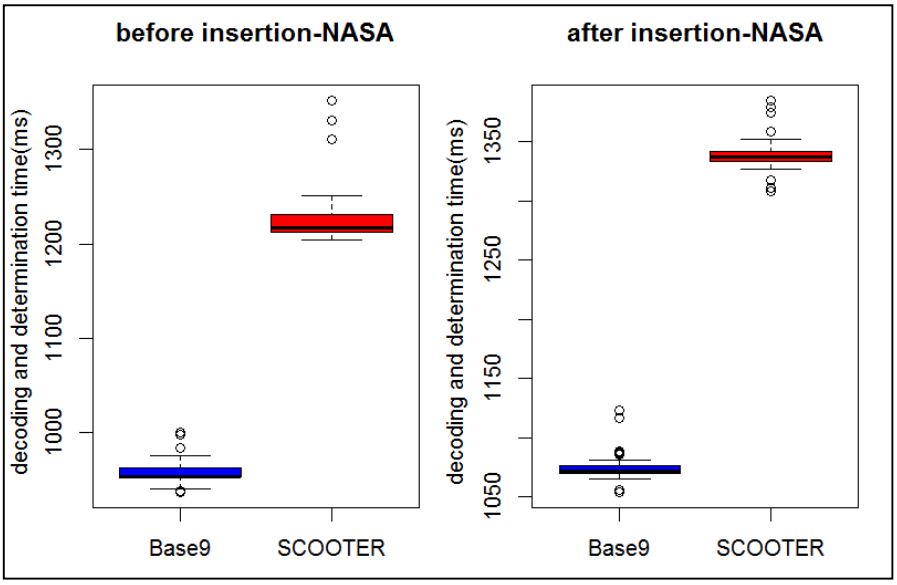


Figure 7.35 Box plot distribution of decoding and determination time

7.3.5.3 Conclusion

The ability to determine the five main relationships (parent/child, sibling, ancestor/descendant, LCA, and document order) was examined in this experiment. When establishing each relationship individually, the compressed Base-9 labels generally sped up the determination process in comparison to SCOOTER both before and after insertion. For the document order (DO), there was no apparent difference between the two schemes.

With regard to the decoding process as a part of the determination time, the Fibonacci decoding proved faster than QED by an average of 47.7% before insertion and 37.28% after insertion (considering the four datasets used). The statistical significance calculation performed on the results confirmed that Base-9 always performed faster than SCOOTER, both before and after insertion, in terms of decoding and determining all relationships between any pair of labels.

7.3.6 Query Performance

As discussed in the previous chapter, this experiment assesses the query response time on the XML-labelled XMark dataset before and after updating. Four main types of XPath queries were used for the reasons stated in Section 6.5.1.

7.3.6.1 Analytical Strategy

The only measurement relevant to this experiment is the query response time (in milliseconds). Taking into account the cache memory and any complex interactions that might occur in terms of computer housekeeping tasks at the time the experiment was executed, each query was separately run for 100 times (after once again excluding the first five runs) for each scheme in order to achieve statistically significant results. The Mann-Whitney U-test was used to gain a statistical analysis of the results obtained for the XPath queries. In addition, the medians of the time taken for 100 runs were compared graphically for the two schemes both before and after insertion.

7.3.6.2 Analysis of the Results

- **Before insertion**

Figure 7.36 illustrates the response times of the XPath queries tested using the Base-9 and SCOOTER schemes over the initial labels of the XMark dataset. This figure shows that Base-9 performs slightly better than SCOOTER in terms of querying time. With the exception of Query1, representing the parent/child relationship (i.e., finding all American (parent) items (child)), both schemes take the same length of time to return the query results. This observation was checked statistically as the Mann-Whitney U-test was applied to each query separately, which obtained p-values as low as 0.01x10^{-5} for queries 2, 3, and 4 but were greater than the significance level 0.05 for query 1 ($p = 0.763$). The medians of both schemes were the same for query 1 (=32 milliseconds) and query 4 (=94 milliseconds).

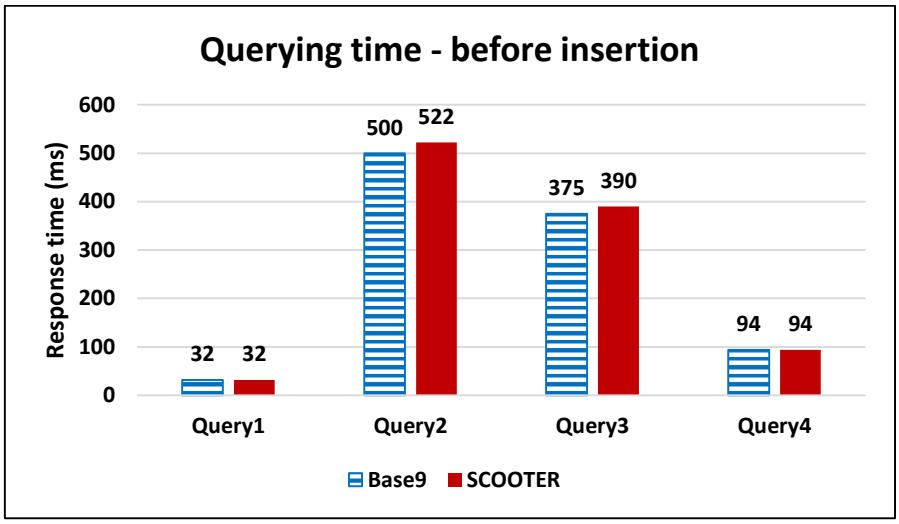


Figure 7.36 Query performance comparison over initial labels

The affect sizes $\hat{A}_{12}$ $(SCOOTER, Base9)$ for queries 2, 3, and 4 were computed, the results for which were 0.82, 0.74, and 0.66, respectively. This confirms the results comparison as it appears in figure 7.36; Base-9 outperformed SCOOTER in returning answers to queries 2, 3, and 4, representing ancestor/descendant, sibling and document order, respectively. The box plot distribution of query 2 response times is presented in figure 7.37. Similar box plots for queries 3 and 4 are shown in appendix B.5.

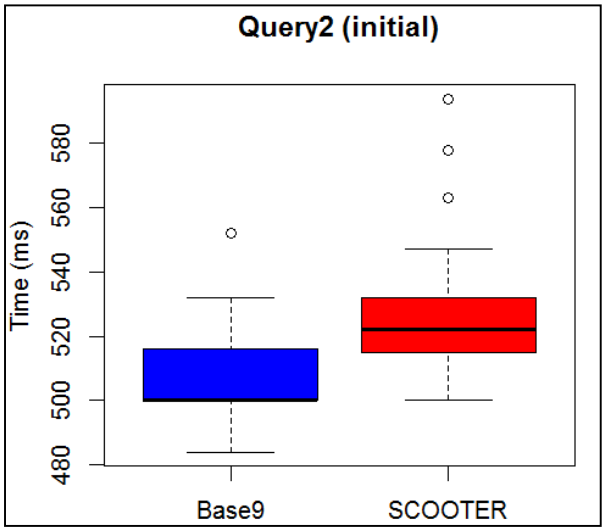


Figure 7.37 Box plot distribution of query 2 response time

- **After insertion**

Skewed insertion (1,000 X 10) was used to add 10,000 nodes to the XMark dataset in order to test the query performance when XML is updated. Figure 7.38 shows the comparison between the two schemes in terms of query response time after insertion. It can be seen from this figure that Base-9 gave a slightly faster response time than SCOOTER for all queries.

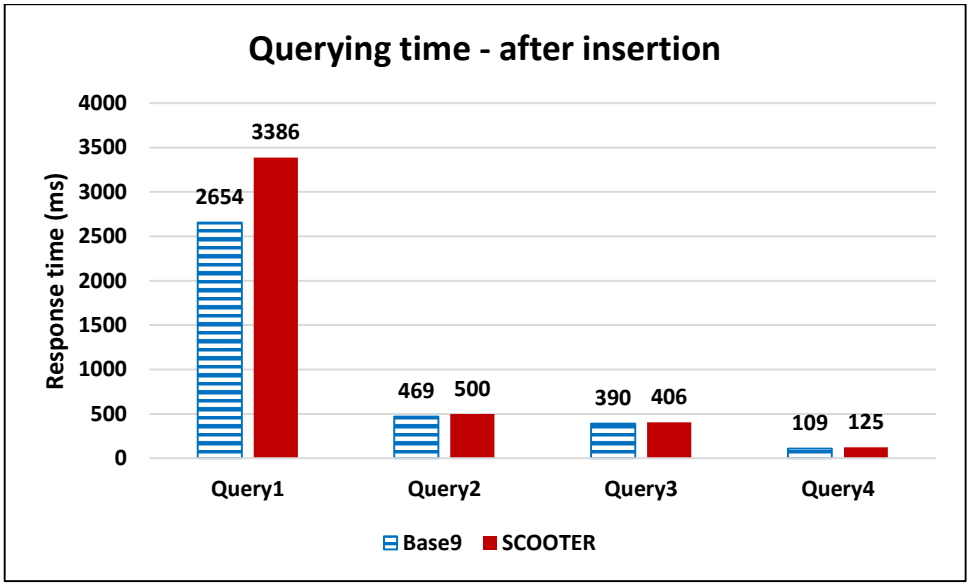


Figure 7.38 Query performance comparison after insertion

Using the Mann-Whitney U-test, the p-value obtained for all queries were $0.01\text{x}10^{-6}$. Therefore, there is a difference in query response time between the Base-9 and SCOOTER schemes. The affect sizes $\hat{A}_{12}$ $(SCOOTER, Base9)$ found were 0.99, 0.96, 0.83, and 0.84 for the queries 1, 2, 3, and 4, respectively. This shows that for over all the queries tested, Base-9 always showed a better performance than SCOOTER in terms of response time. The box plot distribution of response time for query 1 after insertion is illustrated in figure 7.39 and in appendix B.5 for all other queries (as their results are very similar to those of query 1).

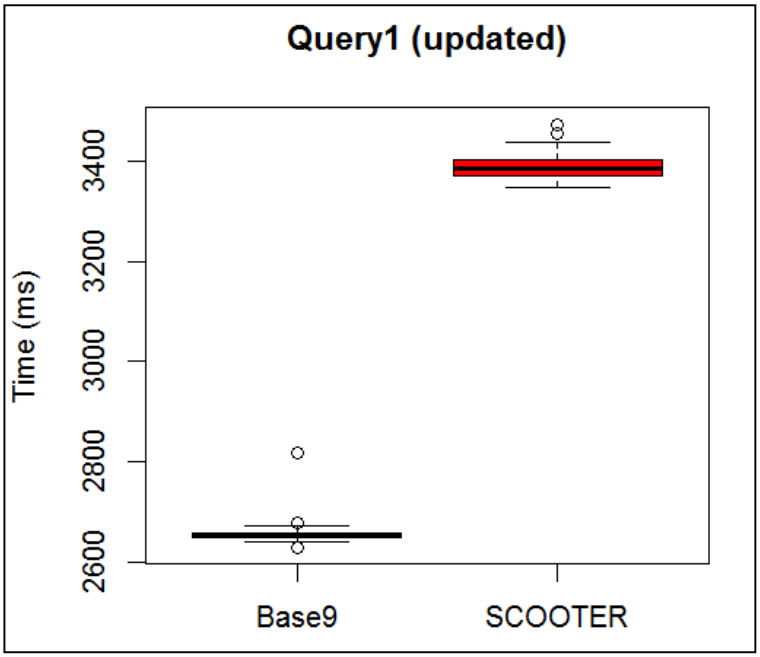


Figure 7.39 Box plot distribution of response time for query 1 after insertion

7.3.6.3 Conclusion

This experiment compared Base-9 with SCOOTER in terms of querying time. The results showed that Base-9 always returns queries concerned with ancestor/descendants, sibling, and document order relationships faster than SCOOTER, both before and after insertion. For queries dealing with parent/child relationships, both schemes behaved similarly for the initial documents but Base-9 became faster than SCOOTER after the XML document had been updated.

Further discussion on the Base-9 scheme's performance evaluation is presented the next chapter.

7.4 Experimental Results of XML Label Compression

This experiment was designed to examine the compression of XML labels using the prefix-encoding methods presented in Chapter 4: Fibonacci coding of order 2 (Fib2) and order 3 (Fib3), Lucas coding, Elias-delta (ED) coding, and Elias Fibonacci coding of order 2 (EF2) and order 3 (EF3). The performances of these prefix-encoding techniques were tested in terms of encoding time, code size, and decoding time (see Section 6.6). Each encoding method was applied to the Dewey order labelling scheme (Tatarinov et al., 2002) and the SCOOTER scheme (O'Connor and Roantree, 2012) separately. In addition to the prefix

encodings tested here, the comparison of results includes the original encoding methods of these two schemes (UTF-8 for Dewey labels and QED for SCOOTER labels; see Chapter 4). The experiments were conducted on the three real-life datasets (NASA, Treebank, and DBLP) described in Section 6.5.1 to study the effect of the XML tree shape over the results obtained.

The effect of the XML dataset size on the compression process was also examined in this experiment. This was achieved by reducing the Treebank and DBLP files to 23MB (equivalent to the NASA file size) whilst their XML tree features were preserved as described in Table 6.3. The encoding and decoding experiments were then repeated over these re-sized datasets and the results were compared with the original ones.

7.4.1 Analytical Strategy

The compression methods experiment was applied to study which encoding method generates the smallest code, and which one(s) process the encoding and/or decoding the fastest. Therefore, two main factors were considered in evaluating the results of this experiment: the execution time (in milliseconds) and the code size (in Kbytes).

For each encoding method, the codes generated for Dewey or SCOOTER labels were found to be fixed values, regardless of the number of times each method was executed, thus the code sizes have been presented graphically.

When the encoding and/or decoding time are considered, it is essential to take into account random effects on the time computed (Li et al., 2005c) and, consequently, specify how many runs are sufficient to obtain statistically significant results. In order to enable the analysis of a statistical hypothesis with minimal statistical power (Dybå et al., 2006) the number of runs had to be at least 10 (Ali et al., 2010) (Wegener et al., 2001). However, it is recommended (Arcuri and Briand, 2014) to use at least 30 runs to reach a more accurate statistical result. To determine a sufficient number of runs required to gain a statistically significant result, the statistical analysis was separately applied on Dewey labels for the NASA dataset with 20, 50, 100, and 150 runs. The analysis included a comparison between the six prefix encodings and UTF-8, each being individually implemented on the Dewey labels for 20, 50, 100 and 150 times. As the medians of the times taken were the same for each encoding method when executed for

20 or more runs, 50 runs (after excluding the first five runs) was selected as an adequate number of runs to evaluate the encoding and decoding time. Because of the non-normal distribution of the time, and because there were more than two encoding methods to be compared, the Kruskal-Wallis test was used to study the statistical significance of the results.

7.4.2 Analysis of the Results

- **Encoding time**

Figures 7.40 and 7.41Figure 7.41 show the median encoding time comparison for the Dewey and SCOOTER labels respectively. As can be seen from these figures, the results were influenced by the different XML tree shapes of the XML datasets tested. For both labelling schemes, the encoding time for the NASA dataset was the fastest as its size is the smallest of all the datasets tested. The encoding time for Dewey labels was the slowest for the Treebank dataset, which has the deepest XML tree structure; this is because more components and separators exist within Treebank's labels. Similar results were achieved on SCOOTER labels with the exception of Lucas encoding, Fibonacci of order 2 (Fib2) and of order 3 (Fib3) encodings, which took more time for the DBLP dataset. SCOOTER labels were computed based on the node child count (see Chapter 3) and so the more children per node that exist (i.e., wider XML tree as in DBLP dataset) the larger the self-label value is. Thus, it seems that the Fibonacci and Lucas encodings were more dependent on self-label sizes than the number of components. Overall, for SCOOTER labels the original QED method achieved the fastest encoding time of all six prefix-encoding methods.

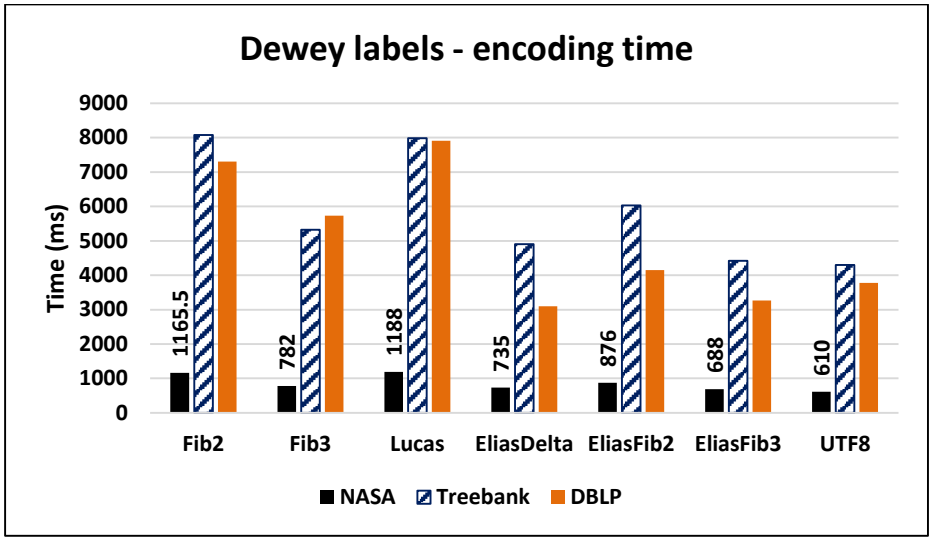


Figure 7.40 Median encoding time comparison for Dewey labels

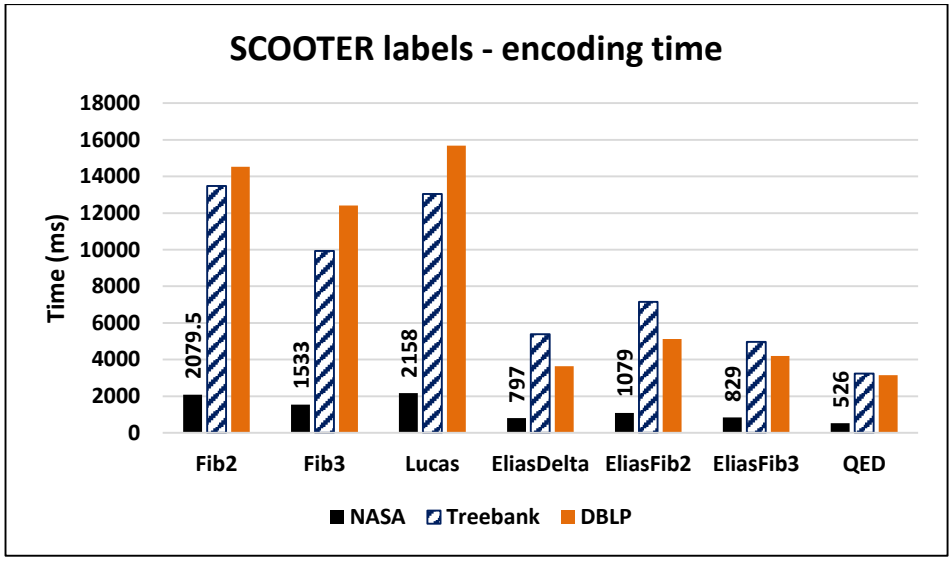


Figure 7.41 Median encoding time comparison for SCOOTER labels

The Kruskal-Wallis test was carried out on the encoding time for each dataset, and the p-value obtained was $p < 0.001$, suggesting there is a difference between at least two encoding methods. Thus, pairwise comparisons via the Manny-Whitney test were performed and the results confirmed that there was very strong evidence (with $p < 0.001$, adjusted using the Bonferroni correction) of a difference between most of the methods. There was no evidence of a difference between Fibonacci coding of order 2 and Lucas coding, where both gave maximal encoding times. Moreover, there was no evidence of any

difference between Elias-delta and Elias-Fibonacci 3 coding. For both schemes, the overall encoding time for the newly implemented Elias-Fibonacci of order 3 had the smallest median value in comparison to the other prefix-encoding methods. The encoding time for the original methods (UTF8 and QED) of the two schemes required the least time of all the prefix encoding methods. Figure 7.42 shows the box plot distribution of the encoding time of Dewey labels for the NASA dataset. Similar box plots were found for the other datasets, so these are only reported in appendix B.6. The statistical measurements obtained in this experiment can also be found in appendix B.6.

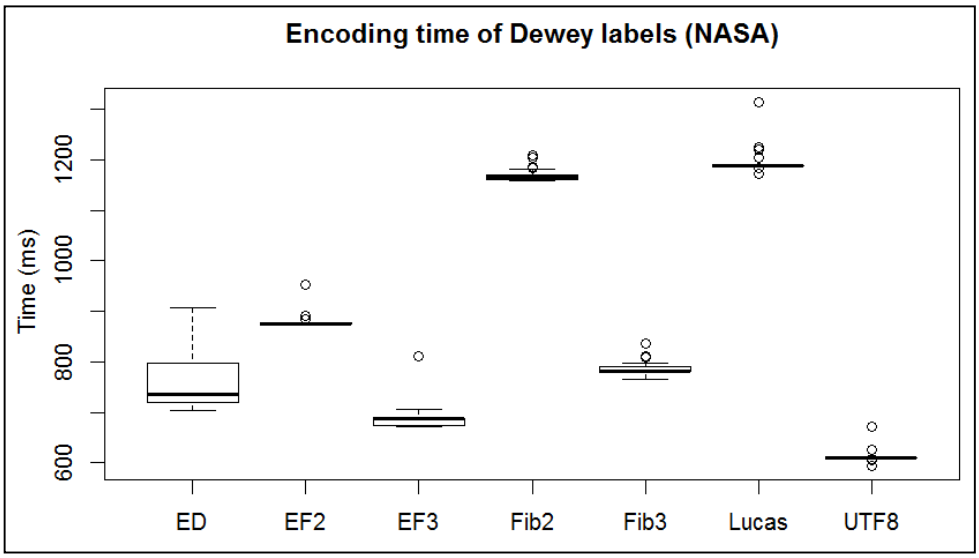


Figure 7.42 Box plot distribution of encoding times of Dewey labels for NASA

- **Decoding time**

Figures 7.43 and 7.44 show the median decoding time comparison for Dewey and SCOOTER labels. As with the encoding, the results here were also affected by the XML tree shape. For the Dewey labelling schemes, the results were similar to those of the encoding process (Figure 7.40), where the NASA dataset was decoded the fastest and Treebank the slowest. Unlike the encoding process, the Lucas and Fibonacci methods gave a better performance than the other methods for the SCOOTER scheme. In general, the UTF-8 and Fibonacci of order 2 achieved the fastest decoding times for Dewey labelling scheme. The Fibonacci of order 2 also decoded the SCOOTER labels faster than the other decoding methods.

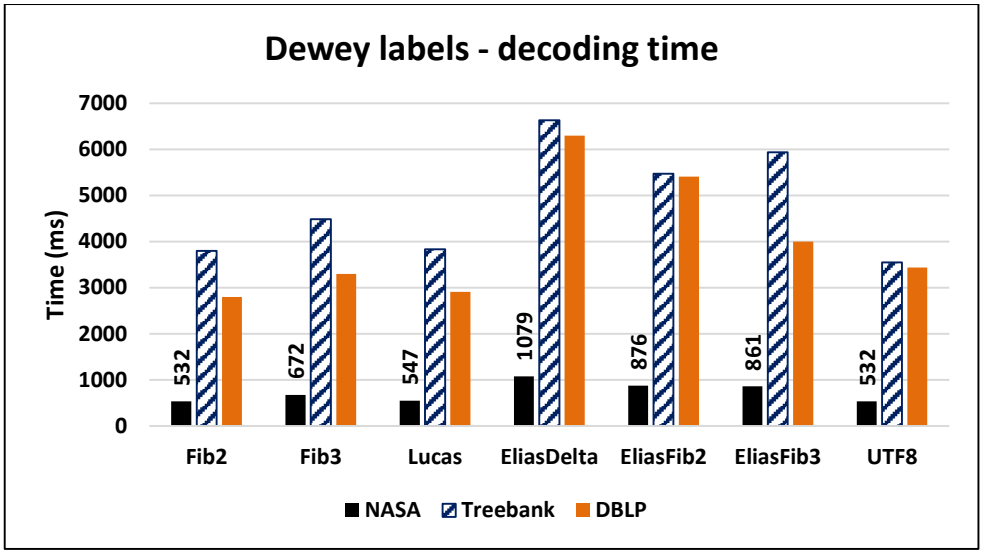


Figure 7.43 Median decoding time comparison for Dewey labels

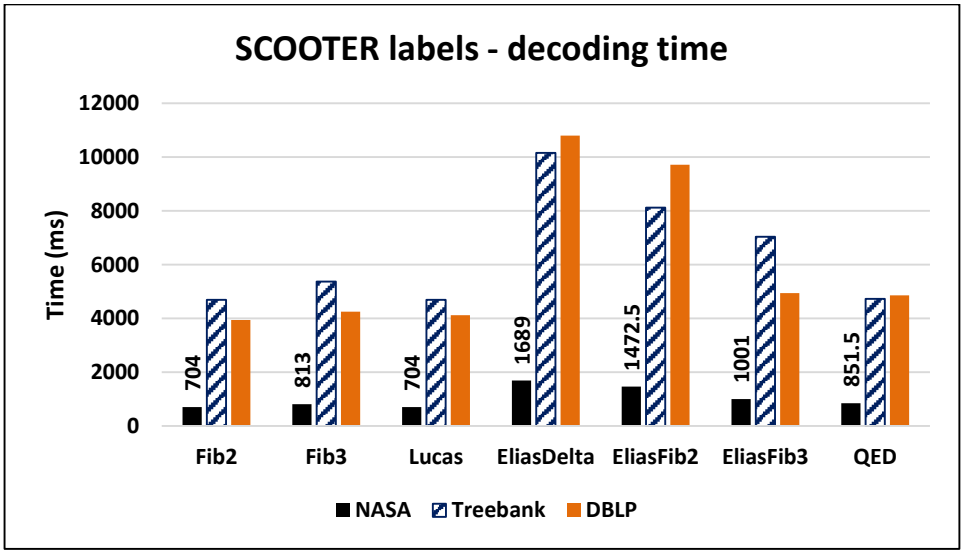


Figure 7.44 Median decoding time for SCOOTER labels

The Kruskal-Wallis test was carried out on decoding time for all datasets individually, and as the p-value obtained was $p < 0.001$ pairwise comparisons via the Manny-Whitney test were performed; the results showed that there was very strong evidence (with $p < 0.001$, adjusted using the Bonferroni correction) of a difference between most of the methods. There was no evidence of any difference between Fibonacci (of order 2), Fibonacci (of order 3) and Lucas coding. Generally, for both schemes, decoding Fibonacci of order 2 was the fastest in comparison to all the other encoding methods tested here. The Elias-

delta was the slowest when decoding XML prefix-based labels. Figure 7.45 presents the box plot distribution of the decoding times of Dewey labels for the NASA dataset. The box plot distribution of decoding times for both schemes across all datasets are similar to figure 7.45, and as such are given in appendix B.6. The statistical measurements obtained in this experiment are also reported in appendix B.6.

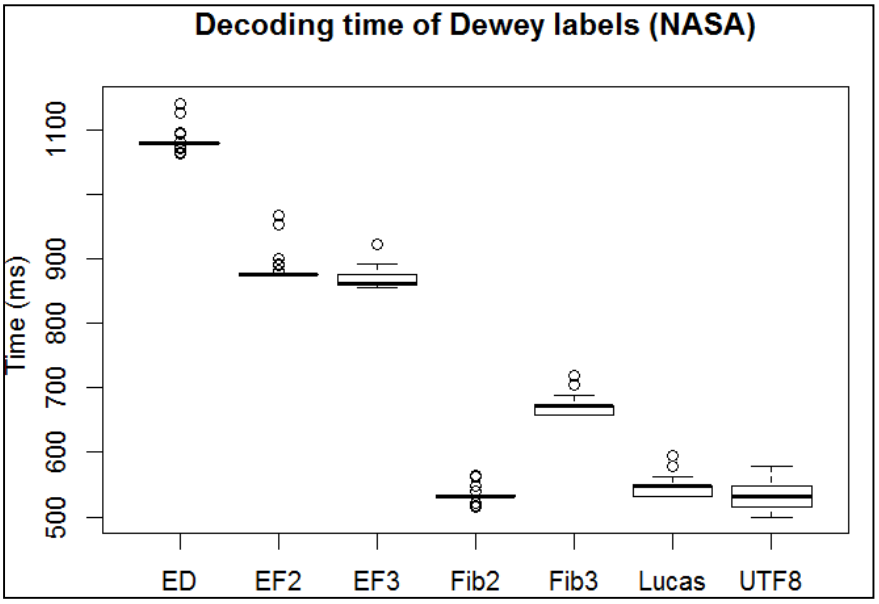


Figure 7.45 Box plot distribution of decoding times of Dewey labels for NASA

- **Code size**

The total code size (in Kbytes) of all the Dewey and SCOOTER labels within each dataset was computed separately. Figures 7.46 and 7.47Figure 7.47 illustrate the results for all encoding methods. For the Dewey labels, all the prefix-encoding methods applied generated smaller code than the original UTF-8 encoding. However, for the SCOOTER labels, the original QED encoding produced the smallest code of all prefix encoding approaches.

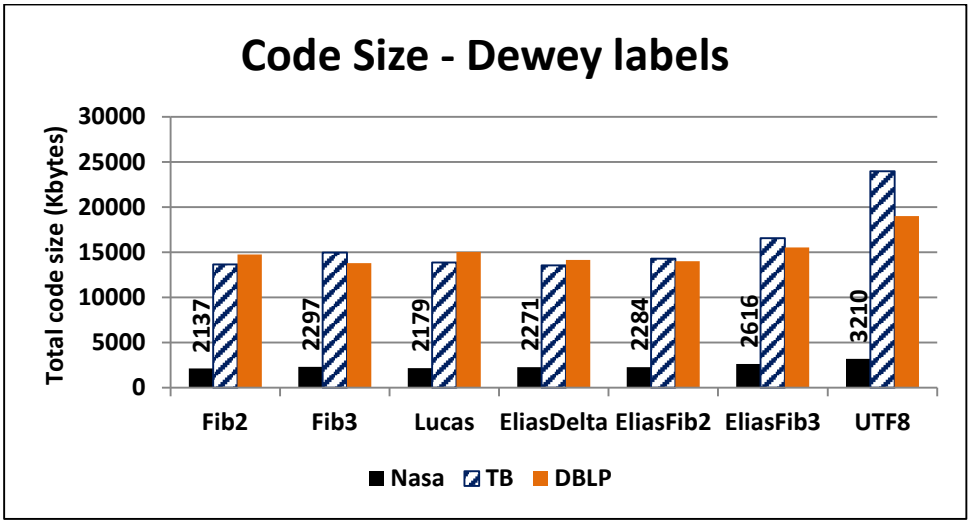


Figure 7.46 Code size comparison for Dewey labels

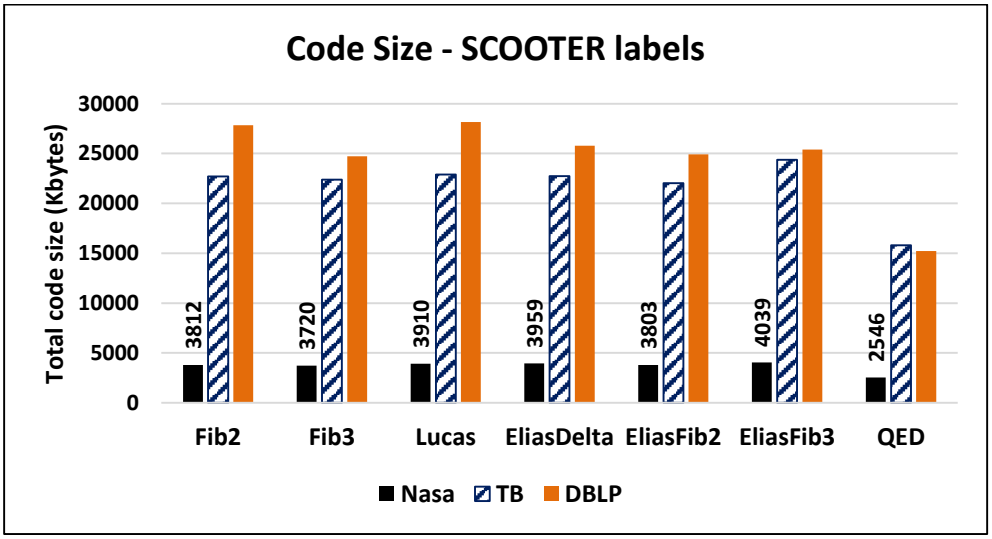


Figure 7.47 Code size comparison for SCOOTER labels

The differences between the total code sizes obtained using the prefix encodings were small (see figures 7.46 and 7.47). The size of self-label values in a label set has an impact on the size of the compressed code. For instance, label sets with shorter self-labels, such as the Dewey labels for the NASA and Treebank datasets using Fibonacci order 2, generated the smallest code. As self-label values get larger (e.g., in SCOOTER labels), Fibonacci of order 3 produced the most compressed code. This observation agrees with the results of the Base-9 label encoding described earlier (see Section 7.3.4). In general, Fibonacci coding

generates the most compressed code in comparison to the other prefix-encoding methods applied. For smaller self-label values, Fibonacci of order 2 is better, whereas Fibonacci of order 3 is recommended for larger self-label values.

- **Database Size**

The prefix-encoding methods were also tested over the three datasets of the same size (i.e., 23 MB) where their XML tree properties were preserved as described in Section 6.5.1. The results of encoding/decoding time and code size were consistent with the originals (discussed earlier). This implies that the XML tree's shape (depth and breadth) influences the encoding/decoding time and code size, but XML document size is not of consequence. A comparison of results is presented graphically in appendix B.6.

7.4.3 Conclusion

Various prefix coding methods were applied for the first time to compress XML labels. The compression process was conducted on three real XML benchmark datasets that vary in their XML tree properties. The results show that the structure of the XML tree representation of a dataset has an impact on the performance of the compression methods, but the XML document size does not. Among the prefix-encoding methods studied the newly implemented Elias-Fibonacci of order 3 achieved the fastest encoding time on average, whilst Fibonacci of order 2 had the best decoding time. In practice, the decoding process is usually performed more often than encoding. Therefore, for faster XML query processing, Fibonacci coding of order 2 is preferable to other encoding methods. In terms of size, Fibonacci of order 3 produced the most compressed codes for larger numbers and Fibonacci of order 2 for small numbers. Consequently, Fibonacci coding is recommended for encoding XML labels because it generates smaller code and produces faster decoding in comparison to the other encoding methods tested here.

The results of this experiment were published in WEBIST 2016 conference (Al-Zadjali and North, 2016).

7.5 Conclusion

To conclude the experimental outcomes: the SCOOTER scheme provided initial labels faster than Base-9, but Base-9 produced more compressed labels.

Uniform and skewed insertions were tested on both schemes when adding both small and large numbers of nodes to each XML experimental dataset. In general, the insertion time was almost the same for the two schemes. As with the initialisation, Base-9 always generated more compact labels than SCOOTER especially when a large number of nodes were inserted. If insertion occurred at the same position as a deleted node, Base-9 guaranteed re-use of at least 98.7% of the deleted labels while SCOOTER re-used only the smallest quaternary labels.

The encoding methods used to store Base-9 labels (i.e., Fibonacci encoding) and SCOOTER labels (QED encoding) were examined in this chapter. The behaviour of these two encoding techniques was consistent in the label encoding experiment and XML label compression experiment in terms of encoding and decoding times over the initial labels. After a large number of insertions, Fibonacci coding outperformed QED encoding in generating labels' codes, particularly Fibonacci of order 3. In general, Fibonacci coding produced smaller codes than QED, either by Fibonacci of order 2 (for smaller self-labels) or of order 3 (for larger self-labels) if not both depending on the XML tree shape of the dataset (as discussed in the last experiment).

The chapter also studied the ability of both schemes to determine relationship. The results suggest that the compressed Base-9 labels have, in general, speeded up the determination process in comparison to SCOOTER both before and after insertion, except when identifying the document order (DO), where two schemes behaved similarly. When considering the decoding process as a part of the determination time, Base-9 always performed faster than SCOOTER both before and after insertion as the Fibonacci decoding was faster than QED. In addition, an assessment of the four XPath queries representing the essential structural relationships was performed to evaluate the queries' response times on both schemes.

Several prefix-encoding methods were implemented to compress the Dewey and SCOOTER labels. Their performances in terms of encoding/decoding time and code sizes were evaluated. The results showed that the shape of an XML tree (but not the size) has an impact on the results of the compression methods. Among the prefix-encoding methods applied, the newly implemented Elias-Fibonacci of order 3 provided the fastest encoding time whilst the Fibonacci of

order 2 had the best decoding time. In terms of size, Fibonacci encoding gave the most compact code.

With a few exceptions, the overall results of the Base-9 scheme were better than those of SCOOTER. A further evaluation of the Base-9 scheme is discussed in the next chapter. Chapter 8 also revisits the research hypotheses and outlines the main findings and limitations, which in turn suggest ideas for future work (presented in Chapter 9).

Chapter 8: Evaluation and Further Discussion

8.1 Introduction

The experimental results were discussed in the previous chapter. This chapter presents a general evaluation of the proposed scheme in the next section. There are many factors that can be considered as 'threats' to the experiments as they may have an impact on the experimental results; these include execution time anomalies and random data (e.g., positions of the new nodes inserted). Section 8.3 describes how these threats were handled to ensure the reliability and scalability of the results. The designs and outcomes of each experiment are individually evaluated in Secion 8.4. The validation of the proposed scheme's properties as a good dynamic XML labelling scheme is given in Section 8.5. The experimental limitations and main findings are discussed in Sections 8.6 and 8.7, respectively. Finally, the chapter is concluded with Section 8.8.

8.2 The Base-9 Scheme's Overall Evaluation

This section gives an overal assessment of the new Base-9 scheme based on the research hypothesis stated in Chapter 1 and described in detail in Chapter 5:

"Providing compact XML labels based on lexicographical order using decimal strings may facilitate query performance and permit multiple insertions without causing any storage overhead. Storing such labels using a Fibonacci prefix-encoding techniques may reduce the storage capacity required and speed up the determination of structural relationships."

As mentioned in Chapter 5, the main aim of this thesis is to limit the occurrence of overflow and increase the efficiency of XML labelling in dynamic environments by focusing on the size of XML labels. The Base-9 scheme was developed to improve the performance of XML labelling by generating shorter labels whilst still allowing insertion and avoiding re-labelling. The underlying principles here are the combination of maintaining the size of the labels via their lexicographical order and then storing these labels in compressed form by using Fibonacci encoding.

To test the research hypothesis, the Base-9 scheme was implemented based on the principles outlined in Chapter 5. The experimental objectives, design and implementations were described in detail in Chapter 6. To allow an even-handed evaluation, the SCOOTER labelling scheme (O'Connor and Roantree, 2012) was also implemented as it contributed to the development of the proposed scheme. The SCOOTER labelling scheme was recently described as the most compact dynamic labelling scheme in controlling the growth of label size when XML is updated (Chiew et al., 2014a). To assess the functionality of the proposed scheme, six experiments were designed and implemented to examine whether the research intentions were achieved and the scheme fulfilled the criteria required to be classified as a good dynamic XML labelling scheme (see Chapters 3 and 5).

Based on the experimental results discussed in Chapter 7, it is clear that the research hypothesis was supported. The proposed scheme gave better performance when either a small or large number of nodes was inserted (see Section 7.3.2). Since the focus of the thesis is on the compactness of the labels, it was essential to measure the size of the intial and updated labels. The results of the average reduction percentage in the labels' sizes when Base-9 was used in preference to SCOOTER was about 30% on initial labels, at least 20% after uniform insertion, and 44% after skewed insertion (see Section 7.3). In all circumstances, the Base-9 scheme generated shorter labels than SCOOTER.

The results of both the label encoding experiment and the XML label compression experiment showed that the XML tree shape has an impact on code size. Based on the results obtained in these two experiments, it can be seen that the performance of the encoding methods was affected by the shape of the tree representing an XML dataset. When encoding initial labels for a deep narrow tree (e.g., the Treebank dataset), Fibonacci of order 2 generated the shortest code among all encoding methods tested in this study. For a wide and shallow XML tree (e.g. the DBLP dataset) Fibonacci of order 3 produced the most compact code. For other shapes of XML tree, Fibonacci of order 2 or of order 3 encoding always performed better than QED in terms of code size. After a large number of insertions, both Fibonacci encodings (particularly Fibonacci of order 3) still generated shorter codes than QED for any shape of XML tree.

Another aspect the research hypothesis considered was that of facilitating query performance and relationship determination. The results of the relationship

determination experiment indicated that, in general, the compressed Base-9 labels effectively speeded up the determination process in comparison to SCOOTER both before and after insertion, except when identifying the document order (DO) where both schemes gave similar results. The effect of the decoding process on facilitating query performance was also tested. Again, Base-9 always gave a better performance than SCOOTER both before and after insertion, as Fibonacci decoding was faster than that of QED. An evaluation of the four XPath queries representing the essential structural relationships in terms of their response times was performed (see Section 7.3.6). The results showed that Base-9 for the most part outperformed SCOOTER in returning answers to the queries both before and after insertions.

According to the experimental design (see Chapter 6) and the results obtained (see Chapter 7), it is clear that the research hypothesis was supported. The proposed scheme was tested on many real and synthetic XML datasets (NASA, Treebank, DBLP, and XMark), which vary in their structure and properties in order to obtain better analytical results and more reliable conclusions. However, the lack of a standard evaluation framework (see Chapter 6) made it challenging to provide a fair comparison with the existing XML labelling schemes. Other threats to the experiments, such as the time restrictions, may have limited the scalability of the results, as will be discussed in the next section.

8.3 Threats to the Experiments

A number of factors may affect the results of a scientific experiment (Johnson, 2002) (Hakim, 2000). Such factors are referred to as threats. For example, execution times rely strongly on complicated interactions between several products that contain the programming application's environment. This comprises memory size and configuration, the type of compiler used, and the operating system (McGeoch, 2001). Another example is when dealing with randomised methods (e.g., in handling insertions experiment), the erraticism between runs of the codes and between randomly generated instances may obscure certain results, making it difficult to derive accurate conclusions (Johnson, 2002).

To ensure the credibility and validity of the research findings, these threats were identified and controlled in each experiment. To ensure the study's outcomes were valid rather than obtained by chance, several principles were followed, as addressed below.

8.3.1 Experimental Implementation

It is important to check the correctness of the algorithm's implementation in order to determine the scalability of the results. For each experiment, before formally collecting the data a set of exploratory experiments (Shneiderman, 1976) were applied to each algorithm. This comprised experimental tests made during debugging and optimising the code by taking into account various scenarios in order to generalise the findings (Johnson, 2002). To control the experimental techniques, the algorithms were first traced manually (using the "School" XML sample – see Chapter 2). Then, each experiment's code was tested and re-tested with various instance types and sizes (e.g., different XML datasets, small and large numbers of insertions).

Due to the lack of any standard evaluation framework for XML labelling schemes (see Section 6.3), it is difficult to provide comparisons with existing schemes in order to verify the proposed scheme's reliability. Therefore, all algorithms (Base-9 and SCOOTER) were implemented in the same machine using the same test instances. When evaluating the correctness of each algorithm's implementation individually, the results had a consistent pattern for any experimental XML dataset used. This supports the reliability of the experimental designs.

8.3.2 Reproducibility

A key aspect of a scientific experimental study is the reproducibility of results (Johnson, 2002) (Saunders, 2011). This implies that when the same code is repeatedly run on the same machine using the same instances/variables/inputs, the results will remain the same (or have the same medians/averages for run-time and randomised data). For more informative results, this reproducibility also means that when the same algorithm is re-tested using a different device, distinct datasets, and even different measuring approaches if possible, then the original results must be consistent with the reproduced results in order to support the same conclusions (Johnson, 2002). This provides confidence that the original results were independent of the details of the experiment.

In this study, each experiment was tested at least three times: once before and twice after upgrading the laptop used. The results were relatively close for all the tests performed on the researcher's laptop. In terms of execution time, the medians were mostly the same for each algorithm using the same

instances/dataset. If there were any distinctions, the results differed by no more than 0.15%. Such differences might seem insignificant, but they could reflect on the scalability of the study. To ensure reliability, the statistical analysis tests (see Chapter 7) were also repeated after each reading.

To gain a better estimate of the true asymptotic running time analysis (Cornell-Tech, 2017), it is essential to understand how the computation of the running times was achieved. In this study, the measurement unit of the execution time was milliseconds, since computer speeds have increased rapidly with the development of modern computers. According to (Johnson, 2002), if a method consumes a second or less, the running time is believed to be irrelevant to any comparison as it does not provide any significant advantage. In such a case, the method should be tested over a larger number of instances to prove that the advantage may persist when instance size grows. Thus, for each experiment, the running time was computed over both small and large datasets. In the label initialisation experiment, different datasets were used that varied in size from 23 MB to 127 MB. The same datasets were also employed in all encoding/decoding experiments as well as all relationship determination experiments. For the individual relationship determination test, only the XML Treebank dataset was used because its deep recursive structure provided sufficient variety in structural relationships (see Section 6.5.2). It was tested with 200,000 initial pairs of labels and then with up to 400,000 pairs of updated labels. The insertion algorithms were examined using different types of insertions, each executed using both small and relatively large numbers of nodes: 500 to 50,000 nodes in uniform insertions and 100X10 up to 10,000X10 for skewed insertions.

For randomised methods (e.g., insertion algorithms and relationship determination processes), to compensate for the variability of the results multiple runs must be performed to gain any understanding of the predictability of the algorithm (Johnson, 2002). As a result, the handling insertion experiment was repeated on every artefact 20 times to measure the run-time performance, and 10 times to evaluate the label size. In the determination experiment, the random selection of labels was compensated for by repeating the process 100 times both before and after insertion. Similarly, for the evaluation of the other experiments multiple runs were used with each being repeated between 20 and 100 times.

It is important to implement each experiment extensively to provide high confidence that the conclusions derived are valid and do not rely in some manner

on the experimental setup. The results of each algorithm, under repeated testing, always supported the same conclusions. This observation supports the scalability of the results and consequently it can be inferred that the experimental findings of this study were reliable (Johnson, 2002).

8.3.3 Comparability

It is important to evaluate the effectiveness of the proposed scheme in comparison to the literature. Hence, it would be ideal be able to provide comparisons between the Base-9 scheme and other existing labelling schemes. Unfortunately, the lack of a standard evaluation framework (as stated in Chapter 6) makes it difficult to adopt a comprehensive comparison. To develop a comparable implementation of the state-of-the-art XML labelling scheme, algorithms of the SCOOTER scheme were studied alongside the proposed scheme. The detailed code for SCOOTER's algorithms (initialisation and insertions) are presented in (O'Connor and Roantree, 2012). All the algorithms were implemented and run in the same environment as that in which the Base-9 scheme was tested. Each experiment was performed equally under each schemes; that is, with the same number of runs and under the same conditions, e.g., hardware type, memory and background noise, laptop on charge or not, and network consumption. All applications not forming a direct part of the experiment were always closed.

The SCOOTER labelling scheme (O'Connor and Roantree, 2012) did not deal with the same test instances as the Base-9 when evaluating its performance. The authors of the SCOOTER scheme focused on evaluating it mainly in terms of performing skewed insertions. Their paper (O'Connor and Roantree, 2012) did not provide any measurement of relationship determination, query performance, or label encoding time. Furthermore, their experiments were based on 10^2 through 10^6 self-labels generated by SCOOTER's initialisation algorithm, rather than using any of the existing experimental benchmarks. Because of this, and the absence of a standard evaluation framework for XML labelling, several difficulties were encountered in ensuring an even-handed comparison between the Base-9 scheme and other schemes in the literature, including SCOOTER.

To ensure reliability, the Base-9 and SCOOTER schemes were implemented individually on the same machine. All experiments were performed for each scheme within the same computational environment. When using different XML

experimental datasets, the comparisons between results were consistent. The statistical significance of the results was sufficient to validate the findings.

8.3.4 The Need for Statistics

Statistical analysis of the experiments permits more control over the compromise between generality and accuracy of the results in order to attain reliable conclusions regarding performance (Bartz-Beielstein et al., 2010). Statistics provides an accepted, powerful mathematical framework with which to analyse experimental data. It is important to consider the true differences between the two methods, particularly for randomised algorithms in which the presence of an effect produced by the algorithm may leads to false conclusions. Statistical tests help to identify whether the data collected are sufficient to establish a difference between the performance of the methods being compared (Ali et al., 2010). They may suggest possibilities for further improvements, or even new experimental directions (Bartz-Beielstein et al., 2010). For instance, the statistical analysis of the query performance experiment on updated data suggested it would be worth re-examining query performance, taking into account the randomness of the insertion mechanism. Accordingly, the query performance experiment was repeated by testing the queries over updated labels. This was performed for 10 insertions, individually, to compensate for the variability of the results. For each query, the response time to return the same number of answers in the two schemes was computed to ensure scalability.

In Chapter 7, several statistical analyses were performed for each experiment. The results obtained supported the findings that the algorithmic behaviour was consistent for all XML datasets in both schemes. The following section illustrates the individual evaluations of the experiments based on the results discussed in the previous chapter.

8.4 Experimental Evaluation

The experimental design (see Chapter 6) and results (see Chapter 7) are evaluated in this section.

8.4.1 Label Initialisation Experiment

The label initialisation experiment was implemented successfully. As the main focus here is the compactness of the label sizes, the aim of this experiment was to generate short initial labels for XML datasets. This experiment was intended to evaluate the Base-9 scheme against SCOOTER, the results of which showed that the experiment met its objective.

This experiment examined two parameters: initialisation time and label size. In terms of time, it was expected that both schemes would behave similarly as the concept of their initialisation mechanisms are analogous, with the exception that Base-9 used decimal strings rather than quaternary strings. The statistical analysis of the results showed that there was a significance difference between the two schemes. In general, SCOOTER generated initial labels faster than Base-9. This could be due to the implementation of the algorithm because Base-9 requires more loop statements than SCOOTER's initialisation algorithm. When the algorithms were applied to relatively large XML datasets (of minimum size 23MB) the difference between the two schemes' performance was significant, by up to 5%. The significance of the results was justified by a statistical test (see Section 7.3.1). On the other hand, as expected, the Base-9 scheme outperformed the SCOOTER scheme by an approximate 30% reduction in the size of the initial labels.

The initial label size comparison by (O'Connor and Roantree, 2012) showed that the V-CDBS scheme (Li et al., 2008) and QED scheme (Li and Ling, 2005b) produced more compact labels than SCOOTER. The percentage difference in total label size generated between the schemes was not reported by (O'Connor and Roantree, 2012). Therefore, based on the literature, it is difficult to obtain comparable results between these schemes and the proposed scheme. This suggests that it is necessary to apply other schemes, such as V-CDBS and QED, in order to generalise the findings.

8.4.2 Handling Insertions Experiment

This experiment was performed to evaluate the dynamic behaviour of the proposed scheme. The results (in Chapter-7) of the experiment support the experimental objectives. Both uniform and skewed insertions were tested in all the experimental datasets. The tests covered inserting both a small and large

number of nodes to obtain general and scalable results. For the most part, both schemes required the same insertion time, but the proposed scheme constantly generated more highly compressed labels than the SCOOTER scheme. The Base-9 scheme improved the skewed insertion performance in compressing XML labels by at least 44% when small (100X10) insertions occurred and up to 95% after large (10,000X10) skewed insertions. As expected, this shows that using the concept of lexicographical comparison in preference to the SCOOTER's adaptive growth technique enhances the performance of the insertion algorithms. Furthermore, increasing the number of digits allowed for labelling to all 10 decimal digits (0 to 9) instead of the 3 quaternary digits (1,2, and 3), giving a wider range of XML labels.

Unlike the initialisation process, the new labels generated in this experiment are dependent on their randomly selected positions. To achieve scalability, the randomness of the algorithm was considered when evaluating label size and insertion time. As discussed earlier, both small and large numbers of insertions were tested in different XML datasets; each test was repeated several times. With regards to the non-normal distribution of the data collected, the medians of the time and size obtained were computed. The consistency of the median values per dataset/artefact supports the reliability of the findings.

To date, the SCOOTER scheme is considered the most compact dynamic labelling scheme that supports skewed insertion (O'Connor and Roantree, 2013) (Chiew et al., 2014a). In (O'Connor and Roantree, 2012), the performance of SCOOTER's insertion algorithm was compared to the V-CDBS (Li et al., 2008), QED (Li and Ling, 2005b), and Vector (Xu et al., 2007) XML labelling schemes. The results of O'Connor and Roantree's comparison showed that SCOOTER always produced more compact labels after both a small (100X10) and large (1000X100) number of insertions. Their experiment was conducted on self-label values rather than the full interval or prefix-based labels. Again, instead of using any existing experimental benchmarks, the self-labels tested were initialy generated using the intialisation algorithms of the schemes being compared.

As Base-9 outperfomed SCOOTER in this respect by always providing shorter labels, it can be inferred that the Base-9 scheme is a more compact labelling scheme than the V-CDBS (Li et al., 2008), QED (Li and Ling, 2005b), and Vector (Xu et al., 2007) schemes. This observation could be verfied by implementing

these schemes on the same machine using the wider range of experimental XML datasets applied in this thesis.

8.4.3 Re-using Deleted Node Labels Experiment

The results of the re-using deleted labels experiment was obtained as expected (see Chapter 7). By using lexicographical comparison techniques to label initial and updated labels the Base-9 scheme allowed the reuse of all the deleted labels (if available). The Base-9 scheme showed that 100% of the deleted labels were re-used, whilst SCOOTER re-used no more than 0.9%.

As discussed in Chapter-5 (Section 5.6.4), very few XML labelling schemes (Hye-Kyeong and SangKeun, 2010) (O'Connor and Roantree, 2010b) (Li et al., 2006b) (O'Connor and Roantree, 2012) considered reusing deleted node labels. At the time this study started, the SCOOTER labelling scheme (O'Connor and Roantree, 2012) was the only successful scheme that was designed to support dynamic XML and reuse deleted labels without generating duplicates (Chiew et al., 2014a). Therefore, based on the results of this experiment, it can be inferred that the Base-9 scheme is the most compact XML labelling scheme that assures re-use of all the available deleted labels.

8.4.4 Label Encoding Experiment

This experiment was implemented with the aim of evaluating the encoding methods applied by the two schemes to store their (prefix-based) XML labels. To obtain scalability, the performance of the encoding methods was compared in terms of encoding time and size for both initial labels and updated labels in all experimental XML datasets.

The simplicity of the QED encoding applied to SCOOTER's labels produced the fastest encoding time when it was used on initial nodes or a small number of inserted nodes. As the SCOOTER labels grow rapidly after a large number of insertions (an average of 48% after 5,000 nodes and 76% after 10,000 nodes), the Fibonacci coding (of order 2 and of order 3) was always faster than the QED encoding.

The tree shape of the XML dataset used affected the size of the encoded labels. The behaviour of these two encoding techniques was consistent between the label encoding experiment and XML label compression experiment in terms of

encoding and decoding times over the initial labels. This supports the scalability of the results. The Treebank dataset has the deepest XML tree with the narrowest width (see Section 6.5.1), which causes its labels to have more separators (particularly at leaf level) and smaller self-label values in comparison to other XML datasets. For such an XML tree structure, the experimental results showed that Fibonacci of order 2 was preferable to other encoding methods, whereas the DBLP dataset has the opposite structure (broad and shallow XML tree); consequently, Fibonacci of order 3 performed better than QED or Fibonacci of order 2 in producing smaller codes. For the NASA and XMark datasets, Fibonacci coding both of order 2 and 3 generated shorter code in comparison to QED, particularly Fibonacci of order 2. A small number (100X10) of skewed insertions did not affect the findings. However, after many insertions, Fibonacci coding has always produced smaller code than QED, showing an average reduction of 61% (see Section 7.3.4). This leads to the conclusion that Fibonacci coding enables the storage of the Base-9 labels in a more compressed form than QED encoding does for the SCOOTER scheme for every shape of XML tree.

This experimental result was expected as the growth ratio for Fibonacci codes of order $m \geq 2$ is $F_{i+1}^{(m)} / F_i^{(m)}$ (where $i \geq 0$) (Klein and Ben-Nissan, 2008). Each Fibonacci coding of order $m \geq 2$ generates $F_{i-1}^{(m)}$ codes of length equal to $i + m$, for $i \geq 1$ and of length 1 for $i = 0$. This is illustrated in Table 8.1, which is adapted from (Klein and Ben-Nissan, 2008). The table presents a sample of Fibonacci codes of order 2 and 3 for integers [1:34].

Table 8.1 Fibonacci codes (m =2 and 3) for various integers (adopted from (Klein and Ben-Nissan, 2008))

Integer (n)	Fibonacci Code $F^{(2)}$ $(m=2)$	Fibonacci code $F^{(3)}$ $(m=3)$	Integer (n)	Fibonacci code $F^{(2)}$ $(m=2)$	Fibonacci code $F^{(3)}$ $(m=3)$
1	11	111	18	0001011	01000111
2	011	0111	19	1001011	11000111
3	0011	00111	20	0101011	00100111
4	1011	10111	21	00000011	10100111
5	00011	000111	22	10000011	01100111
6	10011	100111	23	01000011	00010111
7	01011	010111	24	00100011	10010111
8	000011	110111	25	10100011	01010111
9	100011	0000111	26	00010011	11010111
10	010011	1000111	27	10010011	00110111
11	001011	0100111	28	01010011	10110111
12	101011	1100111	29	00001011	000000111
13	0000011	0010111	30	10001011	100000111
14	1000011	1010111	31	01001011	010000111
15	0100011	0110111	32	00101011	110000111
16	0010011	00000111	33	10101011	001000111
17	1010011	10000111	34	000000011	101000111

As mentioned in Chapter-6 (Section 6.3), although some of the existing schemes have defined their encoding methods for storing their XML labels in the memory, none have really tested them. Since the efficiency of other encoding methods in the literature has never been evaluated, it is difficult to give even general conclusions regarding this experimental finding.

8.4.5 Relationships Determination Experiment

The relationships determination experiment (see Chapter 6) was evaluated on initial and updated labels in Chapter 7. The five essential relationships were examined (see Chapter 2): parent/child (P/C), ancestor/descendent (A/D), sibling, lowest common ancestor (LCA), and document order (DO). The aim of this experiment was to measure how quickly these relationships can be determined directly from the labels. Both the proposed and SCOOTER scheme were used to this end.

Establishing each relationship individually was assessed using 200,000 pair of labels from the Treebank dataset. To examine the scalability of the results, a larger number of pairs (up to 400,000) was used to test the individual relationship's determination on updated labels.

The main measure of this experiment was the determination time. To control the variability of execution time and ensure the validity of the results, for each relationship the test was repeated 100 times. The results were then statistically analysed as reported in Chapter 7. Generally, the proposed scheme was found to be faster in determining P/C, A/D, sibling, and LCA relationships individually both before and after insertion in comparison to SCOOTER. When determining the document order (DO) relationship, there was no significant difference between the two schemes.

Using this same technique to evaluate the determination of individual relationships, DDE (Xu et al., 2009), DFPD (Liu et al., 2013), and DPLS (Liu and Zhang, 2016) XML labelling schemes were tested using the NASA, Treebank, and XMark datasets. These results were found to be consistent for all the XML datasets employed. This observation suggests that examining the establishment of individual relationships on the Treebank dataset alone would be sufficient to validate the findings.

The research hypothesis suggested that Fibonacci coding might speed up the determination process. For this reason, the experiment also evaluated the determination time including the decoding process. At first, all the experimental XML datasets were used separately to measure the time taken to determine the relationship, if any, between a pair of nodes. The assessment then covered the influence of the scheme's decoding approach on the speed of the determination process by adding the decoding time to the determination time. For reliable findings, the same large number of pairs of labels (i.e., 200,000) was used for each dataset both before and after insertion. In all circumstances, Fibonacci decoding was faster than that of QED. This observation agreed with the results of the label encoding experiment. Such consistency in the results supports the scalability of the findings. The comparison in this thesis is limited to that of the SCOOTER scheme, which did not measure its performance in terms of relationship determination.

8.4.6 Query Performance Experiment

This experiment was implemented to assess the response time of four essential XPath queries on the XMark dataset, both before and after insertion (see Chapter-6). The experiment provided a comparison between the proposed scheme and SCOOTER in terms of querying time (see Chapter-7). It was expected that there would be no significant difference in query response time between the two schemes, as both apply lexicographical comparison when accessing components' labels to allocate an answer to the query. To ensure scalability, the time taken for 100 runs was computed for each query test, both before and after insertion.

Working on initial labels, the results showed that Base-9 always returns queries concerned with ancestor/descendants (query 2), sibling (query 3), and document order (query 4) relationships faster than SCOOTER. For queries dealing with parent/child (query 1) relationships, both schemes behaved in a similar fashion.

The experiment was repeated after adding 10,000 nodes using (1,000X10) skewed insertions. This insertion was done 10 times for the XMark dataset in order to compensate for the variability of the results received. After each insertion test, the queries were examined using both the Base-9 and SCOOTER schemes individually. As the positions of the new nodes were affected by the randomness of the algorithm, the number of answers matching the queries were measured for each test. When there was a difference between the two schemes in the numbers of queries' answers returned, the percentage difference was found to be no greater than 7% (an average of 3%).

For each query, the response time to return the same number of answers in the two schemes was computed to ensure scalability. As mentioned earlier, the number of answers matching a query may vary in each of the two schemes as a result of the random behaviour of the insertion algorithm. Thus, after each insertion process, if there was a difference between the two schemes in terms of the number of answers then the lower number of answers (say n) obtained from either scheme was used when computing the query response time. That is, the query response time for returning n answers was measured separately for each scheme. The results showed that Base-9 always outperformed SCOOTER in returning each of the four queries.

As discussed in Chapter-6, these four queries were selected because the axis containment in each query represents the essential structural relationships. The examination of other axes was omitted here as they can be handled in a similar way to these four axes (Min et al., 2009). For further precision, all the queries representing the 13 XPath axes in table 2.1 (see Chapter 2) could be tested.

8.4.7 XML Label Compression Experiment

The aim of the XML label compression experiment was to study the possibility of compressing XML labels using prefix encodings (presented in Section 4.5) with the intention of reducing the storage space and minimising the chances of overflow (see Section 4.3). The experiment tested six prefix encoding methods (described in Chapter 4) when compressing XML labels: Fibonacci coding of order 2 (Fib2) and order 3 (Fib3), Lucas coding, Elias-delta (ED) coding, and Elias-Fibonacci coding of order 2 (EF2) and order 3 (EF3). Three parameters were considered when evaluating these prefix-encoding techniques: code size, and encoding and decoding times. To ensure scalability of the results, two labelling schemes were applied separately: the Dewey order (Tatarinov et al., 2002) and SCOOTER (O'Connor and Roantree, 2012), for the reasons given in Section 6.6. The two schemes were used to label the NASA, Treebank, and DBLP datasets individually to examine the effects of the dataset's tree shape on the results. To generalise the findings, the experiment was repeated on the same datasets, all with the same size of 23 MB but with their original XML tree properties preserved. When evaluating encoding and decoding times, each test was run 50 times to help ensure reliability.

The evaluation of the experiment also included the comparison between UTF-8 encoding for Dewey labels and QED encoding for SCOOTER labels (see Chapters 4 and 6). Since UTF-8 and QED encoding methods were originally chosen by the authors of the Dewey and SCOOTER labelling schemes, respectively, both performed faster than any prefix-encoding approaches in terms of encoding time. Conversely, the Fibonacci of order 2 remained the fastest at decoding XML labels. When the code size was evaluated, in comparison to all other prefix-encoding methods tested, UTF-8 produced the largest code for the Dewey labels, but QED generated the smallest code for the SCOOTER labels. However, the label encoding experiment's results showed that after large insertions, Fibonacci coding always produced smaller code than that of QED.

Therefore, Fibonacci coding is still recommended for storing XML labels in a dynamic XML environment.

As mentioned earlier, all the existing XML labelling schemes did not calculate the actual label sizes in main memory (i.e., after encoding). Thus, the evaluation considered only the encoding methods implemented in this study.

8.5 The Validation of the Base-9 Scheme's Properties

This section provides a further evaluation of the proposed scheme. This is to discuss whether the scheme has the properties that makes it a good dynamic XML labelling approach (outlined in Chapter-3) or otherwise. The assessment of the Base-9 scheme revealed the scheme's limitations, from which further improvements can be recommend (Vlahavas et al., 1999). This evaluation is based on the experimental results discussed in Chapter-7 comparing the scheme's performances before and after XML update. The comparisons are evaluated based on the essential properties of a complete dynamic XML labelling scheme (discussed in Chapter-3): dynamic, compact, deterministic, and efficient. Each of these properties is reviewed individually in the following sections.

8.5.1 Supporting Dynamic Environment

This section evaluates the dynamic characteristic of the proposed scheme. An XML labelling scheme is considered fully dynamic when it completely avoids re-labelling XML tree nodes while XML data is being continuously updated (Härder et al., 2007) (see Chapter-3). Testing this property on the Base-9 scheme was mostly covered by the handling insertion experiment. Other experiments tested the determination and querying ability of the scheme over updated labels. As mentioned earlier, the design of the experiment was successful and the results showed that the Base-9 scheme supports a dynamic XML environment. None of the existing nodes required re-labelling, even when a relatively large number of insertions occurred, i.e., up to 50,000 uniform insertions and 10,000X10 skewed insertions (see Section 7.3.2).

Figures 8.1 and 8.2 show a graphical comparison of the labelling time taken for initialisation and for uniform and skewed insertions, respectively. Overall, there is a linear correlation between the XML dataset size and the labelling time, both before and after insertion. The NASA dataset had the shortest labelling time

whilst the DBLP dataset took the longest. For any insertion type, the time taken to insert less than 10,000 nodes was always less than the total initialisation time, about 77% on average (see figures 8.1 and 8.2)

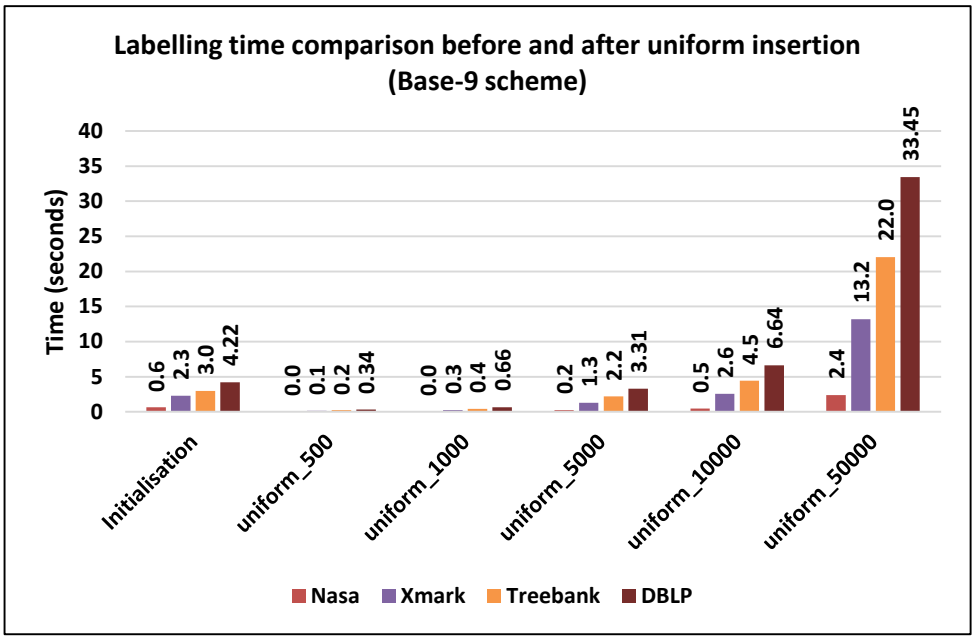


Figure 8.1 Labelling time comparison between initialisation and uniform insertions

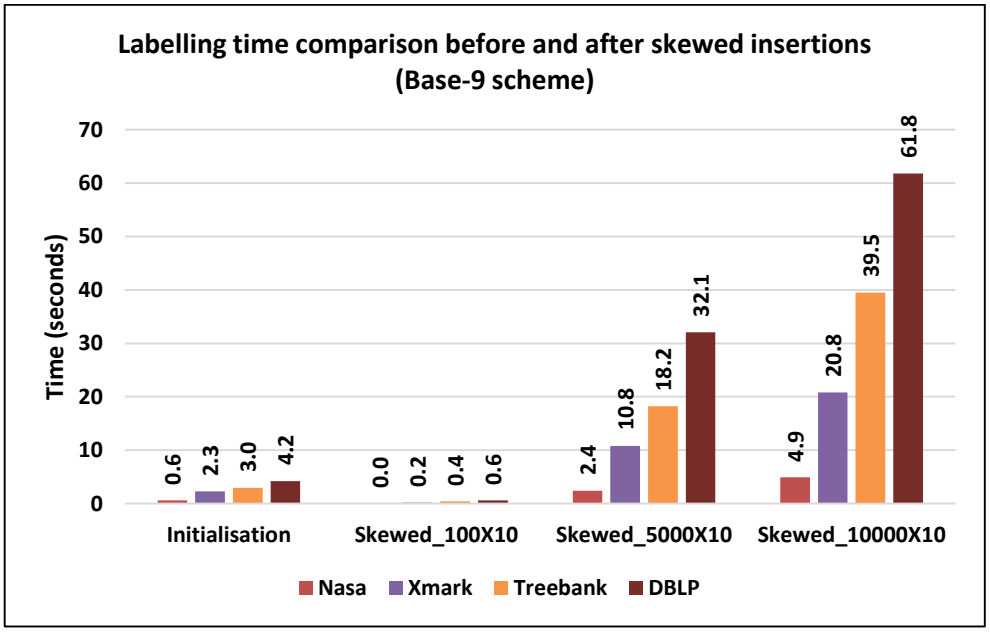


Figure 8.2 Labelling time comparison between initialisation and skewed insertions

In comparison to initialisation time, when a larger numbers of nodes was added, the time taken for labelling new nodes increased rapidly; up to 83% after 50,000 uniform insertions and 81% to 91% after 5,000X10 and 10,000X10 skewed insertions, respectively. Figure 8.3 clarifies this observation by demonstrating the time comparison between generating the initial labels and 50,000 new labels using both uniform and skewed insertion. The chart also shows that the skewed insertion took less time (about 10%) than uniform insertion to add the same number of 50,000 nodes to any XML dataset. Nevertheless, the longest labelling time was just over one minute when the maximum number of 10,000X10 skewed insertions was tested.

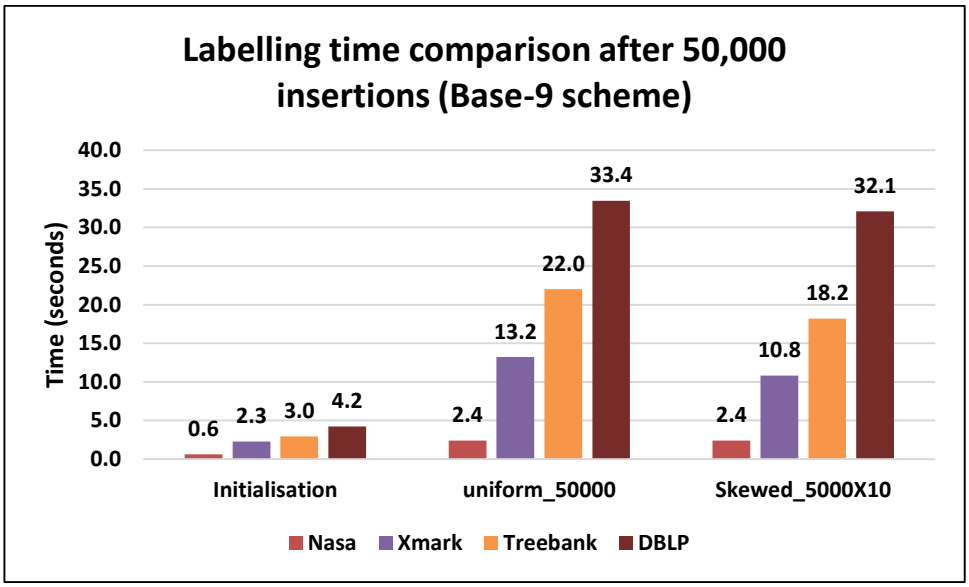


Figure 8.3 Labelling time comparison between initial and after 50,000 insertions

8.5.2 Providing Compact Labels

This section considers the size of the Base-9 labels both before and after insertion. To achieve the compactness requirement for a good XML labelling scheme, the labels generated should be as small as possible (Wu et al., 2004) (see Chapter 3). To study this property within the proposed scheme, the total size of the labels generated was measured both before and after insertion. The initial and updated labels were encoded using both Fibonacci of order 2 and order 3, and the encoded label sizes were also evaluated.

The original sizes of the Base-9 labels for the NASA, XMark, Treebank, and DBLP datasets after initialisation were 0.82 MB, 5.42 MB, 4.04 MB, and 2.85 MB, respectively. As can be seen from Figure 8.4 and Figure 8.5, the Base-9 label

size increases exponentially after insertions. The percentage of the increase on total labels sizes (i.e., $(Size_{updated} + Size_{initial})/Size_{updated}$) for these datasets was an average of 72% after 500 uniform insertions and 86% after 100X10 skewed insertions. As the number of new nodes increased to 50,000, the label size grew by 90% using uniform insertion and 99% using skewed insertion.

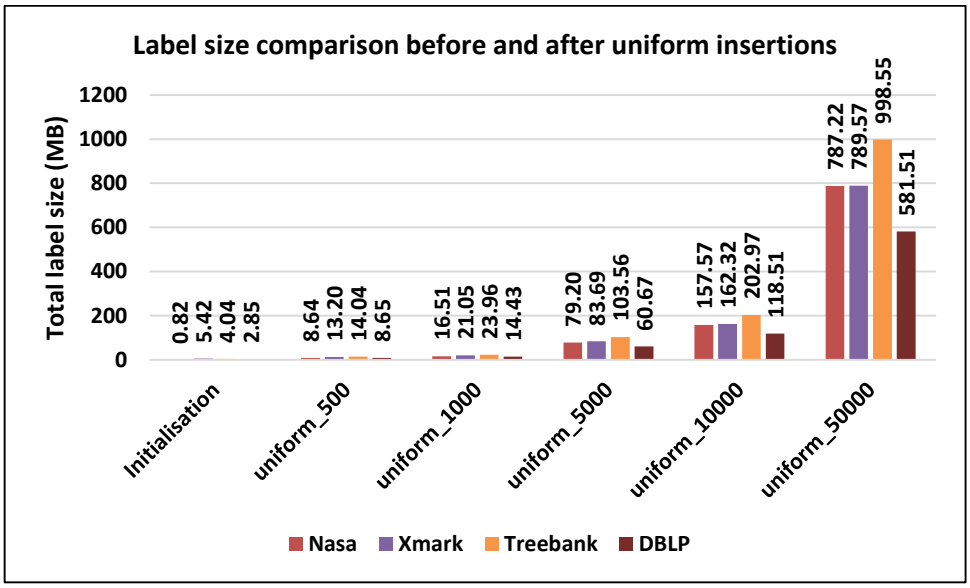


Figure 8.4 Base-9 label size comparison before and after uniform insertion

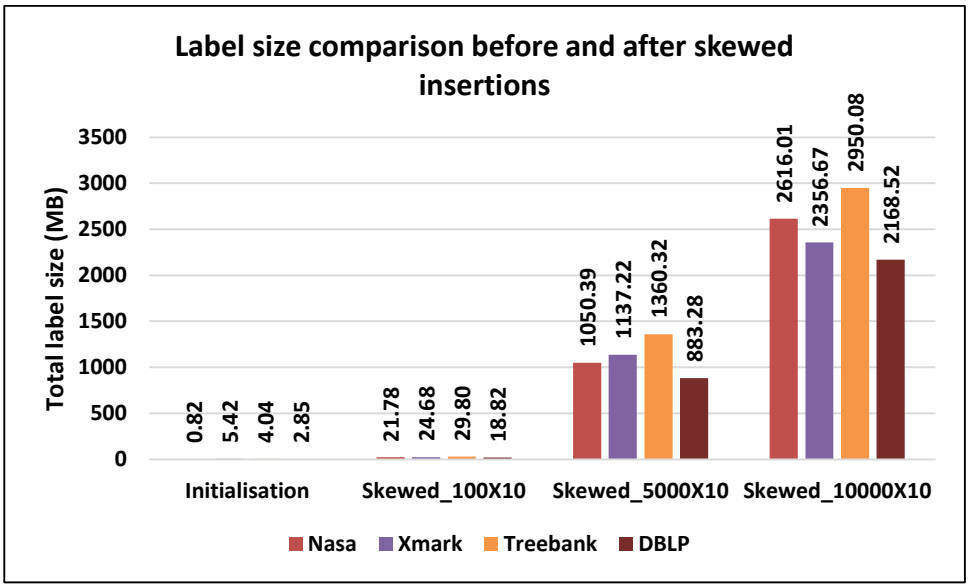


Figure 8.5 Base-9 label size comparison before and after skewed insertion

With regards to encoded label size, the total code size of the initial labels that were actually stored in main memory using Fibonacci of order 2 were: NASA =

2.25 MB, XMark = 8.18 MB, Treebank = 14.37 MB and DBLP = 14.78 MB. As Figures 8.6 and 8.7 show, the Fibonacci code size for the Base-9 labels were slightly increased (by an average of only 0.15%) after a few insertions, i.e., up to 1000 uniform insertions and 100X10 skewed insertions. When as many as 50,000 insertions occurred, the increase in total code size remained insignificant, with an average of 7% whether using uniform or skewed insertions. As the number of nodes inserted doubled to 100,000 by using 10,000X10 skewed insertions, the increase in Fibonacci of order 2 code size was an average of 16%. The total file size stored in memory using both Fibonacci of order 2 and order 3 for each dataset increased by no more than 1.2 MB after 100,000 node insertions. Appendix C presents the increase in file size (in Kbytes) for all the experimental datasets after both uniform and skewed insertions.

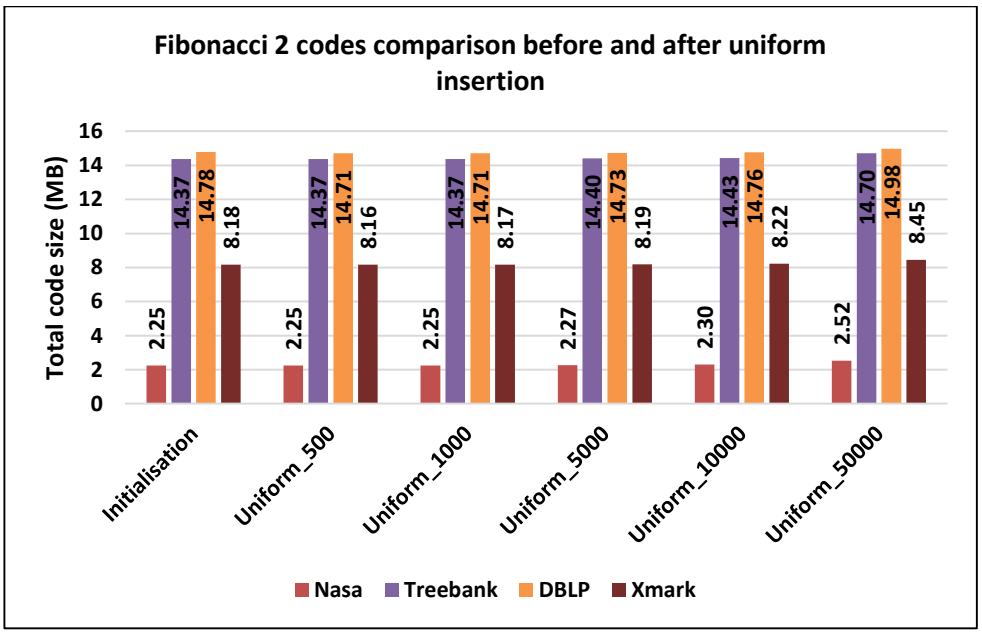


Figure 8.6 Encoded label size comparison before and after uniform insertion using Fibonacci 2

In comparison to Fibonacci of order 2, using Fibonacci of order 3 gave similar measurements. When inserting a small number of nodes (i.e., less than 10,000 nodes) there was no obvious difference between these two encoding schemes in terms of total code size for all the experimental datasets used. As the number of insertions increased to 50,000 nodes regardless of the type of insertion, the Fibonacci of order 3 performed slightly better than Fibonacci of order 2 (about 0.5%). After 100,000 insertions, the percentage difference between the two encoding methods reached around 2%, and Fibonacci of order 3 continued to produce the smallest encoded labels. Due to the similarity between the statistics for these two encoding methods, the graphs illustrating the statistical comparison of the Fibonacci of order 3 are presented in appendix C.

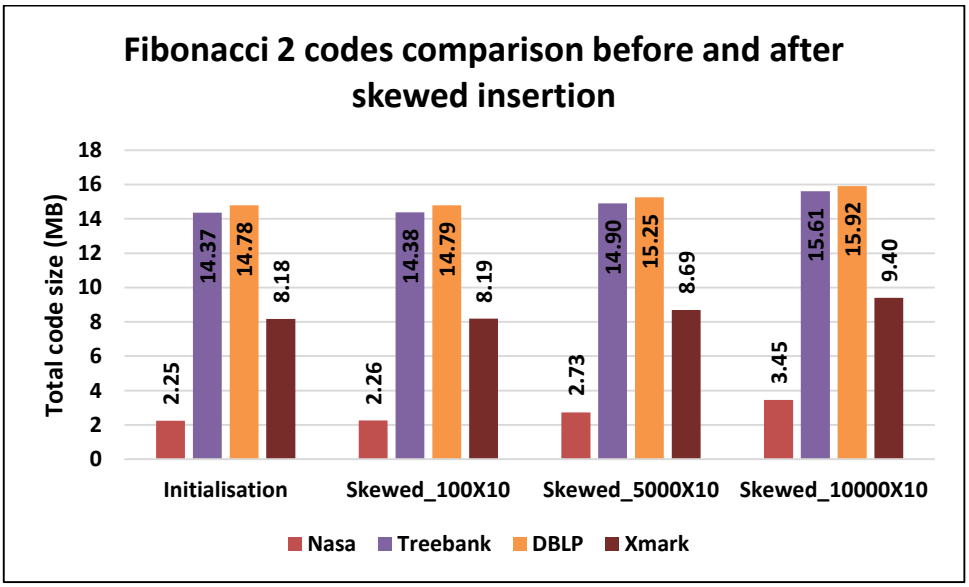


Figure 8.7 Encoded label size comparison before and after skewed insertion using Fibonacci 2

Overall, the maximum encoded label size of the initial Base-9 labels was 20 bytes using Fibonacci of order 2 and 24 bytes using Fibonacci of order 3; both values were obtained using the Treebank dataset. This is because of the high number of separator occurrences within the Treebank prefix-based labels, as this dataset has a relatively deep XML tree structure. Excluding the Treebank dataset, the maximum Fibonacci code for Base-9 labels was no more than 18 bytes after the insertion of 50,000 nodes (see Table 8.2).

Table 8.2 Maximum encoded label size (bytes) of Base-9 labels

Encoding method	Base9-Fib2			Base9-Fib3		
Dataset	Initial	After 50,000 uniform insertion	After 5,000X10 skewed insertion	Initial	After 50,000 uniform insertion	After 5,000X10 skewed insertion
NASA	7	15	14	8	15	13
Treebank	20	36	17	24	42	16
DBLP	6	11	14	6	18	11
XMark	9	17	14	10	18	13

As mentioned earlier, at the time this study was started the SCOOTER labelling scheme was considered the most compact XML labelling scheme that could minimise the growth of label size during skewed insertion (Chiew et al., 2014a). Based on the assessment of the Base-9 scheme in both this and the previous chapters, it can be inferred that Base-9 has enhanced the performance of XML labelling in terms of generating compact XML labels in comparison to SCOOTER.

8.5.3 Determining Relationships

One of the fundamental functions of an XML labelling scheme is establishing the structural relationships between any two nodes efficiently and quickly by directly examining their labels (see Chapter 3). This property was evaluated through the relationship determination experiment in Sections 7.3.5 and 8.4.5. In this section, the Base-9 scheme is assessed in terms of its ability to determine relationships both before and after insertion. Five main relationships were tested: parent/child (P/C), ancestor/descendent (A/D), sibling, lowest common ancestor (LCA), and document order (DO).

The comparison covered the Base-9 scheme using two different sets of labels (i.e., initial and updated). As in the relationship determination experiment (in Section 8.4.5), the Treebank dataset was used for this assessment, where 200,000 pairs of labels were selected randomly to examine the establishment of each relationship individually, both before and after insertion. Again, the main measure in this evaluation was the determination time. For each relationship, the test was repeated 100 times to ensure the validity of the results. Figure 8.8

shows a comparison between the time taken to determine each relationship before and after insertion using the Base-9 scheme. Based on these results, it can be seen that the document order relationship was not affected by the insertion process. The time taken to establish a sibling relationship was increased by 55% after insertion. On the other hand, when determining parent/child, ancestor/descendent and lowest common ancestor relationships, the process took 86%, 92% and 49%, respectively, less time after insertion.

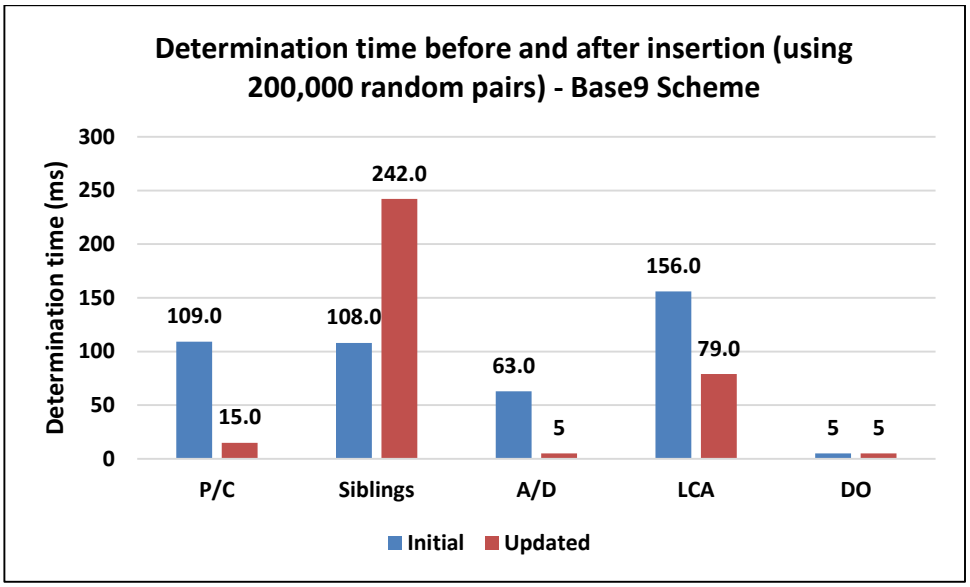


Figure 8.8 Relationships determination time comparison (of 200,000 random pairs) before and after insertion (Base-9 scheme)

However, this result may have been influenced by the random selection of nodes that share the same relationship out of the 200,000 pairs. To investigate whether the randomness of this procedure had an impact on the results, for each relationship the number of pairs with that particular relationship was computed both before and after insertion for 100 runs. The percentage difference between the number of pairs (NP) used to compute the determination time before and after insertion was calculated for each relationship as follows: $(NP_{before} + NP_{after})/NP_{before}$). The number of pairs used were 86%, 88% and 58% fewer after insertion for P/C, A/D and LCA relationships, respectively, and 57% more for the sibling relationship. This could be because of the insertion mechanisim, which focused on adding sibling nodes to the XML tree.

To achieve reliability, it is important to consider a constant number of nodes to test the determination before and after insertion. For each relationship, 20,000

random pairs of labels with the same relationship were selected from initial labels, and the relationship determination time was then computed. The same test was then carried out on the labels after insertion. The test was repeated 100 times, both before and after insertion (separately), to obtain reliable results.

Figure 8.9 shows the relationship determination time comparison for every relationship type using both initial Base-9 labels and updated Base-9 labels. The insertion process did not effect the scheme's performance in establishing the lowest common ancestor or document order relationships. When determining parent/child and sibling relationships, the process consumed 47% and 40%, respectively, more time after insertion. On the other hand, the results showed that there was a 55% improvement in the Base-9 scheme's performance when determining an ancestor/descendent relationship after insertion. Overall, the maximum median determination time was no more than 0.07 seconds after insertion and 0.12 seconds before insertion. Both these maximum times were those which were obtained when establishing the ancestor/descendent relationship.

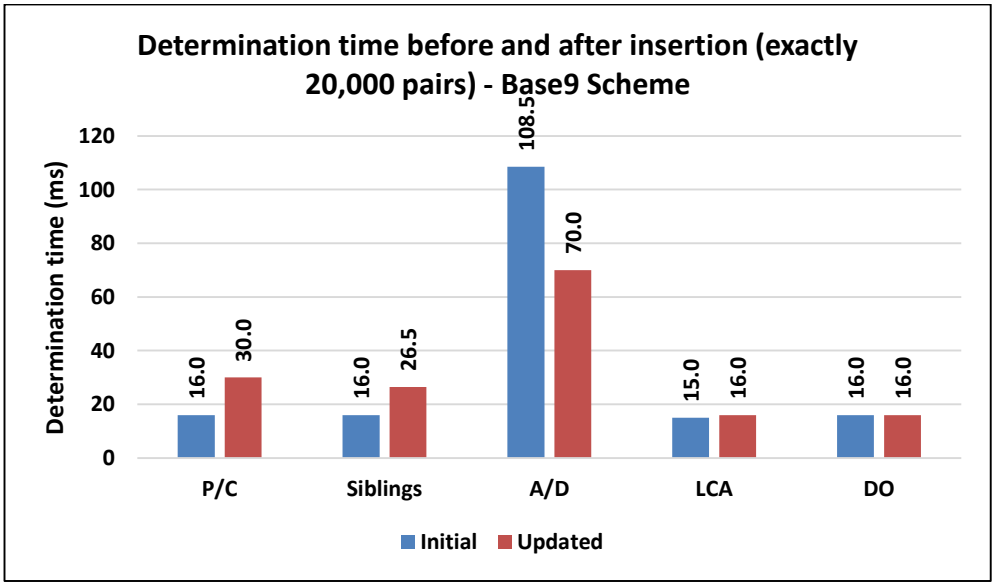


Figure 8.9 Relationships determination time comparison (of 20,000 nodes) before and after insertion (Base-9 scheme)

8.5.4 Query Efficiency

This section discusses the proposed scheme's query efficiency. Any XML labelling scheme must be able to support all kinds of structural relationships queries (see Chapter 3). Four XPath queries representing the main structural

relationships were tested using the XMark dataset: parent/child (query 1), ancestor/descendants (query 2), sibling (query 3), and document order (query 4) relationships. For each query, the response time was taken using the initial Base-9 labels and updated Base-9 labels. Figure 8.10 shows a comparison of response time taken by each query both before and after insertion. It is obvious that the first query (representing the parent/child relationship) was the query most affected by the insertions; the response time increased by 99% on average. For queries 3 and 4, there were smaller increases in their response times after insertion, an average of 4% and 14% respectively. The performance of the second query (representing the sibling relationship) improved slightly after insertion, by about 7%. All queries had a median process time of less than one second, except for the parent/child query, which was as high as 2.65 seconds after insertion.

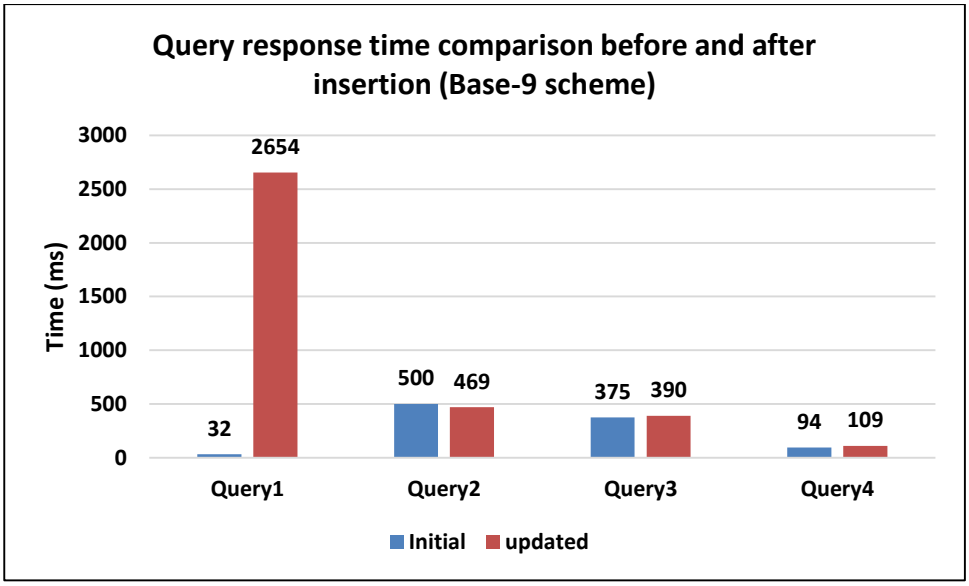


Figure 8.10 Comparison of query response times before and after insertion (Base-9 scheme)

In general, the complexity of a query and the number of nodes examined impacted the query response time. Another influence on the query response time was that the nature of an XML tree's structure may require mulitple structural join operations on values for complex queries (Runapongsa et al., 2006a). This can affect the query performance even further after frequent insertions, which might change the structure of an XML dataset. Such changes are unpredictable as the selection of the inserted nodes' positions are chosen at random. According to the results presented in this section, it can be inferred that the Base-9 scheme permits efficient query processing both before and after insertion.

8.5.5 Conclusion

The evidence so far supports the assertion that the Base-9 scheme can be classified as a fully dynamic XML labelling scheme. The scheme did not cause any re-labelling, even after a relatively large number of insertions. The lexicographical comparison technique applied by the Base-9 scheme kept the labels generated as short as possible both before and after insertion. In a dynamic XML environment, the growth of the storage capacity required to store the Base-9 labels were controlled through the use of Fibonacci coding. It can be seen from the results presented in this chapter and Chapter 7 that the research hypothesis was supported and the use of Fibonacci encoding played an important role in providing compressed XML labels. Furthermore, the scheme's capability to determine relationships and process queries was maintained.

8.6 Limitations of the Experiments

As discussed previously, all the experiments undertaken in this study supported their objectives. Nonetheless, some limitations were found. These restrictions emerged as the experimental designs focused only on the basic aspects required to meet the purposes of each experiment. So as to adhere to the research hypothesis (see Section 8.2), it was necessary that the experimental design be focused and limited in order to assess the hypothesis. Thus, the research first intended to ensure the capabilities of the proposed scheme in supporting a dynamic XML environment before extending it to comprise more complex aspects.

Due to the lack of a standard evaluation framework for XML labelling, all the experiments could be extended by implementing different schemes so as to generalise the findings further. In addition, other existing XML label storage schemes, in particular the ones described in Section 4.4, could be investigated and compared using Fibonacci encoding. For even more elaborate results, the proposed scheme should be tested using more complex and different types of queries. Many mechanical and methodical issues, as previously explained in this chapter, might also be improved and offer suggestions for future work (see Chapter 9).

8.7 The Main Findings of the Experiments

The most important finding is that the experimental results supported the research hypothesis. The various experimental outcomes showed that the Base-9 scheme supports dynamic XML with any shape of XML tree structure. To summarise the findings, it can be stated that the combination of assigning labels based on lexicographical order and storing XML labels via Fibonacci encoding allowed the Base-9 scheme to provide more compact XML labels. Using this approach, the scheme's labels remained relatively small, even after a large number of insertions without requiring any re-labelling in any circumstance. In general, the smaller size of the Base-9 labels helped to speed up the relationship determination and query process (as discussed in Section 8.4.4). The results strongly suggest the effectiveness of the Base-9 scheme as a complete dynamic XML labelling scheme. In the next chapter, the main findings of the entire study are described.

8.8 Conclusion

When developing a new technical or scientific approach, it is essential to evaluate it thoroughly in order to ensure its functionalities, detect and explore its limitations, and highlight further possible improvements. This chapter has evaluated the experiments and their results. The efficiency and scalability of the proposed scheme was assessed in comparison to the SCOOTER scheme. The evaluation of the proposed scheme was also validated in terms of its properties as a complete dynamic XML labelling scheme. The experimental limitations and main findings were identified here. This thesis concludes in the next chapter, in which the main findings of the research are reiterated, with suggestions for future work.

Chapter 9: Conclusions and Future Work

9.1 Introduction

The thesis has illustrated the strengths and weaknesses of XML labelling schemes described in the literature. The study focused on the difficulties related to using an XML labelling scheme in dynamic environments. These difficulties were emphasised by the essential properties required for a fully dynamic XML labelling scheme, i.e., dynamic, compact, deterministic and efficient (detailed in Chapter 3). In an attempt to resolve such difficulties, the Base-9 XML dynamic labelling scheme was proposed in this thesis. This new scheme was an attempt to reduce the occurrence of overflow (see Section 4.3) by employing Fibonacci encoding to store XML labels. In the previous chapters the principles of the proposed scheme along with its objectives, experimental design, implementation, results of testing, and evaluation were described. This chapter completes the thesis by first summarising the research work completed in the following section. The main contributions of this research are highlighted in Section 9.3. Section 9.4 associates the results obtained with the research hypothesis. Future work directions are suggested in Section 9.5. To conclude the thesis, a closing statement is presented in Section 9.6.

9.2 Thesis Summary

As the amount of data on the web has expanded, the use of XML databases became the standard for data transfer and exchange. In many XML database management systems, an XML labelling scheme is recommended for rapid query processing of massive XML documents (Ahn et al., 2017a) (Zhuang and Feng, 2012a). For static XML datasets, query processing can be performed competently using existing labelling schemes such as interval-based labelling schemes (Dietz, 1982) (Li and Moon, 2001) (Zhang et al., 2001), the Dewey Order scheme (Tatarinov et al., 2002), the prime numbering scheme (Wu et al., 2004), and hybrid labelling schemes (Kaplan et al., 2002). As the growing popularity of XML has led to an enormous amount of XML data update (Liu and Zhang, 2016) (Tekli and Chbeir, 2012) (Tatarinov et al., 2001), the need for efficient dynamic XML labelling schemes has become increasingly important in order to support efficient XML queries and updates (O'Connor and Roantree,

2010a) (Subramaniam and Haw, 2014b). Several XML labelling schemes (Xu et al., 2009) (Assefa and Ergenc, 2012) (Li and Ling, 2005b) (O'Connor and Roantree, 2012) have been introduced to process queries efficiently whilst retaining the ability to process order-sensitve updates effecitively (see Chapter-3). However, all the existing labelling schemes suffer from large label sizes, which contributes to the overflow problem particularly under frequent skewed node insertions. One of the reasons behind this is the inadequacy of the encoding techniques used to store the XML labels. These encoding methods have limited storage capacity and do not support frequent skewed insertions in large-scale XML data, particularly in prefix-based labelling schemes (see Chapter-4).

This thesis has attempted to develop an efficient XML dynamic labelling scheme that always generates compressed XML labels both during intialisation and XML update without causing any re-labelling or duplicate labels. This was achievable by the combination of:

- Preserving the node order lexicographically using all the decimal strings including '0' as a part of the labels.
- Using Fibonacci encoding to store the Base-9 labels in a compressed format.

The first chapter outlined the thesis structure and briefly presented the research motivation, objectives and hypothesis statements, which are detailed in Chapter-5. The second chapter provided an overview of XML data and related important topics such as XML syntax, XML parsers, and XML query languages. A comprehensive background on XML labelling schemes and encoding methods in the literature (at the time this study started) is presented in Chapters 3 and 4, respectively.

Chapter 5 introduced the Base-9 scheme including the underlying principles in developing the scheme and defining the rules as to how it would work. The chapter also described the initialisation and insertion mechanisms of the Base-9 scheme. It demonstrated how Fibonacci encoding is employed to store the Base-9 labels. Validation of the Base-9 scheme's ability to determine structural relationships for querying purposes was also given in this chapter.

The experimental objectives, design and implementation were described in detail in Chapter 6. At the time this study was started, the SCOOTER labelling scheme (O'Connor and Roantree, 2012) was described as the most compact dynamic labelling scheme in controlling the growth of label size when XML is updated (Chiew et al., 2014a). To allow comparable evaluation, the SCOOTER labelling scheme was also implemented as it contributed to the development of the proposed scheme. To assess the functionality of the proposed scheme, six experiments were designed and undertaken to examine if the research intentions were achieved and the scheme fulfilled the requirements to be classified as a good dynamic XML labelling scheme. Several experimental XML datasets that vary in their sizes and shapes (see Chapter 6) were used to ensure the scalability of the scheme's performance.

The experimental design and results were evaluated in Chapters 7 and 8. Based on the findings therein, it was demonstrated that the Base-9 scheme outperformed SCOOTER in terms of label size, ability to re-use deleted labels, and determination and query process times both before and after insertion. Moreover, Fibonacci encoding enabled the storage of the Base-9 labels in a more compressed form than allowed by the QED encoding used in the SCOOTER scheme. This applies for any shape of XML tree, even after a large number of insertions. As the Fibonacci decoding was faster than that of QED, Base-9 always showed better performance than SCOOTER during the decoding process, both before and after insertion. Overall, the Base-9 scheme has provided more stability in dynamic XML environments.

Many researchers (Williams and Zobel, 1999) (Chovanec et al., 2010) (Bača et al., 2010) (Scholer et al., 2002) (Walder et al., 2009) (Guttman, 1984) have shown the usefulness of prefix-encoding methods for compression systems. Motivated by this, various existing prefix-encoding methods, including Fibonacci (described in Chapter 4), were studied for the first time in this thesis to encode XML labels. The experimental design is given in Chapter 6. The experimental results were evaluated in Chapters 7 and 8 and has already been published in (Al-Zadjali and North, 2016). The next section illustrates the main contributions of this research.

9.3 The Main Contributions of this Research

This research has addressed some of the problems related to dynamic XML labelling schemes, particularly in terms of compressing XML labels. The study proposed an alternative compression scheme that can reduce label size and simultaneously support large skewed insertion. Consequently, the new Base-9 labelling scheme was introduced, which generates short XML prefix-based labels using decimal strings based on lexicographical order. The labels produced are encoded and stored using Fibonacci encoding. The Fibonacci encoding method allows for a more compressed label code and faster decoding, both of which speed up XML query processing. The main contributions of this thesis can be summarised as follows:

- A new XML labelling scheme, named Base-9, was proposed, which supports static and dynamic XML documents.
- Small or large number of new node insertions in the Base-9 labelling scheme were possible while retaining the lexicographical order of the XML document during both uniform and skewed insertions. The Base-9 labels remain relatively small, even after a large number of insertions and avoid both duplicate labels and re-labelling.
- Fibonacci encoding has been applied for the first time to encode Base-9 XML labels. This encoding method has reduced the storage space required for labels and has accelerated XML querying.
- The Base-9 labelling scheme guarantees the reuse of any deleted node labels.
- The Base-9 labelling scheme establishes all structural relationships quickly and provides for efficient query performance, both in static and dynamic XML environments.
- The effectiveness of the Base-9 labelling scheme and Fibonacci encoding has been measured. The results support the use of the Base-9 scheme as a fully dynamic XML labelling scheme for the four tested datasets. Initially, there is insignificant advantage of using the Base-9 scheme in comparison to SCOOTER. The latter scheme generated initial labels faster than Base-9 by an average of 2.52%.

SCOOTER also succeeded to encode its labels by QED about 53.44% on average faster than Base-9 using Fibonacci encoding. On the contrary, Fibonacci encoding produced about 4.64% shorter codes than QED. Overall, Fibonacci was 27.17% faster in decoding initial labels than QED method. However, the advantages of the Base-9 scheme and Fibonacci encoding amplify when large insertion occurs, particularly after large skewed insertion (which is the main weakness in the current labelling schemes). After 100,000 insertions, the results showed that the Base-9 was 12.78% faster than SCOOTER in assigning new labels. Additionally, the new scheme decreased the average growth rate of the new label sizes by about 97.97% in comparison to SCOOTER. Fibonacci encoding (both of order 2 and of order 3) was faster than QED when encoding and decoding the updated XML datasets labels by 98.7% and 91.20%, respectively. Where Fibonacci encoding always generated shorter label codes by an average of 74.46%.

- Five existing prefix-encoding methods were applied for the first time to compress XML labels: Fibonacci coding of order 2 (Fib2) and order 3 (Fib3), Lucas coding, Elias-delta (ED) coding, and Elias-Fibonacci coding of order 2 (EF2). Among these methods, Lucas coding has not previously been used to compress any kind of data.

- In this thesis, a new Elias-Fibonacci (order 3) encoding was also proposed to encode XML labels. Among the prefix encoding methods studied in this research, the newly implemented Elias-Fibonacci of order 3 achieved the fastest encoding time for the three real XML datasets used.

9.4 Future Work

The novelty of this thesis was based on the application of Fibonacci encoding to store XML labels and control the growth of storage capacity after large insertions. Although the Base-9 labels were generated as lexicographical combinations of decimal strings, Fibonacci encoding treats the labels as integers. The experimental results showed that Fibonacci encoding performed well even after 100,000 insertions; nonetheless, the performance of Fibonacci encoding is still might be subject to integer overflow (Phrack, 2016). Thus, it is worth to

implement Fibonacci encoding with arbitrary precision. Another possible solution would be to define a new encoding technique in which the Base-9 labels could be stored as strings.

It would be reasonable to investigate the possibility of further improvement in Base-9's initialisation and/or insertion algorithms; for instance, using odd decimals for initialisation and preserving the even ones for later XML updates. An interesting approach to gauge the functionality of the Base-9 scheme is by considering the use of XMark with different sizes up to 2GB to study the effectiveness of "Base-9" scheme and Fibonacci encoding over large-scale datasets. To gauge the quality of updating data and label reuse, a more realistic approach could be performed by continuously deleting sequence of subtrees and then reinsert them in a random fashion. This could provide expectation of how label lengths are influenced during such an update process.

With the intention of tackling some of the research limitations discussed above, the following represents further work that could complement the current thesis:

- **Using the Base-9 scheme as an interval-based XML labelling scheme:**

Section 3.3 discussed how interval-based XML labelling schemes establish ancestor/descendant, parent/child and document order more efficiently than other schemes. In general, this approach to labelling is more preferable to be used than prefix-based schemes for query processing of XML keyword searches (Lee et al., 2010), as interval labels contain the path information (Gou and Chirkova, 2007) (Xu and Papakonstantinou, 2005). However, this type of labelling scheme suffers from very long labels. As illustrated in Chapter 3, it is difficult to predict in advance the initial size of the intervals in order to minimise storage cost and avoid repetitive re-labelling in a dynamic XML environment. None of the existing interval-based labelling schemes applied the lexicographical order technique, with which it may be possible to tackle such difficulties. Thus, the Base-9 scheme could be enhanced and applied to an interval-based labelling scheme. In this way, adding a new parent node without causing any re-labelling of the existing nodes (particularly descendant nodes) might also be achievable. Figure 9.1 illustrates a proposed means of adding a new parent node, P_{new}, whilst preserving XML tree details using the Base-9 scheme.

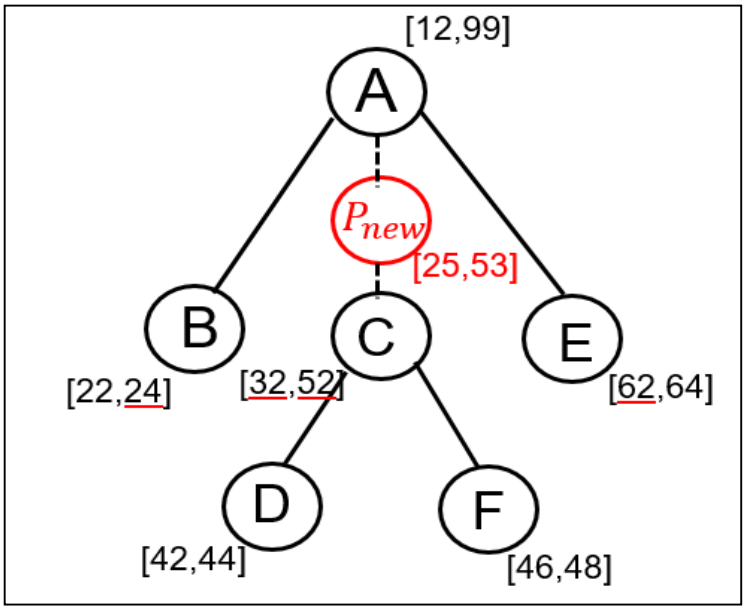


Figure 9.1 Example of inserting a new parent node

Suppose an XML sub-tree is labelled by Base-9 in interval-based format, as shown in Figure 9.1. When node P_{new} is inserted as a new parent to a sub-tree rooted by node C, the start interval value of P_{new} can be computed as an insertion between the end-value of the $label(B)$ (i.e., 24), the adjacent pre-order sibling node to C, and the start-value of $label(C)$ (i.e., 32). The interval's end-value of P_{new} can be calculated by making an insertion between the end-value of $label(C)$ (i.e., 53) and start-value of $label(E)$ (i.e., 62), where E is the adjacent post-order sibling to node C. As can be seen from Figure 9.1, the resulting interval, $label(P_{new})$, satisfies the interval-based labelling's structure and roles (mentioned in Section 3.3) without affecting the existing labels (especially nodes C, D and E). Further study is needed to ensure the effectiveness of the Base-9 scheme as an interval-based labelling scheme in a dynamic XML database.

- **The development of a standard evaluation framework as an XML labelling methodology:**

As discussed in section 6.3, there is no standard framework to evaluate the functionality of XML labelling schemes. This has led to some considerable challenge in validating the proposed scheme's reliability within the literature of XML dynamic labelling schemes. Such as, there was no studies covered the performance of the encoding methods used to store XML labels. Although the

evaluation methodology applied in this thesis was intended to be as comprehensive as possible, further study is needed to determine a general evaluation framework for XML labelling systems. This includes testing the existing encoding methods described in Chapter 4. Analysing the performance of the current XML labelling schemes discussed in Chapter 3 in terms of the essential properties of efficient dynamic XML labelling schemes is also required.

- **An assessment of the Base-9 scheme in comparison to a new XML labelling scheme:**

At the time this study started, the SCOOTER labelling scheme (O'Connor and Roantree, 2012) was described as the most compact dynamic XML labelling scheme that re-uses deleted labels, if available (Chiew et al., 2014a). Thus, the SCOOTER scheme was implemented as it contributed to the development of the proposed scheme (see Chapters 5 and 6). Recently, the DPLS XML labelling scheme (Liu and Zhang, 2016) was developed, also with the aim of re-using deleted labels, using their proposed technique of the reduction of a fraction operation to minimise storage space costs (detailed in Section 3.4). In order to further generalise the findings relative to the current state-of-the-art, particularly Base-9's ability in re-use deleted labels and produce shorter labels, it is important to implement and test the DPLS labelling scheme.

9.5 Conclusion

The current research has investigated a wide range of existing dynamic XML labelling schemes, leading to the identification of various restrictions that motivated the development of the Base-9 scheme and the use of Fibonacci encoding to compress XML labels. The main focus of the research was the compact property of XML labels in a dynamic environment. The proposed scheme, together with Fibonacci encoding, has managed to produce reliable, unique and short XML labels both before and after insertion. The scheme provided efficient performance in labelling time, label size, structural relationship determination and query processing in any XML environment (static or dynamic). This chapter summarised the thesis work and findings, highlighted the research's main contributions, and outlined some directions for future work.

References

ABITEBOUL, S., BUNEMAN, P. & SUCIU, D. 2000. Data on the Web: from relations to semistructured data and XML. *JASIS,* Volume (51)**,** Pages 1050-1052.

ABITEBOUL, S., BUNEMAN, P. & SUCIU, D. 2003. Data on the Web: From Relational to Semistructured Data and XML. *SIGMOD Record,* Volume (32)**,** Pages 109-110.

ABITEBOUL, S., KAPLAN, H. & MILO, T. Compact labeling schemes for ancestor queries. Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, 2001 Washington, D.C., USA. 365529: Society for Industrial and Applied Mathematics, Pages 547-556.

ABITEBOUL, S., QUASS, D., MCHUGH, J., WIDOM, J. & WIENER, J. L. 1997. The Lorel query language for semistructured data. *International Journal on Digital Libraries,* Volume (1)**,** Pages 68-88.

AGRESTE, S., DE MEO, P., FERRARA, E. & URSINO, D. 2014. XML matchers: approaches and challenges. *Knowledge-Based Systems,* Volume (66)**,** Pages 190-209.

AHN, J., IM, D.-H., LEE, T. & KIM, H.-G. 2017a. A dynamic and parallel approach for repetitive prime labeling of XML with MapReduce. *The Journal of Supercomputing,* Volume (73)**,** Pages 810-836.

AHN, J., IM, D.-H., LEE, T. & KIM, H.-G. 2017b. Optimization technique for prime number labeling of directed acyclic graphs. *Journal of Theoretical and Applied Information Technology,* Volume (95)**,** Pages 645.

AL-BADAWI, M. 2010. *A Performance Evaluation of a New Bitmap-based XML Processing Approach.* PhD thesis, The University of Sheffield, UK.

AL-KHALIFA, S., JAGADISH, H., KOUDAS, N., PATEL, J. M., SRIVASTAVA, D. & WU, Y. Structural joins: A primitive for efficient XML query pattern matching. Proceedings of the 18th International Conference on Data Engineering, 2002 San Jose, CA, USA. IEEE, Pages 141-152.

AL-SHAIKH, R., HASHIM, G., BINHURAIB, A. & MOHAMMED, S. A modulo-based labeling scheme for dynamically ordered XML trees. Fifth International Conference on Digital Information Management (ICDIM), 2010 Thunder Bay, Canada. IEEE, Pages 213-221.

AL-ZADJALI, H. & NORTH, S. XML Labels Compression using Prefix-encodings. 12th International Conference on Web Information Systems and Technologies, (WEBIST), 2016 Rome, Italy. Pages 69-75.

ALGHAMDI, N. S., RAHAYU, W. & PARDEDE, E. 2014. Semantic-based Structural and Content indexing for the efficient retrieval of queries over large XML data repositories. *Future Generation Computer Systems,* Volume (37)**,** Pages 212-231.

ALI, S., BRIAND, L. C., HEMMATI, H. & PANESAR-WALAWEGE, R. K. 2010. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering,* Volume (36)**,** Pages 742-762.

ALKHATIB, R. & SCHOLL, M. H. Compacting XML Structures Using a Dynamic Labeling Scheme. 26th British National Conference on Databases, , 2009 Birmingham, UK. Springer, Pages 158-170.

ALMELIBARI, A. 2015. *Labelling Dynamic XML Documents: A GroupBased Approach (Thesis).* PhD thesis, University of Sheffield.

ALSTRUP, S. & RAUHE, T. Improved labeling scheme for ancestor queries. Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, 2002 San Francisco, California. Society for Industrial and Applied Mathematics, Pages 947-953.

AMAGASA, T., YOSHIKAWA, M. & UEMURA, S. QRS: a robust numbering scheme for XML documents. Proceedings. 19th International Conference onData Engineering, 2003 Bangalore, India. Pages 705-707.

AMER-YAHIA, S., BOTEV, C. & SHANMUGASUNDARAM, J. Texquery: a full-text search extension to xquery. Proceedings of the 13th international conference on World Wide Web, 2004 New York, NY, USA. ACM, Pages 583-594.

AMER-YAHIA, S. & LALMAS, M. 2006. XML search: languages, INEX and scoring. *SIGMOD Rec.,* Volume (35)**,** Pages 16-23.

AN, D. & PARK, S. Group-Based Prime Number Labeling Scheme for XML Data. 10th International Conference onComputer and Information Technology (CIT), , 2010 Bradford, West Yorkshire, UK. IEEE, Pages 1639-1644.

AN, D. & PARK, S. 2011. Efficient access control labeling scheme for secure XML query processing. *Computer Standards & Interfaces,* Volume (33)**,** Pages 439-447.

APOSTOLICO, A. & FRAENKEL, A. 1987. Robust transmission of unbounded strings using Fibonacci representations. *IEEE Transactions on Information Theory,* Volume (33)**,** Pages 238-245.

ARCURI, A. & BRIAND, L. A practical guide for using statistical tests to assess randomized algorithms in software engineering. 33rd International Conference on Software Engineering (ICSE), 2011 Hawaii. IEEE, Pages 1-10.

ARCURI, A. & BRIAND, L. 2014. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability,* 24**,** 219-250.

ARION, A., BONIFATI, A., COSTA, G., D'AGUANNO, S., MANOLESCU, I. & PUGLIESE, A. Efficient query evaluation over compressed XML data. International Conference on Extending Database Technology, 2004 Greece. Springer, Pages 200-218.

ARMSTRONG, R. A. 2014. When to use the Bonferroni correction. *Ophthalmic and Physiological Optics,* Volume (34)**,** Pages 502-508.

ARROYUELO, D., CLAUDE, F., MANETH, S., MÄKINEN, V., NAVARRO, G., NGUYỄN, K., SIRÉN, J. & VÄLIMÄKI, N. 2015. Fast in-memory XPath search using compressed indexes. *Software: Practice and Experience,* Volume (45)**,** Pages 399-434.

ASSEFA, B. & ERGENC, B. OrderBased Labeling Scheme for Dynamic XML Query Processing. International Cross-Domain Conference and Workshop on Availability, Reliability, and Security, {CD-ARES}, 2012 Prague, Czech Republic. Springer Berlin Heidelberg, Pages 287-301.

ASSOCIATION, T. F. 2011. *"Representation Theorems"* http://www.fq.math.ca/Books/Fibonacci-Lucas/chap12.pdf [Online]. [Accessed 7/May/2015.

BAČA, R., WALDER, J., PAWLAS, M. & KRÁTKÝ, M. Benchmarking the compression of XML node streams. Database Systems for Advanced Applications, 2010 Berlin Heidelberg. Springer, Pages 179-190.

BANERJEE, S., KRISHNAMURTHY, V., KRISHNAPRASAD, M. & MURTHY, R. Oracle8i-the XML enabled data management system. Proceedings 16th International Conference on Data Engineering, 2000 San Diego, CA, USA. IEEE, Pages 561-568.

BARILLOT, E. & ACHARD, F. 2000. XML: a lingua franca for science. *Trends in biotechnology,* Volume (18)**,** Pages 331-333.

BARTZ-BEIELSTEIN, T., CHIARANDINI, M., PAQUETE, L. & PREUSS, M. 2010. *Experimental methods for the analysis of optimization algorithms*, Springer.

BEECH, G. The Benefits of Using XML Technologies in Astronomical Data Retrieval and Interpretation. Proceedings of the conference on Big Data from Space, 2016 Spain. Publications Office of the European Union, Joint Research Centre, Pages 268-271.

BENEDIKT, M. & CHENEY, J. 2009. Schema-based independence analysis for XML updates. *Proceedings of the VLDB Endowment,* Volume (2)**,** Pages 61-72.

BERTINO, E., CASTANO, S., FERRARI, E. & MESITI, M. 2000. Specifying and enforcing access control policies for XML document sources. *World Wide Web,* Volume (3)**,** Pages 139-151.

BEX, G. J., NEVEN, F. & VAN DEN BUSSCHE, J. DTDs versus XML schema: a practical study. Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS, 2004 New York, NY, USA. ACM, Pages 79-84.

BILLE, P. 2003. Tree edit distance, alignment distance and inclusion [available online via IT University of Copenhagen] . accessed July 2016. Citeseer.

BO, N., XIAOPING, Z. & YIMIN, S. Parallel processing the keyword search in uncertain environment. International Conference on System Science and Engineering (ICSSE), , 2012 Dalian, Liaoning, China. IEEE, Pages 409-414.

BOAG, S., CHAMBERLIN, D., FERNÁNDEZ, M., FLORESCU, D., ROBIE, J., AND SIMÉON, J. . 2007. *"XQuery 1.0: An XML Query Language (2nd edition) [online] at http://www.w3.org/TR/xquery/ "* [Online]. Retrieved 18 July 2016].

BÖHME, T. & RAHM, E. 2000. *Xmach-1: A Benchmark for Xml Data Management [online]* http://dbs.uni-leipzig.de/en/projekte/XML/paper/XMach-1.html [Online]. [Accessed 22 Nov 2016.

BÖHME, T. & RAHM, E. XMach-1: A benchmark for XML data management. Datenbanksysteme in Büro, Technik und Wissenschaft, 2001. Springer, Pages 264-273.

BÖHME, T. & RAHM, E. Multi-user evaluation of XML data management systems with XMach-1. Efficiency and Effectiveness of XML Tools and

Techniques and Data Integration over the Web, 2003. Springer, Pages 148-159.

BONIFATI, A. & CERI, S. 2000. Comparative analysis of five XML query languages. *ACM Sigmod Record,* Volume (29)**,** Pages 68-79.

BÖTTCHER, S. & STEINMETZ, R. 2007. Evaluating xpath queries on XML data streams. *Data Management. Data, Data Everywhere.* Springer.

BOUSQUET-MÉLOU, M., LOHREY, M., MANETH, S. & NOETH, E. 2015. XML compression via directed acyclic graphs. *Theory of Computing Systems,* Volume (57)**,** Pages 1322-1371.

BRAGA, D., CAMPI, A. & CERI, S. 2005. XQBE (XQuery By Example): A visual interface to the standard XML query language. *ACM Trans. Database Syst.,* Volume (30)**,** Pages 398-443.

BRANDES, U., EIGLSPERGER, M., LERNER, J. & PICH, C. 2013. *Graph markup language (GraphML), Chapter 16 pages 517 - 540. Available on line via University of Kanstanz*.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E. & YERGEAU, F. 1998. Extensible markup language (XML). *World Wide Web Consortium Recommendation REC-xml-19980210 http://www.w3.org/TR/1998/REC-xml-19980210,* Volume (16)**,** Page 16.

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E. & YERGEAU, F. 2008. Extensible markup language (XML) 1.0 (5th edition) [online] https://www.w3.org/TR/REC-xml/ Retrieved 18 July 2016. W3C recommendation.

BRESSAN, S., DOBBIE, G., LACROIX, Z., LEE, M. L., LI, Y. G., NAMBIAR, U. & WADHWA, B. XOO7: Applying OO7 Benchmark to XML Query Processing Tools. Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), 2001 Atlanta, GA. ACM.

BRESSAN, S., LI LEE, M., GUANG LI, Y., LACROIX, Z. & NAMBIAR, U. The XOO7 Benchmark. *In:* BRESSAN, S., LEE, M. L., CHAUDHRI, A. B., YU, J. X. & LACROIX, Z., eds. Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web: VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DIWeb Revised Papers, 2003 Berlin, Heidelberg. Springer Berlin Heidelberg, Pages 146-147.

BROWN JR, J. L. 1969. Unique Representations of Integers as Sums of Distinct Lucas Numbers. *The Fibonacci Quarterly,* Volume (7)**,** Pages 243-252.

BRUNO, N., KOUDAS, N. & SRIVASTAVA, D. Holistic twig joins: optimal XML pattern matching. Proceedings of the ACM SIGMOD international conference on Management of data, 2002 Madison, Wisconsin. ACM, Pages 310-321.

CAREY, M. J., DEWITT, D. J. & NAUGHTON, J. F. 1993. *The 007 benchmark*, ACM, Pages 12-21.

CATANIA, B., MADDALENA, A. & VAKALI, A. 2005a. XML document indexes: a classification. *IEEE internet computing,* Volume (9)**,** Pages 64-71.

CATANIA, B., OOI, B. C., WANG, W. & WANG, X. 2005b. Lazy XML updates: laziness as a virtue, of update and structural join efficiency. *Proceedings of the 2005 ACM SIGMOD international conference on Management of data.* Baltimore, Maryland: ACM.

CHAMBERLIN, D., FANKHAUSER, P., MARCHIORI, M. & ROBIE, J. 2001. Xml query requirements [online] http://wiscorp.com/XMLQueryRequirements2.pdf Retrieved 3rd August 2016. *W3C Working Draft,* Volume (15).

CHAMBERLIN, D., ROBIE, J. & FLORESCU, D. Quilt: An XML query language for heterogeneous data sources. International Workshop on the World Wide Web and Databases, 2000 Valencia, Spain. Springer, Pages 1-25.

CHANDRA, P. A. W., ERIC W. 1999. *"Fibonacci Number." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/FibonacciNumber.html* [Online]. Retrieved 07/Sep/2016].

CHAUDHRI, A., ZICARI, R. & RASHID, A. 2003. *XML data management: native XML and XML enabled DataBase systems*, Addison-Wesley Longman Publishing Co., Inc.

CHEN, J., LIANG, W. & YOKOTA, H. 2011. A two-dimension XML encoding method based on Variable length binary code. *Journal of Software,* Volume (6)**,** Pages 2426-2433.

CHEN, Q., LIM, A. & ONG, K. W. D (k)-index: An adaptive structural summary for graph-structured data. Proceedings of the ACM SIGMOD international conference on Management of data, 2003 San Diego, CA, USA. ACM, Pages 134-144.

CHEN, Q., LIM, A. & ONG, K. W. 2008. Enabling structural summaries for efficient update and workload adaptation. *Data & Knowledge Engineering,* Volume (64)**,** Pages 558-579.

CHEN, S., LI, H.-G., TATEMURA, J., HSIUNG, W.-P., AGRAWAL, D. & CANDAN, K. S. Twig 2 Stack: bottom-up processing of generalized-tree-pattern queries over XML documents. Proceedings of the 32nd international conference on Very large data bases, 2006 Seoul, Korea.: VLDB Endowment, Pages 283-294.

CHEN, T., LU, J. & LING, T. W. On boosting holism in XML twig pattern matching using structural indexing techniques. Proceedings of the ACM SIGMOD international conference on Management of data, 2005 Maryland, USA. ACM SIGMOD, Pages 455-466.

CHEN, Y., DAVIDSON, S. B. & ZHENG, Y. Blas: An efficient xpath processing system. Proceedings of the ACM SIGMOD international conference on Management of data, 2004 Paris, France. ACM, Pages 47-58.

CHERGUI, R. 2015. *Zeckendorf arithmetic for Lucas numbers,* https://arxiv.org/pdf/1501.04924.pdf [Online]. [Accessed 7th October 2016.

CHI, Y., YANG, Y. & MUNTZ, R. R. Indexing and mining free trees. Third IEEE International Conference on Data Mining (ICDM) 2003 Florida, USA. IEEE, Pages 509-512.

CHIEN, S.-Y., VAGENA, Z., ZHANG, D., TSOTRAS, V. J. & ZANIOLO, C. Efficient structural joins on indexed XML documents. Proceedings of the 28th international conference on Very Large Data Bases, 2002 Hong Kong SAR, China VLDB Endowment, Pages 263-274.

CHIEW, W.-S., HAW, S.-C., SUBRAMANIAM, S. & CHUA, F.-F. 2014a. Labeling schemes for XML dynamic updates: A survey and open discussions. *E-Commerce, E-Business and E-Service***,** Page 79-83.

CHIEW, W.-S., YEOW, W.-Y., HAW, S.-C., SUBRAMANIAM, S. & CHUA, F.-F. 2014b. Storing and retrieval of hybrid XML databases: A performance evaluation. *E-Commerce, E-Business and E-Service,* Volume (1)**,** Page 91.

CHOI, H., LEE, K.-H. & LEE, Y.-J. 2014. Parallel labeling of massive XML data with MapReduce. *The Journal of Supercomputing,* Volume (67)**,** Pages 408-437.

CHOI, R. H. & WONG, R. K. 2015. VXQ: A visual query language for XML data. *Information Systems Frontiers,* Volume (17)**,** Pages 961-981.

CHOVANEC, P., KRÁTKÝ, M. & WALDER, J. Lossless R-tree compression using variable-length codes. International Conference for Internet

Technology and Secured Transactions (ICITST), 2010 London, United Kingdom. IEEE, Pages 1-8.

CHUNG, C.-W., MIN, J.-K. & SHIM, K. APEX: An adaptive path index for XML data. Proceedings of the ACM SIGMOD international conference on Management of data, 2002 Madison, WI, USA. ACM, Pages 121-132.

CIANCARINI, P., RIZZI, A. & VITALI, F. 1998. Proceedings of the Seventh International World Wide Web ConferenceAn extensible rendering engine for XML and HTML. *Computer Networks and ISDN Systems,* Volume (30)**,** Pages 225-237.

COHEN, E., KAPLAN, H. & MILO, T. 2010. Labeling dynamic XML trees. *SIAM Journal on Computing,* Volume (39)**,** Pages 2048-2074.

COHEN, S., MAMOU, J., KANZA, Y. & SAGIV, Y. XSEarch: A semantic search engine for XML. Proceedings of the 29th international conference on Very large data bases, 2003 Berlin, Germany. VLDB Endowment, Pages 45-56.

CONNOLLY, T. M. & BEGG, C. E. 2005. *Database systems: a practical approach to design, implementation, and management*, Pearson Education.

COOPER, B. F., SAMPLE, N., FRANKLIN, M. J., HJALTASON, G. R. & SHADMON, M. A fast index for semistructured data. in Proceedings of the 27th International Conference on Very Large Data Bases, 2001 Rome, Italy. VLDB, Pages 341-350.

CORNELL-TECH. 2017. *Introduction to Asymptotic Analysis* http://www.cs.cornell.edu/courses/cs312/2004fa/lectures/lecture16.htm , *Cornell University, Ithaca, NY 14853* [Online]. Cornell University, Ithaca, NY 14853 [Accessed February 2017.

CUEVAS, A., FEBRERO, M. & FRAIMAN, R. 2004. An anova test for functional data. *Computational Statistics & Data Analysis,* Volume (47)**,** Pages 111-122.

CZERWINSKI, W., MARTENS, W., NIEWERTH, M. & PARYS, P. Minimization of Tree Pattern Queries. Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, 2016 San Francisco, California, USA. ACM, Pages 43-54.

DALAMAGAS, T., CHENG, T., WINKEL, K.-J. & SELLIS, T. 2006. A methodology for clustering XML documents by structure. *Information Systems,* Volume (31)**,** Pages 187-228.

DATE, C. J. & DARWEN, H. 1993. *A guide to the SQL Standard: a user's guide to the standard relational language SQL*, Addison-Wesley Longman.

DAVIS, K. C., ZHAN, Y. & DAVIS, R. B. An XML/XPath query language and XMark performance study. Proceedings. Symposium on Applications and the Internet, 2003 Orlando, FL, USA. IEEE, Pages 422-427.

DBLP. 2013. *The Dblp Computer Science Bibliography* http://dblp.uni-trier.de/db/ [Online]. [Accessed 22 Nov 2016].

DEUTSCH, A., FERNANDEZ, M., FLORESCU, D., LEVY, A. & SUCIU, D. 1998. Xml-ql: A query language for xml [online] https://www.w3.org/TR/1998/NOTE-xml-ql-19980819/ Retrieved August 2016.

DEUTSCH, A., FERNANDEZ, M., FLORESCU, D., LEVY, A. & SUCIU, D. 1999. A query language for XML. *Computer networks,* Volume (31)**,** Pages 1155-1169.

DEWEY, M. 1876. A Classification and Subject Index for Cataloguing and Arranging the Books and Pamphlets of a Library [Dewey Decimal Classification]. *Brick row book shop, http://www.gutenberg.org/files/12513/12513-h/12513-h.htm accessed on 15th September 2014*.

DIETZ, P. F. Maintaining order in a linked list. Proceedings of the fourteenth annual ACM symposium on Theory of computing, 1982 San Francisco, California, USA. ACM, Pages 122-127.

DUONG, M. & ZHANG, Y. LSDX: a new labelling scheme for dynamically updating XML data. Proceedings of the 16th Australasian database conference 2005 Newcastle, Australia. Australian Computer Society, Inc., Pages 185-193.

DUONG, M. & ZHANG, Y. 2008. Dynamic Labelling Scheme for XML Data Processing. *In:* MEERSMAN, R. & TARI, Z. (eds.) *On the Move to Meaningful Internet Systems: OTM 2008.* Springer Berlin Heidelberg.

DYBÅ, T., KAMPENES, V. B. & SJØBERG, D. I. 2006. A systematic review of statistical power in software engineering experiments. *Information and Software Technology,* Volume (48)**,** Pages 745-755.

ELIAS, P. 1975. Universal codeword sets and representations of the integers. *Information Theory, IEEE Transactions on,* Volume (21)**,** Pages 194-203.

ELMASRI, R. 2008. *Fundamentals of database systems*, Pearson Education India.

EVJEN. B., K. S., THIRU THANGARATHINAM 2007. *Professional XML [online] via https://books.google.co.uk/books?isbn=0470167386 retrieved July 2016*.

FAN, W., GAROFALAKIS, M. N. & XIONG, M. 2015. Grammar and method for integrating XML data from multiple sources [online] https://www.google.com/patents/US8949710 Retrieved 28 July 2016. Google Patents.

FERNANDEZ, M. & SUCIU, D. Optimizing regular path expressions using graph schemas. Proceedings 14th International Conference on Data Engineering, 1998. , Orlando, FL, USA. IEEE, Pages 14-23.

FISCHER, J. 2009. Short labels for lowest common ancestors in trees. *Algorithms-ESA.* Springer.

FISHER, R. A. 1925. *Statistical methods for research workers,* Edinburgh, UK, Genesis Publishing Pvt Ltd.

FLORESCU, D. & KOSSMANN, D. 1999. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin, Special Issue on,* Volume (1060)**,** Pages 27-34.

FLORESCU, D., KOSSMANN, D. & MANOLESCU, I. 2000. Integrating keyword search into XML query processing. *Computer networks,* Volume (33)**,** Pages 119-135.

FRAENKEL, A. S. 1985. Systems of numeration. *American Mathematical Monthly***,** Pages 105-114.

FRAENKEL, A. S. & KLEIN, S. T. 1985. *Robust universal complete codes as alternatives to Huffman codes*, Department of Applied Mathematics, Weizmann Institute of Science.

FRAIGNIAUD, P. & KORMAN, A. 2016. An Optimal Ancestry Labeling Scheme with Applications to XML Trees and Universal Posets. *Journal of the ACM (JACM),* Volume(63)**,** Page 6.

FRANCESCHET, M. XPathMark: an XPath benchmark for the XMark generated data. International XML Database Symposium, 2005 Trondheim, Norway. Springer, Pages 129-143.

FREDRIKSSON, K. & NIKITIN, F. Simple compression code supporting random access and fast string matching. International Workshop on Experimental and Efficient Algorithms, 2007 Rome, Italy. Springer, Pages 203-216.

FUHR, N., GRO, K. & JOHANN. XIRQL: a query language for information retrieval in XML documents. Proceedings of the 24th annual international

ACM SIGIR conference on Research and development in information retrieval, 2001 New Orleans, Louisiana, USA. ACM, Pages 172-180.

FUHR, N. & GROßJOHANN, K. Xirql-an extension of xql for information retrieval. ACM SIGIR Workshop On XML and Information Retrieval, 2000 NY, USA. Citeseer.

GAGIE, T., NAVARRO, G., NEKRICH, Y. & ORDÓNEZ, A. 2015. Efficient and compact representations of prefix codes. *IEEE Transactions on Information Theory,* Volume (61)**,** Pages 4999-5011.

GENEVÈS, P. & LAYAÏDA, N. 2006. A system for the static analysis of XPath. *ACM Transactions on Information Systems (TOIS),* Volume (24)**,** Pages 475-502.

GHALEB, T. A. & MOHAMMED, S. Novel scheme for labeling XML trees based on bits-masking and logical matching. World Congress on Computer and Information Technology (WCCIT), 2013 Tunisia, Sousse. IEEE, Pages 1-5.

GHALEB, T. A. & MOHAMMED, S. 2015. A Dynamic Labeling Scheme Based on Logical Operators: A Support for Order-Sensitive XML Updates. *Procedia Computer Science,* Volume (57)**,** Pages 1211-1218.

GHASEMI, A. & ZAHEDIASL, S. 2012. Normality tests for statistical analysis: a guide for non-statisticians. *International journal of endocrinology and metabolism,* Volume (10)**,** Pages 486-489.

GHEERBRANT, A., LIBKIN, L. & SIRANGELO, C. Reasoning About Pattern-Based XML Queries. *In:* FABER, W. & LEMBO, D., eds. Proceedings Web Reasoning and Rule Systems: 7th International Conference, RR, Mannheim, Germany, , 2013 Berlin, Heidelberg. Springer Berlin Heidelberg, Pages 4-18.

GIBBONS, J. D. & CHAKRABORTI, S. 2011. *Nonparametric statistical inference*, Springer.

GOG, S. 2009. Broadword computing and Fibonacci code speed up compressed suffix arrays. *Experimental Algorithms.* Springer.

GOLDMAN, R., MCHUGH, J. & WIDOM, J. 1999. From semistructured data to XML: Migrating the Lore data model and query language. *ACM SIGMOD Workshop on The Web and Databases, ,* Volume (2)**,** Pages 153-163.

GOLDMAN, R. & WIDOM, J. Dataguides: Enabling query formulation and optimization in semistructured databases. Proceedings of the 23th VLDB, 1997 San Francisco, CA. Pages 436–445

GOLDMAN, R. & WIDOM, J. Approximate dataguides. Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, 1999 Jerusalem, Israel. , Pages 436-445.

GOTTLOB, G., KOCH, C. & PICHLER, R. 2005. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems (TODS),* Volume (30)**,** Pages 444-491.

GOU, G. & CHIRKOVA, R. 2007. Efficiently querying large XML data repositories: A survey. *IEEE Transactions on Knowledge and Data Engineering,* Volume (19)**,** Pages 1381-1403.

GRAHAM, I. S. 1995. *The HTML sourcebook*, John Wiley & Sons, Inc.

GREEN, T. J., MIKLAU, G., ONIZUKA, M. & SUCIU, D. Processing XML streams with deterministic automata. International Conference on Database Theory, 2003 Italy. Springer, Pages 173-189.

GRIJZENHOUT, S. & MARX, M. 2013. The quality of the XML Web. *Web Semantics: Science, Services and Agents on the World Wide Web,* Volume (19)**,** Pages 59-68.

GRISSOM, R. J. & KIM, J. J. 2005. Effect sizes for research. *A broad practical approach. Mah*.

GUO, L., SHAO, F., BOTEV, C. & SHANMUGASUNDARAM, J. XRANK: Ranked keyword search over XML documents. Proceedings of the ACM SIGMOD international conference on Management of data, 2003 San Diego, California. ACM, Pages 16-27.

GUPTA, A. K. & SUCIU, D. Stream processing of XPath queries with predicates. Proceedings of the ACM SIGMOD international conference on Management of data, 2003 San Diego, California. ACM, Pages 419-430.

GUTTMAN, A. 1984. *R-trees: a dynamic index structure for spatial searching*, ACM.

HACHICHA, M. & DARMONT, J. 2013. A survey of XML tree patterns. *IEEE Transactions on Knowledge and Data Engineering,* Volume (25)**,** Pages 29-46.

HAKIM, C. 2000. *Research design: Successful designs for social and economic research*, Psychology Press.

HAKUTA, S., MANETH, S., NAKANO, K. & IWASAKI, H. XQuery streaming by forest transducers. 30th International Conference on Data Engineering (ICDE), , 2014 Chicago, USA. IEEE, Pages 952-963.

HALL, D. & STRÖMBÄCK, L. Generation of synthetic XML for evaluation of hybrid XML systems. International Conference on Database Systems for Advanced Applications, 2010 Tsukuba, Japan. Springer, Pages 191-202.

HAN, J. Y., LIANG, Z. P. & QIAN, G. 2006. A multiple-depth structural index for branching query. *Information and Software Technology,* Volume (48)**,** Pages 928-936.

HÄRDER, T., HAUSTEIN, M., MATHIS, C. & WAGNER, M. 2007. Node labeling schemes for dynamic XML documents reconsidered. *Data & Knowledge Engineering,* Volume (60)**,** Pages 126-149.

HAROLD, E. R., MEANS, W. S. & UDEMADU, K. 2004. *XML in a Nutshell*, O'reilly Sebastopol, CA.

HAW, S.-C. & AMIN, A. 2015. Node Indexing in XML Query Optimization: A Review. *Indian Journal of Science and Technology,* Volume (8).

HAW, S.-C. & LEE, C.-S. 2011. Data storage practices and query processing in XML databases: A survey. *Knowledge-Based Systems,* Volume (24)**,** Pages 1317-1340.

HAW, S. C. & RAO, G. R. K. A comparative study and benchmarking on xml parsers. The 9th International Conference on Advanced Communication Technology, 2007 Gangwon-Do, Korea (South). IEEE, Pages 321-325.

HE, H., WANG, H., YANG, J. & YU, P. S. Compact reachability labeling for graph-structured data. Proceedings of the 14th ACM international conference on Information and knowledge management, 2005 Bremen, Germany. ACM, Pages 594-601.

HE, H. & YANG, J. Multiresolution indexing of XML for frequent queries. 20th International Proceedings Conference on Data Engineering, 2004 USA. IEEE, Pages 683-694.

HE, Y. A Novel Encoding Scheme for XML Document Update-supporting. International Conference on Advances in Mechanical Engineering and Industrial Informatics (AMEII), 2015 Zhengzhou. Atlantis Press, Volume (15).

HENZINGER, M. R., HENZINGER, T. A. & KOPKE, P. W. Computing simulations on finite and infinite graphs. Proceedings., 36th Annual Symposium on Foundations of Computer Science, 1995 Milwaukee. IEEE, Pages 453-462.

HIDDERS, J. & PAREDAENS, J. 2014. Xpath/xquery. *Encyclopedia of Social Network Analysis and Mining***,** Pages 2425-2432.

HOWELL, D. 2012. *Statistical methods for psychology*, Cengage Learning.

HSU, W.-C. & LIAO, I.-E. 2013. CIS-X: A compacted indexing scheme for efficient query evaluation of XML documents. *Information Sciences,* Volume (241)**,** Pages 195-211.

HUANG, X., BAO, Z., DAVIDSON, S. B., MILO, T. & YUAN, X. Answering regular path queries on workflow provenance. 31st International Conference on Data Engineering, 2015 Seoul, Korea (South). IEEE, Pages 375-386.

HYE-KYEONG, K. & SANGKEUN, L. 2010. A Binary String Approach for Updates in Dynamic Ordered XML Data. *Knowledge and Data Engineering, IEEE Transactions on,* Volume (22)**,** Pages 602-607.

IVES, Z., LEVY, A. & WELD, D. 2000. Efficient evaluation of regular path expressions on streaming XML data, University of Washington.

JIANG, Y., MIN ZENG, Z. & ZHANG, D. Z. An Efficient Encoding and Labeling Based Upon Continued Fraction for Dynamic XML Data. WRI World Congress on Software Engineering, 2009 Xiamen, China. IEEE, Pages 324-328.

JITTRAWONG, K. & WONG, R. K. Optimizing XPath queries on streaming XML data. Proceedings of the eighteenth conference on Australasian database, 2007 Ballarat, Victoria, Australia. Australian Computer Society, Inc., Pages 73-82.

JOHNSON, D. S. 2002. A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges,* Volume (59)**,** Pages 215-250.

JOHNSON, J. R., MILLER, A., KHAN, L. & THURAISINGHAM, B. Extracting semantic information structures from free text law enforcement data. International Conference on Intelligence and Security Informatics (ISI), , 2012 Washington, DC, USA. IEEE, Pages 177-179.

JONES, B. M., SAWICKI, M. & LITTLE, R. A. 2008. System and method for validating an XML document and reporting schema violations [online] https://www.google.com/patents/US7373595 Retrieved 20 July 2016. Google Patents.

KAMPS, J., MARX, M., RIJKE, M. D. & SIGURBJÖRNSSON, B. 2006. Articulating information needs in XML query languages. *ACM Transactions on Information Systems (TOIS),* Volume (24)**,** Pages 407-436.

KANNAN, S., NAOR, M. & RUDICH, S. 1992. Implicat representation of graphs. *SIAM Journal on Discrete Mathematics,* Volume (5)**,** Pages 596-603.

KAPLAN, H., MILO, T. & SHABO, R. A comparison of labeling schemes for ancestor queries. Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, 2002 San Francisco, California. Pages 954-963.

KARPINSKI, M. A. Y. N. 2009 A Fast Algorithm for Adaptive Prefix Coding. *Algorithmica,* Volume (55)**,** Pages 29-41.

KAUSHIK, R., BOHANNON, P., NAUGHTON, J. F. & KORTH, H. F. Covering indexes for branching path queries. Proceedings of the ACM SIGMOD international conference on Management of data, 2002b Madison, WI, USA. ACM, Pages 133-144.

KAUSHIK, R., SHENOY, P., BOHANNON, P. & GUDES, E. Exploiting local similarity for indexing paths in graph-structured data. Proceedings. 18th International Conference on Data Engineering, 2002a Washington, DC, USA IEEE, Pages 129-140.

KELLER, T. J. 1972. Generalizations of Zeckendorf's theorem. *Fibonacci Quarterly,* Volume (10)**,** Pages 95-102.

KHA, D. D., YOSHIKAWA, M. & UEMURA, S. A structural numbering scheme for XML data. the 8th International Conference on Extending Database Technology, 2002 Prague, Czech Republic. Springer, Pages 91-108.

KHAING, A. A. & NI LAR, T. A Persistent Labeling Scheme for Dynamic Ordered XML Trees. IEEE/WIC/ACM International Conference on Web Intelligence, 2006 Hong Kong. IEEE, Pages 498-501.

KHARE, R. & RIFKIN, A. 1997. XML: A door to automated Web applications. *IEEE Internet Computing,* Volume (1)**,** Pages 78-87.

KISELYOV, O. A better XML parser through functional programming. International Symposium on Practical Aspects of Declarative Languages, 2002 Portland Springer, Pages 209-224.

KLAIB, A. & LU, J. Investigation into Indexing XML Data Techniques. Proceedings on the International Conference on Internet Computing (ICOMP), 2014. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing.

KLEIN, S. T. & BEN-NISSAN, M. K. Using Fibonacci compression codes as alternatives to dense codes. Data Compression Conference. DCC 2008, Snowbird, UT, USA. IEEE, Pages 472-481.

KLEIN, S. T. & BEN-NISSAN, M. K. 2010. On the usefulness of fibonacci compression codes. *The Computer Journal,* Volume (53)**,** Pages 701-716.

KLETTKE, M. & MEYER, H. XML and object-relational database systems enhancing structural mappings based on statistics. International Workshop on the World Wide Web and Databases, 2000 Berlin Heidelberg. Springer, Pages 151-170.

KNOTT, R. 1998. *Who was Fibonacci? "http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibBio.html"* [Online]. [Accessed 30/04/2015].

KNUTH, D. E. 1979. Lexicographic permutations with restrictions. *Discrete Applied Mathematics,* Volume (1)**,** Pages 117-125.

KOBAYASHI, K., LIANG, W., KOBAYASHI, D., WATANABE, A. & YOKOTA, H. VLEI Code: An Efficient Labeling Method for Handling XML Documents in an RDB. Proceedings. 21st International Conference on Data Engineering, 2005 Tokyo, Japan. IEEE, Pages 386-387.

KOCHMER, C. & FRANDSEN, E. 2002. *JSP and XML: From Web Services to XML in Your JSP Application*, Addison-Wesley Longman Publishing Co., Inc.

KURTEV, I. 2001. Logical and Physical Structure of XML Documents [online] Retrieved July 2016.

LAM, T. C., DING, J. J. & LIU, J.-C. 2008. XML Document Parsing: Operational and Performance Characteristics. *IEEE Computer,* Volume (41)**,** Pages 30-37.

LAMORTE, W. W. 2016. *Nonparametric Tests http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Nonparametric/BS704_Nonparametric4.html, Boston University School of Public Health* [Online]. Bosten: Boston University School of Public Health. [Accessed December 2016].

LASSILA, M., JUNKKARI, M. & KEKÄLÄINEN, J. 2015. Comparison of two XML query languages from the perspective of learners. *Journal of Information Science,* Volume (41)**,** Pages 584-595.

LAWRENCE, R. 2004. The space efficiency of XML. *Information and Software Technology,* Volume (46)**,** Pages 753-759.

LEE, D. & CHU, W. W. 2000. Comparative analysis of six XML schema languages. *SIGMOD Record,* Volume (29)**,** Pages 76-87.

LEE, D. & CHU, W. W. 2001. CPI: constraints-preserving inlining algorithm for mapping XML DTD to relational schema. *Data & Knowledge Engineering,* Volume (39)**,** Pages 3-25.

LEE, K.-H., WHANG, K.-Y., HAN, W.-S. & KIM, M.-S. 2010. Structural consistency: enabling XML keyword search to eliminate spurious results consistently. *The VLDB Journal,* Volume (19)**,** Pages 503-529.

LEE, Y. K., YOO, S.-J., YOON, K. & BERRA, P. B. Index structures for structured documents. Proceedings of the first ACM international conference on Digital libraries, 1996 Bethesda, MD, USA. ACM, Pages 91-99.

LELEWER, D. A. & HIRSCHBERG, D. S. 1987. Data compression. *ACM Computing Surveys (CSUR),* Volume (19)**,** Pages 261-296.

LI, C. & LING, T. An Improved Prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML. Database Systems for Advanced Applications, 2005a. Springer Berlin Heidelberg, Pages 125-137.

LI, C. & LING, T. W. QED: a novel quaternary encoding to completely avoid re-labeling in XML updates. Proceedings of the 14th ACM international conference on Information and knowledge management, 2005b Bremen, Germany. ACM, Pages 501-508.

LI, C., LING, T. W. & HU, M. Efficient Processing of Updates in Dynamic XML Data. Proceedings of the 22nd International Conference on Data Engineering, 2006a Atlanta, Georgia. IEEE, Pages 13-13.

LI, C., LING, T. W. & HU, M. Reuse or never reuse the deleted labels in XML query processing based on labeling schemes. Database Systems for Advanced Applications, 2006b. Springer, Pages 659-673.

LI, C., LING, T. W. & HU, M. 2008. Efficient updates in dynamic XML data: from binary string to quaternary string. *The VLDB Journal—The International Journal on Very Large Data Bases,* Volume (17)**,** Pages 573-601.

LI, C., LING, T. W., LU, J. & YU, T. On reducing redundancy and improving efficiency of XML labeling schemes. Proceedings of the 14th ACM international conference on Information and knowledge management, 2005c Bremen, Germany. ACM, Pages 225-226.

LI, G., FENG, J., WANG, J. & ZHOU, L. Effective keyword search for valuable lcas over xml documents. Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, 2007 Lisbon, Portugal. ACM, Pages 31-40.

LI, J., LIU, C., ZHOU, R. & WANG, W. 2014. XML keyword search with promising result type recommendations. *World Wide Web,* Volume (17)**,** Pages 127-159.

LI, Q. & MOON, B. Indexing and Querying XML Data for Regular Path Expressions. Proceedings of the 27th International Conference on Very Large Data Bases, 2001. Morgan Kaufmann Publishers Inc., Pages 361-370.

LI, S., YANG, D., WANG, T. & WANG, Y. Highly efficient processing of XML path/twig queries using Index Caches. 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2012 Chongqing, Sichuan, China. IEEE, Pages 2939-2943.

LI, Y. G., BRESSAN, S., DOBBIE, G., LACROIX, Z., LEE, M. L., NAMBIAR, U. & WADHWA, B. XOO7: applying OO7 benchmark to XML query processing tool. Proceedings of the tenth international conference on Information and knowledge management, 2001 Atlanta, USA. ACM, Pages 167-174.

LIEFKE, H. & SUCIU, D. XMill: an efficient compressor for XML data. ACM Sigmod Record, International Conference on Management of Data, 2000 New York, USA. ACM, Pages 153-164.

LIN, R.-R., CHANG, Y.-H. & CHAO, K.-M. A Compact and Efficient Labeling Scheme for XML Documents. Database Systems for Advanced Applications, 2013 Wuhan, China. Springer, Pages 269-283.

LIU, J., MA, Z. M. & QV, Q. 2014. Dynamically querying possibilistic XML data. *Information Sciences,* Volume (261)**,** Pages 70-88.

LIU, J., MA, Z. M. & YAN, L. 2013. Efficient labeling scheme for dynamic XML trees. *Information Sciences,* Volume (221)**,** Pages 338-354.

LIU, J. & YAN, D. 2016. Answering Approximate Queries Over XML Data. *IEEE Transactions on Fuzzy Systems,* Volume (24)**,** Pages 288-305.

LIU, J. & ZHANG, X. X. 2016. Dynamic labeling scheme for XML updates. *Knowledge-Based Systems,* Volume (106)**,** Pages 135-149.

LIU, Z. & CHEN, Y. 2012. Exploiting and Maintaining Materialized Views for XML Keyword Queries. *ACM Transactions on Internet Technology (TOIT),* Volume (12)**,** Page 6.

LIU, Z. H. & GAWLICK, D. Management of Flexible Schema Data in RDBMSs-Opportunities and Limitations for NoSQL. The 7th biennial Conference on Innovative Data Systems Research, 2015 Asilomar, California.

LIZHEN, F. & XIAOFENG, M. Triple Code: An Efficient Labeling Scheme for Query Answering in XML Data. 10th Web Information System and

Application Conference (WISA), , 2013 Yangzhou, China. IEEE, Pages 42-47.

LLOYD, C. M., HALSTEAD, M. D. & NIELSEN, P. F. 2004. CellML: its future, present and past. *Progress in biophysics and molecular biology,* Volume (85)**,** Pages 433-450.

LOHREY, M., MANETH, S. & PETERNEK, F. Compressed tree canonization. International Colloquium on Automata, Languages, and Programming, 2015 Kyoto, Japan. Springer, Pages 337-349.

LOHREY, M., MANETH, S. & SCHMIDT-SCHAUß, M. 2012. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences,* Volume (78)**,** Pages 1651-1669.

LOU, Y., LI, Z. & CHEN, Q. 2012. Semantic relevance ranking for XML keyword search. *Information Sciences,* Volume (190)**,** Pages 127-143.

LU, J. 2013. XML Labeling Scheme. *An Introduction to XML Query Processing and Keyword Search.* Springer.

LU, J., CHEN, T. & LING, T. W. TJFast: effective processing of XML twig pattern matching. Special interest tracks and posters of the 14th international conference on World Wide Web, 2005a Chiba, Japan. ACM, Pages 1118-1119.

LU, J. & LING, T. W. Labeling and querying dynamic XML trees. Asia-Pacific Web Conference, 2004 Hangzhou, China. Springer, Pages 180-189.

LU, J., LING, T. W., BAO, Z. & WANG, C. 2011a. Extended XML Tree Pattern Matching: Theories and Algorithms. *IEEE Transactions on Knowledge and Data Engineering,* Volume (23)**,** Pages 402-416.

LU, J., LING, T. W., CHAN, C.-Y. & CHEN, T. From region encoding to extended dewey: On efficient processing of XML twig pattern matching. Proceedings of the 31st international conference on Very large data bases, 2005b Trento, Italy. VLDB Endowment, Pages 193-204.

LU, J., MENG, X. & LING, T. W. 2011b. Indexing and querying XML using extended Dewey labeling scheme. *Data & Knowledge Engineering,* Volume (70)**,** Pages 35-59.

LU, W., CHIU, K. & PAN, Y. A parallel approach to XML parsing. 7th IEEE/ACM International Conference on Grid Computing, 2006 Barcelona, Spain. IEEE, Pages 223-230.

LUO, C. 2007. *On XML selectivity estimation*, , ProQuest, pages 6-9 [online] http://books.google.co.uk Retrieved 10th August 2016]

LUO, C., JIANG, Z., HOU, W.-C., YU, F. & ZHU, Q. A sampling approach for XML query selectivity estimation. Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, 2009 Saint Petersburg, Russia. ACM, Pages 335-344.

MACTUTOR. 1996. *Edouard Lucas* *http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Lucas.html* [Online]. [Accessed 7/May/2015].

MANI, M. & SUNDARESAN, N. 2003. System and method for query processing and optimization for XML repositories [online] https://www.google.com/patents/US6654734 Retrieved 2nd August 2016]. Google Patents.

MARTINS, L. F. 2009. The Chinese Remainder Theorem. *sage,* Volume (1)**,** Pages 15-38.

MATHGOODIES. 2015. *Percentage changes [online]: http://www.mathgoodies.com/* [Online]. [Accessed January 2017].

MATHIS, C., HÄRDER, T., SCHMIDT, K. & BÄCHLE, S. 2015. XML indexing and storage: fulfilling the wish list. *Computer Science - Research and Development,* Volume (30)**,** Pages 51-68.

MATSUDA, T. 2007. Devices for interpreting and retrieving XML documents, methods of interpreting and retrieving XML documents, and computer product, [online] https://www.google.com/patents/US7228296 retrieved 27 July 2016. Google Patents.

MCGEOCH, C. C. 2001. Experimental analysis of algorithms. *Notices of the AMS,* Volume (48)**,** Pages 304-311.

MCGILL, R., TUKEY, J. W. & LARSEN, W. A. 1978. Variations of box plots. *The American Statistician,* Volume (32)**,** Pages 12-16.

MCPHERSON, J., JETZ, W. & ROGERS, D. J. 2004. The effects of species' range sizes on the accuracy of distribution models: ecological phenomenon or statistical artefact? *Journal of applied ecology,* Volume (41)**,** Pages 811-823.

MEGGINSON, D. 2000. *About SAX via http://sax.sourceforge.net/* [Online]. 19 March 2015].

MEIER, W. eXist: An open source native XML database. Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World, 2002 Germany. Springer, Pages 169-183.

MERIALDO., P. 1999. *Acm Sigmod Record: Xml Version* http://www.dia.uniroma3.it/Araneus/Sigmod/ [Online]. [Accessed 22 Nov 2016].

MICROSOFT, O. 2016. *Microsoft* http://office.microsoft.com/en-gb/?CTT=97 [Online]. [Accessed 13 July 2016].

MILO, T. & SUCIU, D. Index structures for path expressions. International Conference on Database Theory, 1999 Jerusalem, Israel. Springer, Pages 277-295.

MIN, J.-K., LEE, J. & CHUNG, C.-W. 2007. An Efficient Encoding and Labeling for Dynamic XML Data. *In:* KOTAGIRI, R., KRISHNA, P. R., MOHANIA, M. & NANTAJEEWARAWAT, E. (eds.) *Advances in Databases: Concepts, Systems and Applications.* Springer Berlin Heidelberg.

MIN, J.-K., LEE, J. & CHUNG, C.-W. 2009. An efficient XML encoding and labeling method for query processing and updating on dynamic XML data. *Journal of Systems and Software,* Volume (82)**,** Pages 503-515.

MIRABI, M., IBRAHIM, H., MAMAT, A., UDZIR, N. I. & FATHI, L. 2010. Controlling label size increment of efficient XML encoding and labeling scheme in dynamic XML update. *Journal of Computer Science,* Volume (6)**,** Pages 1535-1540.

MIRABI, M., IBRAHIM, H., UDZIR, N. I. & MAMAT, A. 2012. An encoding scheme based on fractional number for querying and updating XML data. *Journal of Systems and Software,* Volume (85)**,** Pages 1831-1851.

MOHR, S. F., KAY, M., LIVINGSTONE, S. & LOESGEN, B. 2000. *Professional XML*, Wrox Press Ltd.

MOTULSKY, H. & SEARLE, P. 2003. The InStat guide to choosing and interpreting statistical tests. *GraphPad Software, San Diego, CA*.

MOTWANI, R. & RAGHAVAN, P. 1996. Randomized algorithms. *ACM Computing Surveys (CSUR),* Volume (28)**,** Pages 33-37.

NA, N. & GUOQING, D. A new labeling scheme for XML trees based on mesh partition. 2nd International Conference on Future Computer and Communication (ICFCC), 2010 Wuhan, China. IEEE, Pages 353-356.

NACHAR, N. 2008. The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution. *Tutorials in Quantitative Methods for Psychology,* Volume (4)**,** Pages 13-20.

NAMBIAR, U., LACROIX, Z., BRESSAN, S., LEE, M. L. & LI, Y. G. Efficient XML data management: an analysis. International Conference on Electronic

Commerce and Web Technologies, 2002 London, UK. Springer, Pages 87-98.

NASA. 2001. *Gsfc Open Source Software* http://opensource.gsfc.nasa.gov/ [Online]. [Accessed 22 Nov 2016].

NI, Y.-F., FAN, Y.-C., TAN, X.-C., CUI, J. & WANG, X.-L. 2012. Numeric-based XML labeling schema by generalized dynamic method. *Journal of Shanghai Jiaotong University (Science),* Volume (17)**,** Pages 203-208.

NICOLA, M. & JOHN, J. Xml parsing: a threat to database performance. Proceedings of the twelfth international conference on Information and knowledge management, 2003 New Orleans, LA, USA. ACM, Pages 175-178.

NICOLA, M., KOGAN, I., RODRIGUES, V. & LIU, M. 2007c. *Transaction Processing over XML (TPoX) Benchmark: XML Data Generation* http://tpox.sourceforge.net/TPoX_DataGeneration_v1.2.pdf [Online]. [Accessed 22 Nov 2016].

NICOLA, M., KOGAN, I. & SCHIEFER, B. An XML transaction processing benchmark. Proceedings of the ACM SIGMOD international conference on Management of data, 2007b China. ACM, Pages 937-948.

NOOR EA THAHASIN, S. & JAYANTHI, P. Vector based labeling method for dynamic XML documents. International Conference on Information Communication and Embedded Systems (ICICES), 2013 Chennai, Tamilnadu, India. IEEE, Pages 217-221.

O'CONNOR, M. F. & ROANTREE, M. Desirable properties for XML update mechanisms. Proceedings of the EDBT/ICDT Workshops, 2010a Lausanne, Switzerland. ACM, Pages 1-9.

O'NEIL, P., O'NEIL, E., PAL, S., CSERI, I., SCHALLER, G. & WESTBURY, N. ORDPATHs: insert-friendly XML node labels. Proceedings of the ACM SIGMOD international conference on Management of data, 2004 Paris, France. ACM, Pages 903-908.

O'CONNOR, M. & ROANTREE, M. EBSL: Supporting Deleted Node Label Reuse in XML. Database and XML Technologies, 2010b. Springer Berlin Heidelberg, Pages 73-87.

O'CONNOR, M. & ROANTREE, M. SCOOTER: A Compact and Scalable Dynamic Labeling Scheme for XML Updates. Database and Expert Systems Applications, 2012. Springer Berlin Heidelberg, Pages 26-40.

O'CONNOR, M. & ROANTREE, M. FibLSS: A Scalable Label Storage Scheme for Dynamic XML Updates. Advances in Databases and Information Systems, 2013 Genoa, Italy. Springer Berlin Heidelberg, Pages 218-231.

ONIZUKA, M. Light-weight XPath processing of XML stream with deterministic automata. Proceedings of the twelfth international conference on Information and knowledge management, 2003 New Orleans, Louisiana, USA. ACM, Pages 342-349.

ORACLE. 2014. *Package org.w3c.dom; https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html#package_description* [Online]. 19 March 2015].

OZTUNA D, E. A., TUCCAR E. 2006. Investigation of four different normality tests in terms of type 1 error rate and power under different distributions. *Turkish Journal of Medical Sciences. ,* volume (36)**,** Pages 171–6.

PAN, Y., ZHANG, Y. & CHIU, K. Hybrid parallelism for XML SAX parsing. IEEE International Conference on Web Services, 2008 Beijing, China. Pages 505-512.

PELEG, D. Informative labeling schemes for graphs. International Symposium on Mathematical Foundations of Computer Science, 2000. Springer, Pages 579-588.

PHRACK. 2016. *Basic Integer Overflows http://phrack.org/issues/60/10.html* [Online]. Phrack Magazine. [Accessed 14th Feb 2017].

POTOK, T. E., ELMORE, M. T., REED, J. W. & SAMATOVA, N. F. An ontology-based HTML to XML conversion using intelligent agents. Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002 Big Island, HI. IEEE, Pages 1220-1229.

PRZYWARSKI, R., GRABOWSKI, S., NAVARRO, G. & SALINGER, A. FM-KZ: An even simpler alphabet-independent FM-index. Stringology Conference, 2006 Czech Technical University, Prague. Citeseer, Pages 226-241.

QADAH, G. Z. 2016. Indexing techniques for processing generalized XML documents. *Computer Standards & Interfaces,* Volume (49)**,** Pages 34-43.

QIN, Z., TANG, Y., TANG, F., XIAO, J., HUANG, C. & XU, H. 2017. Efficient XML query and update processing using a novel prime-based middle fraction labeling scheme. *China Communications,* Volume (14)**,** Pages 145-157.

QIN, Z., TANG, Y. & WANG, X. A String Approach for Updates in Order-Sensitive XML Data. Proceedings International Conference on

Information Technology and Software Engineering, 2013. Springer, Pages 153-164.

RADIYA, A. D., V. 2000. *The Basics of Using Xml Schema to Define Elements* *http://www.ibm.com/developerworks/xml/library/xml-schema/index.html: IBM.* [Online]. [Accessed 14 July 2016].

RAFIEI, D., MOISE, D. L. & SUN, D. Finding syntactic similarities between xml documents. 17th International Workshop on Database and Expert Systems Applications, 2006 USA. IEEE, Pages 512-516.

RAMANAN, P. Covering indexes for XML queries: bisimulation - simulation = negation. Proceedings of the 29th international conference on Very large data bases, 2003 Berlin, Germany. VLDB Endowment, Pages 165-176.

RANGAN, C. A. & JAYANTHI, J. A Generic Parser to parse and reconfigure XML files. Recent Advances in Intelligent Computational Systems (RAICS), 2011 Trivandrum, India. IEEE, Pages 823-827.

RAY, E. T. 2003. *learning XML*, " O'Reilly Media, Inc.".

REIS, D. D. C., GOLGHER, P. B., SILVA, A. S. & LAENDER, A. Automatic web news extraction using tree edit distance. Proceedings of the 13th international conference on World Wide Web, 2004 New York, NY, USA. ACM, Pages 502-511.

REN, J., YIN, X. & GUO, X. 2006. A dynamic labeling scheme for XML document. *Journal of Communication and Computer,* Volume (3)**,** Pages 61-65.

RIZZOLO, F. & MENDELZON, A. O. Indexing XML Data with ToXin. WebDB, 2001 Santa Barbara, California, USA. VLDB, Pages 49-54.

ROBIE, J., CHAMBERLIN, D., DYCK, M., AND SNELSON, J. . 2007. *W3C Recommendation "XML Path Language (XPath) 3.0 http://www.w3.org/TR/2014/REC-xpath-30-20140408/ "* [Online]. Retrieved 18 July 2016.].

ROBIE, J., DERKSEN, E., FANKHAUSER, P., HOWLAND, E., HUCK, G., MACHERIUS, I., MURATA, M., RESNICK, M. & SCHÖNING, H. 1999. XQL (XML query language) http://www.w3.org/TandS/QL/QL98/pp/xql.html. Retrieved August 2016.

ROY, J. & RAMANUJAN, A. 2001. XML schema language: taking XML to the next level. *IT professional,* Volume (3)**,** Pages 37-40.

RUNAPONGSA, K., PATEL, J. M., JAGADISH, H., CHEN, Y. & AL-KHALIFA, S. 2006a. The Michigan benchmark: towards XML query performance diagnostics. *Information Systems,* Volume (31)**,** Pages 73-97.

RUNAPONGSA, K., PATEL, M., JAGADISH, H., CHEN, Y. & S., A.-K 2006b. *The Michigan Benchmark [Online]. Available: http://www.eecs.umich.edu/db/mbench/description.html* [Online]. [Accessed 22 Nov 2016.

RUSU, L. I., RAHAYU, W. & TANIAR, D. Warehousing dynamic XML documents. International Conference on Data Warehousing and Knowledge Discovery, 2006 Krakow, Poland. Springer, Pages 175-184.

SALL, K. B. 2002. *XML family of specifications, Chapter XML Syntax and Parsing Concepts [online] http://www.informit.com/articles/article.aspx?p=27006&seqNum=3 Retrieved July 2016*, Addison-Wesley Longman Publishing Co., Inc.

SALMINEN, A. & TOMPA, F. 2012. *Communicating with XML, Chapter 1: Setting the Stage*, Springer Science & Business Media.

SANS, V. & LAURENT, D. 2008. Prefix based numbering schemes for XML: techniques, applications and performances. *Proc. VLDB Endow.,* Volume (1)**,** Pages 1564-1573.

SAUNDERS, M. N. 2011. *Research methods for business students, 5/e*, Pearson Education India.

SCHMIDT, A. 2003. *Xmark — an Xml Benchmark Project http://www.xml-benchmark.org/* [Online]. [Accessed 22 Nov 2016.

SCHMIDT, A., KERSTEN, M. & WINDHOUWER, M. Querying XML documents made easy: nearest concept queries. Proceedings 17th International Conference on Data Engineering, 2001b Washington, DC, USA. IEEE, Pages 321-329.

SCHMIDT, A., WAAS, F., KERSTEN, M., CAREY, M. J., MANOLESCU, I. & BUSSE, R. XMark: A benchmark for XML data management. Proceedings of the 28th international conference on Very Large Data Bases, 2002. VLDB Endowment, Pages 974-985.

SCHMIDT, A., WAAS, F., KERSTEN, M., FLORESCU, D., CAREY, M. J., MANOLESCU, I. & BUSSE, R. 2001a. Why and how to benchmark XML databases. *ACM Sigmod Record,* Volume (30)**,** Pages 27-32.

SCHOLER, F., WILLIAMS, H. E., YIANNIS, J. & ZOBEL, J. 2002. Compression of inverted indexes For fast query evaluation. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval.* Tampere, Finland: ACM.

SELIGMAN, L. & ROENTHAL, A. 2001. XML's impact an databases and data sharing. *Computer,* Volume (34)**,** Pages 59-67.

SHENG, Y., MINGHUI, W. & LIN, L. 2011. An extended byte carry labeling scheme for dynamic XML Data. *Procedia Engineering,* Volume (15)**,** Pages 5488-5492.

SHICHUAN, L., DONGQING, Y., TENGJIAO, W. & YUE, W. Highly efficient processing of XML path/twig queries using Index Caches. 9th International Conference on Fuzzy Systems and Knowledge Discovery, 2012 Chongqing, Sichuan, China. IEEE, Pages 2939-2943.

SHIRRELL, O. 2016. XML: A Deeper Understanding, Chapter 7 DTDs and Schema [on line] www.xmlbook.info.

SHNAIDERMAN, L. & SHMUELI, O. 2015. Multi-Core Processing of XML Twig Patterns. *IEEE Transactions on Knowledge and Data Engineering,* Volume (27)**,** Pages 1057-1070.

SHNEIDERMAN, B. 1976. Exploratory experiments in programmer behavior. *International Journal of Computer & Information Sciences,* Volume (5)**,** Pages 123-143.

SILBERSTEIN, A., HAO, H., KE, Y. & JUN, Y. BOXes: efficient maintenance of order-based labeling for dynamic XML data. Proceedings 21st International Conference on Data Engineering 2005. Pages 285-296.

SILVASTI, P., SIPPU, S. & SOISALON-SOININEN, E. Schema-conscious filtering of XML documents. 12th International Conference on Extending Database Technology: Advances in Database Technology, 2009 EDBT Saint Petersburg, Russia. ACM, Pages 970-981.

SIRKIN, R. M. 2005. *"Two-sample t tests" Statistics for the social sciences*, Sage Publications.

SOMASUNDARAM, K. & DOMNIC, S. 2007. Extended golomb code for integer representation. *IEEE transactions on multimedia,* Volume (9)**,** Pages 239-246.

SONAWANE, V. R. & RAO, D. 2015. A Comparative Study: Change Detection and Querying Dynamic XML Documents. *International Journal of Electrical and Computer Engineering,* Volume (5)**,** Page 840.

ST.LAURENT, S. 1998. *Why XML* *http://www.simonstl.com/articles/whyxml.htm* [Online]. [Accessed 18th Feb 2017.

STEEDMAN, M., OSBORNE, M., SARKAR, A., CLARK, S., HWA, R., HOCKENMAIER, J., RUHLEN, P., BAKER, S. & CRIM, J. Bootstrapping statistical parsers from small datasets. Proceedings of the tenth conference on European chapter of the Association for Computational

Linguistics, 2003 USA. Association for Computational Linguistics, Pages 331-338.

SU-CHENG, H. & CHIEN-SING, L. 2009. Node labeling schemes in XML query optimization: a survey and trends. *IETE Technical Review,* Volume (26)**,** Pages 88-100.

SU CHENG, H. & KRISHNA RAO, G. S. V. R. A Comparative Study and Benchmarking on XML Parsers. The 9th International Conference on Advanced Communication Technology, 2007. IEEE, Pages 321-325.

SUBRAMANIAM, S., HAW, S.-C. & SOON, L.-K. ReLab: A subtree based labeling scheme for efficient XML query processing. IEEE 2nd International Symposium on Telecommunication Technologies (ISTT), 2014a Langkawi, Malaysia. Pages 121-125.

SUBRAMANIAM, S. & HAW, S. C. ME labeling: A robust hybrid scheme for dynamic update in XML databases. IEEE 2nd International Symposium on Telecommunication Technologies (ISTT), 2014b Langkawi, Malaysia. Pages 126-131.

SUCIU, D. 2002. *Xml Data Repository University of Washington* http://www.cs.washington.edu/research/xmldatasets/ [Online]. [Accessed 22 Nov 2016.

SUN, C., CHAN, C.-Y. & GOENKA, A. K. Multiway slca-based keyword search in xml data. Proceedings of the 16th international conference on World Wide Web, 2007 Banff, AB, Canada. ACM, Pages 1043-1052.

TAHRAOUI, M. A., PINEL-SAUVAGNAT, K., LAITANG, C., BOUGHANEM, M., KHEDDOUCI, H. & NING, L. 2013. A survey on tree matching and XML retrieval. *Computer Science Review,* Volume (8)**,** Pages 1-23.

TAKASE, T., MIYASHITA, H., SUZUMURA, T. & TATSUBORI, M. An adaptive, fast, and safe XML parser based on byte sequences memorization. Proceedings of the 14th international conference on World Wide Web, 2005 New York, USA. ACM, Pages 692-701.

TATARINOV, I., IVES, Z. G., HALEVY, A. Y. & WELD, D. S. Updating XML. Proceedings of the ACM SIGMOD international conference on Management of data, 2001 Santa Barbara, California, USA. ACM, Pages 413-424.

TATARINOV, I., VIGLAS, S. D., BEYER, K., SHANMUGASUNDARAM, J., SHEKITA, E. & ZHANG, C. Storing and querying ordered XML using a relational database system. Proceedings of the ACM SIGMOD

international conference on Management of data, 2002 Madison, Wisconsin. Pages 204-215.

TEKLI, J. & CHBEIR, R. 2012. A novel XML document structure comparison framework based-on sub-tree commonalities and label semantics. *Web Semantics: Science, Services and Agents on the World Wide Web,* Volume (11)**,** Pages 14-40.

TEOREY, T. J., LIGHTSTONE, S. S., NADEAU, T. & JAGADISH, H. 2011. *Database modeling and design: logical design*, Elsevier.

THIMMA, M., TSUI, T. K. & LUO, B. HyXAC: a hybrid approach for XML access control. Proceedings of the 18th ACM symposium on Access control models and technologies, 2013 Amsterdam, Netherlands. ACM, Pages 113-124.

THONANGI, R. A Concise Labeling Scheme for XML Data. In International Conference on Management of Data 2006 India. Computer Society of India, Pages 4-14.

TIDWELL, D. 2002. *Introduction to Xml* *http://www.ibm.com/developerworks/xml/tutorials/xmlintro/section2.html.* [Online]. IBM. [Accessed 13 July 2016].

TIZAG. 2003. *XML family tree* *http://www.tizag.com/xmlTutorial/xmltree.php* [Online]. [Accessed 27 July 2016].

TONG, T., EGUCHI, G., CHEON, J., CALLAHAN, J. & LEFF, L. 2006. Rules about XML in XML. *Expert Systems with Applications,* Volume (30)**,** Pages 397-411.

TREEBANK. 1999. *The Penn Treebank Project* *http://www.cis.upenn.edu/~treebank/* [Online]. [Accessed 22 Nov 2016].

TRIPPE, B. & WALDT, D. 2008. *Using XML and Databases* *http://gilbane.com/whitepapers/EMC/* [Online]. Gilbane Group, Inc. [Accessed 18 Feb 2017].

TROTMAN, A. & SIGURBJÖRNSSON, B. NEXI, now and next. International Workshop of the Initiative for the Evaluation of XML Retrieval, 2004. Springer, Pages 41-53.

VALENTINE;, C., DYKES;, L. & TITTEL, E. 2001. XML Schemas, Chapter 5: Understanding XML Schema [online] http://www.eyrolles.com/Chapitres/9780782140453/chap05.pdf Retreive 14 July 2016.

VARGHA, A. & DELANEY, H. D. 1998. The Kruskal-Wallis test and stochastic homogeneity. *Journal of Educational and Behavioral Statistics,* Volume (23)**,** Pages 170-192.

VARGHA, A. & DELANEY, H. D. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics,* Volume (25)**,** Pages 101-132.

VLAHAVAS, I., STAMELOS, I., REFANIDIS, I. & TSOUKIÀS, A. 1999. ESSE: an expert system for software evaluation. *Knowledge-based systems,* Volume (12)**,** Pages 183-197.

W3C. 2005. *Document Object Model (DOM)* http://www.w3.org/DOM/ [Online]. 19 March 2015].

W3C. 2007. *XML Query (XQuery) Requirements* http://www.w3.org/TR/xquery-requirements/ [Online]. W3C. [Accessed 3rd August 2016].

W3C. 2011. *XQuery and XPath Full Text 1.0* http://www.w3.org/TR/2011/REC-xpath-full-text-10-20110317/ [Online]. W3C Recommendation 17 March 2011. [Accessed 3rd August 2016].

W3C. 2014. *XQuery 3.0: An XML Query Language* https://www.w3.org/TR/xquery-30/. W3C.[Accessed 4th August 2016].

W3C. 2016a. *The Extensible Stylesheet Language Family (XSL)* http://www.w3.org/Style/XSL/ [Online]. [Accessed 18 Feb 2017].

W3C. 2016b. *W3C (http://www.w3.org/).* [Online]. W3C. [Accessed 13 July 2016].

W3SCHOOLS. 2016c. *W3Scools XML attributes* http://www.w3schools.com/xml/xml_attributes.asp [Online]. W3Scools. [Accessed 13 July 2016].

W3SCHOOLS. 2016d. *W3Schools DTD* http://www.w3schools.com/xml/xml_dtd.asp [Accessed 13 July 2016.

W3SCHOOLS. 2016e. *W3Schools XML Schema* http://www.w3schools.com/Xml/schema_intro.asp[Accessed 14 July 2016].

W3SCHOOLS. 2016f. *W3School XML tree* http://www.w3schools.com/xml/xml_tree.asp [Accessed 15 July 2016].

W3SCHOOLS. 2016a. *HTML(5) Toturial* http://www.w3schools.com/html/ [Online]. W3Schools. [Accessed 13 July 2016].

W3SCHOOLS. 2016b. *W3Scools XML* http://www.w3schools.com/xml/ [Online]. W3Scools. [Accessed 13 July 2016].

WALDER, J., KRÁTKÝ, M. & BACA, R. Benchmarking Coding Algorithms for the R-tree Compression. DATESO, 2009 Czech Republic. Pages 32-43.

WALDER, J., KRÁTKÝ, M., BAČA, R., PLATOŠ, J. & SNÁŠEL, V. 2012. Fast decoding algorithms for variable-lengths codes. *Information Sciences,* Volume (183)**,** Pages 66-91.

WALDT, D. 2010. *Six Strategies for Extending Xml Schemas in a Single Namespace* http://www.ibm.com/developerworks/xml/library/x-xtendschema/index.html: *IBM* [Online]. [Accessed 14 July 2016].

WALSH, N. 2016. *A Technical Introduction to XML* http://www.xml.com/index.csp [Online]. O'Reilly Media, Inc. [Accessed July 2016].

WANG, F., LI, J. & HOMAYOUNFAR, H. 2007. A space efficient XML DOM parser. *Data & Knowledge Engineering,* Volume (60)**,** Pages 185-207.

WANG, H., PARK, S., FAN, W. & YU, P. S. ViST: a dynamic index method for querying XML data by tree structures. Proceedings of the ACM SIGMOD international conference on Management of data, 2003 San Diego, California. Pages 110-121.

WEGENER, J., BARESEL, A. & STHAMER, H. 2001. Evolutionary test environment for automatic structural testing. *Information and Software Technology,* Volume (43)**,** Pages 841-854.

WEIGEL, F., SCHULZ, K. & MEUSS, H. The BIRD Numbering Scheme for XML and Tree Databases – Deciding and Reconstructing Tree Relations Using Efficient Arithmetic Operations. Database and XML Technologies, 2005. Springer Berlin Heidelberg, Pages 49-67.

WEISSTEIN, E. W. 1999a. *"Zeckendorf Representation." From MathWorld--A Wolfram Web Resource.* http://mathworld.wolfram.com/ZeckendorfRepresentation.html *[*07/Nov/2014].

WHATLEY , K. 2009. Xml Basics for New Users [Online]. IBM. Retrieved July 2016.

WHITMER, R. 2004. Document Object Model (DOM) Level 3 XPath Specification. *W3C,* http://www*. w3. org/TR/DOM-Level-3-XPath*.

WILCOXON, F. 1945. Individual comparisons by ranking methods. *Biometrics bulletin,* Volume (1)**,** Pages 80-83.

WILLIAMS, H. E. & ZOBEL, J. 1999. Compressing integers for fast file access. *The Computer Journal,* Volume (42)**,** Pages 193-201.

WONG, R. K., LAM, F. & SHUI, W. M. Querying and maintaining a compact XML storage. Proceedings of the 16th international conference on World Wide Web, 2007 Banff, Alberta, Canada,. ACM, Pages 1073-1082.

WU, X., LEE, M.-L. & HSU, W. A prime number labeling scheme for dynamic ordered XML trees. Proceedings. 20th International Conference on Data Engineering, 2004 Boston, USA.: IEEE, Pages 66-78.

WU, X. & LIU, G. 2008. XML twig pattern matching using version tree. *Data & Knowledge Engineering,* Volume (64)**,** Pages 580-599.

XU, L., BAO, Z. & LING, T. A Dynamic Labeling Scheme Using Vectors. Database and Expert Systems Applications, 2007 Germany. Springer Berlin Heidelberg, Pages 130-140.

XU, L., LING, T. W., BAO, Z. & WU, H. Efficient label encoding for range-based dynamic XML labeling schemes. the 15th International Conference on Database Systems for Advanced Applications, 2010 Japan. Springer, Pages 262-276.

XU, L., LING, T. W. & WU, H. 2012. Labeling dynamic xml documents: an order-centric approach. *Knowledge and Data Engineering, IEEE Transactions on,* Volume (24)**,** Pages 100-113.

XU, L., LING, T. W., WU, H. & BAO, Z. DDE: from dewey to a fully dynamic XML labeling scheme. Proceedings of the ACM SIGMOD International Conference on Management of data, 2009 Providence, Rhode Island, USA. ACM, Pages 719-730.

XU, Y. & PAPAKONSTANTINOU, Y. Efficient keyword search for smallest LCAs in XML databases. Proceedings of the ACM SIGMOD international conference on Management of data, 2005 USA. Pages 527-538.

YANGHUA, X., JI, H., WANYUN, C., ZHENYING, H., WEI, W. & GUODONG, F. Branch Code: A Labeling Scheme for Efficient Query Answering on Trees. 28th International Conference on Data Engineering (ICDE), 2012 USA. IEEE, Pages 654-665.

YAO, B. B., OZSU, M. T. & KHANDELWAL, N. XBench benchmark and performance testing of XML DBMSs. Proceedings. 20th International Conference on Data Engineering, 2004 USA. IEEE, Pages 621-632.

YERGEAU, F. 2003. *UTF-8, a transformation format of ISO 10646,* https://tools.ietf.org/html/rfc3629 [Online]. [Accessed 24/OCT/2014.

YOUNAS, M., SHAKSHUKI, E., CHAO, K.-M., PARDEDE, E., RAHAYU, J. W. & TANIAR, D. 2008. Web and Mobile Information Systems XML data update management in XML-enabled database. *Journal of Computer and System Sciences,* Volume (74)**,** Pages 170-195.

YU, J. X., LUO, D., MENG, X. & LU, H. 2005. Dynamically Updating XML Data: Numbering Scheme Revisited. *World Wide Web,* Volume (8)**,** Pages 5-26.

YUN, C., YIRONG, Y. & MUNTZ, R. R. HybridTreeMiner: an efficient algorithm for mining frequent rooted trees and free trees using canonical forms. Proceedings 16th International Conference on Scientific and Statistical Database Management., 2004 Santorini Island Greece. Pages 11-20.

YUN, J.-H. & CHUNG, C.-W. 2008. Dynamic interval-based labeling scheme for efficient XML query and update processing. *Journal of Systems and Software,* Volume (81)**,** Pages 56-70.

ZENG, Y., BAO, Z. & LING, T. W. Supporting range queries in XML keyword search. Proceedings of the Joint EDBT/ICDT Workshops, 2013. ACM, Pages 97-104.

ZHANG, B., GENG, Z. & ZHOU, A. SIMP: efficient XML structural index for multiple query processing. The Ninth International Conference on Web-Age Information Management, 2008 Zhangjiajie, China. IEEE, Pages 113-118.

ZHANG, C., NAUGHTON, J., DEWITT, D., LUO, Q. & LOHMAN, G. 2001. On supporting containment queries in relational database management systems. *SIGMOD Rec.,* Volume (30)**,** Pages 425-436.

ZHANG, J.-X. & SUN, X. Keyword Retrieval Technology Research of XML Document. 3rd International Workshop on Intelligent Systems and Applications (ISA), 2011 Wuhan, China. IEEE, Pages 1-3.

ZHANG, N., HAAS, P. J., JOSIFOVSKI, V., LOHMAN, G. M. & ZHANG, C. Statistical learning techniques for costing XML queries. Proceedings of the 31st international conference on Very large data bases, 2005 Trondheim, Norway. VLDB Endowment, Pages 289-300.

ZHANG, P. & DONG, G. A new labeling scheme using vectors based on polar coordinate system for dynamic XML data. Second Pacific-Asia Conference on Circuits, Communications and System (PACCS), 2010 Beijing, China. IEEE, Pages 167-170.

ZHOU, J., WANG, W., CHEN, Z., YU, J. X., TANG, X., LU, Y. & LI, Y. 2016. Top-Down XML Keyword Query Processing. *IEEE Transactions on Knowledge and Data Engineering,* Volume (28)**,** Pages 1340-1353.

ZHUANG, C. & FENG, S. Full Tree-Based Encoding Technique for Dynamic XML Labeling Schemes. Database and Expert Systems Applications, 2012a Austria. Springer, Pages 357-368.

ZHUANG, C. & FENG, S. Reuse the Deleted Labels for Vector Order-Based Dynamic XML Labeling Schemes. *In:* LIDDLE, S., SCHEWE, K.-D.,

TJOA, A. & ZHOU, X., eds. Database and Expert Systems Applications, 2012b Austria. Springer Berlin Heidelberg, Pages 41-54.

ZHUANG, C., LIN, Z. & FENG, S. Insert-friendly XML containment labeling scheme. Proceedings of the 20th ACM international conference on Information and knowledge management, 2011 Glasgow, United Kingdom. Pages 2449-2452.

ZISMAN, A. 2000. An overview of XML. *Computing & Control Engineering Journal,* Volume (11)**,** Pages 165-167.

ZOU, Q., LIU, S. AND CHU, W.W. Ctree: a compact tree for indexing XML data. In Proceedings of the 6th annual ACM international workshop on Web information and data management 2004 Washington, DC, USA. Pages 39-46.

Appendix A: Summary of Current XML Labelling Evaluation Framework

Table A displays evaluation framework of various existing prefix-based labelling schemes presented in chapter 3, section 3.4, such that:

- Data type used for labelling XML documents
- Encoding method used to store XML labels
- Dynamic column presents how the XML labelling scheme tested handling insertions to support XML update.
- Query performance shows how the XML labelling scheme is validated in terms of supporting XML querying and structural relationships determination.
- Datasets used by the labelling scheme to apply the experimental tests.

Table A Evaluation framework of the existing prefix-based labelling schemes

XML Labelling Scheme	Data type	Encoding method	Dynamic	Query performance	Datasets
Dewey (Tatarinov et al., 2002)	Integer	Utf8	Static	Selected XPath/XQuery from Shakespeare dataset translated to SQL	Shakespeare
ORDPATHs (O'Neil et al., 2004)	Integer	Prefix-free Ordpaths coding	Theoretical	Theoretically: using index and XPath axes	NONE
Extended Dewey (Lu et al., 2005)	Integer	Utf8	Static	Selected: Path queries from XMark and Twig queries from DBLP and Treebank	XMark DBLP Treebank
DDE/CDDE (Xu et al., 2009)	Integer	Prefix-free Ordpaths coding	Uniform/skewed insertions between two siblings: time and size	Determining relationships over first 10000 labels	XMark NASA Treebank

XML Labelling Scheme	Data type	Encoding method	Dynamic	Query performance	Datasets
DFPD (Liu et al., 2013)	Float-point	Prefix-free Ordpaths coding	Uniform insertions between two siblings Skewed insert after/before random nodes: time and size	Determining relationships over randomly chosen labels pairs	XMark NASA Treebank Actor
DPLS (Liu and Zhang, 2016)	Float-point	Prefix-free Ordpaths coding	Skewed insertions between two siblings: time and size	Determining relationships over randomly chosen labels pairs	XMark NASA Treebank Actor
Fractional (Mirabi et al., 2012)	Integers-fractions	Mapping into bit string	Small (10 X 10) skewed insertions between two siblings, leaf and parent nodes insertions: time and size	Selected queries	XMark Shakespeare TCP-H SIGMOD
DPESP (He, 2015)	Alphanumeric-fractions	Not defined	Theoretical: insert between two nodes	Theoretical	XMark SIGMOD NASA Hamlet
LSDX (Duong and Zhang, 2005)	Alphanumeric	Not defined	Inserting single nodes or sub-trees considering different size of XMark dataset: time only	Theoretical	XMark
OrderBased (Assefa and Ergenc, 2012)	Alphanumeric	Not defined	Insert a sub-trees as child to the root: time only	Required time to return all descendants of the root in different size of XMark dataset	XMark Shakespeare
Com-D (Duong and Zhang, 2008)	Alphanumeric	Not defined	Theoretical	Selected XPath queries from Shakespeare dataset	XMark Shakespeare
Persistent (Khaing and Ni Lar, 2006)	Alphanumeric	Not defined	Theoretical	Theoretical	NONE
ImprovedBinary (Li and Ling, 2005a)	Binary string	Stored as binary bits	Focused on re-labelling required during insertions	Selected XPath queries from Shakespeare dataset	Shakespeare SIGMOD, Hamlet, NASA,

XML Labelling Scheme	Data type	Encoding method	Dynamic	Query performance	Datasets
					Actor, Movie Bib, Club, Company, Department
Cohen's (Cohen et al., 2010)	Binary string	Stored as binary bits	Theoretical	Theoretical	NONE
VLEI (Kobayashi et al., 2005)	Binary string	Stored as binary bits	Theoretical	Theoretical	NONE
IBSL (Hye-Kyeong and SangKeun, 2010)	Binary string	Stored as binary bits	Leaf nodes insertions: time and size	Selected XPath queries from Shakespeare dataset	XMark Shakespeare SIGMOD, Hamlet, NASA, Actor, Club, DBLP, Department
EBSL (O'Connor and Roantree, 2010b)	Binary string	Stored as binary bits	Theoretical	Theoretical	NONE
V-CDBS (Li et al., 2008)	Binary string	Stored as binary bits	Uniform/skewed insertions: time and size increment	Selected XPath queries from Shakespeare dataset	XMark Shakespeare Hamlet, DBLP Treebank
XDAS (Ghaleb and Mohammed, 2013)	Binary string	Stored as binary bits	Static	Determining AC, PC, and sibling relations over randomly chosen labels pairs	XMark DBLP Treebank
Dynamic XDAS (Ghaleb and Mohammed, 2015)	Binary string	Stored as binary bits	Simple insertions in different positions focusing on insertion time required	Theoretically as in XDAS but not tested after insertions	XMark DBLP Treebank
QED (Li and Ling, 2005b)	Quaternary string	QED separators	Uniform/skewed insertions between two nodes: focusing on re-labelling	Theoretical	XMark Shakespeare,

XML Labelling Scheme	Data type	Encoding method	Dynamic	Query performance	Datasets
			required and insertion time		Hamlet, NASA, Company, DBLP, Treebank
SCOOTER (O'Connor and Roantree, 2012)	Quaternary string	QED separators	Skewed insertions focusing on storage size	Theoretical	Generated by the author
Base-9 (proposed in this thesis)	Decimal string	Fibonacci coding	Uniform/Skewed insertions: insertion/ decoding time and size increment before and after decoding new labels	Determining relationships over randomly chosen labels pairs and selected XPath queries from XMark	XMark NASA DBLP Treebank

Appendix B: Statistical Analysis Graphs

Appendix B.1 Labels Initialisation Statistical Results

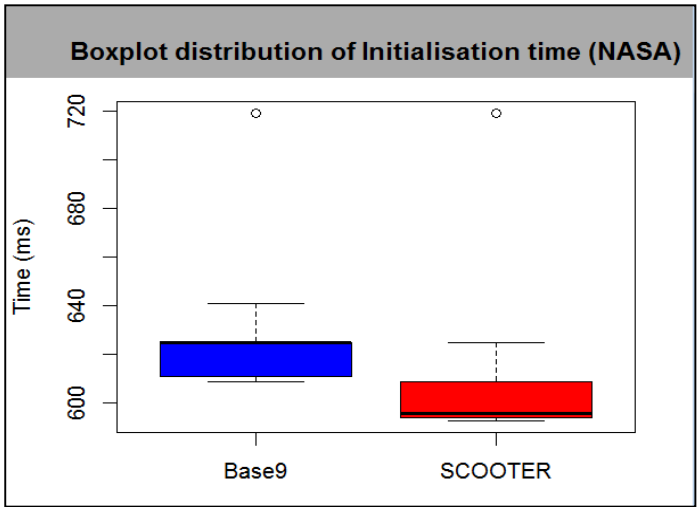
- **Initialisation time: Mann-Whitney U-test result:**

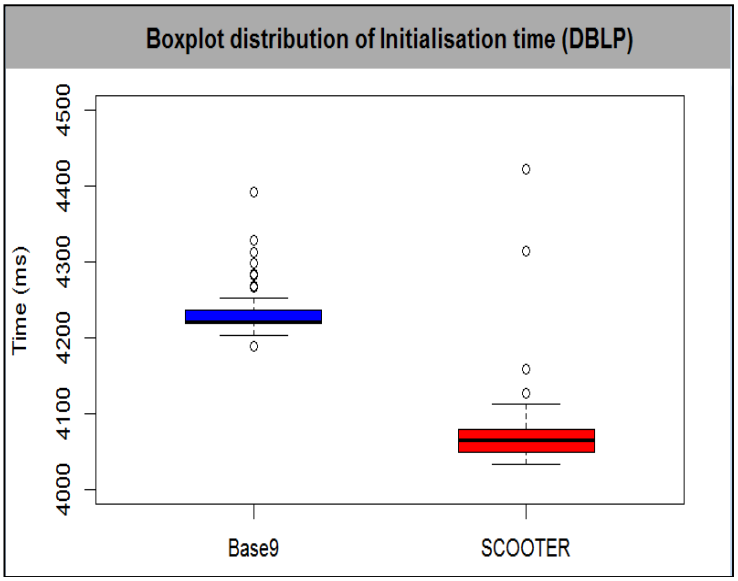
Hypothesis Test Summary

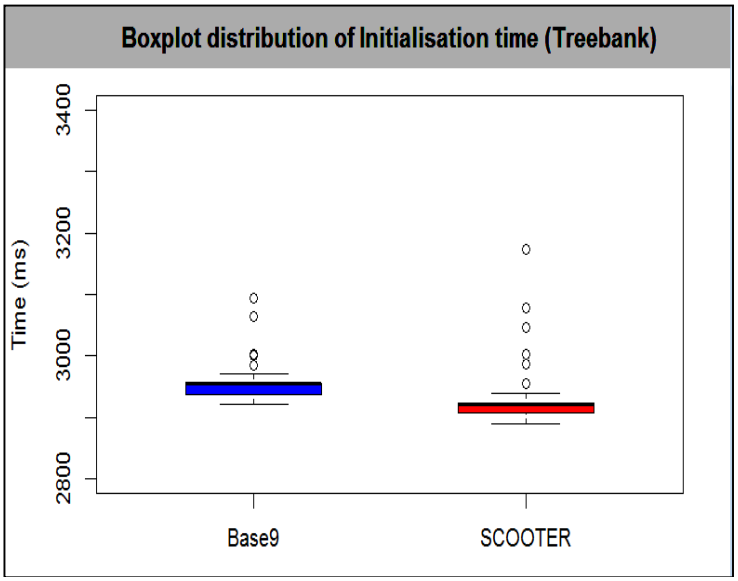
	Null Hypothesis	Test	Sig.	Decision
1	The distribution of NASA is the same across categories of Scheme.	Independent-Samples Mann-Whitney U Test	.000	Reject the null hypothesis.
2	The distribution of Treebank is the same across categories of Scheme.	Independent-Samples Mann-Whitney U Test	.000	Reject the null hypothesis.
3	The distribution of DBLP is the same across categories of Scheme.	Independent-Samples Mann-Whitney U Test	.000	Reject the null hypothesis.
4	The distribution of XMark is the same across categories of Scheme.	Independent-Samples Mann-Whitney U Test	.000	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

- **Box plot distribution of initialisation time:**



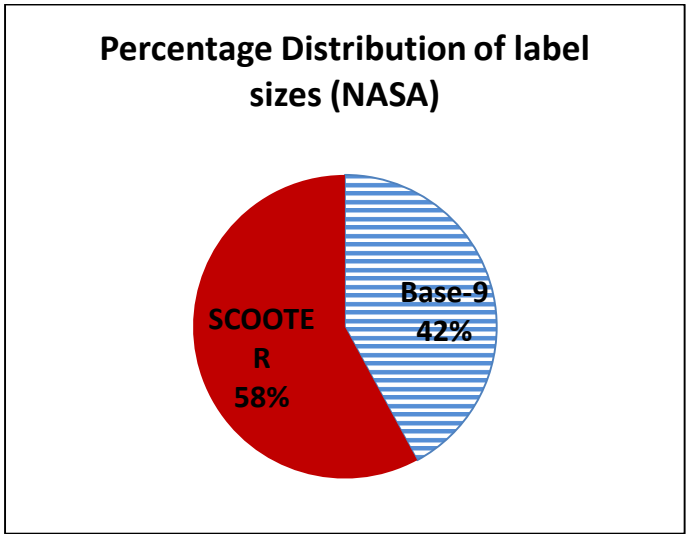


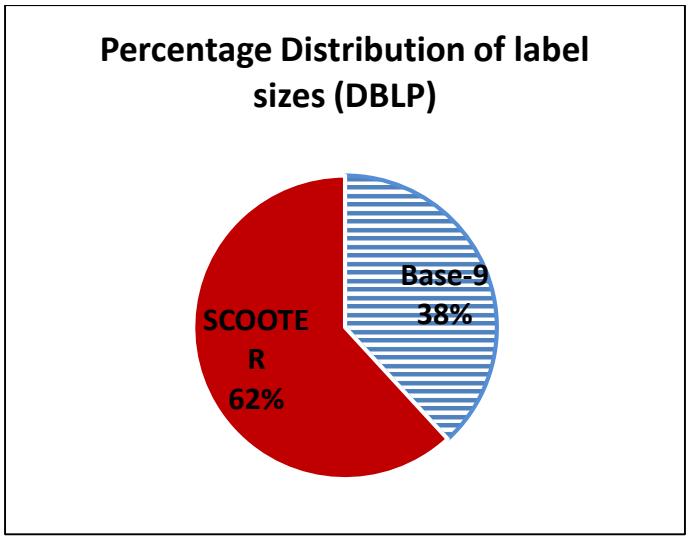


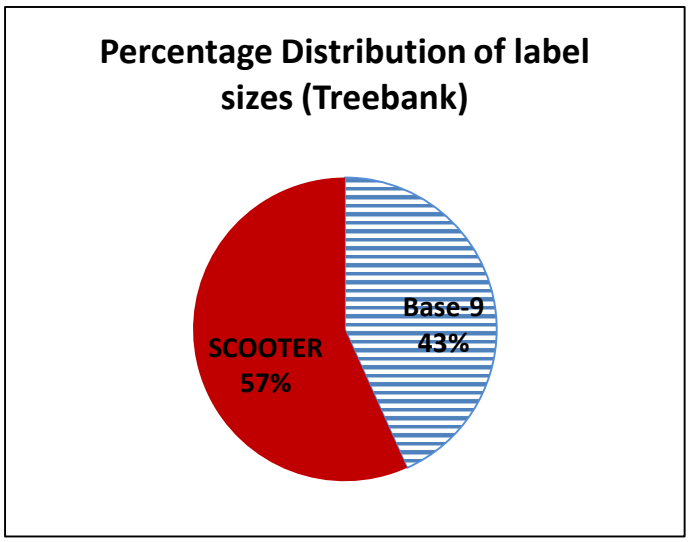
- **Statistic descriptions of initialisation time**

Statistics						
Scheme			**NASA**	**Treebank**	**DBLP**	**XMark**
SCOOTER	N	Valid	100	100	100	100
		Missing	0	0	0	0
	Median		596.0000	2922.0000	4064.0000	2220.0000
	Variance		201.380	7274.438	37503.812	235.818
Base9	N	Valid	100	100	100	100
		Missing	0	0	0	0
	Median		625.0000	2954.0000	4221.0000	2283.0000
	Variance		274.263	7760.061	882.603	122.163

- **The percentage distribution of initial label sizes:**

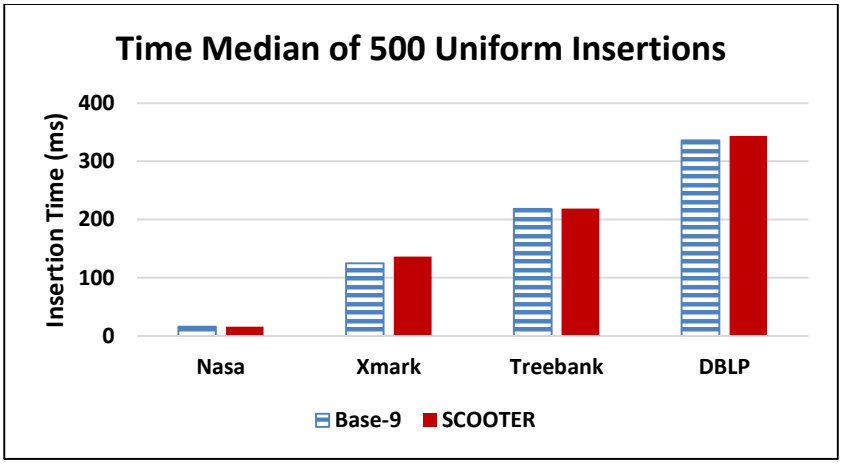


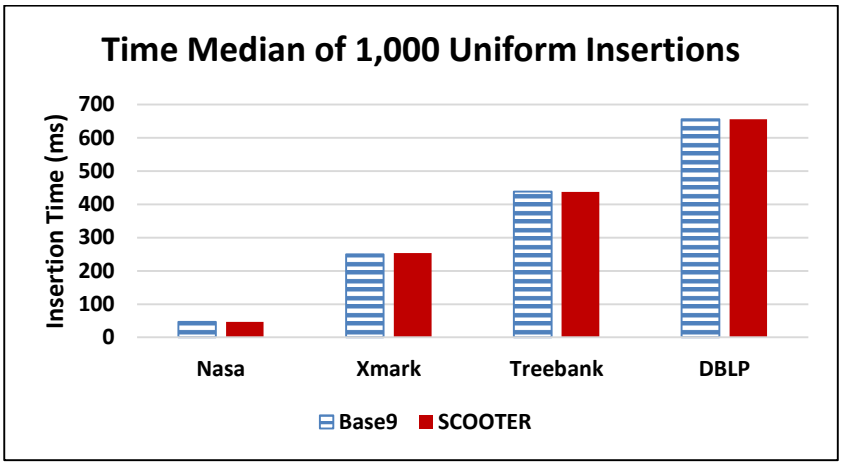


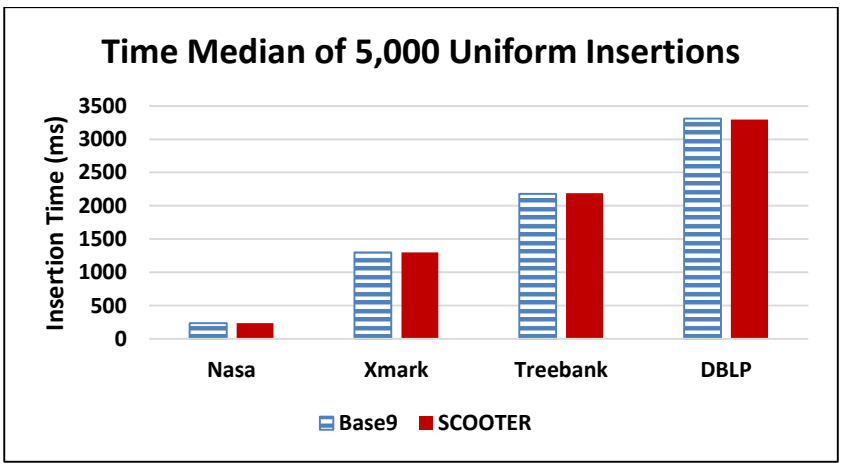


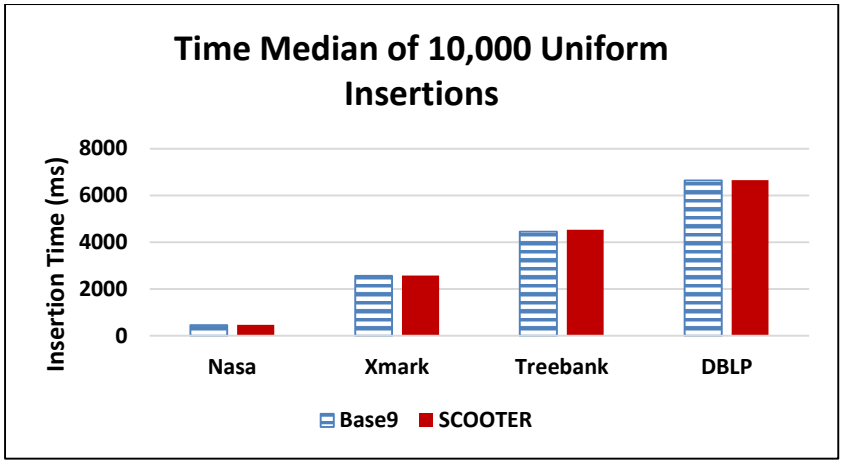
Appendix B.2 Handling Insertions Statistical Results

- **Uniform insertion time comparisons:**

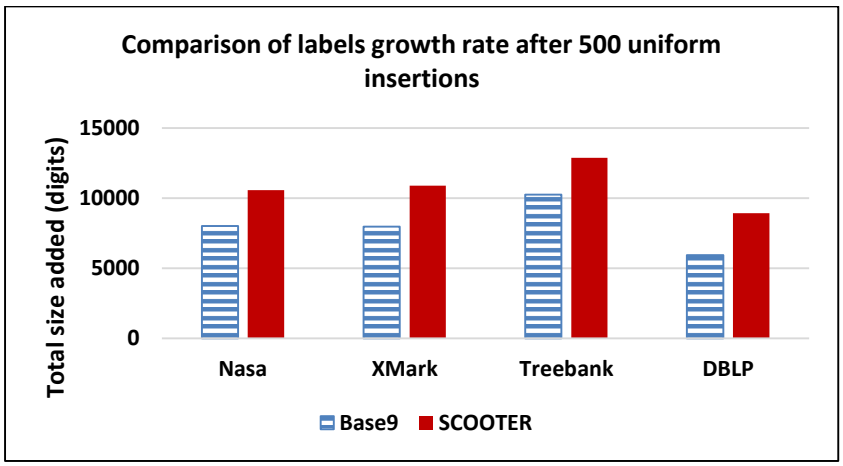


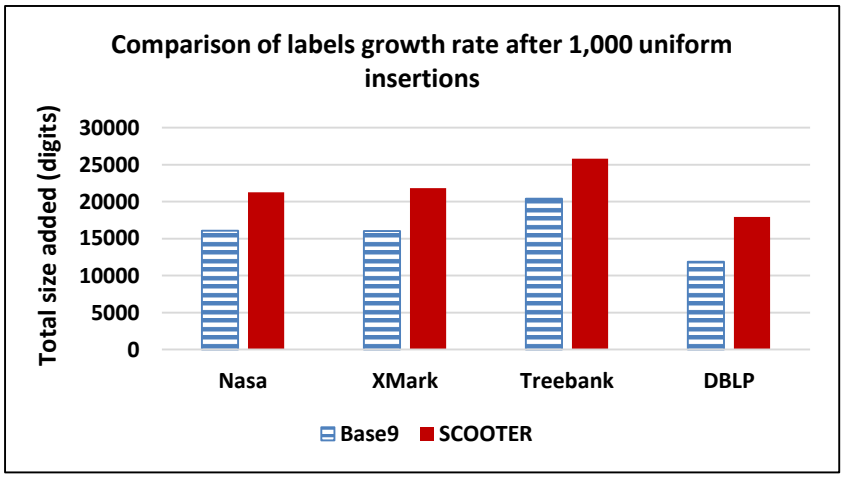


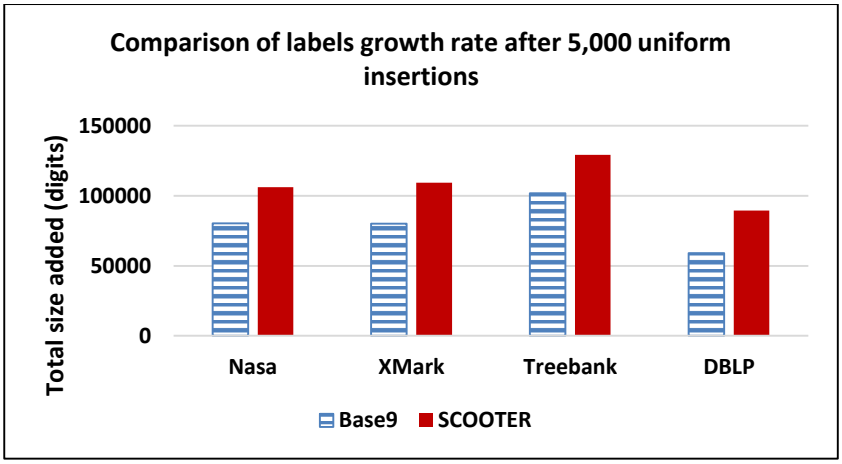


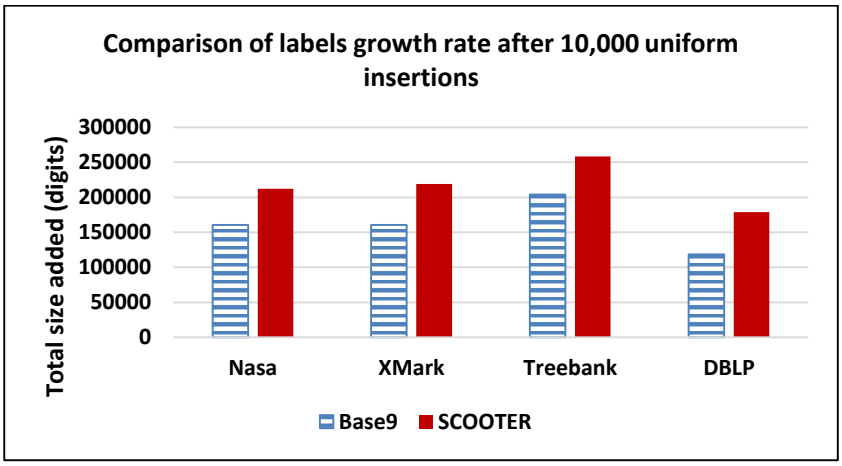


- **Uniform insertion – label size comparisons:**

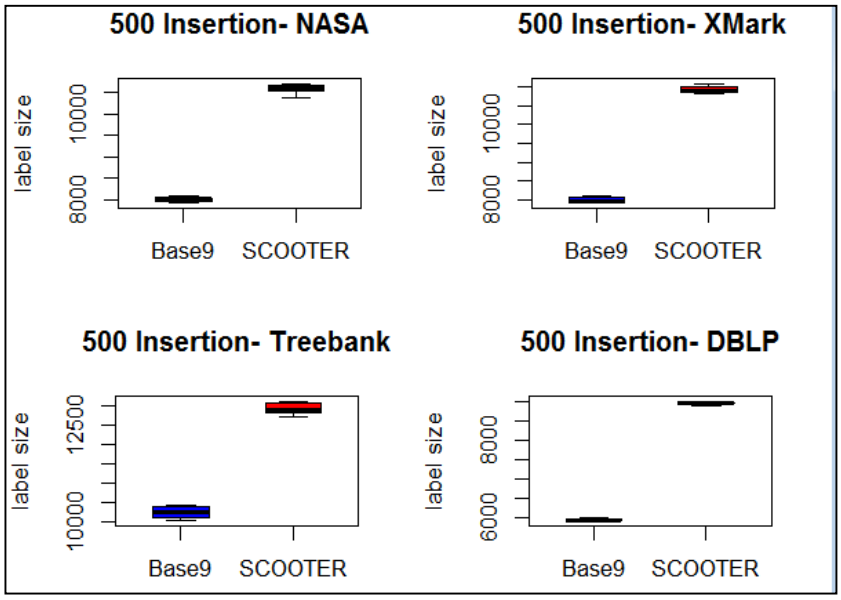


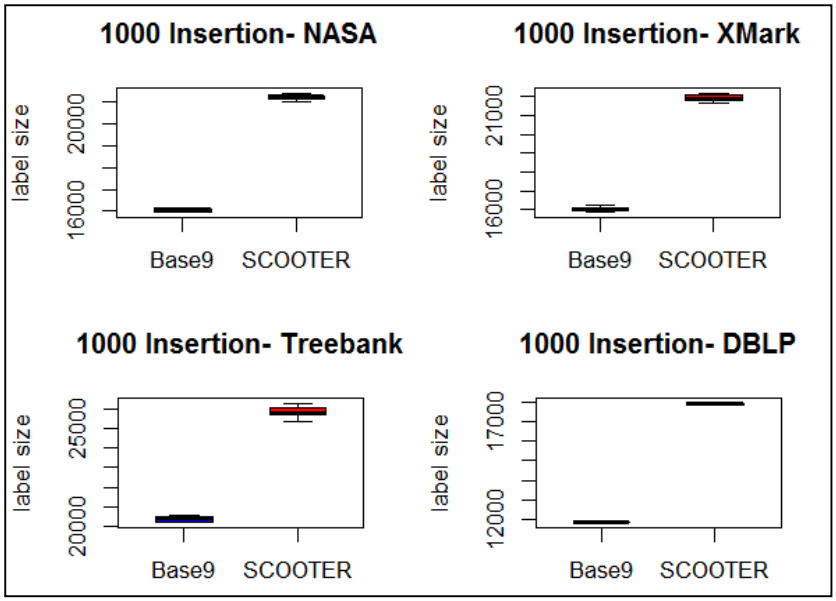


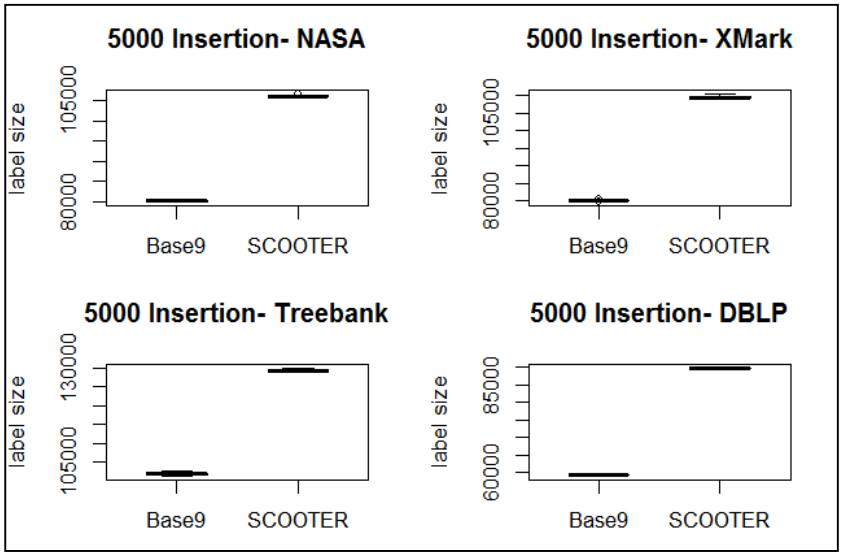


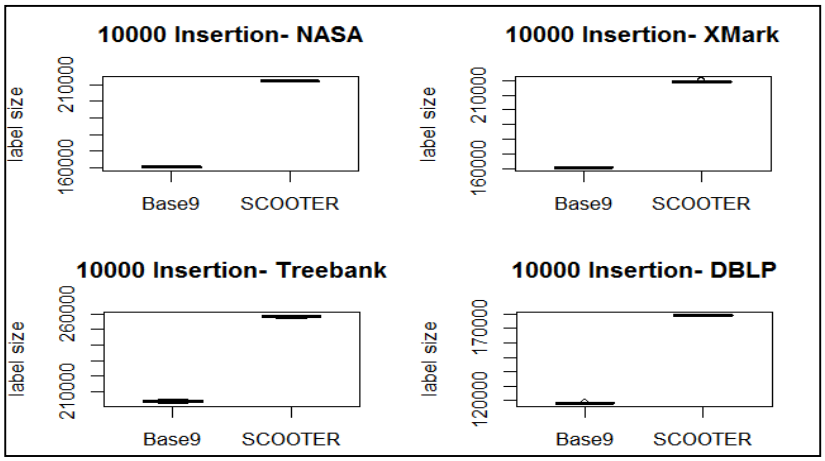


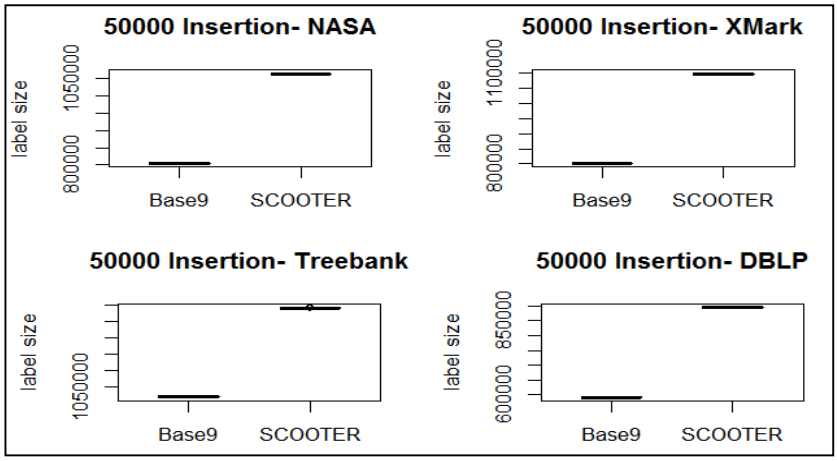
- **Box plot distribution of label size after uniform insertions:**



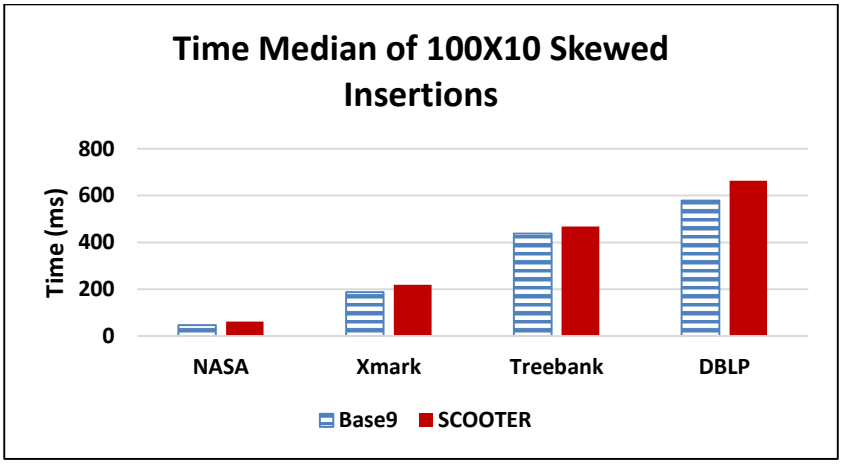

1000 Insertion- NASA
label size
16000
20000
Base9
SCOOTER
1000 Insertion- XMark
label size
16000
21000
Base9
SCOOTER
1000 Insertion- Treebank
label size
20000
25000
Base9
SCOOTER
1000 Insertion- DBLP
label size
12000
17000
Base9
SCOOTER

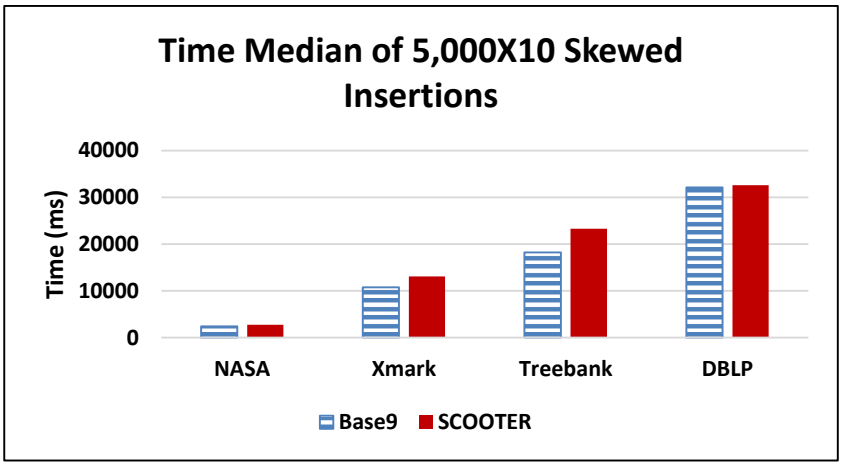

5000 Insertion- NASA
label size
80000
105000
Base9
SCOOTER
5000 Insertion- XMark
label size
80000
105000
Base9
SCOOTER
5000 Insertion- Treebank
label size
105000
130000
Base9
SCOOTER
5000 Insertion- DBLP
label size
60000
85000
Base9
SCOOTER


10000 Insertion- NASA
label size
160000
210000
Base9
SCOOTER
10000 Insertion- XMark
label size
160000
210000
Base9
SCOOTER
10000 Insertion- Treebank
label size
210000
260000
Base9
SCOOTER
10000 Insertion- DBLP
label size
120000
170000
Base9
SCOOTER

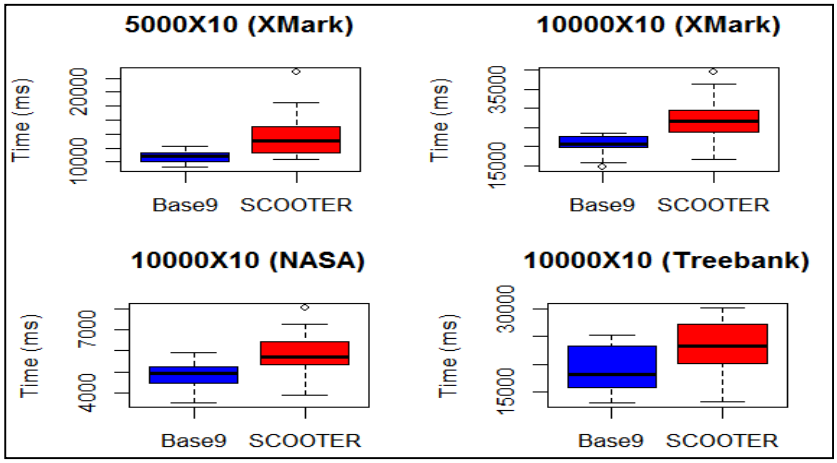


- **Skewed insertion time comparisons:**

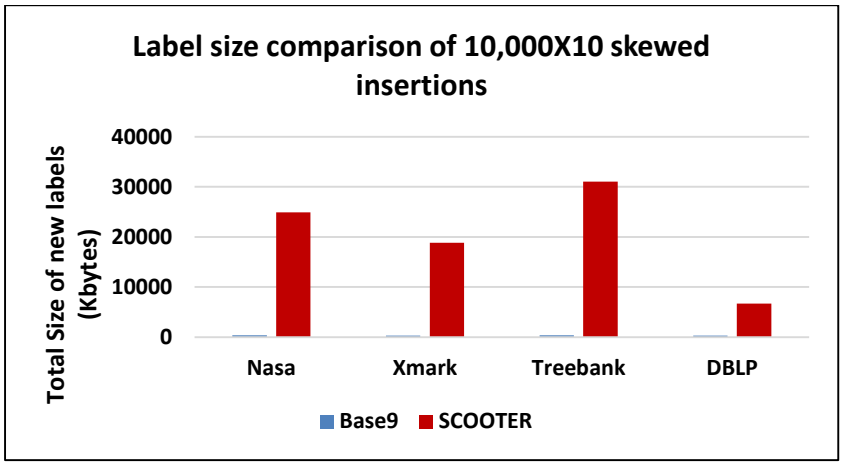




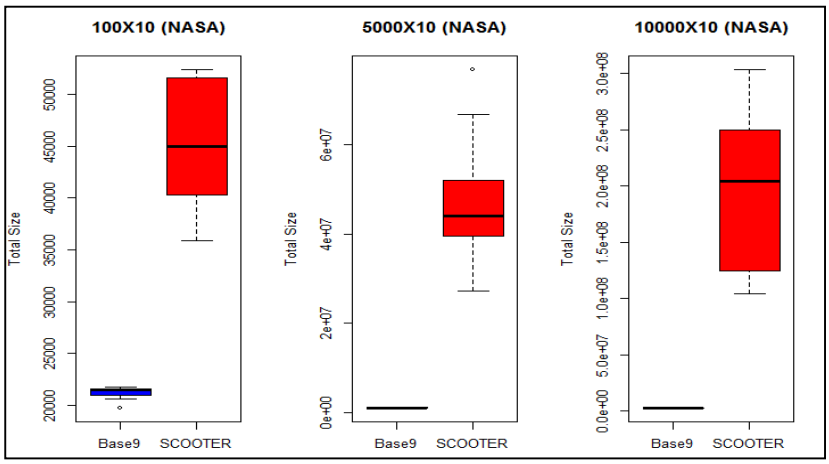
- **Box plot distribution of time in skewed insertions cases with $p < 0.05$**

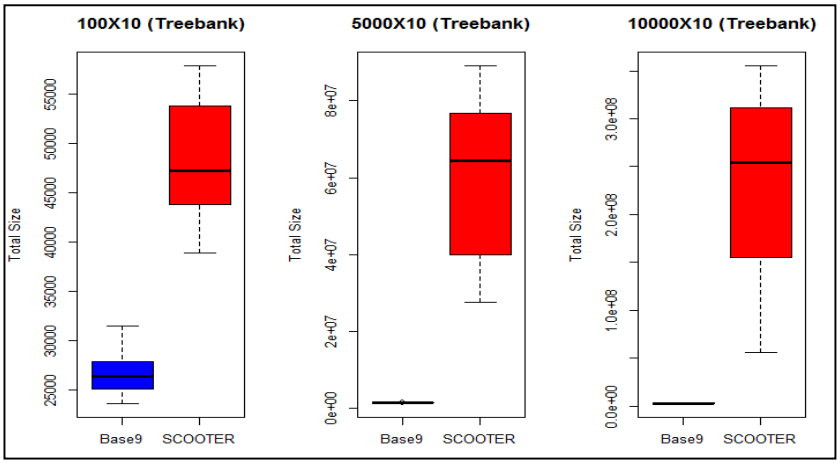


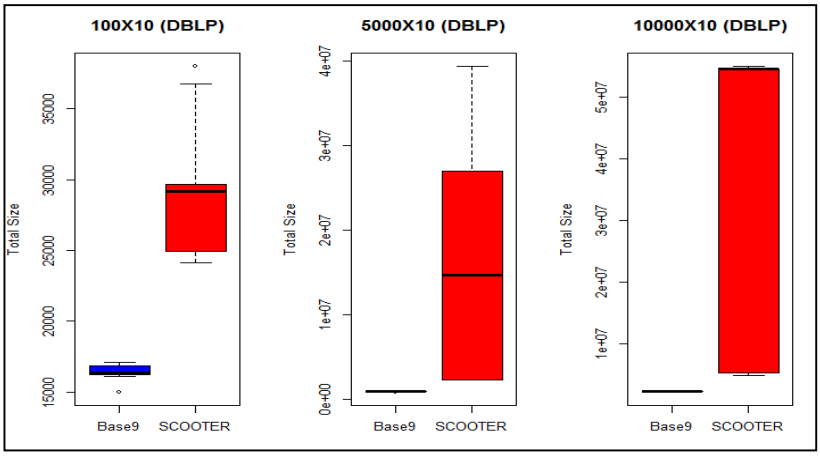
- **Skewed insertion - labels size comparisons:**



- **Box plot distribution of label size after skewed insertions:**

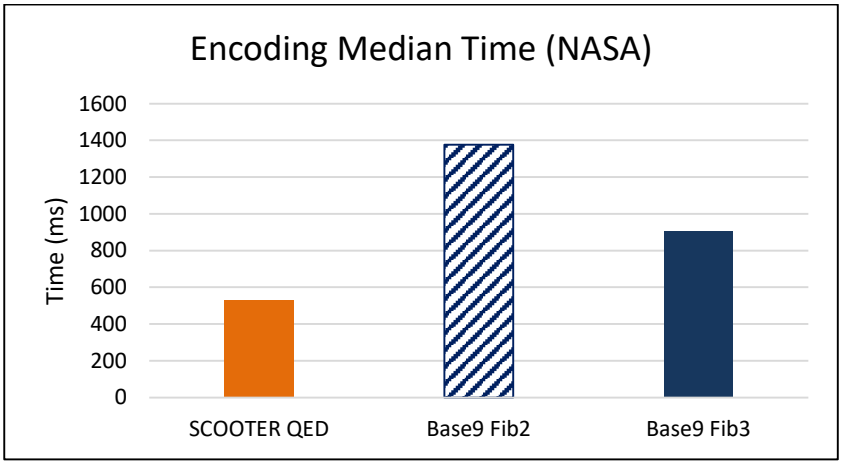


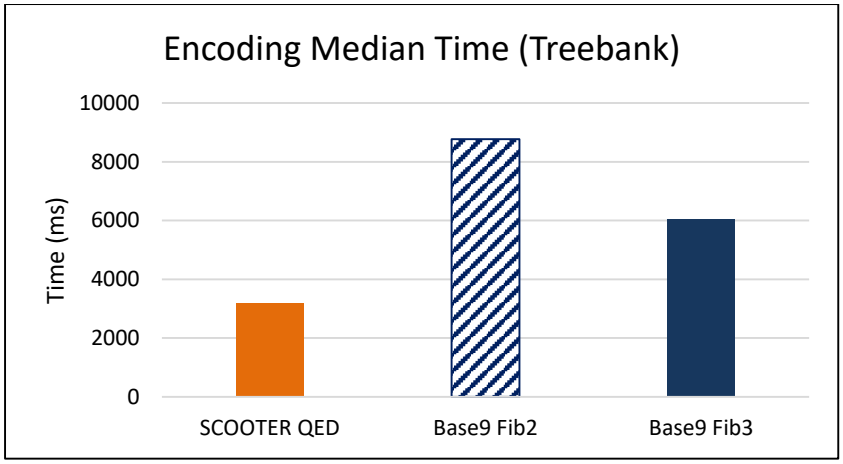


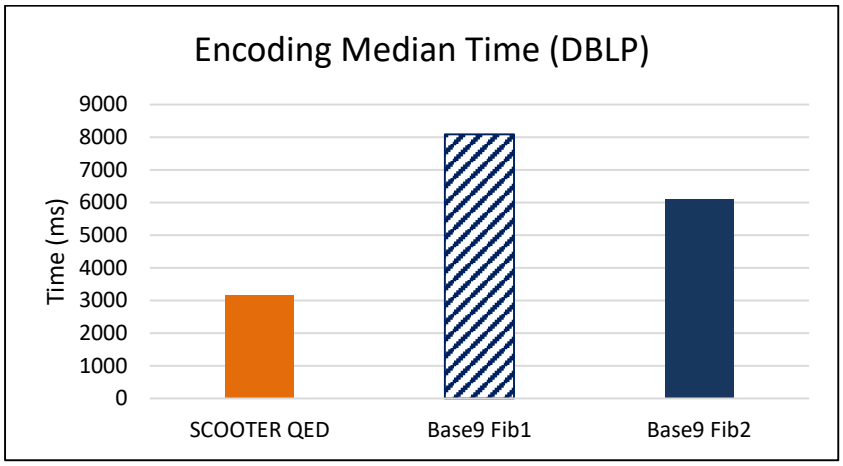


Appendix B.3 Encoding labels

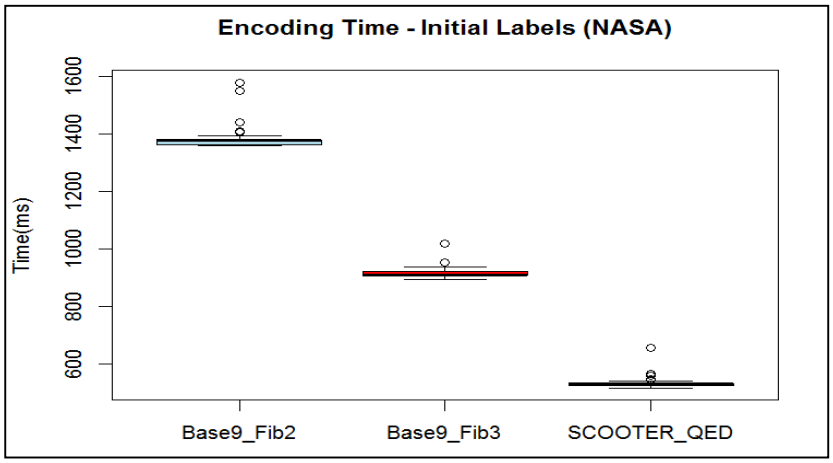
- **Encoding time comparisons (initial labels):**

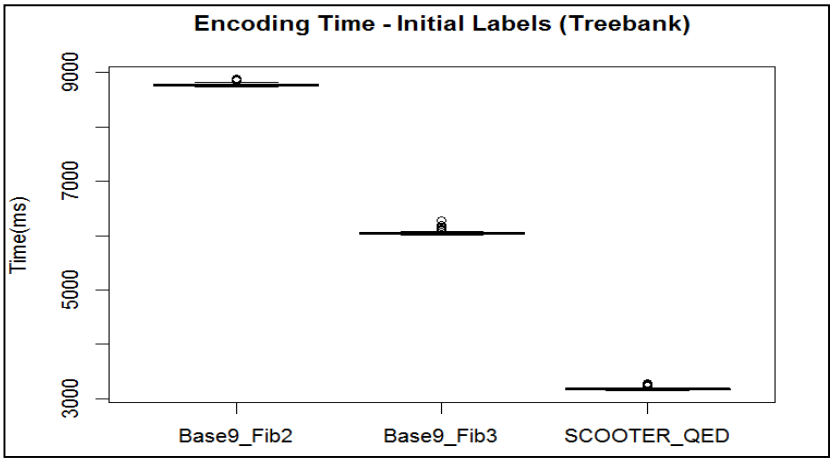


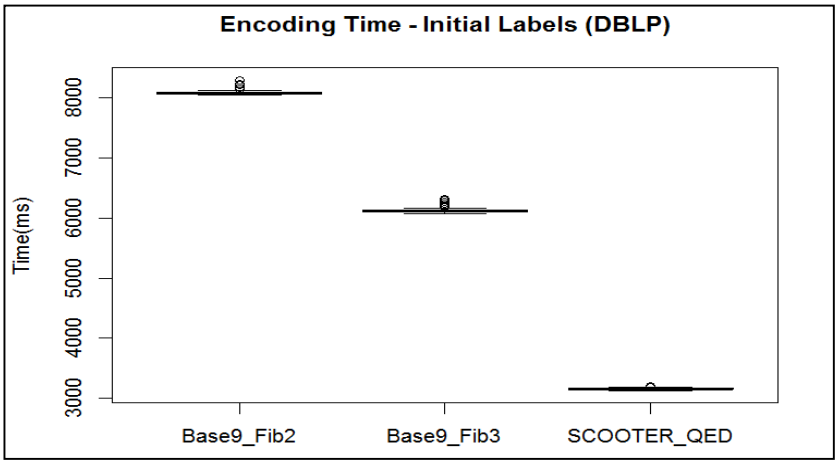




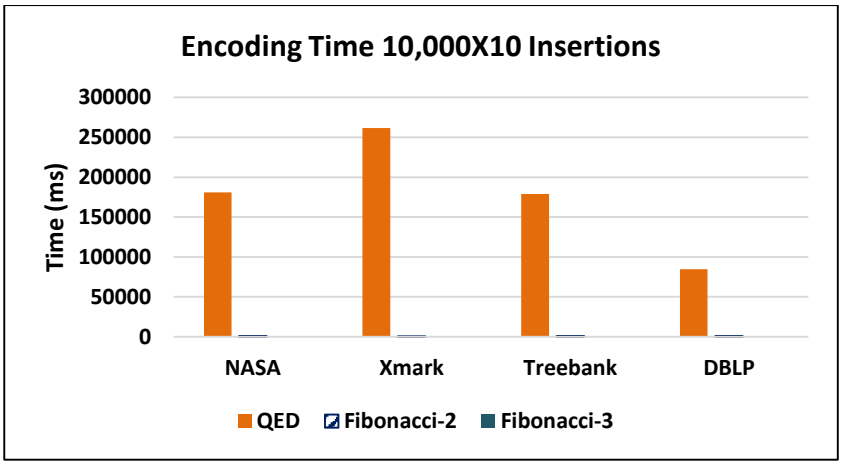
- **Box plot distribution of initial labels' encoding time:**



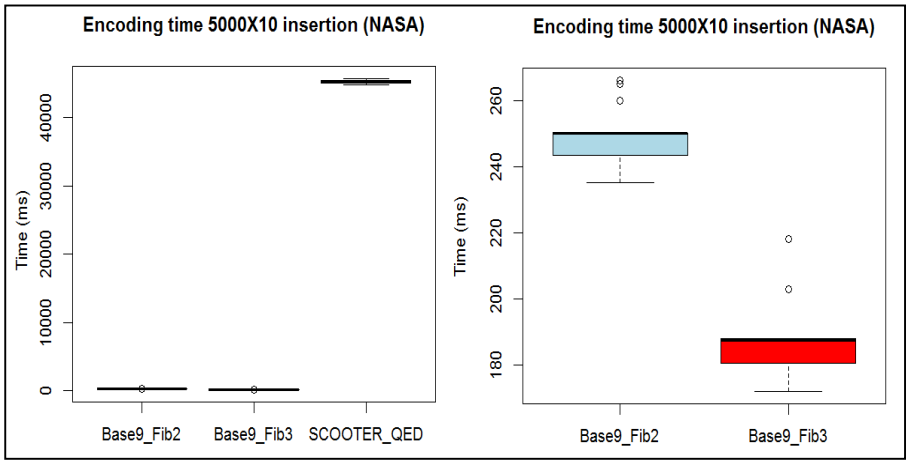


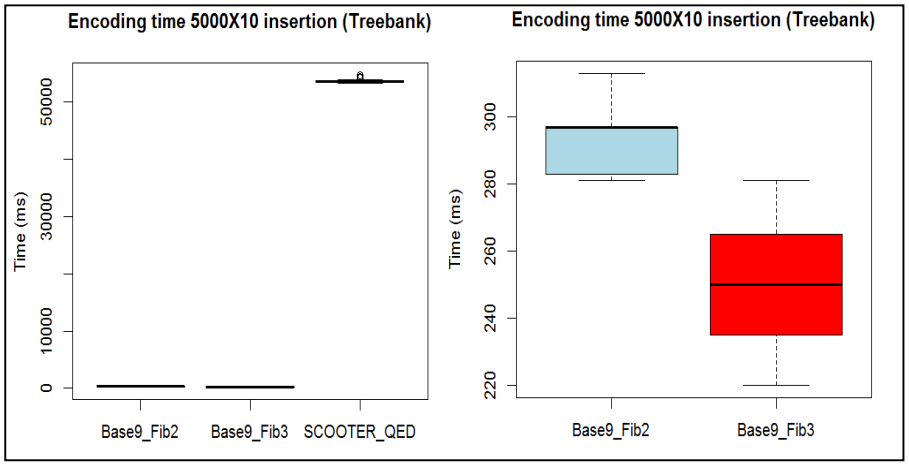


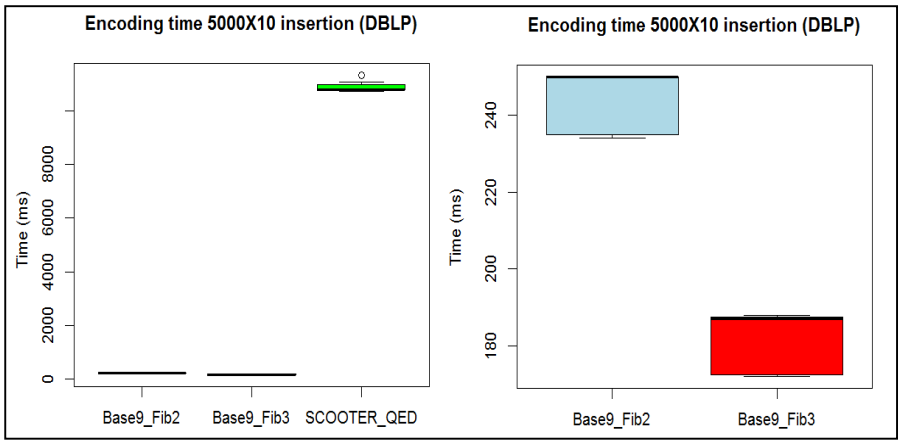
- **Encoding Median time comparison after insertion:**

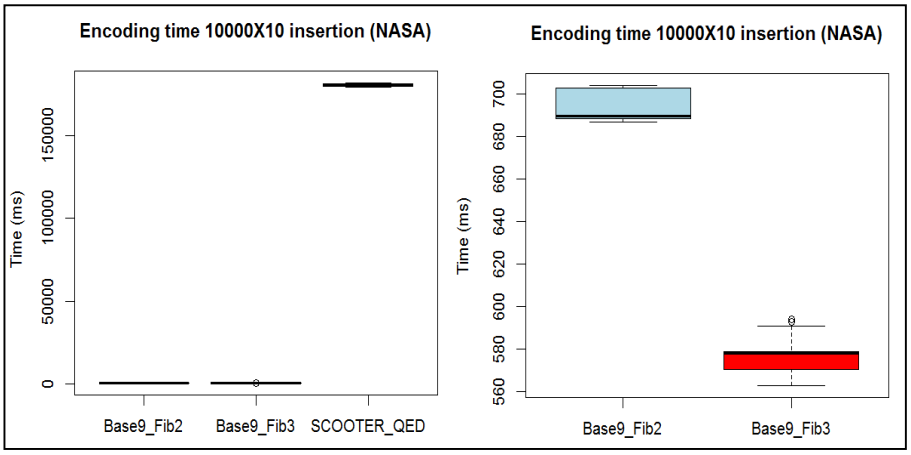


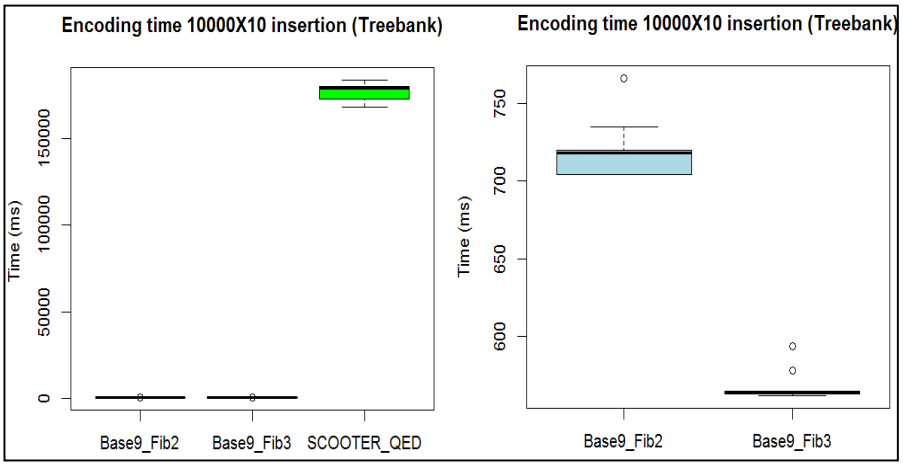
- **Boxplot distribution of encoding time after insertion:**

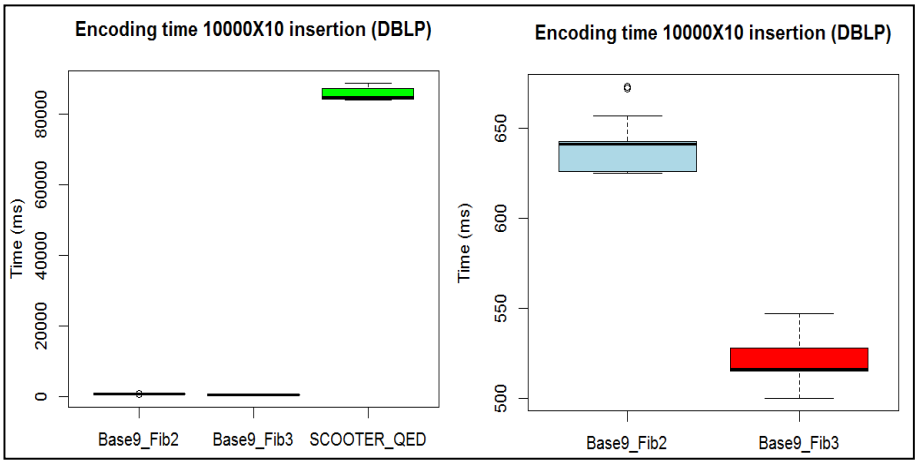


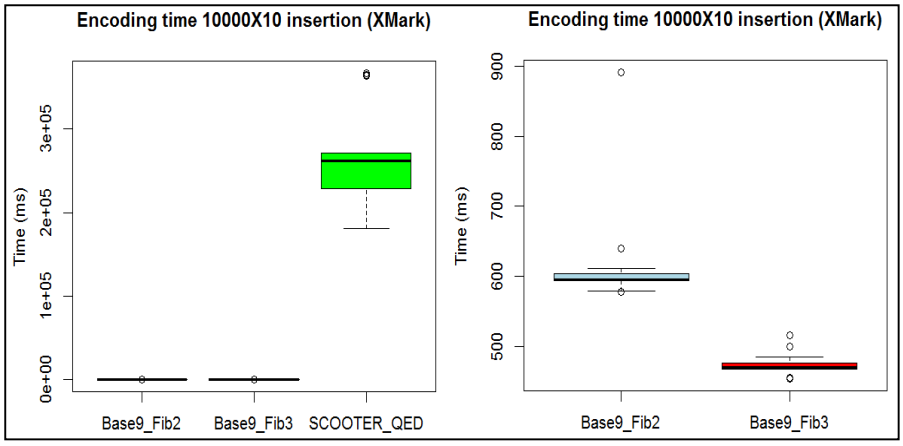





Encoding time 10000X10 insertion (NASA)
Time (ms)
0
50000
100000
150000
Base9_Fib2
Base9_Fib3
SCOOTER_QED
Encoding time 10000X10 insertion (NASA)
Time (ms)
560
580
600
620
640
660
680
700
Base9_Fib2
Base9_Fib3


Encoding time 10000X10 insertion (Treebank)
Time (ms)
0
50000
100000
150000
Base9_Fib2
Base9_Fib3
SCOOTER_QED
Encoding time 10000X10 insertion (Treebank)
Time (ms)
600
650
700
750
Base9_Fib2
Base9_Fib3

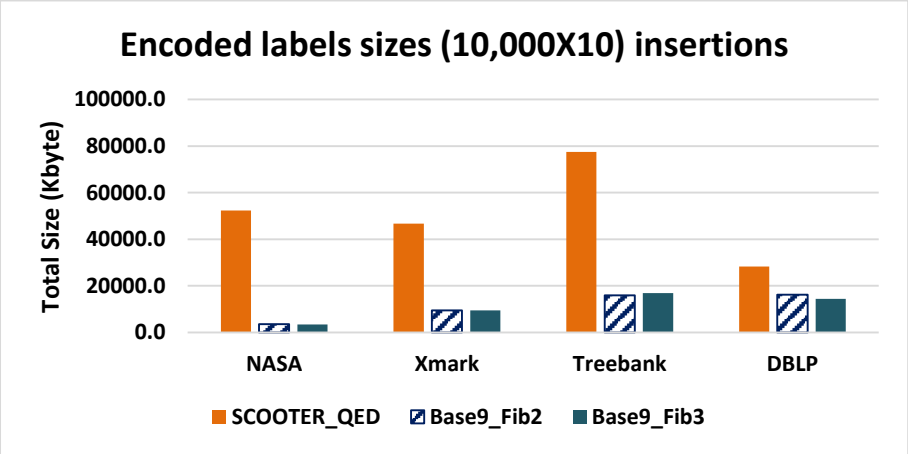

Encoding time 10000X10 insertion (DBLP)
Time (ms)
0
20000
40000
60000
80000
Base9_Fib2
Base9_Fib3
SCOOTER_QED
Encoding time 10000X10 insertion (DBLP)
Time (ms)
500
550
600
650
Base9_Fib2
Base9_Fib3



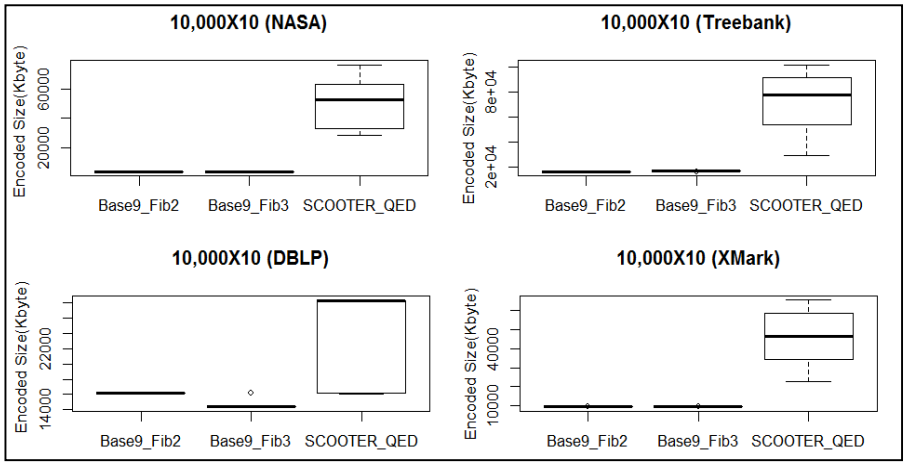
- **Percentage difference of encoding time (after large skewed insertions) between encoding methods**

	5,000X10 insertion			10,000X10 insertion		
Datasets	**QED vs Fib2**	**QED vs Fib3**	**Fib2 vs Fib3**	**QED vs Fib2**	**QED vs Fib3**	**Fib2 vs Fib3**
NASA	99.45%	99.59%	25.20%	99.62%	99.68%	16.17%
XMark	99.10%	99.29%	21.21%	99.77%	99.82%	21.09%
Treebank	99.45%	99.53%	15.82%	99.60%	99.68%	21.45%
DBLP	97.68%	98.27%	25.20%	99.24%	99.39%	19.50%

- **Encoded label size comparison after insertions:**



- **Boxplot distribution of encoded labels size after insertion:**



Appendix B.4 Relationships determination

- **Statistics description of each relationship determination time before insertion (Treebank) – using SPSS**

Statistics

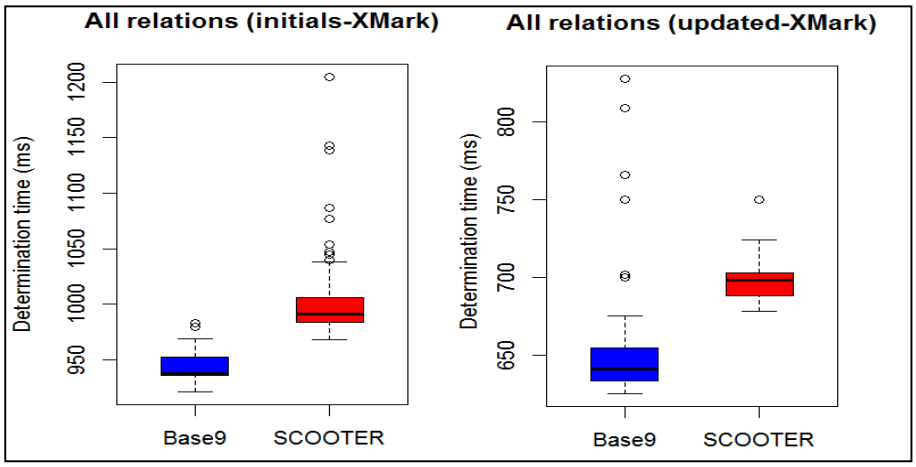
Scheme			PC	sib	AD	LCA	DO
SCOOTER	N	Valid	100	100	100	100	100
		Missing	0	0	0	0	0
	Mean		123.0000	117.7200	82.7400	132.8500	13.4100
	Median		124.0000	110.0000	79.0000	139.0000	16.0000
	Std. Deviation		45.66169	38.66329	37.04778	39.01473	5.89486
Base9	N	Valid	100	100	100	100	100
		Missing	0	0	0	0	0
	Mean		106.9500	104.5200	71.0700	123.3400	13.7700
	Median		108.5000	95.5000	63.5000	122.5000	16.0000
	Std. Deviation		38.55532	37.32183	29.91545	39.79930	5.53018

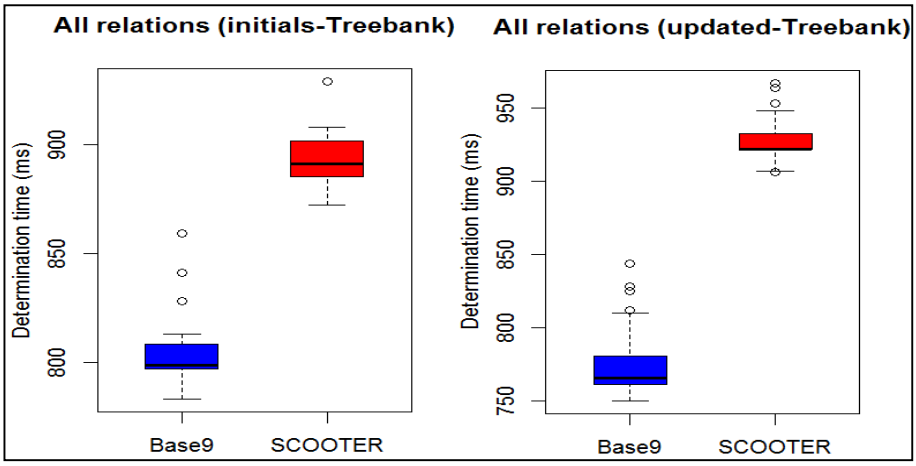
- **Statistics description of each relationship determination time after insertion (Treebank) – using SPSS**

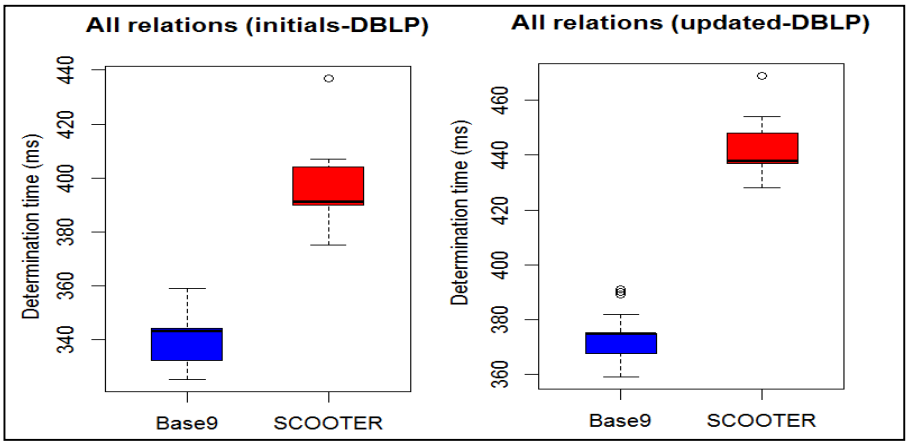
Statistics

Scheme			PC_af	sib_af	AD_af	LCA_af	DO_af
SCOOTER	N	Valid	100	100	100	100	100
		Missing	0	0	0	0	0
	Mean		77.0600	385.6200	71.5500	258.940	21.0600
	Median		77.0000	390.5000	64.0000	253.000	16.0000
	Std. Deviation		32.02739	66.99054	31.04066	51.9384	8.01365
Base9	N	Valid	100	100	100	100	100
		Missing	0	0	0	0	0
	Mean		58.4200	329.010	61.5500	225.630	20.8400
	Median		61.0000	335.000	62.5000	233.000	16.0000
	Std. Deviation		30.0767	63.2543	31.0618	50.0074	7.39058

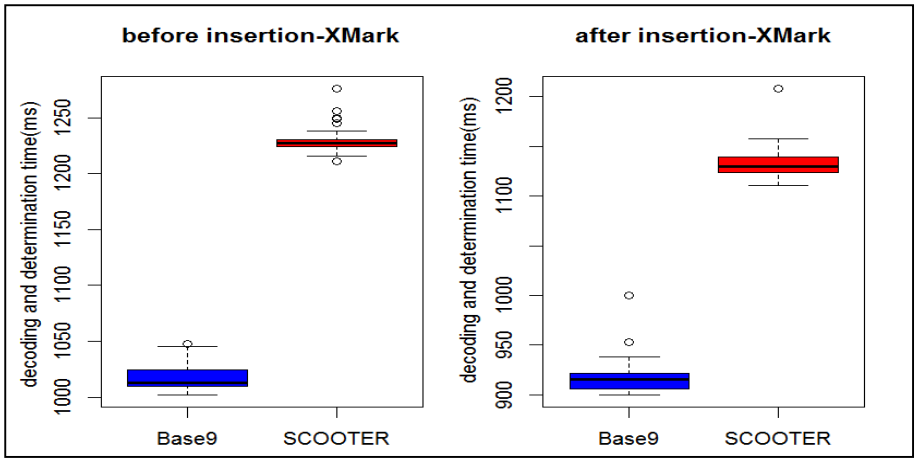
- **Boxplot distribution of determination time (all relations after decoding) before and after insertion**

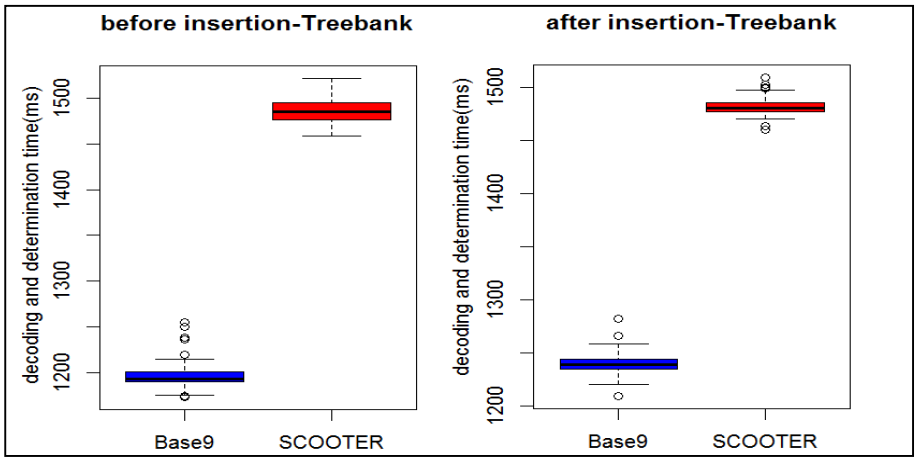


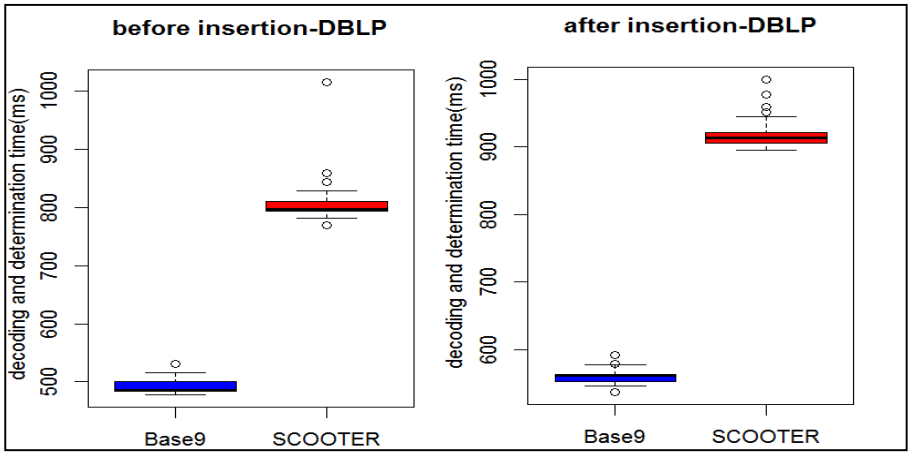

All relations (initials-Treebank)
Determination time (ms)
800
850
900
Base9
SCOOTER
All relations (updated-Treebank)
Determination time (ms)
750
800
850
900
950
Base9
SCOOTER


All relations (initials-DBLP)
Determination time (ms)
340
360
380
400
420
440
Base9
SCOOTER
All relations (updated-DBLP)
Determination time (ms)
360
380
400
420
440
460
Base9
SCOOTER

- **Boxplot distribution of decoding and determination time (combined) before and after insertion:**

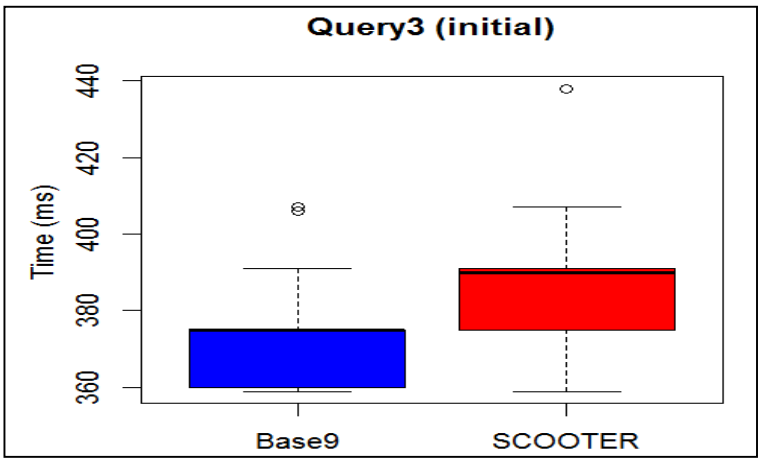


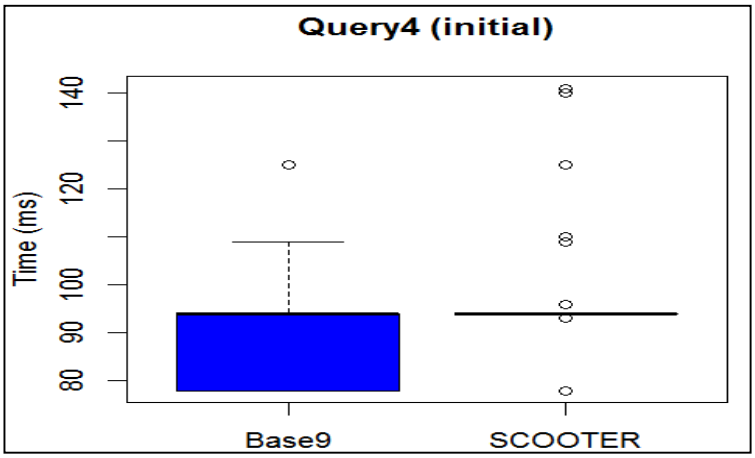


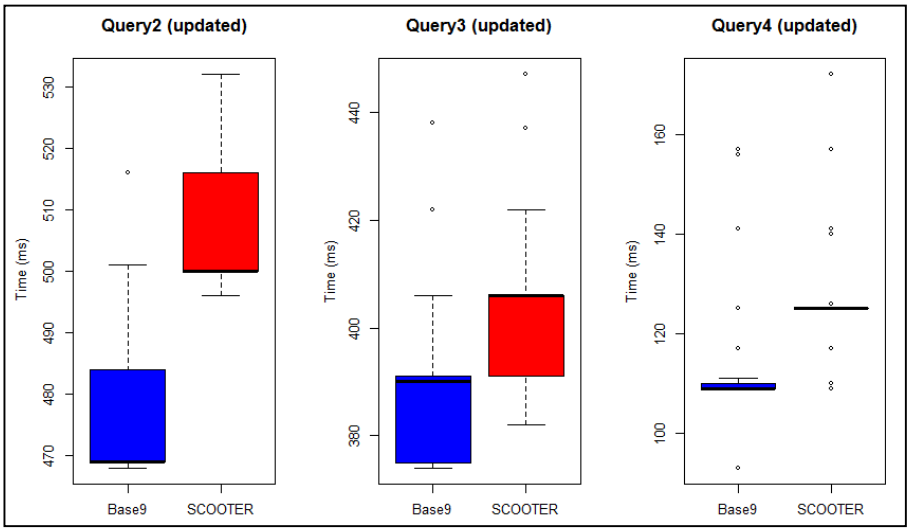


Appendix B.6 Query performance statistic

- **Boxplot distribution query response time**







Appendix B.7 XML labels compression – prefix encodings: statistic

- **Encoding times – Dewey labels**

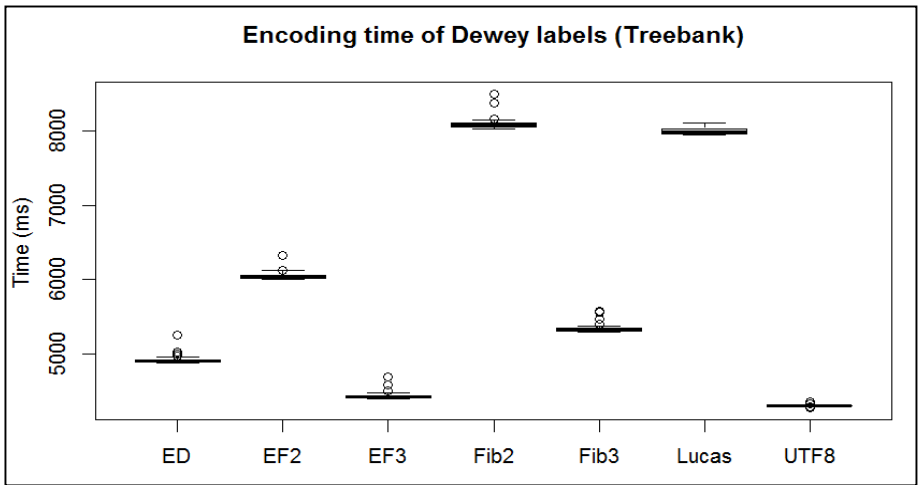
Statistics of Dewey labels encoding times					
Method			**NASA**	**Treebank**	**DBLP**
Fib2	N	Valid	50	50	50
		Missing	0	0	0
	Mean		1168.1200	8099.2000	7323.8600
	Median		1165.5000	8080.0000	7306.0000
	Std. Deviation		10.12703	77.54972	71.23259
Fib3	N	Valid	50	50	50
		Missing	0	0	0
	Mean		785.5600	5341.4000	5753.8800
	Median		782.0000	5325.5000	5732.0000
	Std. Deviation		12.34233	56.80902	99.48616
Lucas	N	Valid	50	50	50
		Missing	0	0	0
	Mean		1190.9600	8007.8000	7926.0600
	Median		1188.0000	7986.0000	7907.0000
	Std. Deviation		20.41804	44.31796	71.37610
ED	N	Valid	50	50	50
		Missing	0	0	0
	Mean		765.5000	4923.1600	3105.8200
	Median		735.0000	4907.0000	3095.0000
	Std. Deviation		58.77256	58.83819	32.65696
EF2	N	Valid	50	50	50
		Missing	0	0	0
	Mean		879.2400	6050.2200	4164.7000
	Median		876.0000	6033.0000	4147.5000
	Std. Deviation		11.78300	50.53925	56.17220
EF3	N	Valid	50	50	50
		Missing	0	0	0
	Mean		686.2400	4432.6200	3269.0200
	Median		688.0000	4423.0000	3267.0000
	Std. Deviation		20.05661	48.90594	25.45223
UTF8	N	Valid	50	50	50
		Missing	0	0	0
	Mean		613.4600	4301.3200	3788.9000
	Median		610.0000	4298.0000	3782.0000
	Std. Deviation		10.69085	16.65428	18.61670

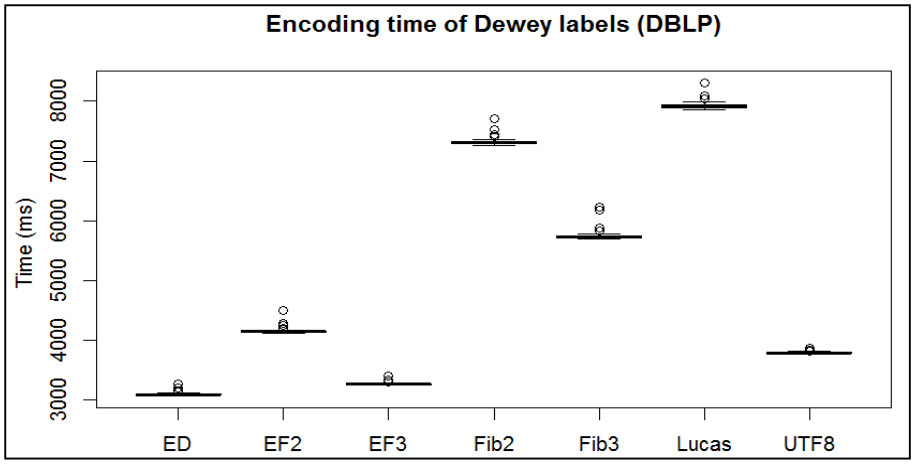
- **Encoding times – SCOOTER labels**

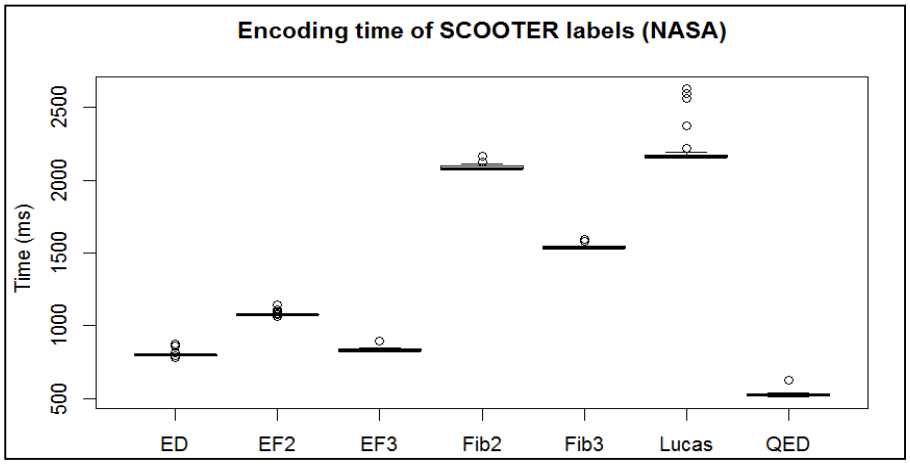
Statistics					
Method			**NASA**	**Treebank**	**DBLP**
Fib2	N	Valid	50	50	50
		Missing	0	0	0
	Mean		2086.4400	13502.0400	14549.0200
	Median		2079.5000	13483.5000	14533.0000
	Std. Deviation		14.95184	78.87939	63.10357
Fib3	N	Valid	50	50	50
		Missing	0	0	0
	Mean		1537.9400	9947.0600	12435.7400
	Median		1533.0000	9938.0000	12409.5000

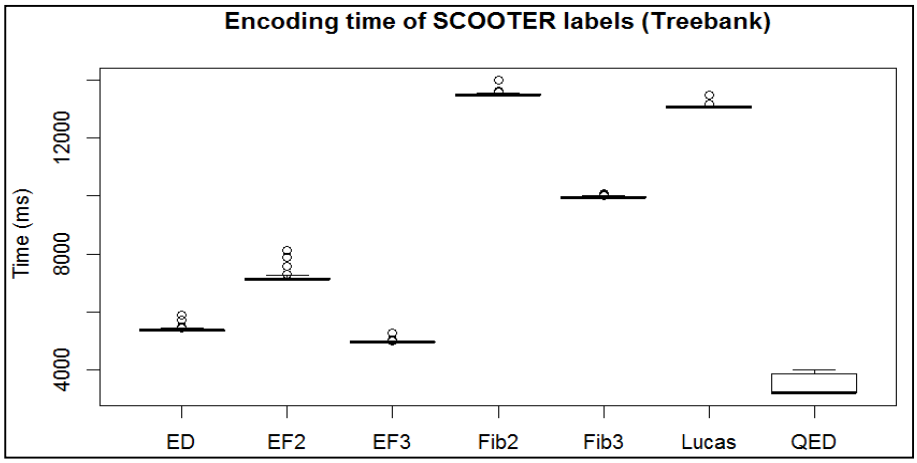
	Std. Deviation		12.14943	34.19775	86.98435
Lucas	N	Valid	50	50	50
		Missing	0	0	0
	Mean		2210.8400	13060.9200	15697.8600
	Median		2158.0000	13047.0000	15688.0000
	Std. Deviation		135.52714	65.18008	85.38747
ED	N	Valid	50	50	50
		Missing	0	0	0
	Mean		799.1600	5403.0800	3824.9600
	Median		797.0000	5376.0000	3642.0000
	Std. Deviation		16.79208	92.42092	372.82616
EF2	N	Valid	50	50	50
		Missing	0	0	0
	Mean		1080.7000	7208.0600	5132.0000
	Median		1079.0000	7141.0000	5126.0000
	Std. Deviation		14.86847	208.81184	24.23693
EF3	N	Valid	50	50	50
		Missing	0	0	0
	Mean		833.7800	4987.8200	4193.3600
	Median		829.0000	4970.0000	4188.0000
	Std. Deviation		10.51198	48.37215	36.39172
QED	N	Valid	50	50	50
		Missing	0	0	0
	Mean		526.0200	3472.3800	3182.2600
	Median		526.0000	3225.0000	3144.5000
	Std. Deviation		15.88413	337.48070	138.31003

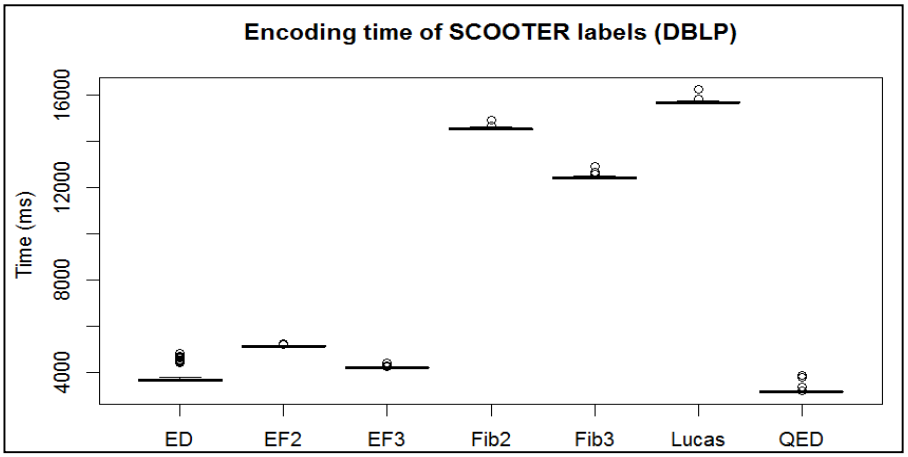
- **Encoding times – boxplots distribution:**




Encoding time of Dewey labels (DBLP)
Time (ms)
3000
4000
5000
6000
7000
8000
ED
EF2
EF3
Fib2
Fib3
Lucas
UTF8


Encoding time of SCOOTER labels (NASA)
Time (ms)
500
1000
1500
2000
2500
ED
EF2
EF3
Fib2
Fib3
Lucas
QED


Encoding time of SCOOTER labels (Treebank)
Time (ms)
4000
8000
12000
ED
EF2
EF3
Fib2
Fib3
Lucas
QED



- **Decoding times – Dewey labels**

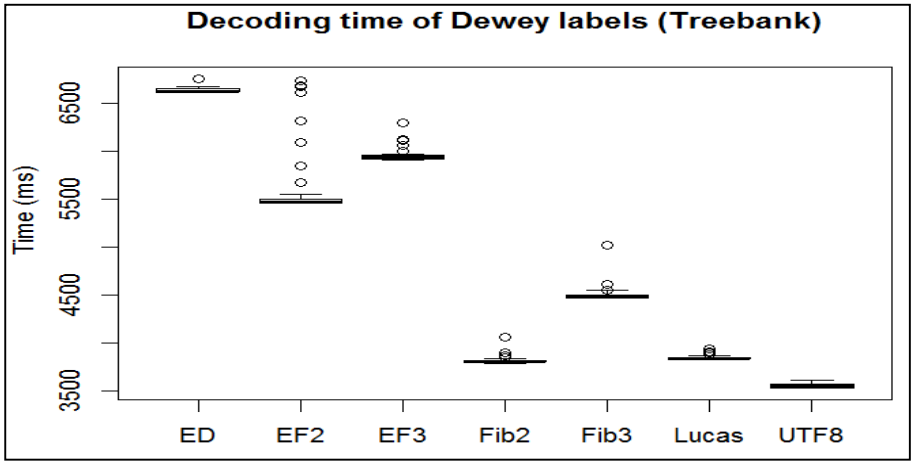
Statistics					
Method			**NASA**	**Treebank**	**DBLP**
Fib2	N	Valid	50	50	50
		Missing	0	0	0
	Median		532.0000	3798.0000	2798.0000
Fib3	N	Valid	50	50	50
		Missing	0	0	0
	Median		672.0000	4485.0000	3298.0000
Lucas	N	Valid	50	50	50
		Missing	0	0	0
	Median		547.0000	3830.0000	2908.0000
ED	N	Valid	50	50	50
		Missing	0	0	0
	Median		1079.0000	6626.0000	6298.0000
EF2	N	Valid	50	50	50
		Missing	0	0	0
	Median		876.0000	5470.0000	5407.0000
EF3	N	Valid	50	50	50
		Missing	0	0	0
	Median		861.0000	5938.0000	4001.0000
UTF8	N	Valid	50	50	50
		Missing	0	0	0
	Median		532.0000	3548.0000	3439.0000

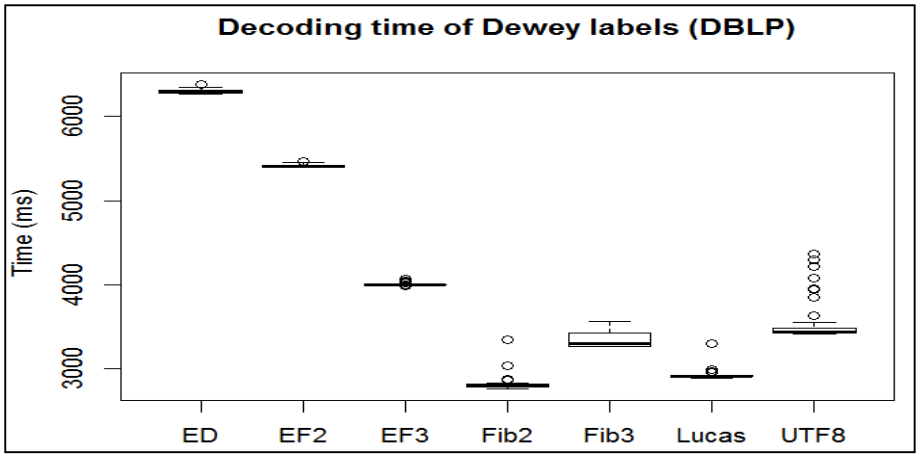
- **Decoding times – SCOOTER labels**

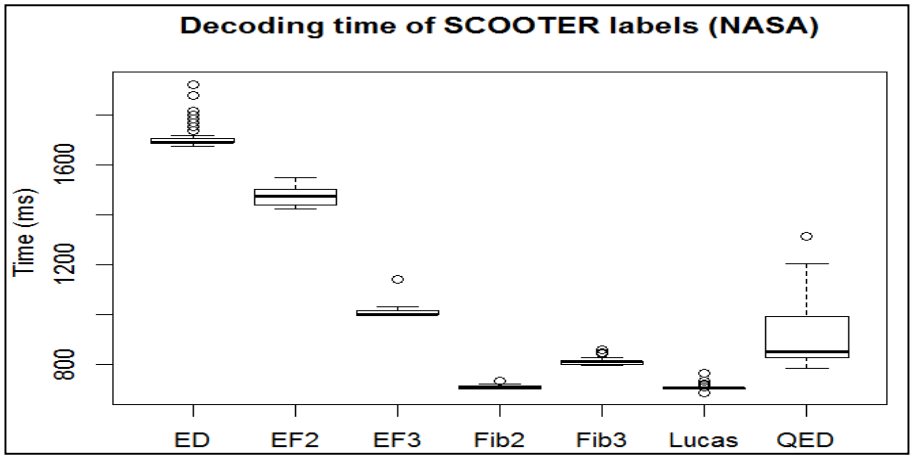
Statistics					
Method			**NASA**	**Treebank**	**DBLP**
Fib2	N	Valid	50	50	50
		Missing	0	0	0
	Median		704.0000	4699.5000	3939.0000
Fib3	N	Valid	50	50	50
		Missing	0	0	0
	Median		813.0000	5373.5000	4251.0000
Lucas	N	Valid	50	50	50
		Missing	0	0	0
	Median		704.0000	4689.0000	4126.0000

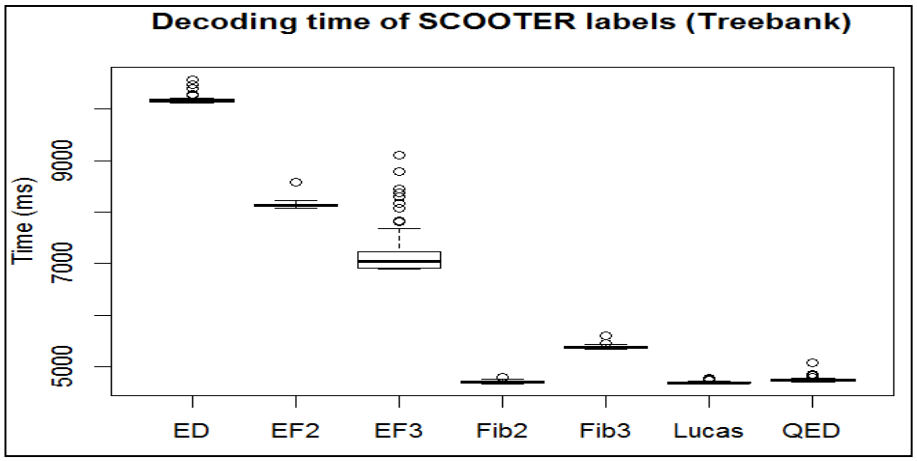
ED	N	Valid	50	50	50
		Missing	0	0	0
	Median		1689.0000	10157.0000	10798.0000
EF2	N	Valid	50	50	50
		Missing	0	0	0
	Median		1472.5000	8126.0000	9720.0000
EF3	N	Valid	50	50	50
		Missing	0	0	0
	Median		1001.0000	7040.0000	4939.0000
QED	N	Valid	50	50	50
		Missing	0	0	0
	Median		851.5000	4730.5000	4861.0000

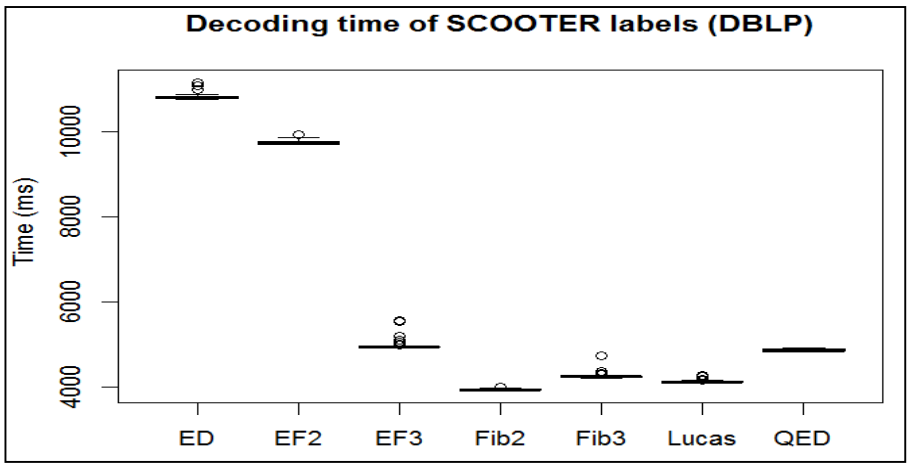
- **Boxplot distribution of decoding time**



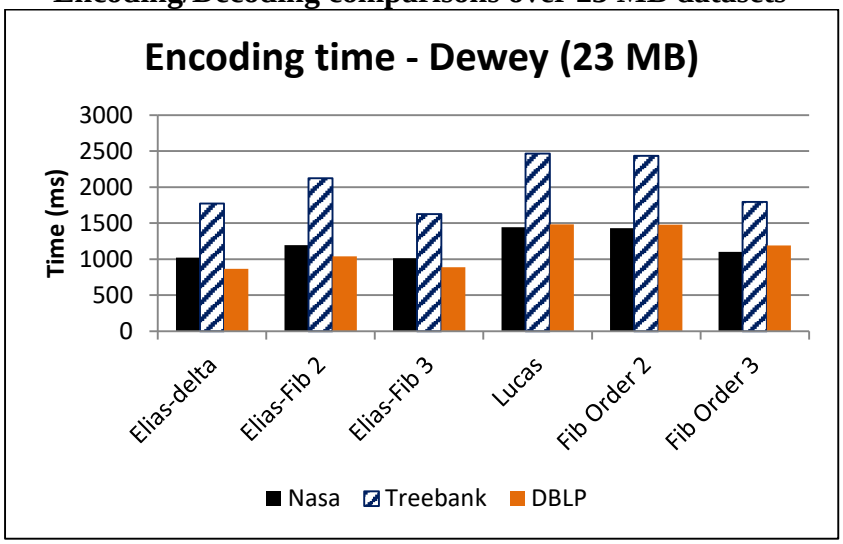


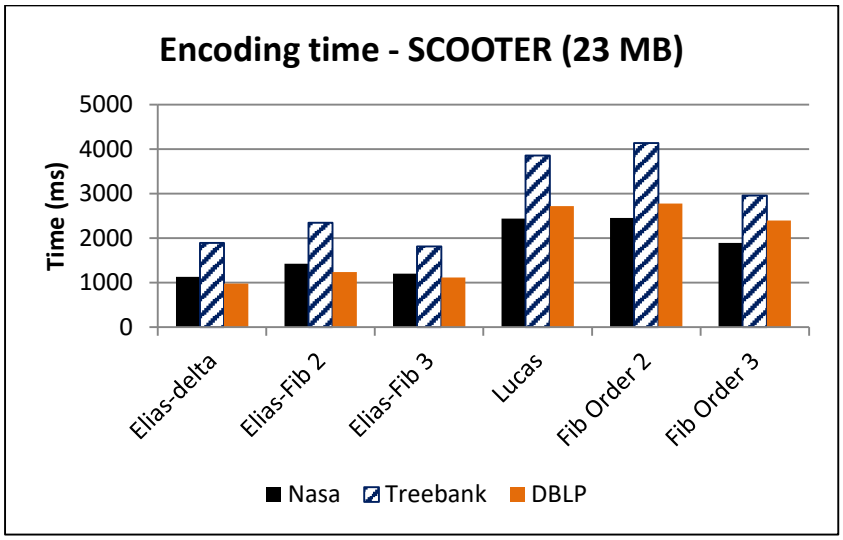

Decoding time of SCOOTER labels (NASA)
Time (ms)
800
1200
1600
ED
EF2
EF3
Fib2
Fib3
Lucas
QED

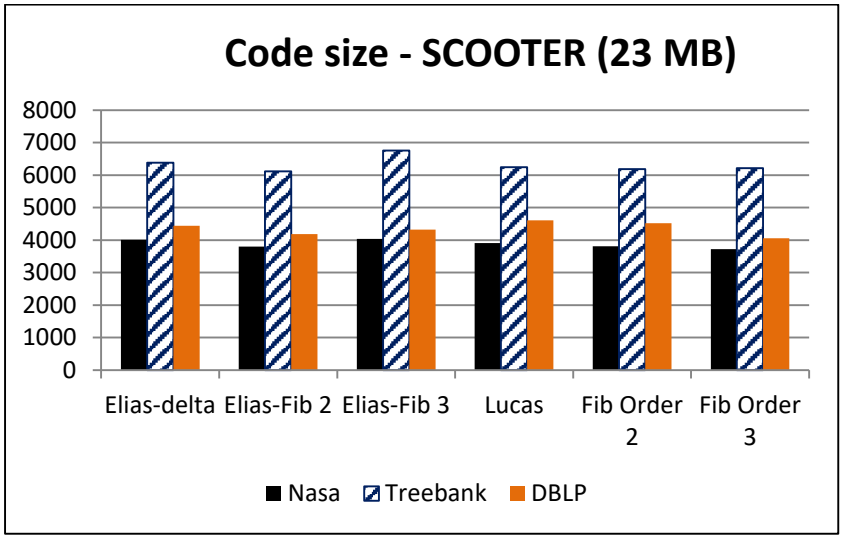

Decoding time of SCOOTER labels (Treebank)
Time (ms)
5000
7000
9000
ED
EF2
EF3
Fib2
Fib3
Lucas
QED

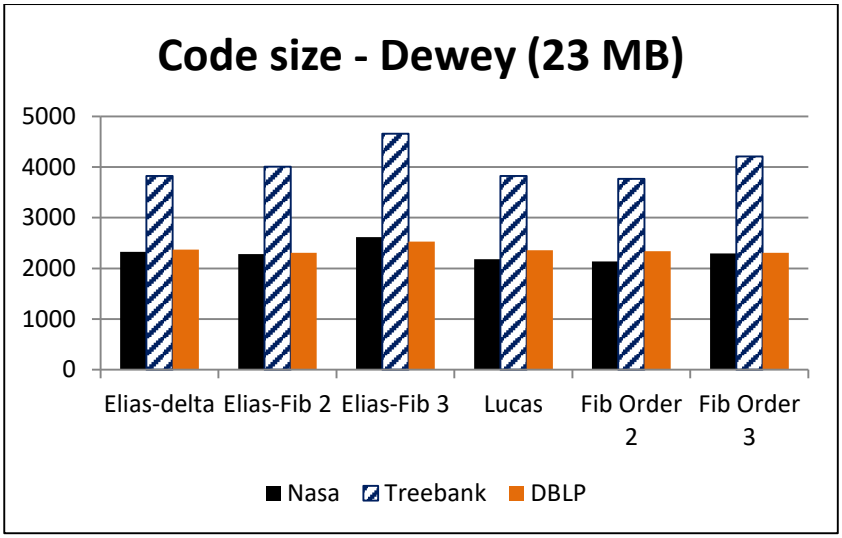

Decoding time of SCOOTER labels (DBLP)
Time (ms)
4000
6000
8000
10000
ED
EF2
EF3
Fib2
Fib3
Lucas
QED

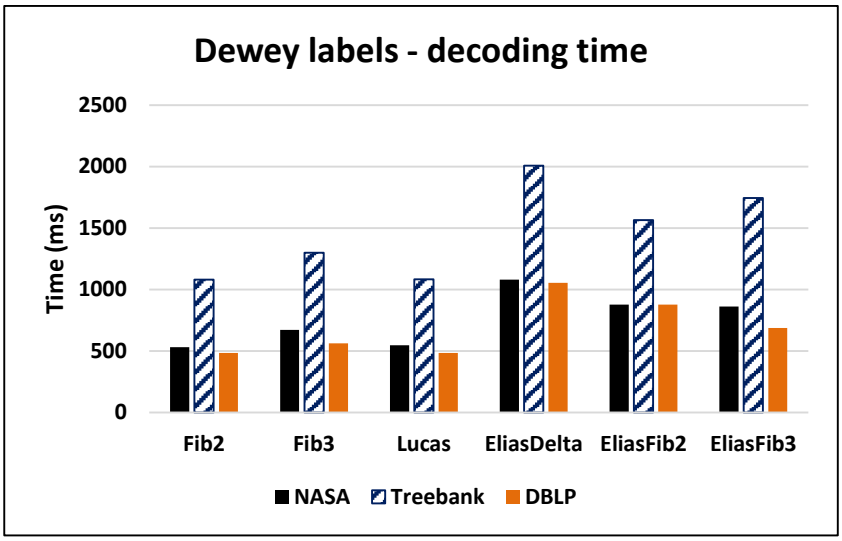
- **Encoding/Decoding comparisons over 23 MB datasets**

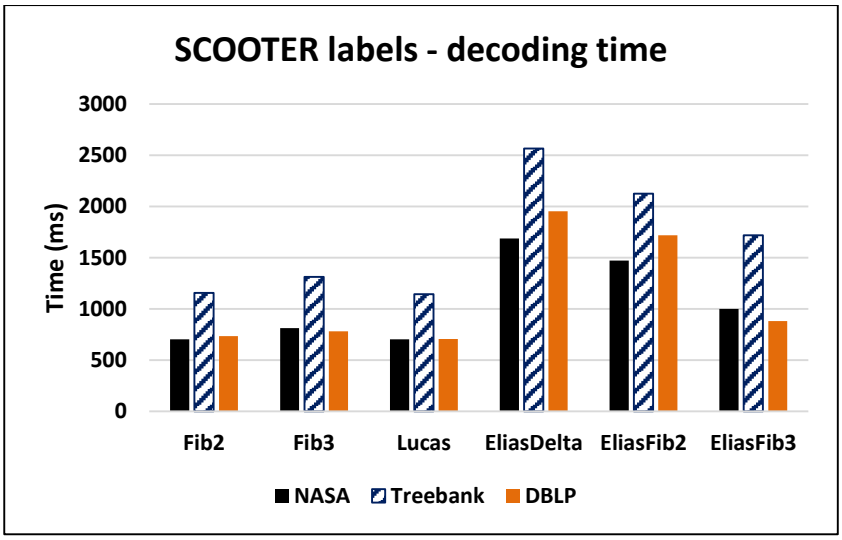




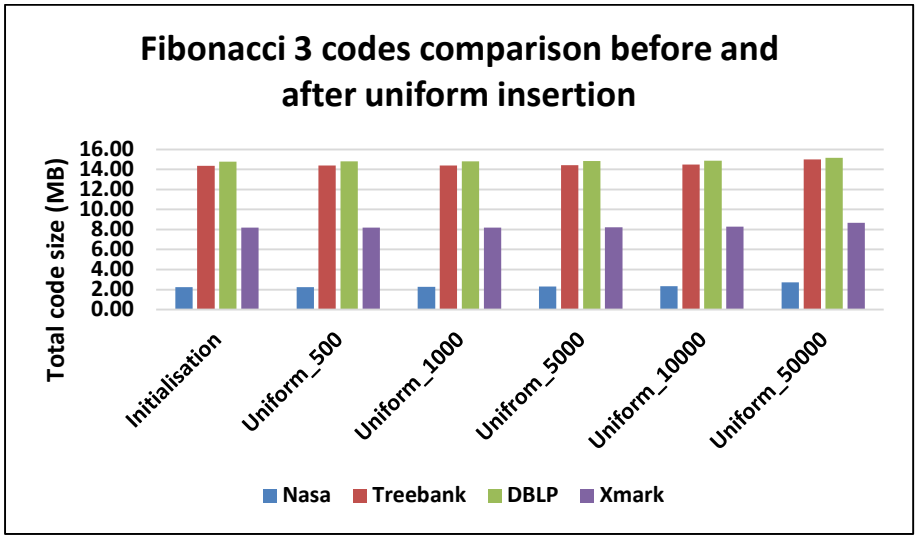



Code size - Dewey (23 MB)
5000
4000
3000
2000
1000
0
Elias-delta
Elias-Fib 2
Elias-Fib 3
Lucas
Fib Order 2
Fib Order 3
Nasa
Treebank
DBLP


Dewey labels - decoding time
Time (ms)
2500
2000
1500
1000
500
0
Fib2
Fib3
Lucas
EliasDelta
EliasFib2
EliasFib3
NASA
Treebank
DBLP


SCOOTER labels - decoding time
Time (ms)
3000
2500
2000
1500
1000
500
0
Fib2
Fib3
Lucas
EliasDelta
EliasFib2
EliasFib3
NASA
Treebank
DBLP

Appendix C: Self Evaluation of the Base-9 scheme



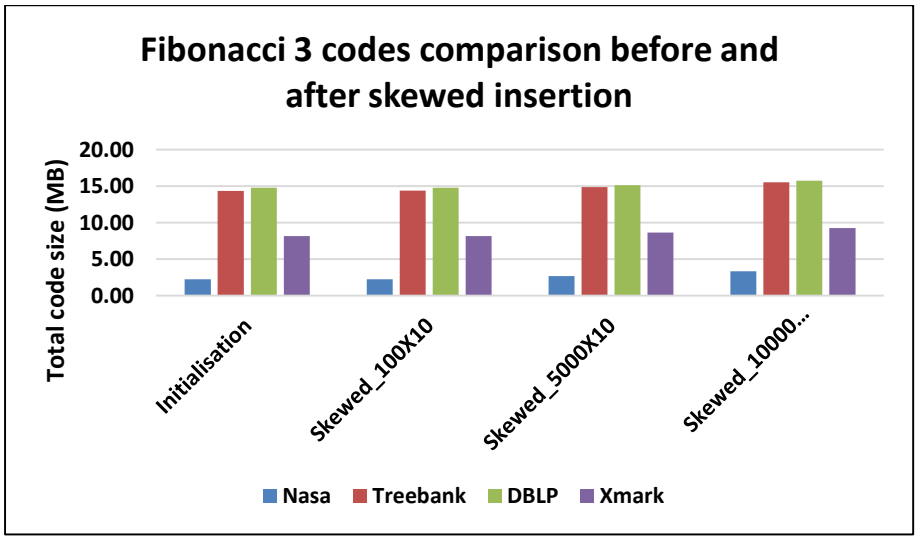


Table C1: 0increase size (KB) in memory (Base-9 labels) after skewed insertion

Number of insertion	Skewed 100X10		Skewed 5,000X10		Skewed 10,000X10	
dataset	Fib2	Fib3	Fib2	Fib3	Fib2	Fib3
NASA	10.00	9.00	496.00	453.00	1233.00	1107.00
Treebank	11.00	11.00	544.00	509.00	1273	1206
DBLP	9.00	8.00	474.00	390.00	1159	982
XMark	10.00	9.00	529.00	496.00	1252.00	1099.00

Table C2: increase size (KB) in memory (Base-9 labels) after uniform insertion

Number of insertion	Uniform 500		Uniform 1,000		Uniform 5,000		Uniform 10,000		Uniform 50,000	
Dataset	FIB 2	FIB 3	FIB 2	FIB 3	FIB 2	FIB 3	FIB2	FIB3	FIB2	FIB3
NASA	5.0	5.0	10.0	10.0	48.0	49.0	96.0	97.0	478.0	483.0
Treebank	7.0	7.0	13.0	13.0	61.0	64.0	121.0	127.0	602.0	628.0
DBLP	5.0	4.0	9.0	8.0	43.0	39.0	86.0	77.0	428.0	381.0
XMark	6.0	6.00	11.0	11.0	50.0	50.0	100.0	101.0	500.0	500.0