THE UNIVERSITY OF HULL

ACCELERATING DATA RETRIEVAL STEPS IN XML DOCUMENTS

being a Thesis submitted for the Degree of
Doctor of Philosophy
in the University of Hull

by

Yun Shen

January, 2005

Except where otherwise indicated, this thesis is my own original work.

Yun Shen

January 2005

To my parents, *Gang Shen* and *Kongmin Zhou*, who made all this possible

Acknowledgments

Writing up this thesis marks the end of the journey I started in 2001. However, for those from whom I have greatly gained support and guidance during these years, my appreciation will never go to an end.

First of all, I would like to thank my PhD supervisor, Dr Bing Wang, for choosing this research topic and providing guidance and critical feedback on the research problems and issues studied in the thesis, and also, importantly, for his patience. With another advisor, I might never have been able to reach the very end.

Besides my supervisor, the other members of my PhD panel, Dr David Grey and Dr Len Bottaci, thank you for having provided insightful feedback concerning my research direction and procedure. I am also grateful to my department for the past three years' tuition fee support and lots of help more than that. In addition to my panel, Dr Qingde Li has been an unflagging source of encouragement, inspiration, and advice in mathematics. Especially, I am indebted to Professor Ling Feng at University of Twente. You directly and practically shape who I am now. During my PhD study, I am also fortunate to have chances to discuss with several professors at Southwest University of Finance and Economics in China: Professor Danqing Du and Professor Lujun Zhou. They point out a correct direction for me in proving several lemmas in Chapter 5.

My brother, Tao Shen, I owe you a lot whenever a problem needs a Statistics solution. My wife, Rong Zhou, you always support and encourage me whenever I am depressed. Last and definitely not least, without my parents, Gang Shen and Kongmin Zhou, I would not be able to come this far and see a happy end.

Abstract

ACCELERATING DATA RETRIEVAL STEPS IN XML DOCUMENTS

by

Yun Shen

Doctor of Philosophy

University of Hull, January, 2005

The aim of this research is to accelerate the data retrieval steps in a collection of XML (eXtensible Markup Language) documents, a key task of current XML research. The following three inter-connected issues relating to the state-of-the-art XML research are thus studied: semantically clustering XML documents, efficiently querying XML document with an index structure and self-adaptively labelling dynamic XML documents, which form a basic but self-contained foundation of a native XML database system.

This research is carried out by following a divide-and-conquer strategy. The issue of dividing a collection of XML documents into sub-clusters, in which semantically similar XML documents are grouped together, is addressed at first. To achieve this purpose, a semantic component model to model the implicit semantic of an XML document is proposed. This model enables us to devise a set of heuristic algorithms to compute the degree of similarity among XML documents. In particular, the newly proposed semantic component model and the heuristic algorithms reflect the inaccuracy of the traditional edit-distance-based clustering mechanisms. After similar XML documents are grouped into sub-collections, the problem of querying XML documents with an index structure is carefully studied. A novel geometric sequence model is proposed to transform XML documents into numbered geometric sequences and XPath queries into geometric query sequences. The problem of evaluating an XPath query in an XML document is theoretically proved to be equal to the problem of finding the subsequence matchings of a geometric query sequence in a numbered geometric document sequence. This geometric sequence model then enables us to devise two new stack-based algorithms to perform both top-down and bottom-up XPath evaluation in XML documents. In particular, the algorithms treat an XPath query as a whole unit, avoiding resource-consuming join operations and generating all the answers without semantic errors and false alarms. Finally the issue of supporting update functions in XML documents is tackled. A new Bayesian allocation model is introduced for the index structure generated in geometric sequence model. Based on k-ary tree data structure and the level traversal mechanism, the correctness and efficiency of the Bayesian allocation model in supporting dynamic XML documents is theoretically proved. In particular, the Bayesian allocation model is general and can be applied to most of the current index structures.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The goal (of XML) is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML.

– XML Specification

As the World Wide Web has become one of the most important communication media, there is an exponential increase in the amount of electronic data which are in a web-compliant format, such as HTML [RHJ99] and SGML (Standard Generalized Markup Language). However, the increasing relevance of the Web as a means of sharing data has posed a number of new issues to the database research community [ABS00, Via01]. Traditional database systems, such as relational, object-oriented, and object-relational systems, of handling highly structured data (i.e. relational data), are insufficient and ineffective [Suc98, ABS00, Via01] whereas Web data are semistructured [ABS00] and encoded using different formats (i.e. HTML [RHJ99] and XML [W3C00]).

Especially, XML [W3C00] is a novel data representation standard where data are bounded by user-defined, meaningful tags. In recent years, it has gained increasing relevance as a means of exchanging information [Suc98, ABS00, Via01]. The datatype underlying the XML paradigm, namely tree, is expressive enough to capture the structure of diverse data sources, yet simple enough to be processed by recursive algorithms.

information

name	age	address	postcode
John	18	-	HU6 7RX
John	18	5 Carrington Ave.	HU5 2AE
William	17	15 Exmouth Street	-

Table 1.1: Relational Table: *information*

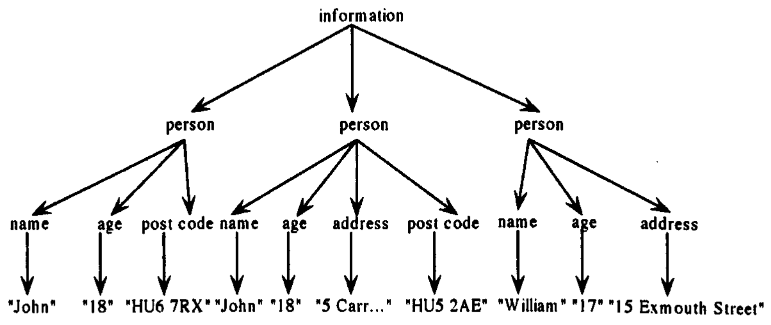


Figure 1.1: Representing *information* in XML Tree Model

1.1 Motivation of Research

XML was not originally derived from the database community [W3C00]. In fact, XML has its roots in the document management community, and is derived from a language for structuring large documents known as SGML. However, XML is simple, lightweight, self-describing and allows users to define their own tags to describe the meaning of data (with/without specific schema). Consider the example in Table 1.1 and Figure 1.1, the hierarchical structure of XML enables it to capture more detailed structural information and not to suffer from unnecessary null information in traditional relational systems.

To cope with the tree-structure of XML documents, several query languages have been proposed to retrieve data [AQM+97, DFF+99, RLS99, W3C99c, W3C03]. However, retrieving data from tree structured XML data is not a trivial task. The recursion inherent in the tree-like data types [W3C00] as well as in the operations over these types turns out to be a challenge for the traditional database community [Suc98, ABS00, Via01]. Consider again the example in Table 1.1 and Figure 1.1, if a user is to "find a person whose age is 18 and lives in HU6 7RX", matching such a query pattern against a document tree without a preprocessing mechanism is equivalent to the tree inclusion problem and has been proved to be NP-complete [AHU74, SWG02].

To address the difficulties, in this thesis, a novel data retrieval mechanism is developed to efficiently search all the answers that match the query conditions precisely in a collection of XML documents. Throughout the thesis, diverse issues needed to support such data retrieval mechanism in the XML tree data model is carefully studied.

1.2 Research Problems

To exploit the power of XML, a data retrieval mechanism for XML documents should efficiently support queries with content and structural constraints [LM01, NND02, WPFY03, JLW03] supported by specialised query languages [AQM+97, DFF+99, RLS99, W3C99c, W3C03] and index structures. The introduction of a query evaluation in the XML data tree model brings up several challenging technical issues as follows:

- **Semantically Clustering XML Data**

  The traditional information retrieval community utilised the vector model [DAM02] on the basis of keywords to model XML documents and the edit distance algorithms to measure the degree of the similarity among documents. However, how to devise a mechanism that semantically and ac-

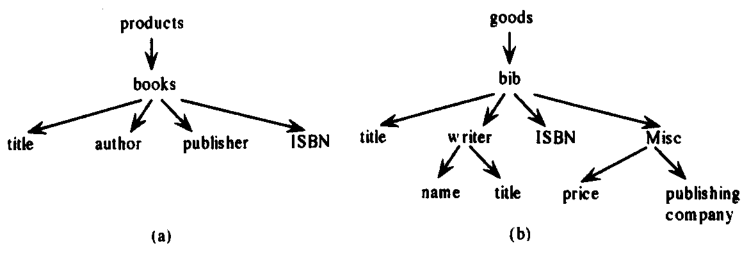


Figure 1.2: Two Similar XML Data Trees

curately clusters XML documents based on the semantic of XML data is an open research problem because there is no formal semantic defined for XML data [W3C00] since XML is originally designed to improve the functionality of the Web by providing more flexible and adaptable information identification. For instance, the example in [LLK01] is used. In Figure 1.2, two sample XML documents about books are illustrated. It is clear that once apparently different sets of elements are "normalised" using synonyms (i.e. "write" equals to "author"), they may constitute rather similar documents. Secondly, their structures are similar considering how the elements are nested. Assume the existence of the functions **Generate()**, **Compute()** and **Cluster()**, in general, a methodology computes similarity among a collection of XML documents $\mathbb{D}$ by taking account of XML semantic works as follows.

1. ***Generate***: $\mathbb{D} \rightarrow \mathbb{S}$, where $\mathbb{S}$ denotes the semantic of $\mathbb{D}$.
2. ***Compute***: $\mathbb{S} \rightarrow \mathbb{M}$, where $\mathbb{M}$ denotes the similarity matrix computing from $\mathbb{S}$.
3. ***Cluster***: $\mathbb{M} \rightarrow \mathbb{C}$, where $\mathbb{C}$ denotes the clusters computing from $\mathbb{M}$.

Therefore, a mechanism that accurately models the semantic of an XML document and can be further computed plays an essential role in dividing a large collection of XML documents. Ranges of queries can be dramatically decreased to applicable documents to accelerate the data retrieval steps. In this thesis, we thus focus on studying the problem of how to model and accurately compute semantic similarity among a collection of XML data.

- **Querying XML Data With An Index Structure**

  A query compatible to XPath [W3C99c] is modelled as a tree, called *twig pattern*, and can be complicated when wildcards "*", self-or-descendent axis("//") and content constraints are presented. Evaluating such a twig pattern in a document tree without a preprocessing mechanism (i.e. an index structure) is equivalent to the tree inclusion problem and has been proved to be NP-complete [AHU74, SWG02]. Current XML query processing techniques are divided into two groups: navigation-based algorithms and index-based algorithms. Navigation-based methods [AF00, CFGR02, LP02, PC03] generate the results by processing an input document one tag at a time. It implies that their processing performance degrades with the increasing number of tags. In contrast to navigation-based algorithms, index-based mechanisms [ZND+01, LM01, KYU01, NND02, CTZZ02, JLW03] utilise the index structures over the input XML documents to accelerate data retrieval steps. Assuming the existence of the function **Query()**, **GenerateIndex()** and the index structure $\mathbb{I}$, the querying process can be simplified as follows:

```
1. $\mathbb{I} \leftarrow$ GenerateIndex($\mathbb{D}$)
2. While(there exists a potential answer in $\mathbb{I}$)
       Query($\mathbb{Q}$, $\mathbb{I}$, $\mathbb{D}$)
```

  Therefore, a mechanism for accurately and efficiently retrieving data from

XML documents plays an important role in enhancing the performance of XPath evaluation steps with the presence of a light-weight index structure, and thus is the core part of the study in this thesis. The problem to devise elegant twig pattern matching algorithms in which an XPath tree structure (i.e. a twig pattern) is treated as a whole unit to avoid expensive join operations is carefully studied.

- **Labelling Dynamic XML Data**

A number of index structures based on corresponding labelling schemes have been proposed to accelerate the query evaluation performance in static XML documents [LM01, KYU01, Gru02, JLW03]. However, XML data in the Web are dynamic [ABS00, CKM02]. It implies that both the structure and the content of an XML document may be evolving with time. Therefore, it is important to devise a labelling scheme to support update operations on dynamic XML documents without affecting the querying performance. Few research efforts have been proposed to solve this problem [LM01, KYU01, CKM02, JLW03, WPFY03]. Assuming the existence of the functions **LabellingTree()** and **Update()**, the whole process of labelling dynamic XML documents can be simplified as follows:

1. ***LabellingTree***($\mathbb{D}$): $\mathbb{D} \rightarrow \mathbb{I}$, where $\mathbb{I}$ stands for the index structure
2. ***LabellingTree***($\mathbb{F}$): $\mathbb{F} \rightarrow \mathbb{I}_{\mathbb{F}}$, where $\mathbb{F}$ stands for an XML fragment and $\mathbb{I}_{\mathbb{F}}$ for its index
3. ***Update***($\mathbb{I}_{\mathbb{F}}$, $\mathbb{I}$): $\mathbb{I} + \mathbb{I}_{\mathbb{F}} \rightarrow \mathbb{I}_{new}$

Therefore, techniques for efficiently labelling dynamic XML data play an important role in both minimising the cost of updating the index structure and accelerating the index-based data retrieval steps in a dynamic environment. The goal in this thesis is to devise such an efficient and general labelling scheme for dynamic XML documents.

1.3 Contributions

The main aim of the thesis is to efficiently index and query XML documents by following a divide-and-conquer [AHU74] strategy. Its main contributions are summarised as follows:

- **Semantically Clustering XML Documents**
  A collection of XML documents is divided into specific groups in which semantically similar XML documents are aggregated together. The range of queries can be thus decreased to applicable documents after relevant documents are grouped into sub-collections. The semantic component model [SW03] is core to this contribution. The proposed set of heuristic algorithms make the computing of similarity among XML documents practical. Its unique feature on the detection of tree inclusion and tree isomorphism renders previous approaches inapplicable. Extensive experimental results using both real and synthesised documents have demonstrated the accuracy of the proposed methods. The tradeoff among various parameters is also investigated. More importantly, the proposed methodology offers opportunities in optimising information retrieval, document classification, and data mining.

- **Holistic XPath Evaluation**
  After relevant documents are aggregated together, an efficient querying mechanism [WFS05] is proposed to retrieve XML data with an index struc-

ture. XML documents and XML queries are transformed into geometric sequences. By performing subsequence matching on a set of sequences in the database, all the occurrences of an XPath query pattern (i.e. twig pattern) in the database are retrieved. The approach allows matching a twig pattern query without breaking down the pattern into root-to-leaf paths and processing these paths individually. Furthermore, it is theoretically proved that all correct answers are found without false answers and semantic false, and, most importantly, without refinement or post-processing. An elegant use of stack mechanism is also proposed to support such efficient data retrieval purpose.

- **Self-adaptive Labelling Scheme for Dynamic XML Documents**
  XML data in the Web is dynamic. It implies that the structure and the content of an XML document are evolving over time. A self-adaptive scope allocation scheme [SFSW04] for labelling dynamic XML documents is introduced to tackle the updating problem. It is general, light-weight and can be built upon existing data retrieval mechanisms. The Bayesian inference is initiatively used to compute the actual scope allocated for labelling a certain node based on both the prior information and the actual document. Through theoretical proofs and extensive experiments, it is proved that the proposed Bayesian allocation model can practically and significantly improve the performance of the conventional fixed scope allocation models.

1.4 Outline of Thesis

The overall approach of the research described in this thesis was to divide a collection of semantically different XML documents into sub-collections, and then index these XML documents to support data retrieval functions and update functions. This divide-and-conquer approach [Llo84] gives rise to the following thesis structure.

Following the short introduction of this chapter, the first part of Chapter 2 presents a background review of XML and its related schema languages and query languages. The second part of Chapter 2 reviews the research issues related to indexing and querying XML documents. Throughout this thesis, all XML documents are treated as data files except where otherwise indicated.

Chapter 3 considers how to divide a collection of XML documents into groups of semantically similar XML documents. The deficiency of the state-of-the-art clustering techniques for XML documents is thoroughly examined. An advanced semantic component model is introduced to model the semantic expressed in an XML document. After that, a set of heuristic algorithms are presented to compute the semantic similarity among a collection of XML documents. A heuristic clustering technique is utilised to generate the final clusters.

Chapter 4 describes a method for efficiently indexing and querying XML documents after similar XML documents are grouped together. In this chapter, the shortcomings of current XML data retrieval mechanisms is briefly reviewed. Then a novel geometric sequence model is introduced to transform both XML documents and XPath queries. It is theoretically proved that the geometric sequence model turns a twig pattern matching problem into a subsequence matching problem. Finally, two efficient stack-based algorithms are presented to extract data from XML documents based on the geometric sequence model and a trie index structure.

Chapter 5 considers how to support update functions in XML documents. Previous research efforts in supporting update functions in XML documents are reviewed. Then a novel Bayesian allocation model is proposed to self-adaptively label XML documents by embedding the index structure into a K-ary tree with the level traversal. Finally, the correctness and efficiency of the allocation model is theoretically and experimentally proved.

Chapter 6 summarises the contributions of the thesis and proposes the future work.

CHAPTER 2

Background

Knowledge is a process of piling up facts; wisdom lies in their simplification.

– Martin Fischer

The use of semistructured data [ABS00] is motivated by the challenges posed by existing electronic data (e.g. HTML [RHJ99] and SGML) which do not conform to the traditional relational data model. It requires the research and development of the corresponding semistructured data models with query languages adapted to these models. The emergence of XML [W3C00] has facilitated the electronic data publication by providing a simple syntax for data which is both machine- and human-understandable. Though introduced in a document community, XML is quickly gaining popularity in data representation and exchange on the Web [Suc98]. Interestingly, semistructured data and XML share considerable similarity [GMW99]. Previous research efforts on semistructured data [HGMI+95, QWG+96, MAG+97, FFLS97, AV97, Suc98, BFS00] may thus offer some solutions to the data-centric challenges posed by XML, though the efforts may not be trivial [Suc98, Via01]. In this chapter, an overview of semistructured data and XML is presented. After that a thorough literature review on the substantial research efforts on the issues covered in this thesis is studied.

2.1 Basics of Semistructured Data

The publication of electronic data has been becoming universal and pervasive in the Web. However, these data lie outside of the conventional relational database management systems but inside structured documents (e.g. in format of HTML or SGML). From the perspective of the database community, it is the Web that led to the development of semistructured data models and the relevant query languages, which are proposed to tackle the core issues on efficient data retrieval, version control, change detection, indexing, and data management of semistructured data, which can not be directly supported by conventional relational database [ABS00].

Semistructured Data is often explained as "schema-less" or "self-describing", terms that indicate that there is no separate description of the type or structure of data [ABS00] in contrast to the relational model. An example of the semistructured data fragment is shown below.

```
&book1{
   author: ''Serge Abiteboul, Peter Buneman and Dan Suciu",
   book: ''Data on the Web: from relations to semistructured data
          and XML",
   publisher: ''Morgan Kaufmann Publishers",
}
```

Typically, the data fragment is graphically represented by a node which is viewed as an object and inter-connected with the other objects by labelled edges. Each node can be atomic or complex using the syntax [ABS00] below. For example, "&book1" is an oid, "author" is a label and "Serge Abiteboul, Peter Buneman and Dan Suciu" is an atomicvalue.

```
<ssd-expr>  ::=  <value>  | oid<value> | oid
<value>::=atomicvalue | <complexvalue>
<complexvalue> ::= {label : <ssd-expr>, ..., label : <ssd-expr>}
```

These inter-connected objects are normally represented by a graph. A graph G(V, E) consists of a set of V nodes (objects) and a set of E edges. Each edge connects a pair of ordered nodes, the source node (parent node) v_s and the target node (child node) v_t. A path is a sequence of edges starting from the source node v_s to the target node v_t, denoted as e_s, e_{s+1}, ..., e_t. A node r is a root if there is a path from r to v for every $v \in V$. A cycle in a graph is a path e_s, e_{s+1}, ..., e_s. A graph with no cycles is called acyclic. To summarise, the data model above is an edge-labelled graph. The graph example of the above data fragment is illustrated below in Figure 2.1.

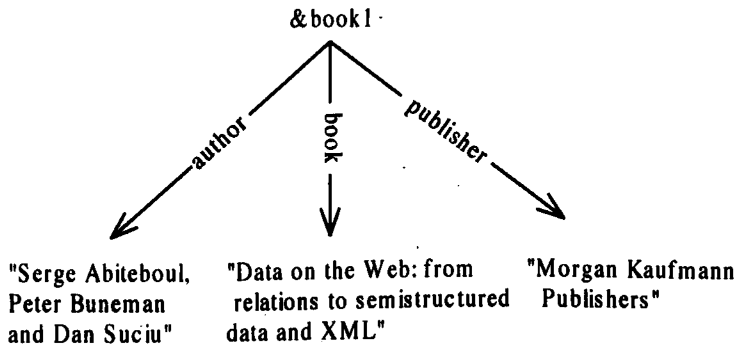

&book1
author
book
publisher
"Serge Abiteboul, Peter Buneman and Dan Suciu"
"Data on the Web: from relations to semistructured data and XML"
"Morgan Kaufmann Publishers"

Figure 2.1: A Semistructured Data Model

The research on semistructured data started with the OEM (Object Exchange Model) model in Tsimmis project at Stanford in 1995 [HGMI+95], following by a related, successful Lore project in Stanford [QWG+96, MAG+97], in which an edge-labelled variant of OEM was proposed. Later, UnQL [BDFS97, BFS00], developed at the University of Pennsylvania, introduced an alternative data model based on bisimulation.

Most of the research following the above projects has focused on the logical data models and the corresponding query languages. Several query languages have been proposed: UnQL [BFS00], StruQL [FFLS97], MSL [PAGM96], and Lorel [AQM$^+$97]. Other research includes heterogeneous data integration [HGMI$^+$95], Web site management [FFLS97], schema formalisms [BDFS97], indexing [MS99], and optimisations [AV97, Suc98].

2.2 Basics of XML

2.2.1 XML Premier

Unlike the semistructured data, XML [W3C00] does not originate in the database community. It was introduced in the document community as a subset of SGML and is more like an augmentation of HTML allowing annotation of data with information about its meaning rather than just its presentation [Via01]. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification, and should be versatile enough to be customised for domains as diverse as web sites, electronic data interchange, vector graphics, object serialisation, remote procedure calls and genealogy.

XML defines a generic syntax used to mark up documents with simple, human readable tags. The users can define their own tag names, contrasting to HTML in which the terms are fixed. Data is included in XML documents as strings of text, and is surrounded by the above mentioned text markup (tag name) that describe the data. This particular unit is called an element. The XML specification [W3C00] defines the exact syntax for: how elements are delimited by tags, what a tag looks like, how elements are nested, and so forth. The markup permitted in a particular XML document can also be stated in a document type

```
<A>
  <B>
    <D>indexing</D>
    <E>and</E>
    <F>querying</F>
  </B>
  <B>
    <D>XML</D>
    <E>documents</E>
  </B>
  <J>CIKM</J>
</A>
```

Figure 2.2: An Example XML Document

definition (DTD), one of the most popular XML schema languages. It lists all valid markup and designate where and how these markup may be included in an XML document. Additionally, DTD provides a limited support for data types in an XML document. An example XML document is shown in Figure 2.2,

2.2.2 XML Data Model

Data in XML is grouped into elements delimited by tags and elements can be nested according to the user's specification. However, XML is only a meta-markup language, and did not have an associated data model when introduced. In this section, a data model for XML and some preliminaries for the rest of the thesis is set up.

An XML document is modelled as an ordered, node labelled and rooted tree. More formally, consider a graph $\mathbf{T} = (\mathbf{V_G}, \mathbf{V_T}, \mathbf{v_r}, \mathbf{E_G}, \mathbf{label_{node}}, \mathbf{nid}, \sum_\mathbf{T})$. V_G is the set of element nodes and V_T is the set of text nodes. $\forall v \in V_T$, v has no outgoing edge. v_r is root of the XML data tree, where there exists a path from v_r to v, $\forall v \in V_G \cup V_T$. It implies that v_r has no incoming edge. Each node $v \in V_G \cup V_T$

is labelled through the function $label_{node}$ over the set of terms, $\sum_T$. The label of a node $v \in V_G$ is referred to the tag name. The label of $v \in V_T$ is referred to a distinguishing keyword contained in the corresponding text. Quotation mark in future figures is used to distinguish the label of V_T. Each edge e, $e \in E_G$, is a parent-to-child edge, denoting the parent-child relationship. The parent node is denoted as v_{e_p}, and the child node is denoted as v_{e_c}. A path is a sequence of edges starting from the node v_i to the node v_j, denoted as e_i, e_{i+1}, ..., e_j. A node v_i is ancestor of v_j *iff* a path to v_j goes through v_i. The order among the sibling nodes is distinguished. Each node is assigned a unique nid number for indexing and querying purpose. T_{v_i} refers to the subtree induced by node v_i.

Figure 2.3 shows an example of our data model. The solid edges represent E_G. The dashed edge denotes a edge e, $v_{e_p} \in V_G$ and $v_{e_c} \in V_T$. The quoted string represents a label of a node v, $v \in V_T$.

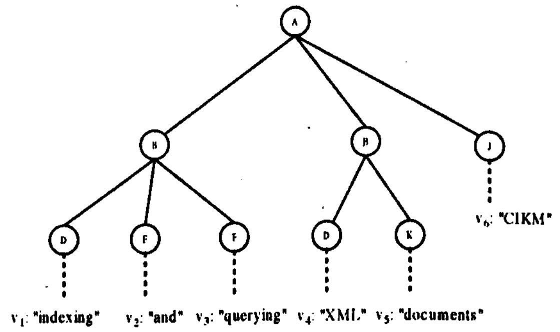


Figure 2.3: An Example XML Document in Tree Structure

2.2.3 XML Schema Languages

An XML schema language [W3C01b] aims to present a formalisation of the constraints, expressed as rules or a model of structures, which applies to a class of XML documents. In this section, a brief overview of a number of schema lan-

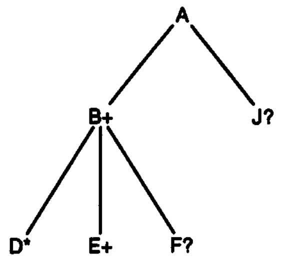


Figure 2.4: An Example of DTD Tree

guages have been proposed to model XML documents in both structures and datatypes is presented.

DTD This grammar is known as a document type definition, or DTD [W3C00]. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. DTD is written in a formal BNF syntax that explains precisely which elements and entities may appear in the document and what the elements' contents and attributes are. Simple cardinality constraints can be imposed on the elements using regular expression operators (?, *, +). Elements can be grouped as ordered sequences (a,b) or as choices (a|b). Elements have attributes with properties type (PCDATA, ID, IDREF, ENUMERATION), cardinality (#REQUIRED, #FIXED, #DEFAULT), and any default value. An example DTD of Figure 2.2 is shown below and its tree structure is shown in Figure 2.4. It states that an element called A may have two kinds of child elements, respectively B and J, and B may have three kinds of child elements, respectively D, E, and F.

```
<!ELEMENT A (B+, J?)>
```

```
<!ELEMENT B (D*, E+, F?))>
<!ELEMENT D (#PCDATA))>
<!ELEMENT E (#PCDATA))>
<!ELEMENT F (#PCDATA))>
<!ELEMENT J (#PCDATA))>
```

DTD supports limited data types and constraints definitions. However, it, a subset of SGML DTD, is the de facto standard XML schema language of the past and present and is most likely to thrive until XML-Schema finally arrives. Mignet et al. [MBV03] conclude that the use of XML Schema, the new mechanism for specifying the schema for an XML document, is insignificant compared to the usage of DTD. Hence, this thesis mainly focuses on DTD. The brief summary of type definitions in DTD is listed below.

- <!DOCTYPE root-element [doctype-declaration...]> determines the name of the root element and contains the document type declarations.
- <!ELEMENT element-name content-model> associates a content model to all elements of the given name. An element type declaration constrains the element's content.
- <!ATTLIST element-name attr-name attr-type attr-default ...> declares which attributes are allowed or required in which elements. XML attribute types are of three kinds: a string type, a set of tokenised types, and enumerated types.

XML Schema Language XML Schema is the W3C's object-oriented description for XML [W3C01b]. It extends the functionality provided by DTD, most importantly, with the addition of a type system. An XML Schema is based on the fact that the DTD is not an XML document; it is written in

Extended BNF notation [W3C00, CEM03]. By contrast, the XML Schema is itself an XML document. Moreover, XML schema is more complex than DTD, using a uniform XML syntax and supporting a complicated notion of derivation of document types. It is thus not likely to simply model XML Schema as a tree. W3C [W3C01b] specifies that an XML Schema is modelled as a set of schema components. Each component is a building block that comprises the schema. W3C [W3C01b] also states there are 13 kinds of components in all, falling into three groups. These components constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content and attributes and their values [W3C01b]. In summary, XML Schema is both about structure and datatyping, and is compatible to DTD.

Miscellaneous Schema Languages However, DTD and XML Schema languages do not provide all the facilities that might be needed by any application. Some applications may require constraint capabilities not expressible in DTD or XML Schema, and so may need to perform their own additional validations. Document Content Description (DCD) [BFM98] proposes a structural schema facility to specify rules covering the structure and content of XML documents with additional basic datatypes, and is consistent with the ongoing W3C RDF (Resource Description Framework) [W3C04] effort. Schema for Object-Oriented XML (SOX) [W3C99b] extends the language of DTDs by supporting an extensive (and extensible) set of datatypes, and defines the syntactic structure and partial semantics of XML document types. Document Definition Markup Language (DDML) [W3C99a] encodes the logical (as opposed to physical) content of DTDs in an XML document, allowing schema information to be explored and used with widely available

XML tools. RELAX NG [CM01] was based on TREX [CM01] designed by James Clark and RELAX designed by MURATA Makoto [Mur00]. RELAX NG is supported by clean principles of hedge automata [Mur00] and has the capability of expressing context-sensitive schema rules by means of non-terminal symbols.

2.2.4 XML Query Languages

Several query languages, i.e. Lorel [AQM+97], Qulit [CRF01], XML-QL [DFF+99], XQL [RLS99], XPath [W3C99c] and XQuery [W3C03], have been proposed to retrieve data from XML documents and semistructured data. The common features of these languages are the use of regular path expressions and the ability to extract information about the schema from the data [LM01]. The database community is now well underway to adapt its technology to host large XML stores and to query these stores efficiently, preferably using query languages developed in the XML domain: XPath and XQuery.

XPath XPath [W3C99c] is a language for addressing parts of an XML document. It gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document. In support of its primary purpose of addressing parts of an XML document, XPath also provides basic facilities for manipulation of strings, numbers and booleans. It uses a compact, non-XML syntax to facilitate the use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax.

XPath operates on an XML document as a tree. Each node in the tree has one of the seven types; root, element, attribute, text, namespace, processing instruction and comment. Starting from a so called context node, XPath

/A/B	Select all elements B which are children of the root element A
//D/B	Select all elements B which are children of the element D
/*/*/*/B	Select all elements B which have 3 ancestors
//B[@id='XML']	Select B elements which have attribute id with value XML

Table 2.1: Example of XPath Queries

query traverses its input documents using a number of location paths. The result of a location path is the node-set containing the nodes selected by the location path, which is further specified by its location steps. In each step, an *axis* describes which document nodes (and the subtrees below these nodes) form the intermediate result forest for this step. [W3C99c] specifies 13 different axes: child, descendant, parent, following-sibling, preceding-sibling, following, preceding, attribute, namespace, self, descendant-or-self, and ancestor-or-self. Importantly, the ancestor, descendant, following, preceding and self axes partition a document (ignoring attribute and namespace nodes); they do not overlap and together they contain all the nodes in the document. "." is short for self::node(), "/" is short for children::nodes(), and "//" is short for /descendant-or-self::node()/. In the rest of the thesis, the abbreviations are used without further explanation. Several examples are listed in Table.2.1.

XQuery XQuery [W3C03] is designed to be broadly applicable across many types of XML data sources. It is designed to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. It also uses the structure of XML intelligently to express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware [W3C03]. XQuery comes from an XML query language called Quilt [CRF01], which inherits several other query languages, including XPath [W3C99c], XQL [RLS99], XML-QL [DFF+99], SQL [ISO99], and OQL [Cat96]. The current version of XQuery (version 1.0) is an extension of XPath Version 2.0. It implies that any expression that is syntactically valid and executes successfully in both XPath and XQuery will return the same result in both languages.

Miscellaneous Query Languages Lorel [AQM+97] was originally designed for querying semistructured data, and has been extended to support XML data [GMW99]. XML-QL [DFF+99] was introduced in Studel project. It extends SQL with an explicit CONSTRUCT clause with the capability of integrating XML data from different sources [DFF+99]. The XML Query Language (XQL) [RLS99] is a notation for addressing and filtering the elements and text of XML documents, and can be treated as a natural extension to XSLT syntax. Quilt [CRF01] is designed as a small, implementable language that meets the requirements identified by the W3C XML Query Working Group. XML-GL [CCD+99] is a graph-based query languages with both of its syntax and semantics defined in term of graph structures and operations. It is well suitable for supporting a user-friendly interface.

2.3 Some of the Research Issues Involved in XML

Researchers have recently observed a striking similarity [ABS00] between semistructured data models and XML [W3C00]: their structures may be irregular, may not always be known ahead of time, and may change frequently and without notice [Suc98]. However, XML and its applications require solutions to problems which the research on semistructured data [HGMI+95, QWG+96, MAG+97, FFLS97, AV97, Suc98, BFS00] has not yet addressed (e.g. type inference), or has not found efficient solutions (e.g. data mining, efficient indexing and querying). However, adapting the previous research efforts on semistructured data is not trivial. In this section, we briefly review and summarise some of the research issues involved in current XML research and covered in this thesis: (1) Clustering XML data, (2) Labelling XML data and (3) Querying XML data with index structure.

2.3.1 Clustering XML Data

Document clustering has been widely recognised as a powerful data mining technique and has been studied extensively in a number of different areas of text mining and information retrieval in recent years. The main objective of document clustering is to group similar documents into a certain partition. It has several interesting applications, such as the management of Web data (especially XML data) and knowledge discovery. With the advent of XML as a standard for data representation and exchange in the web, the traditional approaches to clustering documents are ineffective in the new context of structured documents. In such context, current research efforts on clustering XML documents focus on two research directions: (1) Text-centric clustering and (2) Structure-centric clustering.

Text-centric Clustering Text-centric clustering technique treats XML documents as pure text documents. The hierarchical structure of an XML document is ignored when calculating the similarity among a collection of documents.

- Antoine Doucet et al. [DAM02] proposed an approach of clustering a homogenous collection of text-centric XML documents. Each document is represented by an N-dimensional vector, with N being the number of *document features* in the collection. Since the number of dimensions of the vector, N, would easily place the computational efficiency at stake, all the words in an XML document are not selected as features. Antoine Doucet et al. [DAM02] used three different document feature sets: (1) "text features only": it is achieved by ignoring words less than three characters and longer words with a weak discriminative power and reducing words into a canonical form; (2) "tag features only": it is achieved by collecting unique tag names in an XML document; (3) "text+tags": it is achieved by simply combining "text features" and "tag features" together. The vectors are then filed with (term frequency - inverted frequency) tf-idf measures, and the k-means clustering technique is utilised to partition the dataset. In this mechanism, the hierarchical structure of an XML document is completely ignored.

Structure-centric Clustering Structure-centric clustering techniques consider to structurally measure the similarity among structured documents which are modelled as trees or graphs.

- Damien Guillaume et al. [GM00] proposed an approach to clustering inter-linked AML (astronomical markup language) documents by giving these links weight. The clustering problem is then transformed into a *graph partition problem*: a graph of connected nodes[1], with weighted edges (links) between them. Damien Guillaume et al. partitioned this graph so that it can be displayed on a density map which shows clusters of similar documents. This approach falls into the web structure clustering technique, and

[1]XML documents in the physical level

does not contribute a lot, considering the structural originality of an XML document.

- Jong Yoon et al. [YRCK01] proposed a bitmap indexing technique, *BitCube*, to cluster XML documents. An XML document is defined as a sequence of root-to-content *ePaths* with associated element contents. Each document is indexed by using a 2-dimensional bitmap index where a bit represents an ePath and a row represents an XML documents. Since an XML document database contains a set of XML documents, 2-dimensional bitmap index is further extended into a 3-dimensional BitCube. The similarity of two XML documents (bitmap rows in bitcube), d_i and d_j, is defined as sim(d_i, d_j) = 1 - $|XOR(d_i, d_j)|/MAX([d_i], [d_j])$. The experiments showed that bitcube works fine in clustering a large number of XML documents, however, it suffers from enormous '0' bits when certain epaths do not exist in a collection of XML documents.

- Jung-Won Lee et al. [LLK01] proposed a methodology for quantitative determination of similarity between XML documents. The *minimal hierarchical structure* of an XML document is generated using specialised automata. Elements are sorted on path by tree levels. By traversing elements from the root element to elements with atomic data, large 1-paths, large 2-paths, ..., large n-paths are obtained through an adapted sequential pattern mining algorithm. *Nested path expressions* of the base document (NPE_B) is generated from the transformed path expressions of the base document (PE_B). To qualify the similarity between XML documents, Jung-Won Lee et al. [YRCK01] used the following computing phase:

$$\sum_{k=1}^{L(NPE)} \frac{1}{S(n)} \times V(E_k) \tag{2.1}$$

S(n) is a number of siblings of 1st level and L() is length function.

$$V(E_k) = \begin{cases} 1, & \text{if } E_k \text{ is a atomic and selected;} \\ 1, & \text{if } E_k \text{ is a atomic and not selected;} \\ \frac{1}{L(E_k)} \sum_{i=1}^{L(E_k)} V(E_i), & \text{if } E_k \text{ is a list.} \end{cases}$$

The experiments claim that this approach greatly outperforms the traditional vector-space model in clustering accuracy.

- Sergio Flesca et al. [FMM+02] proposed a completely different approach in determining the similarity between XML documents. Sergio Flesca et al. [FMM+02] represented the structure of an XML document as a time series. The structure of an XML document is captured by using *multilevel encoding* technique. The multilevel encoding of a document d is a sequence $[S_0, S_1, S_2, ..., S_n]$ where

$$S_i = \gamma(t_i) \times B^{maxdepthd-l_{t_i}} + \sum_{t_j \in nest_d(t_i)} \gamma(t_i) \times B^{maxdepthd-l_{t_j}} \tag{2.2}$$

where B = $|tnams(d)| + 1$ and γ is a tag encoding function (i.e. mapping tag names to integer set). Sergio Flesca et al. [FMM+02] defined the *Discrete Fourier Transform distance* of the documents as the approximation of the difference of the magnitudes of the two signals which are produced in a depth-first traversal. The experiments demonstrate that this approach clusters XML documents very efficiently, however, it fails to semantically compute the similarity and can not solve both tree inclusion and tree isomorphism issues.

2.3.2 Labelling XML Data

The objective of designing labelling schemes for XML trees is to allow quick determination of the relationships between two element nodes specified in twig patterns without actually accessing the XML file. In addition, labelling schemes of XML trees have three desiderata: ease of computing, short code size and extensibility. Existing labelling schemes for XML data tree fall into the following three categories: (1) Interval-based labelling schemes, (2) Prefix-based labelling schemes and (3) Complete-tree-based labelling schemes. More recently, few research efforts also tackle the problem of labelling dynamic XML trees within the above three categories of labelling scheme.

Interval-based Labelling Schemes The label of a node in a tree T is given an interval label in this kind of labelling schemes. The key insight of interval-based labelling schemes is due to Dietz [Die82], who noted that the ancestor relationship between two nodes in a tree can be determined by using the following fact: for two nodes v_i and v_j of a rooted tree T, v_i is an ancestor of v_j if and only if v_i occurs before v_j in the pre-order traversal of T and after v_j in the post-order traversal of T.

- Chun Zhang et al. [ZND+01] proposed to label each element node in an XML data tree with a (*begin*, *end*, *level*) label and each text node with a (*wordno*, *level*) label. This labelling scheme is achieved by doing a depth-first traversal of the tree, and sequentially assigning a number at each visit. These numbers could be physical or logical offsets of the begin and the end of the nodes in the XML data tree. Given two nodes, v_i and v_j, v_i is *ancestor* of v_j *iff* v_i.begin < v_j.start < v_i.end holds; and v_i is *parent* of v_j *iff* v_i.level = v_i.level - 1 $\wedge$ v_i.begin < v_j.start < v_i.end holds. Similar labelling schemes are also used in [YASU01, AKJKP02, NND02, JLW03]

- Quanzhong Li et al. [LM01] proposed an alternative labelling scheme, in which each node is given a (*order*, *size*, *level*) label, where *order* is the pre-order number and *size* is the number of descendants. The labelling scheme is also achieved by doing a depth-first traversal and the numbers could be physical or logical offsets of the begin and the end of the nodes in the XML data tree. For a tree node v_i and its parent v_j, order(v_j) < order(v_i) and order(v_i) + size(v_i) $\leq$ order(v_j) + size(v_j) and v_i.level = v_j.level + 1. In the other words, interval [order(v_i), order(v_i) + size(v_i)] is contained in the interval [order(v_j), order(v_j) + size(v_j)]. For any given two nodes v_i and v_j of a tree T, v_j is an ancestor of v_i if and only if order(v_j) < order(v_i) $\leq$ order(v_j) + size(v_j). Similar labelling schemes are also used in [CTZZ01, WPFY03, WM05].

- Torsten Grust [Gru02] proposed to give each node in an XML data tree a (*pre*, *post*, *level*) label. The labelling scheme is achieved by assigning a node a pre-order sequence number when parsing its open tag and a post-order sequence number when parsing its close tag. Given any two nodes v_i and v_j in a tree T, v_i is a descendant of v_j if and only if v_i.pre > v_j.pre $\wedge$ v_i.post < v_j.post. The labelling scheme is also used in [FLSW03] to efficiently order XML data, in which limited update functions are supported. The above two kinds of interval labelling schemes suffer from the global update problem. The whole index has to be re-calculated if data in an XML document is updated.

- Dao Dinh Kha et al. [KYU01] proposed a *Relative Region Coordinate (RRC)* labelling scheme, contrasting to the above absolute region coordinate (ARC) labelling schemes to tackle this global update problem. Each node v of a tree T is given a (c_1, c_2) label, where c_1 (respectively c_2) is the number of physical bytes from the starting byte of the parent node of v to the starting byte (respectively, the end byte) of v. RRC represents the relative location within the space of its parent, contrasting to ARC which represents the location of a node by distance to the start of the XML documents. The ancestor-descendant (parent-child) relationship can be determined by induction. ARC can be computed into RRC: Assume we compute the ARC of node v, the path of v is n_1, n_2, ..., n_l, where n_l = n, and RRC of each node n_i is $[r_i,\ s_i]$, where $i \in [1, l]$, then ARC of v is $[\sum_{i=1}^{l} r_i - (l-1),\ \sum_{i=1}^{l-1} r_i + b_l - (l-1)]$. In case we change the length of data in the node v with the node path P: in the same level with node v_i (i=1,2,..,k), besides v_i, only the right siblings of v_i have their RRC changed. Therefore, the total number of RRC-updated for v is the sum of the numbers of RRC-updated nodes for v in each level.

Prefix-based Labelling Schemes Unlike interval-based labelling schemes, prefix-based labelling scheme targets to encode the root-to-node path information for each node in a data tree. One can check whether a node v_i is ancestor of v_j in T in practically constant time complexity by checking whether a label of v_i is a prefix of v_j's. Prefix-based labelling schemes are always utilised to support dynamic XML trees which are subject to updates.

- *Dewey Decimal Coding (DDC)* has been widely used by librarians. Igor Tatarinov et al. [TVB+02] exploited DDC to capture document order in the relational data model. The labelling scheme is achieved by assigning a vector that represents the path from the document's root to the node. Though DDC identifies each node with a unique label and overcomes the overhead problem incurred by updates, one of the potential disadvantages is the extra space required to store paths from the root to each node. Igor Tatarinov et al. [TVB+02] utilised UTF-8 based representation to minimise this overhead.

- Edith Cohen et al. [CKM02] presented a *persistent structural labelling scheme* to label nodes of an XML tree which is subjected to insertions and deletions of nodes. A persistent structural labelling scheme is a pair $\langle p, L \rangle$ where p is a binary predicate over strings. The labelling function L, however, behaves different from interval-based labelling schemes. Rather than getting as input a full tree, L gets a sequence of insertions of nodes into an initially empty tree. As each node is inserted, L assigns it a binary string [2]. The label can not be changed subsequently. Given any two nodes v_i and v_j in the resulting tree, p(L(v_i), L(v_j)) evaluates to be TRUE iff v_i is an ancestor of v_j. Edith Cohen et al. [CKM02] proved that the maximum length of a label without clues in such a persistent labelling scheme is at most 4dlog($\triangle$), d being the maximal depth of a tree and $\triangle$ the maximum out-degree of a node. Edith Cohen et al. [CKM02] also proved that the maximum length of a label with sub-tree clues requires $\Omega(\log^2 n)$ bits for each label, which asymptotically matches the bound for static tree labelling and the maximum length of a label with sibling clues is with tight lower and upper bound of $\Theta(\log n)$. This persistent labelling scheme supports unlimited update functions in an unordered XML document.

Complete-tree-based Labelling Schemes Unlike interval-based labelling schemes and prefix-based labelling schemes, complete-tree-based labelling schemes aim to utilise the perfect mathematical aspects of the k-ary complete tree.

[2]The children of a node v have the label concatenated with the string s attached to their incoming edge. Given s(1)=0, to obtain s(i+1), the binary number represented by s(i) is increased 1 and if the representation of s(i)+1 consists of all ones, its length is double.

- Yong Kyu Lee et al. proposed a labelling scheme to interpret an XML document structure as a complete k-ary tree and assign each node a unique element identifier (UID) according to the order of the level order tree traversal. Suppose the maximum depth of a data tree is l and the total number of nodes in a data tree is k, $\frac{(k^l-1)}{(k-1)}$ - n virtual nodes would be introduced in the worst case. The UID's of the parent and the j-th child of a node whose UID is i can be obtain by the following functions.

$$\begin{aligned} parent(i) &= \lfloor \frac{(i-2)}{k} + 1 \rfloor && (2.3) \\ child(i,j) &= k(i-1) + j + 1 && (2.4) \end{aligned}$$

- Wei Wang et al. [WPFY03] proposed a novel labelling scheme PBiTree to *binarise* an XML data tree into a perfect binary tree. The labelling scheme is achieved by (1) embedding the current node and its child nodes and (2) recursively embedding the subtrees rooted at each child node during the in-order traversal of a binary tree. The criteria of embedding the child nodes of the current node is: if the level of the mapped parent is l, the first possible level to place all the m child nodes is l + k, where k satisfies $2^k \geq$ m. Each node of a data tree is assigned a unique *code* after the *binarisation* in the PBiTree. For a given node v_i of a PBiTree, its ancestor v_j at a given height h_j can be computed by,

$$\mathcal{F}(v_i, h_j) = 2^{h_j} \cdot \lfloor v_i / 2^{h_j+1} \rfloor + 2^{h_j} \qquad (2.5)$$

Given two nodes v_i and v_j in a PBiTree, v_i is an ancestor of v_j if and only if $v_i = \mathcal{F}(v_j, \text{height}(v_i))$, where height$(v_i)$ is the position of the rightmost '1' bit in its binary representation.

2.3.3 Querying XML Data with Index Structure

XML is emerging as one of the dominant data formats for data representation and exchange in the Internet. To query XML data, several query languages, i.e. Lorel [AQM$^+$97], Quilt [CRF01], XML-QL [DFF$^+$99], XQL [RLS99], XPath [W3C99c] and XQuery [W3C03], have been proposed. However, conventional traversal-based techniques are insufficient and not efficient to meet the processing requirement under heavy requests in semistructured data environment. In any DBMS, the tradeoff between efficient data querying and space complexity, i.e. index structure, must be considered. Indexing allows fast and efficient access to data by purposely summarising or replicating part of the source data. State-of-the-art research efforts on indexing and querying XML data have two major research directions: the navigation-based querying mechanism and the index-based querying mechanism. In this chapter, the focus is on reviewing the index-based querying mechanisms. This kind of querying mechanism have two major research efforts: (1) Path-based indexing and querying, and (2) Node identification based indexing and querying.

Path Based Indexing and Querying The path based indexing and querying techniques share common properties: (1) They treat paths as basic units when indexing and querying; (2) They focus on structural indexing and querying semistructured data and depend on the specialised data structure; and (3) They all can efficiently process single path evaluation in semistructured data, however, their performance degrades when tree queries are involved.

- **PIndex** Path based indexing and querying was first studied by Jason McHugh et al. [MW99, MWA$^+$98]. In contrast to conventional relational DBMS in which index is created on an attribute, Jason McHugh et al.

proposed Vindex (value index), Tindex (text index), Lindex (link index) and Pindex (path index) on the basis of the OEM data model to enable a fast and efficient access to semistructured data. Pindex is core of indexing semistructured data since finding all objects reachable by a given labelled path through the database is an indispensable part of semistructured data querying. Pindex is enabled by using DataGuides [GW97] which concisely and accurately maintains dynamic structural summaries of graph modelled semistructured data. In DataGuides, (1) two labelled paths point to the same object in the DataGuide object if and only if the target sets of both labelled paths are the same; (2) all the valid paths of a certain object are indexed. Svetlozar Nestorov [NUWC97] et al. proved that creating a DataGuides over a source database is equivalent to converting of a non-deterministic finite automaton (NFA) to a deterministic finite automaton (DFA) [AHU74]. It may be able to compute a minimal DataGuide even though there may exist a number of DataGuides for a single semistructured database.

- **T-index** PIndex requires a powerset construct over the underlying database, which, in the worst case, can be of exponential cost. T-index (template index) [MS99], another research effort of path based indexing and querying graph modelled semistructured data, was proposed by Tova Milo et al. to tackle the problem. The core of T-index is computing bi-simulation [PT87] (denoted as $\approx_b$) or simulation (denoted as $\approx_s$) refined semistructured data graph [PT87, HHK95] given a path template $\boxed{T_0}\ x_0 \ldots \boxed{T_k}\ x_k$. T-index guarantees linear size of index structure over a semistructured data in contrast to exponential size of DataGuide [GW97] in the worst case. Special cases of T-index include 1-index (indexing all absolute paths in semistructured data) and 2-index (indexing all relative paths connecting ancestors and descendants). However, T-index only supports paths matching a single path template $\boxed{T_0}\ x_0 \ldots \boxed{T_k}\ x_k$. It implies that all the paths which do not conform to the template can not be efficiently evaluated. Tova Milo et al. [MS99] proposed a partial solution, called prefix replacement, to rewrite the query, allowing queries with a prefix matching of the path template be rewritten and evaluated in T-index already created.

- **Index Fabric** Index Fabric [CSF+01] is a state-of-the-art research effort in path based indexing and querying. All the root-to-leaf paths in an XML documents are encoded as strings by using designators (i.e. B for <Buyers>). A well-known Patricia trie is utilised to index these encoded paths. Raw paths (encoded root-to-leaf paths) and refined paths (specialised paths to optimise frequently occurring access patterns) are proposed to accelerate a single path evaluation in XML documents. To achieve graceful scaling properties, Index Fabric proposed a layered Patricia trie, allowing a search to proceed directly into a block-sized subtries. Since Index Fabric is designed for top-down path evaluation, evaluating twig pattern queries in XML data by using Index Fabric may thus involve backtracking and require mechanisms of upward navigation. XRel [YASU01] proposed another similar path based approach to decompose XML documents into nodes with path information and store these fragments into a relational database to support data storage and retrieval in XML documents.

Node Identification Based Indexing and Querying Node identification based indexing and querying techniques share the following common feature: unique node identification (label) is assigned to each node in a data tree (graph) to determine the relationships between any given two nodes in twig patterns.

State-of-the-art research efforts tackles two major research problems: (1) Indexing and querying with support of RDBMS (i.e. relational database enabled indexing and querying) and (2) Indexing and querying with support of B+Tree and R+Tree index structure or their variants (i.e. native XML database).

- **RDBMS Supported Indexing and Querying** Daniela Florescu et al. [FK99] and Jayavel Shanmugasundaram et al. [STZ+99] firstly proposed the idea of storing and querying XML data with the support of RDBMS using binary mapping or DTD schema.

  - XRel [YASU01] is the first research effort which utilises numbered labelling scheme to support data access to XML data. Four fixed relational schemas, *Element(docID, pathID, start, end, index, reindex)*, *Attribute(docID, pathID, start, end, value)*, *Text(docID, pathID, start, end, value)*, *Path(pathID, pathexp)* are proposed to decompose and store XML data graph. Each element or text node is assigned with a region which is a pair of two identical numbers equal to the start position of the parent element node plus one. In XRel, queries are later evaluated by transferring *XPathCoreGraph* into corresponding SQL expressions to access data in the XML documents. The advantage of this approach is that it can easily identify whether a node is in a path from another node by using θ-joins (to test the containment- relationship using $>$, $<$). But θ-joins are considered as more costly than *equi-joins.* In addition, off-the-shelf database management systems usually do not have a special index mechanism to support containments.
  - Chun Zhang et al. [ZND+01] treated XPath expression queries as containment queries and proposed *MPMGJN* (Multi-Predicate Merge

Join) algorithm to access XML data with the support of inverted index (E-index and T-index), which is created on the basis of a pre-order numbering scheme where each node is labelled with (docno, begin:end, level). The two fixed schemes of MPMGJN are *ELEMENTS(term, docno, begin, end, level) and TEXTS(term, docno, wordno, level).* Different from XRel, queries are *joined* based on the containment of any two nodes specified in an XPath expression. Shurug Al-Khalifa et al. [AKJKP02] also proposed a general tree-merge algorithm, which is similar to MPMGJN, to efficiently find all the occurrences of parent-child and ancestor-descendant relationships in an XML database.

  - Igor Tatarinov et al. [TVB+02] proposed an improved order encoding methods to efficiently support data access to the ordered XML data model in relational database. Wei Wang et al. [WPFY03] proposed two models: interval model and position model, and a set of estimation methods to estimate the size of results of a containment query. These two research [TVB+02, WPFY03] efforts can be treated as complimentary to Chun Zhang et al. [ZND+01]. Moreover, all the research efforts in this part share common properties: (1) The pre-order numbering scheme is exploited and (2) RDBMS is fully utilised or adapted to support binary *join* operations.

- **B+Tree and R-Tree Supported Indexing and Querying** Despite of the previous research efforts on bridging XML and RDBMS, it is widely believed that the relational database technology failed to deliver necessary functionality of efficiently processing regular path expression queries. The primitive tree structured relationships are parent-child and ancestor-descendant, and finding all occurrences of these relationships in an XML database is thus a core operation for XML query processing.

- Quanzhong Li et al. proposed *XISS* (XML Indexing and Storage System) [LM01] to index and store XML data on the basis of the pre-order numbering scheme. All the nodes in an XML data tree are labelled with (pre-order, size, level) and are indexed in B+Tree. Regular expression queries are later processed by using (1) $\varepsilon\varepsilon$-Join (searching paths from an element to another), (2) εA-Join (searching element-attribute pairs), and (3) *KC*-Join (finding Kleene-Closure on repeated paths or elements) with the support of the above index. Interestingly, both XISS and MPMGJN break down the XPath expression queries into binary twigs and join the temporary results of individual twigs on the basis of a corresponding numbering scheme.
- Shurug Al-Khalifa et al. [AKJKP02] improved XISS by proposing a set of stack-tree algorithms to efficiently find all the occurrences of binary parent-child and ancestor-descendant relationships. These structural join algorithms guaranteed the linear I/O and CPU complexities of finding binary twigs. Both the space and CPU time complexity of the stack-tree algorithms are O($|A|$ + $|D|$ + k) where $|A|$ and $|D|$ are the number of ancestor list A and the descendant list D, k is the size of output list. The corresponding I/O complexity is O($\frac{|A|}{B}$ + $\frac{|D|}{B}$ + $\frac{k}{B}$).
- Shu-Yao Chien et al. [CVZ+02] pointed out that previous structural join algorithms [AKJKP02, ZND+01, TVB+02] do not utilise index structures but sequentially scan the input lists. I/O's can be wasted for scanning elements that do not participate in the join. Shu-Yao Chien et al. [CVZ+02] proposed several index structures on the basis of B+Tree and R-Tree to efficiently support structural join opera-

tions and optimise the corresponding performance of data accessing by skipping descendants and some ancestors without matches in the lists (Ancestor-List and Descendant-List) by probing the B+-tree indexes. Recently, Yuqing Wu et al. [WPJ03] further optimised the work of Shurug Al Khalifa et al. [AKJKP02] and Shu-Yao Chien [CVZ+02] and proposed five algorithms leading to structural join order optimisation for XML tree pattern matching on the basis of dynamic programming.

– Nicolas Bruno et al. [NND02] points out two problems of the above binary structural join techniques: (1) matching binary twigs against an XML documents is inefficient and (2) the intermediate result sizes can get very large. Nicolas Bruno et al. [NND02] proposed top-down *holistic twig join* algorithms to efficiently solve the above problems in querying stream XML data. *PathStack* is proposed to compactly represent partial results to root-to-leaf query paths, which are then composed to obtain matches for the twig pattern. *TwigStack* further improved the *PathStack* and only generated solutions that are parts of the final result, ensuring that before a node h_q is pushed on its stack S_q: (1) h_q has a descendant $h_{q'}$ in the input list $T_{q'}$, where q' is a child of q; and (2) each $h_{q'}$ satisfies the property (1). Nicolas Bruno et al. [NND02] proposed *XB-Tree* to solve the suboptimality for parent-child edges in a twig pattern. More recently, Haifeng Jiang et al. [JLW03] proposed *XR-Tree* to improve the I/O performance of holistic twig join algorithms on all/partly indexed XML documents. Byron Choi et al. [CMW03] studies the problem of optimising the performance of holistic twig join algorithms in current data streaming models and shows that the top-down computation of twig queries is not memory-bounded.

- More recently, Haixun Wang et al. [WPFY03] proposed *ViST*, a novel indexing and querying technique, to accelerate XPath evaluation steps. ViST transforms both XML documents and XPath queries into structure encoded sequences utilizing pre-order traversal. Structure encoded document sequences are sorted into a trie where the corresponding B+Tree index structure is built on. This mechanism turns twig pattern matching problem into subsequence pattern matching problem and greatly improves all the previous searching mechanisms since no join operations are in need.
- Haixun Wang et al. [WM05] points out that subsequence matching in ViST is not equal to the actual query evaluation because of the existence of false alarms and needs post-processing phases to eliminate the false answers. Haixun Wang et al. [WM05] defined the *sibling-cover* relationship between two nodes in the trie, and proposed a specialised trie+list data structure to find *sibling-cover* to remove false answers. However, ViST related techniques lack flexibility since a specialised data structure must be involved and the space complexity of the index structure in the worst case is O(n^2).

2.4 Summary

The semistructured data is motivated by the challenges posed by existing electronic data (e.g. HTML and SGML) The emergence of XML has remarkably facilitated the above electronic data publication by providing a simple syntax for data which is both machine- and human-understandable. However, the emergence of XML brings interesting challenges for the database community in efficient data retrieval, version control, change detection, indexing, and data management.

Those challenges can not be efficiently solved using current techniques. In this chapter, an overview of semistructured data, XML data model, its schema and query languages is briefly presented and substantial research efforts on the issues covered in this thesis: clustering XML documents, labelling XML documents and querying XML data using an index structure are carefully reviewed. These research efforts surveyed so far still need improvement and their deficiencies are the motivations of this thesis.

CHAPTER 3

Clustering XML Documents

Birds of a feather flock together.

– GoEnglish.com Pocket English Idioms

Clustering is the process of grouping the data into classes or clusters so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters [HK00]. Data clustering contributes to a number of areas of research including data mining, statistics, machine learning, spatial database, biology and markets [Dun03]. It has been widely recognised as a powerful data mining technique and has been studied extensively in recent years. With the arrival of XML data, the challenges of clustering become more evident; due to the different hierarchical structures of XML data, it is often more difficult to cluster XML data by only clustering the data according to their structural and content characteristics, more desirably, according to their implied semantics. In this chapter, the issue of semantically clustering the increasing number of the schemaless (i.e. no separate description of the type or structure of data such as DTD or XML schema) XML documents is discussed. The technical contributions in this chapter are as follows:

- An advance semantic component model [SW03] is proposed to model an XML document. An XML document is then represented by a series of semantic components, in which the semantic of an XML document is maximally preserved.

- A set of heuristic algorithms are introduced to compute the similarity among documents. These algorithms rely on the semantic component model in computing the degrees of similarity in a document collection, while preserving the linguistic, structural, content and semantic information in the components when clustering. Theoretical bounds of the algorithms as well as details of implementations are presented.

- An approximate mechanism is presented to compute the problem of the tree inclusion and the isomorphic tree similarity. In this approach, a tree inclusion range value is utilised to detect the tree inclusion while a best matching schedule is proposed to compute the similarity among isomorphic trees. The experiment results provide valuable insights into the tradeoff between computing efficiency and clustering accuracy.

3.1 Motivation

XML [W3C00] has become the standard of data representation and exchange. It allows users to define elements utilising words and organise them in a nested structure. Nowadays, the subset of the Web formed by these XML documents, hereafter called the XML web [MBV03], is growing into a large XML data repository. Consequently, integrating the XML documents to realise a Web database is desirable. However, Mignet et al. [MBV03] show that only 48% of the documents contain the links to the specific schemas. It implies that integrating the enormous schemaless and semantically different documents is a difficult task.

The above integrating task can be relieved by utilising data clustering technique [HK00, Dun03, BYRNRN99], which assembles together the related documents. It implies that relevant documents which are highly similar to each other are grouped in the same cluster. A very large XML data collection can thus be automatically divided into smaller sub-collections. The benefits of document clustering are obvious and are the motivations to perform this research:

- Efforts in integrating XML documents with different structures and semantics can be alleviated because reconciling analogous and relatively small document set is easier.
- Ranges of queries can be dramatically decreased to applicable documents after relevant documents are aggregated together.

3.2 Related Work

Significant challenges for clustering XML data exist on how to abstract and represent distinctive semantic characteristics and to design an effective yet efficient similarity measure based on them. Several research approaches [LLK01, LYHY02, FMM+02, Xyl01, LCMY04, DAM02, FGM03] had tackled the issue. However, the deficiencies in these efforts are not trivial.

Xyleme [Xyl01] is a dynamic warehouse for XML data in the Web. Its data integration method is mainly based on natural language and machine learning techniques. Firstly, DTDs are classified into domains based on a statistical analysis of the ontology similarities between words in different DTDs. Secondly, an abstract DTD for a specific domain is manually constructed by administrator. Finally, the semantic connections (mappings) between elements in the abstract DTD and elements in concrete ones are generated. However, Xyleme's semantic preservation mainly relies on human interaction, and its accuracy and extensibility degrade with the increasing number of DTDs.

Mong Li Lee et al. introduce XClust [LYHY02] to automatically cluster DTDs for a data integration system. The approach models a DTD as a tree T(V, E) where V is a set of nodes and E is a set of edges. To facilitate schema matching, [LYHY02] propose a set of transformation rules to determine the degree of similarity of two elements that have AND-OR nodes. Finally a similarity matrix among DTDs is computed and a hierarchical clustering technique is utilised to group DTDs into clusters. This approach considers not only the linguistic and structural information of DTD elements but also the context of a DTD element. However, its loss of semantic information is inevitable due to its transformation rules. Furthermore, to adapt XClust for the schemaless documents is not trivial due to its computational complexity which is only acceptable in case of DTDs (i.e. a relatively small number of nodes in a DTD).

Laurent Mignet et al. [MBV03] conclude that only 48% XML documents in the web contain the links to the schemas. This implies a huge amount of XML documents are schemaless. Thus providing an efficient mechanism to cluster these schemaless XML documents is very necessary.

To tackle the problem, Jung-Won Lee et al. [LLK01] propose a methodology for computing the similarity between XML documents by computing a minimal hierarchical structure of an XML document using automata, then adapting a sequential pattern mining algorithm to compute the similarity between XML documents. However, the algorithm failed to detect isomorphic tree structures though the semantic is partly considered.

Jong P. Yoon et al. [YRCK01] describe a bitmap indexing technique to cluster the XML documents. In this approach, an XML document is firstly defined as a sequence of ePaths with associated element contents. Secondly, a bitmap index is constructed from all the ePaths in the documents. Finally, a bitcube is assembled when element contents of the ePaths are integrated into the bitmap index. Similarity and popularity operations are defined to compute the similarity among these documents. Statistical analysis is also provided. Though Bitcube can support clustering XML documents, it contains many unnecessary zero bits, especially for the word dimension. The size of Bitcube impedes both its extensibility and computational efficiency.

Antoine Doucet et al. [DAM02] address the problem of clustering a homogenous collection of text-based XML documents. In this approach each document is represented by an n-dimensional vector, which is generated by using three different feature (defined as most significant words in the document) sets in an XML document: 1) text features only; 2) tag features only; 3) text and tags. Secondly, the document vectors are filled with normalised frequency and inverted document frequency measures. Finally, k-means clustering algorithm is utilised to cluster these documents. This approach is purely text-based and ignores the structural information in XML documents. Thus the semantic information of an XML document is completely ignored.

Sergio Flesca et al. [FMM+02] propose a technique for detecting the similarity in the structure of XML documents. This approach represents the structure of an XML document as a time series. Each impulse in the time series corresponds to a certain tag in the document. The similarity between documents is computed by analysing the frequencies of the consonant Fourier transform. Though this approach can approximately determine the structural similarity between XML documents, it can not detect isomorphic tree structures, and in particular, the semantics are not preserved due to its linear encoding mechanism.

To accurately cluster schemaless XML documents, the proposed technique should:

- preserve the semantics defined in the XML documents when clustering,
- support structural information related computation, and
- find a balance between efficiency and accuracy.

More importantly, there is not much work on semantically classifying schemaless XML documents. To overcome these deficiencies, an advance and accurate clustering technique is proposed to semantically assemble the schemaless XML documents.

However, the above approaches surveyed so far do not match the above requirements due to their computational complexity and loss of semantic information. To address these problems, the semantic component model is thus proposed.

3.3 Semantic Component Model

In this chapter, semantic is defined as the machine-understandable information which describes the meaning of the corresponding XML data. However, XML has no formal semantic which implies that a practical definition on the semantic must be provided. Moreover, it is difficult to define and compute the similarity among the schemaless (i.e. no separate description of the type or structure of data [ABS00]) XML documents as the semantic is concerned. To tackle the problem, an accurate clustering technique is proposed to group these schemaless XML documents on the basis of a competent, semantic-preserving document modelling mechanism.

3.3.1 Requirements

XML is pervasively modelled as a tree. However, computing the degrees of the similarity among the trees is not feasible due to the computational complexity. It implies that an efficient mechanism should be provided to model an document tree, and ideally preserve its semantic. The proposed model should,

- maintain the linguistic information in an XML document.
- maximally prevent the semantic information loss.
- efficiently support a set of algorithms to possess an acceptable tradeoff between computational complexity and accuracy when clustering a large amount of documents.

Currently, there is no formal semantic for an XML document. It implies that a practical definition on the semantic must be provided. In this thesis, it is proposed that the semantic expressed in an XML document should be related to:

- terminological information in XML data. It includes the terms used in the set of tag names and the keywords of the data.
- structural information in XML data. It includes the nested structure of XML data, describing how the tags are organised and nested.

To match the above three requirements, a semantic component model is introduced to semantically simplify an XML document tree.

```
<Order SEQNUM = "ifrtei39djued">
    <Billto BCODE = "ELEC86">
        <Customer MEBREF = "cus0122001skfd">
            <FirstName>Andrew</FirstName>
            <LastName>Angel</LastName>
        </Customer>
    </Billto>
    <Payment CUR = "EURO">
        <RateStamp> bst0407032001ssrf </RateStamp>
    </Payment>
    <Shipto SCODE = "OSEA44">
        <Customer REFNO = "elec3293402stet" CONCODE = "21">
            <FirstName>Johnson</FirstName>
            <LastName>Will</LastName>
        </Customer>
        <ExtraInfo>Return Customer</ExtraInfo>
    </Shipto>
</Order>
```

Figure 3.1: An Example of XML Data

3.3.2 Specification of Semantic Component Model

Semantic component model is based of an unordered, node labelled, rooted tree. More formally, consider a graph $\mathbf{T} = (\mathbf{V_G}, \mathbf{V_T}, \mathbf{v_r}, \mathbf{E_G}, \mathbf{label_{node}}, \mathbf{nid}, \sum_\mathbf{T})$.

- V_G is the set of element nodes and V_T is the set of text nodes. $\forall v \in V_T$, v has no outgoing edge. v_r is root of the XML data tree, where there exists a path from v_r to v, $\forall v \in V_G \cup V_T$. It implies that v_r has no incoming edge. Each node $v \in V_G \cup V_T$ is labelled through the function $label_{node}$ over the set of terms, $\sum_T$. The label of a node $v \in V_G$ is referred to the tag name. The label of $v \in V_T$ is referred to a distinguish keyword contained in the corresponding text. Quotation mark in future figures is used to distinguish the label of V_T.

- Each edge e, $e \in E_G$, is a parent-to-child edge, denoting the parent-child relationship. The parent node is denoted as v_{e_p}, and the child node is denoted as v_{e_c}. A path is a sequence of edges starting from the node v_i to the node v_j, denoted as e_i, e_{i+1}, ..., e_j. A node v_i is ancestor of v_j *iff* a path to v_j goes through v_i.

- The order among siblings is ignored due to the observation that the order will cause semantic information loss when computing the similarity value between two documents. An example is given in Section 3.4.3.

An example XML document is listed in Figure 3.1, and its data tree is shown in Figure 3.2. The value of text node is not shown in Figure 3.2. In the rest of this chapter, an XML document is denoted as D_T. Let $\sum_{D_T}$ denotes the set of tag and attribute names in a document D_T. In the rest of the section, the heuristic rule of generating semantic components is firstly presented, then

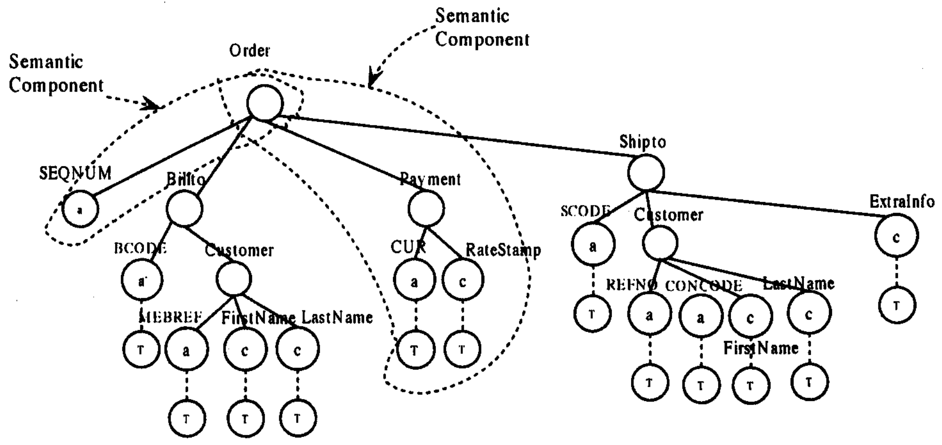


Figure 3.2: An XML Data Tree

three specific definitions on path component, attribute component, and content component used in semantic component model is proposed and discussed.

We observe the fact that the nodes without any direct attribute nodes or content nodes are mostly utilised to clarify the structure of an XML document. Their semantic information is less important compared with the nodes having attributes or content, in which the essential contextual information of such a node is described in detail. Therefore, only the significant semantic information in the document needs to be preserved when modelling an XML document as a series of semantic components. In this chapter, a semantic component is generated only at a certain node v_i which has attributes or content nodes. It implies that *attribute component* or *content component* may be empty when a node does not have any attributes or content nodes, and the nodes without any attribute nodes or content nodes are not eligible for semantic component generating. Thus the semantic information in these nodes are lost. Nevertheless, in the next section the semantic loss of these nodes can be recaptured when computing the similarity among *semantic components* similarity.

Definition 3.1 Path Component Given a node v_i in an XML document tree D_T, the path component $P_{v_i}^{D_T}$ of the node v_i is defined as an *ordered* sequence of tag names from root to node v_i, denoted as

$$P_{v_i}^{D_T} = (v_0, v_1, ..., v_i), v_k \in \sum_{D_T}, k \in [1...i]$$

$P_{v_i}^{D_T}$ describes the structural information on how a node v_i is nested in an XML document. Given the node v_l and the node v_p, v_l is *nested* in v_p when l>p $\forall v_l$, $v_p \in P_{v_i}^{D_T}$.

Example 3.1 Consider, for example, in Figure 3.2. The $P_{Customer}^{D_T}$ of node "Customer" in the "Shipto" element is {Order, Shipto, Customer}. It means the node "Customer" is nested in a node "Shipto", which is further nested in a node "Order".

Definition 3.2 Attribute Component Given a node v_i in an XML document tree D_T, the attribute component $A_{v_i}^{D_T}$ of the node v_i is defined as a set of its attribute names, and denoted as

$$A_{v_i}^{D_T} = \{a_0, a_1, ..., a_l\}, a_k \in \sum_{D_T}, k \in [1...l]$$

Example 3.2 Consider, for example, in Figure 3.2. The $A_{Customer}^{D_T}$ of node "Customer" in the "Billto" element is {MEBREF}. It means the node "Customer" have only one attribute, called "MEBREF". Contrasting to the path component, the order in the attribute component is ignored in further similarity computing algorithms.

Definition 3.3 Content Component Given a node v_i in an XML document tree D_T, the content component $C_{v_i}^{D_T}$ of the node v_i is defined as a set of the tag names of its direct descendants, whose direct descendants are content (text node) in the data model, and is denoted as

$$C_{v_i}^{D_T} = \{c_0, c_1, ..., c_r\}, c_k \in \sum_{D_T}, k \in [1...r]$$

Example 3.3 Consider, for example, in Figure 3.2. The $C_{Customer}^{D_T}$ of node "Customer" in the "Shipto" element is {FirstName, LastName}. More importantly, the duplicate element names are not omitted in the content.

Definition 3.4 Semantic Component (SC) Given a node v_i in an XML document tree D_T, a semantic component of node v_i is uniformly defined as a record with three fields:

```
Semantic Component{
    path component: structural information
    attribute component: contextual information
    content component: contextual and content information
}
```

An XML document is later modelled into a series of semantic components, denoted as

$$D_T = \{SC_0^{D_T}, SC_1^{D_T}, ..., SC_t^{D_T}\}, t = |D_T|$$

When an XML document consists of only a root node, t = 0. The order in D_T is also ignored to support the isomorphic tree similarity and the tree inclusion similarity.

Example 3.4 Consider, for example, in Figure 3.2. The semantic component of node "Customer" nested in the "Shipto" element is {(Order, Shipto, Customer), {REFNO, CONCODE}, {FirstName, LastName}}. The semantic component of node "Payment" is ((Order, Payment), {CUR}, {RateStamp}). The complete list of semantic components of Figure 3.2 is shown in Table 3.1.

SC	Components
SC_1	{(Order), {SEQNUM}, {}}
SC_2	{(Order, BillTo), {BCODE}, {}}
SC_3	{(Order, Billto, Customer), {MEBREF}, {FirstName, LastName}}
SC_4	{(Order, Payment), {CUR}, {RateStamp}}
SC_5	{(Order, Shipto), {SCODE}, {ExtraInfo}}
SC_6	{(Order, Shipto, Customer), {REFNO, CONCODE}, {FirstName, LastName}}

Table 3.1: List of Components of Figure 3.2

3.4 Semantic Component Similarity

After an XML document is modelled as a series of semantic components, the document similarity problem is further turned into a semantic component similarity problem. To accurately compute the similarity, the algorithms should:

- compute the linguistic similarity between the tag names. This characteristic targets to the linguistical accuracy of clustering.
- preserve the semantic of the semantic components. This characteristic targets to the semantical correctness and accuracy of clustering.
- provide an acceptable computational complexity. This characteristic targets to the tradeoff between the computational efficiency and accuracy of clustering.

To meet the above requirements, two algorithms are presented in this section. The first one is to compute the degree of the similarity between two path components and the second one is to uniformly compute the similarity value between two attribute components (respectively, content components). A weight mechanism is also proposed to flexibly combine these similarity values in different contextual models. The semantic component similarity is formally defined as follows: given two semantic components, $SC_{v_i}^{D_{T_m}} = (P_{v_i}^{D_{T_m}}, A_{v_i}^{D_{T_m}}, C_{v_i}^{D_{T_m}})$ and $SC_{v_j}^{D_{T_l}} = (P_{v_j}^{D_{T_l}}, A_{v_j}^{D_{T_l}}, C_{v_j}^{D_{T_l}})$ The similarity between $SC_{v_i}^{D_{T_m}}$ and $SC_{v_j}^{D_{T_l}}$ is defined as:

$$\begin{aligned} SCSimilarity(SC_{v_i}^{D_{T_m}}, SC_{v_j}^{D_{T_l}}) = Weight(&PathSimilarity(P_{v_i}^{D_{T_m}}, P_{v_j}^{D_{T_l}}) \\ &AttrSimilarity(A_{v_i}^{D_{T_m}}, A_{v_j}^{D_{T_l}}), \\ &ContSimilarity(C_{v_i}^{D_{T_m}}, C_{v_j}^{D_{T_l}})) \end{aligned} \tag{3.1}$$

3.4.1 Name similarity

The metric of computing the name similarity between tag names is firstly discussed. This metric is mainly focused on the linguistic similarity among the terms which are used to define element tag names (i.e. attributes, content etc.). This kind of similarity has been broadly utilised in many applications, such as information retrieval, information extraction, text classification, word sense disambiguation, example-based machine translation, etc. In this chapter, the WordNet [Fel98] API which returns the synonym set is utilised to support the purpose. Additionally, the terms in an XML document in order is pre-processed to eliminate abbreviated words which can not be processed by WordNet. For example, "MEMREF" is normalised to "reference".

The algorithms are given in Algorithm 3.1 and Algorithm 3.2. The name similarity value is denoted as θ_{name} in the rest of the chapter. The computational complexity is connected with the number of synonyms of a certain word. Consider, for example, in the algorithm, NameSimilarity(Bill, Invoice) returns 0.9 if

```
input  : $e_1$, $e_2$: the names of two elements. $\beta$: the base ontology
         similarity value, the default value is 0.9. $\delta$: the search depth,
         the default maximum value is 4.
output: $\theta_{name}$: the name similarity value
if $e_1 = e_2$ then
    return 1.0
else
    $Set_{Syn} \leftarrow$ the synonym set of $e_2$;
    $\theta_{name} \leftarrow$ VocabularySimilarity ($e_1$, $Set_{Syn}$, 1, $\beta$);
return $\theta_{name}$
```

Algorithm 3.1: NameSimilarity

β in Algorithm 3.1 is set to 0.9, NameSimilarity(Allocate, Ship) returns 0.81, and NameSimilarity(FirstName, Order) returns 0. In the rest of the chapter, all the default values are based on the experience of processing XML documents.

3.4.2 Path Component Similarity

The path similarity metric is carried out on the fact that a path component describes the detailed information of how an element is nested in an XML document due to its order (Definition 3.1). The structural similarity between two nodes are therefore captured by computing the path component similarity. In this thesis, it is connected with:

- Given the name similarity value θ_{name}, if θ_{name} is greater than the threshold value (τ_{name}), it is defined that a *matching point* is reached. If not, a *gap* is obtained.
- The path alignment is then connected with the gaps between continuous matching points.

```
input : e: an element name. $Set_{syn}$: a synonym set. $\delta$: the
        search depth, the default maximum value is 4. $\beta$: the
        base vocabulary similarity value, the default value is 0.9
output: $\theta_{vocabulary}$: the vocabulary similarity
if $\delta \leq 4$ then
    if $e \in Set_{Syn}$ then
        return $\beta^{\delta}$;
    else
        $Set_{Syn_{new}} \leftarrow \emptyset$;
        for each $s_i \in Set_{Syn}$ do
            $Set_{s_i} \leftarrow$ the synonym set of $s_i$;
            $Set_{Syn_{new}} \leftarrow Set_{Syn_{new}} \cup Set_{s_i}$;
        VocabularySimilarity($e$, $Set_{Syn_{new}}$, $\delta + 1$, $\beta$);
else
    return $0$;
```

Algorithm 3.2: VocabularySimilarity

To match the above two requirements, a heuristic algorithm for path component similarity is given in algorithm 3.3. The computational complexity is O(mn), where m and n stand for the individual lengths of two path sequences.

Consider for example, in Figure 3.3. Though the two path components have the same matching points, however, P_1 is better than P_2 for there is no gap between matching points in P_1 while there exists one gap in P_2. Thus the algorithm has a subtle weight mechanism not only for the name similarity but also for the structural similarity when comparing two path sequences. Moreover, though the nodes without any attribute nodes or content nodes are not eligible for semantic component generation, their semantics are thus partly re-captured

```
input : $P_{v_i}, P_{v_j}$: the path components; penalty: a penalty value
        for mismatching of tag names; $\tau_{name}$: the threshold
        value, default value is 0.6.
output: $\theta_{path}$: the path sequence similarity value.
$m \leftarrow LEN(P_{v_i})$;
$n \leftarrow LEN(P_{v_j})$;
Initialise an m-by-n array $S_{m \times n}$ to zero;
$r, t \leftarrow 0$;
for $r \leftarrow 1$ to $m$ do
    for $t \leftarrow 1$ to $n$ do
        S(r, t) ← $MAX$(S(r-1, t) + penalty, S(r, t-1)) + penalty,
        S(r-1, t-1) + Select ( $P_{v_i}(r)$, $P_{v_j}(t)$, $\tau_{name}$));
$\theta_{path} \leftarrow S(m, n)/MAX(m, n)$;
return $\theta_{path}$
```

Algorithm 3.3: PathSimilarity

when computing the path similarity because they are encoded within the path components.

3.4.3 Attribute(Content) Component Similarity

the similarity between attribute(content) components is motivated by the fact that edit distance is not sufficient to detect the similarity between attribute (respectively content) components. For example, given two XML fragments,

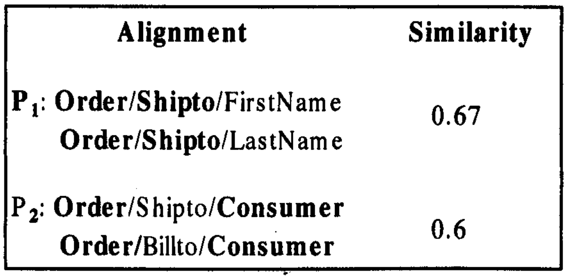
$$< PDATEDAY = "02"MONTH = "Feb"YEAR = "1998"/ > (PDATE_1)$$

and

$$< PDATEMONTH = "May"DAY = "12"YEAR = "2003"/ > (PDATE_2)$$

```
input  : $e_1, e_2$: element names. $\tau_{name}$: the threshold value, de-
         fault value is 0.6
output: value: selected name similarity value
if ((value ← NameSimilarity ($e_1$, $e_2$, $\delta$, $\beta$))> $\tau_{name}$) then
    return value;
else
    return 0;
```

Algorithm 3.4: Select



Alignment	Similarity
P_1: **Order/Shipto/**FirstName **Order/Shipto/**LastName	0.67
P_2: **Order/**Shipto/**Consumer** **Order/**Billto/**Consumer**	0.6

Figure 3.3: Path Alignment Weight

They are different localisations of an certain XML document on payment date and contain the same semantic information. But if edit distance algorithm [FMM+02, Dun03] is utilised to compute the similarity, the semantic information is ignored. To maximally prevent semantic information loss, an efficient algorithm is proposed in Algorithm 3.5. The computational complexity is O(mn), where m and n stand for the individual size of two attribute (respectively, content) components.

Consider, for example. Given two attribute components, $A_{v_i} = \{Send, Payment\}$, $A_{v_j} = \{Bill, Ship\}$ and τ_{name} 0.6, ACSimilarity(A_{v_i}, A_{v_j}) = 0.9. We observe that the edit distance algorithms output 0 since the order is indispensable. Consider again the above two pay date fragments, A^{PDATE_1} = {DAY, MONTH, YEAR}

```
input  : $A(C)_{v_i}, A(C)_{v_j}$: attribute(content) components
output: $\theta_{ac}$: attribute(content) similarity value
$m \leftarrow LEN(A(C)_{v_i})$;
$n \leftarrow LEN(A(C)_{v_j})$;
for $r \leftarrow 1$ to $m$ do
    for $t \leftarrow 1$ to $n$ do
        if ((value $\leftarrow$ NameSimilarity $(A(C)_{v_i}(r), A(C)_{v_j}(t))$)
        $> \tau_{name}$) then
            $count \leftarrow count + 1$;
            $\theta_{ac} \leftarrow \theta_{ac} + value$;
$para \leftarrow$ Max$(count, m, n)$;
return $\theta_{ac}/para$;
```

Algorithm 3.5: ACSimilarity

and $A^{PDATE_2} = \{\text{MONTH, DAY, YEAR}\}$. ACSimilarity($A^{PDATE_1}$, A^{PDATE_2}) = 1.0 when τ_{name} is set to 0.6.

3.4.4 Semantic Component Similarity

The semantic component similarity is finally inspected on the basis of the above two algorithms. This technique is motivated by the fact that the most important semantic information of an XML document is reflected by the nodes that have either attribute nodes or content nodes. Definition 3.4 shows that the semantic information of a certain node v_i is preserved in its semantic component $(P_{v_i}^{D_T}, A_{v_i}^{D_T}, C_{v_i}^{D_T})$. When generating the final result, a weight mechanism is proposed to combine the path similarity and the attribute(content) similarity to obtain the final similarity value. In this chapter w_p, w_a, and w_c are respec-

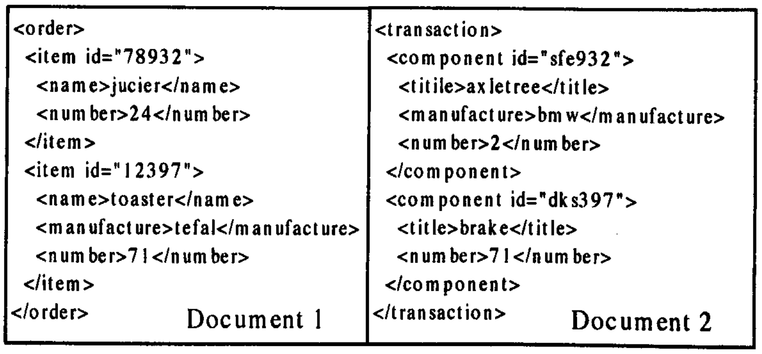
```
input  : $SC_{v_i}$, $SC_{v_j}$: two semantic components; $w_p$, $w_a$, $w_c$:
         weight value.
output: $\theta_{SC}$: SC similarity value.
$\theta_{path} \leftarrow$ PathSimilarity ($P_{v_i}$, $P_{v_j}$);
if $A_i = \emptyset$ or $A_j = \emptyset$ then
    $\theta_{atti} \leftarrow -0.1$;
else
    $\theta_{atti} \leftarrow$ ACSimilarity ($A_{v_i}$,$A_{v_j}$);
if $C_i = \emptyset$ or $C_j = \emptyset$ then
    $\theta_{cont} \leftarrow -0.1$;
else
    $\theta_{cont} \leftarrow$ ACSimilarity ($C_{v_i}$,$C_{v_j}$);
$\theta_{SC} \leftarrow (w_p \times \theta_{path} + w_a \times \theta_{attr} + w_c \times \theta_{cont})$;
return $\theta_{SC}$;
```
Algorithm 3.6: SCSimilarity

tively used to represent the weight values for P_{v_i}, A_{v_i}, and C_{v_i} where w_p, w_a, $w_c \in [0,1] \wedge w_p \geq w_a \geq w_c$. By using the weight mechanism, we can flexibly adjust how the three sequences are combined. The complete algorithm is given in algorithm 3.6.

Example 5. Consider, for example, given two semantic components in Table 3.2, if given $w_p = 1.0$, $w_a = 0.9$, $w_c = 0.8$, SCSimilarity(SC_{v_i}, SC_{v_j}) = ($w_p\times$ PathSimilarity({Order, Billto, Customer}, {Order, Shipto, Customer}) + $w_a\times$ ACSimilarity({MEBREF}, {REFNO, CONCODE}) + $w_c\times$ ACSimilarity({FirstName, LastName}, {FirstName, LastName}))/3 = (1.0× 1.0 + 0× 1.0 + 1× 1)/3 = 0.67.

SC_1	path = (Order, Billto, Customer)
	attribute = MEBREF
	content = FirstName, LastName
		SC_2	path = (Order, Shipto, Customer)
			attribute = REFNO, CONCODE
			content = FirstName, LastName

Table 3.2: An Example of Two Semantic Components



```
<order>
  <item id="78932">
    <name>jucier</name>
    <number>24</number>
  </item>
  <item id="12397">
    <name>toaster</name>
    <manufacture>tefal</manufacture>
    <number>71</number>
  </item>
</order>
```

Document 1

```
<transaction>
  <component id="sfe932">
    <titile>axletree</title>
    <manufacture>bmw</manufacture>
    <number>2</number>
  </component>
  <component id="dks397">
    <title>brake</title>
    <number>71</number>
  </component>
</transaction>
```

Document 2

Figure 3.4: Two Sample XML Documents

3.5 Clustering Schemaless XML Documents

3.5.1 Document Similarity

Given two XML documents,D_{T_r} = (SC_{r_1}, SC_{r_2}, ..., SC_{r_m}) and D_{T_t} = (SC_{t_1}, SC_{t_2}, ..., SC_{t_n}) The similarity between D_{T_r} and D_{T_t}) is formally defined as:

$$DocSimilarity(D_{T_r}, D_{T_t}) = \sum_{1 \le i \le m, 1 \le j \le n} select(SCSimilarity({SC_{r_i}}^{D_{T_r}}, {SC_{t_j}}^{D_{T_t}})) \tag{3.2}$$

The approximate tree inclusion and the isomorphic tree similarity are also supported in the algorithm. A complete algorithm is listed in Algorithm 3.7. The computational complexity is O(mn), where m and n stand for the number of semantic components in two documents.

Example 6 Consider, for example, in Figure 3.4. Two sample documents are

```
input: $D_{T_i}$, $D_{T_j}$: two XML documents. $\tau_{SC}$: SC threshold value.
       $\tau_{inclusion}$: tree inclusion threshold value, default value is
       0.9. $range$: tree inclusion range, its default value is 0.05
output: $\theta_{doc}$: document similarity value
$m \leftarrow LEN(D_{T_i})$;
$n \leftarrow LEN(D_{T_j})$;
$count \leftarrow 0$;
$uniset \leftarrow \emptyset$;
$unicount, \theta_{inclusion} \leftarrow 0$;
for $r \leftarrow 1$ to $m$ do
    $flag \leftarrow$ false;
    for $t \leftarrow 1$ to $n$ do
        if $((value \leftarrow \mathtt{SCSimilarity}\ (D_{T_i}(r),\ D_{T_j}(t))) > \tau_{SC})$
        and $(flag = false)$ and $(t \notin uniset)$ and $value \geq \tau_{inclusion}$
        then
            $\theta_{inclusion} \leftarrow \theta_{inclusion} + value$;
            $unicount \leftarrow unicount + 1$;
            $uniset \leftarrow uniset \cup \{t\}$;
            $flag \leftarrow$ true;
        $\theta_{doc} \leftarrow \theta_{doc} + value$;
        $count \leftarrow count + 1$;
$k \leftarrow \mathtt{Min}(m, n)$;
if $unicount \in [(1 - range) \times k,\ (1 + range) \times k]$ then
    return $\tau_{inclusion}/unicount$
$\theta_{doc} \leftarrow \theta_{doc}/\mathtt{Max}(count, m, n)$;
return $(\theta_{doc})$;
```

Algorithm 3.7: DocSimilarity

given as follows:

$$D_{T_1} = \{SC_{11}, SC_{12}\}(D_{T_1})$$

and

$$D_{T_2} = \{SC_{21}, SC_{22}\}(D_{T_2})$$

Let SC_{11} = {(order, item), {id}, {name, number}}, SC_{12} = {(order, item), {id}, {name, number, manufacture}}, SC_{21} = {(transaction, component), {id}, {title, number, manufacture}}, SC_{22} = {(transaction, component), {id}, {title, number}} and given $w_p = 1.0$, $w_a = 0.9$, $w_c = 0.8$ and $\tau_{SC} = 0.72$, $\tau_{name} = 0.6$, $\tau_{inclusion} = 0.9$. DocSimilarity(D_{T_1},D_{T_2}) is (0.688 + 0.73 + 0.778 + 0.688)/4 = 0.721. Moreover, we can discern that D_{T_1} and D_{T_2} are isomorphic trees when comparing their data trees.

3.5.2 Clustering an XML Document Collection

Given a collection of XML documents, $D=\{D_{T_1}, D_{T_2}, ..., D_{T_n}\}$ and the similarity matrix $M_{n \times n}=\{m_{ij}|m_{ij}$ = DocSimilarity(D_{T_i}, D_{T_j}), $1 \leq i,j \leq n\}$ The XML documents with similar semantic should be clustered together by analysing the similarity matrix M. Once these documents are grouped together, the benefit is obvious: the speed of data retrieving can be increased because range of a certain query can be dramatically decreased only to semantically applicable documents. The algorithm is given in Algorithm 3.8.

After the similarity matrix of the XML document collection is generated, a hierarchical clustering technique [Dun03] is utilised to group these XML documents on the basis of the matrix. In a hierarchical classification, the data are not partitioned into a particular number of clusters at single step. Instead the classification consists of a series of partitions which may run from a single cluster containing all individuals, to n clusters each containing a single individual.

```
input: $D$: an XML document collection
output: $M$: similarity matrix
$n \leftarrow LEN(D)$;
for $i \leftarrow 1$ to $n$ - 1 do
    for $j \leftarrow i+1$ to $n$ do
        M(i,j) = DocSimilarity ($D_{T_i}$, $D_{T_j}$);
return $M$
```

Algorithm 3.8: SimilarityMatrix

However, a hierarchical clustering technique suffers from its quadratic complexity seen as its main weakness. In general, hierarchical clustering has been considered as the best quality clustering approach, and "offline" clustering XML documents as a preparation step is acceptable in most systems. In this thesis, the foucs are not on the complete hierarchy but only of certain partitions obtained from it.

3.6 Experimental Results

In order to evaluate the performance of the semantic component decomposition and clustering, several experiments are performed. 100 XML documents are collected from the Internet and 100 schemaless documents are generated by specialised designed program. The above real world documents and synthesised documents are then randomly mixed to set up a general data environment. On the basis of the data sets, the execution time of semantic component decomposition and the clustering accuracy are evaluated. The computing environment is a Celeron 400MHZ machine with 128MB memory.

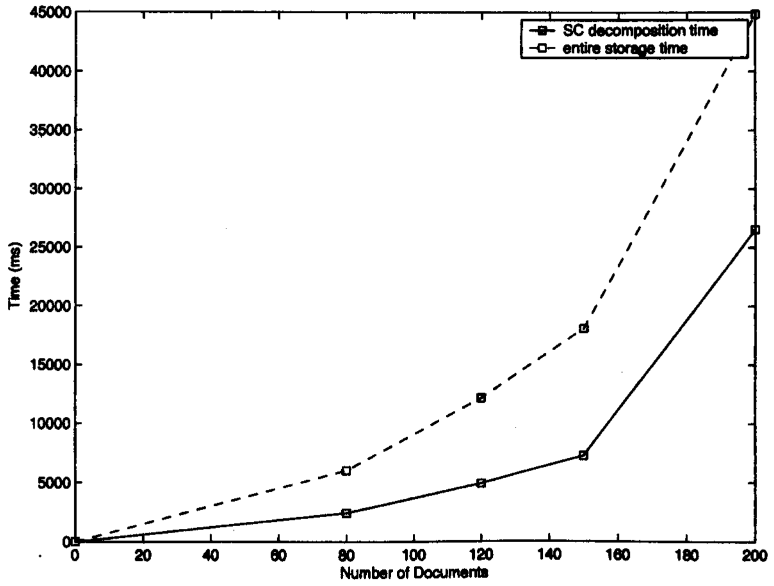


Figure 3.5: Entire Storage Time

Number of Documents	Size in Total(KB)	Time(ms)
80	14,238	2,444
120	23,924	4,928
150	36,321	7,261
200	59,114	26,548

Table 3.3: Semantic Component Decomposition Time

3.6.1 Performance of Semantic Component Decomposition

In this experiment, the execution time of the semantic component decomposition when storing an XML document into a local storage system is investigated. Four different data sets are initialised and each data set contains 80, 120, 150 or 200 documents. Two criteria are essential to evaluate the performance of semantic component decomposition: the reasonable computing time concerning the resource provided and the number of semantic components with respect to the number of individual nodes in a document. Two experimental results are obtained: (1) the semantic component decomposition time; and (2) the entire storage time. The experimental result on semantic component decomposition is listed in Table 3.3 and the entire storage time is in Figure 3.5. The storage time increases slightly when semantic component decomposition is involved but is still within the acceptable range.

The relationship between the number of SCs and the number of nodes in an XML document is also evaluated. A semantic component is treated as a unit when computing in the clustering algorithm. Thus, efficiently generating a number of semantic components with respect to the number of nodes is vital to the accuracy and efficiency of clustering. The experimental result is shown in Figure 3.6. The number of semantic components is less than 30% of the number of nodes in an XML document. Consequently, the efficiency of computing is guaranteed.

3.6.2 Performance of Clustering Efficiency and Accuracy

In this experiment, the accuracy of clustering schemaless XML documents is investigated. Firstly, the documents are manually classified into l clusters and these clusters are constructed as "ground truth". Secondly, the algorithms are executed to automatically group the documents. Finally, by specifying the various

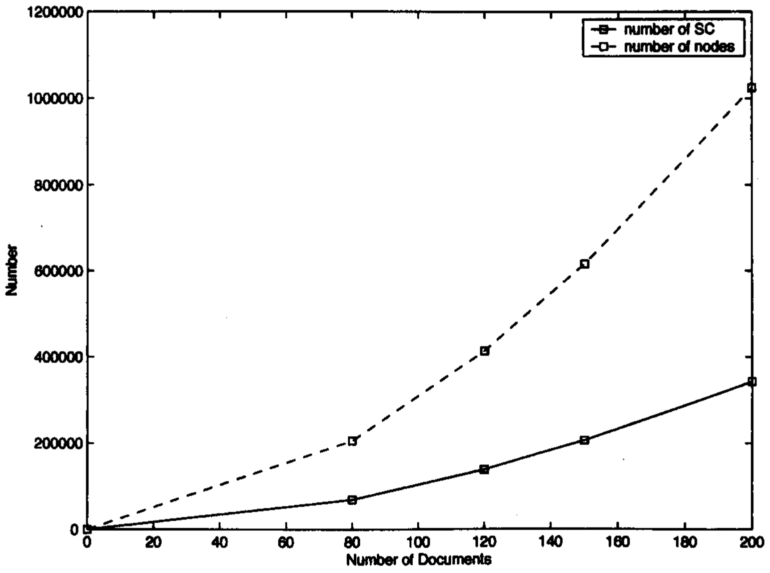


Figure 3.6: The Number of SCs with respect to the Number of Nodes in XML Documents

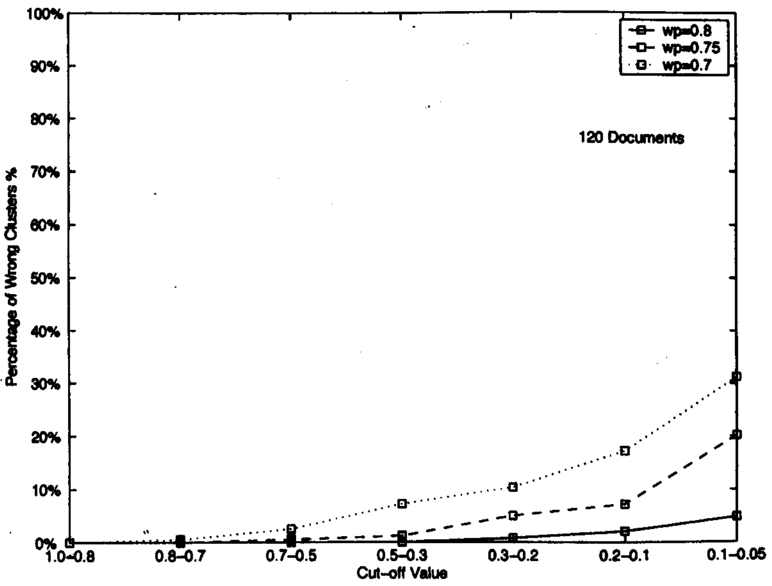


Figure 3.7: Clustering 120 Documents

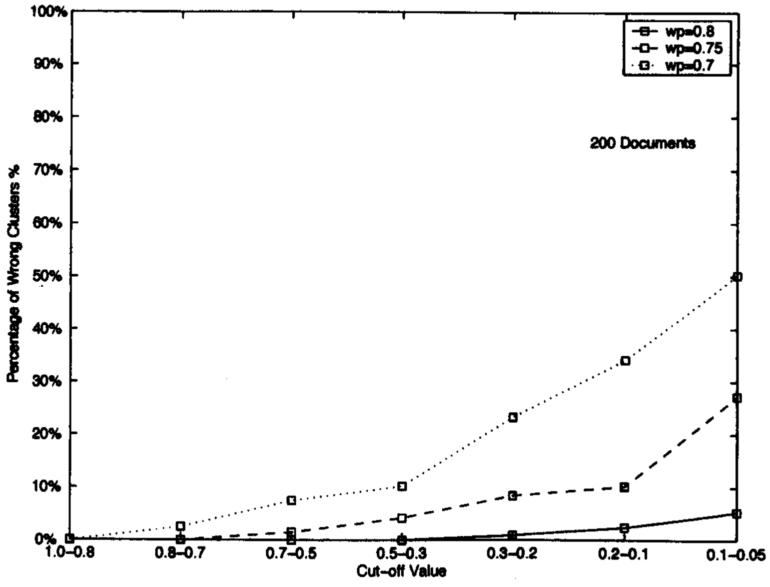


Figure 3.8: Clustering 200 Documents

cut-off values, the different number of clusters is generated. By comparing with the manual results, the accuracy of the algorithms can thus be defined. Figure 3.7 and Figure 3.8 respectively are the preliminary results for 120 and 200 documents. Three various values for w_c are experimented in each data set.

According to Figure 3.7 and Figure 3.8, the accuracy of clustering degrades when the number of documents in a collection increases. However, this problem can be partly solved when a statistical pre-processing mechanism is applied, i.e. pre-dividing a large document collection into small groups based on a statistical analysis of the words used in a certain domain. Also the accuracy of clustering degrades when the value of w_c is decreased. This shows that the direct descendant is an important part of a node's semantics which SC well preserves and the weight mechanism maximally preserves the semantic information within an XML document.

3.7 Summary

In this chapter, the comprehensive study on clustering schemaless documents is reported. Approximate modelling the semantic of an XML document is the key part of schemaless XML document clustering. The unique feature on the detection of tree inclusion and tree isomorphic renders previous approaches inapplicable. The proposed algorithms make the computing of similarity among XML documents practical. Extensive experimental results using both real and synthesised documents has demonstrated the effectiveness of the proposed methods. The tradeoff among various parameters is also investigated in the chapter. The proposed method offers opportunities in optimising information retrieval, document classification, and data mining.

CHAPTER 4

Holistic XPath Evaluation

You can never solve a problem on the level on which it was created.

– Albert Einstein

XML in data representation and exchange in large-scale scientific and commercial applications has become pervasive. Substantial research efforts have been made to provide efficient indexing and querying mechanisms to retrieve data from XML-formatted data. The ViST approach, proposed by Wang *et al.* [WPFY03], is an initiative and advanced [RM04] index method for querying XML data by tree structures. However, ViST suffers from the following three shortcomings. (i) Semantic flaw problem caused by ViST's inherent structure-encoded sequence model. That is, an XML document which semantically satisfies an XML query may not be returned; (ii) False answer (alarm) problem, i.e., documents which do not satisfy a query may be wrongly returned. Additional time-consuming refinement or post-processing phase has to be called to eliminate false query answers [WM05]; (iii) ViST cannot guarantee the linear size complexity of structure-encoded sequences. In the worst case, its space complexity reaches $O(n^2)$ (where n is the total number of nodes in an XML data tree).

Inspired by the promising indexing and querying performance of ViST among the rest of the approaches in the literature, and meanwhile to overcome its shortcomings, in this chapter, another efficient and novel geometric sequence mechanism [WFS05], which transforms XML documents and XPath queries into the corresponding geometric data/query sequences, is proposed. XML querying is thus converted to finding non-contiguous geometric subsequence matches. The proposed approach ensures correct (i.e., without semantic false) and fast (i.e., without the costly post-processing phase) evaluation of XPath queries, while at the same time guaranteeing the linear space complexity. The significant performance improvement of the proposed approach is demonstrated through a set of experiments on both synthetic and real-life data. The main contributions of this chapter are summarised as follows.

- XML documents and XPath queries are transformed into geometric sequences. Twig matches can be found by performing subsequence matching on the set of sequences. Together with a set of stacks proposed to efficiently accelerate the query steps, the proposed method finds all the answers without semantic false and false alarms, without performing post-processing phase. Moreover, the algorithms depend only on B+Tree index structure, which is well-supported by DBMS.

- Geometric sequence model also guarantees the linear complexity of the total size of geometric sequence in the number of nodes in the XML data trees. The length of a geometric sequence is proved to be $2n$ - 1 where n is the number of nodes in an XML tree.

4.1 Motivation

With the advent of XML as a standard for data representation and exchange on the Web, indexing and querying XML documents becomes increasingly important for current and future data-centric applications. Substantial research efforts

[CSF+01], [Gru02], [GW97], [MS99] have been conducted to structurally index and retrieve data from XML documents.

The first problem of retrieving data from XML documents is how to deal with specific queries containing constraints related to the content of the documents. Providing a uniform index structure [WPFY03] for both the structure and content information of an XML document is therefore desirable. More importantly, the mechanism should be preferably implemented using some well-supported DBMS data structures like B+Tree.

The second problem is that a query compatible to XPath is modelled as a tree, referred to as a *twig*, and can be complicated when wildcards "*" and self-or-descendent axis("//") are presented (for example, Q5 in Table 4.9). To match such a complex query against a document tree without corresponding preprocessing mechanism is equivalent to the tree inclusion problem and has been proved to be NP-complete [AHU74].

Previous research efforts have been devoted to twig pattern matching for several years. XISS [LM01] is the first to break twig pattern query into binary twigs, and "stitch" the binary twigs (i.e. two nodes with parent-child relationship) together to obtain the final results. State-of-the-art mechanisms, i.e. structural join [CTZZ02], holistic twig join [NND02], have been proposed to stitch root-to-node paths together by using specially designed stacks. Additionally, some index structures, such as XR-Tree [JLW03] and XB-Tree [NND02], have been proposed to optimise the above twig join operations. However, the performance of all the above mechanisms suffers from the time-consuming join operations.

Wang et al. proposed a novel ViST mechanism [WPFY03], which transforms both XML documents and XPath queries into structure-encoded sequences so that the twig pattern matching problem is converted to subsequence matching problem. The advantage of this approach is that it does not need to break down a twig pattern into root-to-leaf paths and process them individually, thus avoiding the heavy join operations to join intermediate results. This method improves all the previous searching mechanisms significantly. However, ViST has several major shortcomings. Firstly, its structure-encoded sequence model does not uniquely transform an XML document into an strucuture-encoded sequence and can cause the semantic false problem. That is, an XML fragment which semantically matches a query may not be returned. ViST may also return false answers (false alarms) because its encoding method can not fully maintain the structures of XML data trees. Time-consuming refinement phase or post-processing phase has to be called to eliminate the false answers. Although Wang et al. [WM05] further proposed a way to eliminate the post-processing phase with $O(n^2)$ total size complexity (where n is the total node number in a data tree), it depends on specialised index structure to find sibling-cover in the trie and remove the false answers, in which the semantic false problem still exists. Finally, in the worst case, the total size of structure-encoded sequence is $O(n^2)$ when a document is a unary tree.

To overcome the above three problems, in this chapter, another encoding mechanism is introduced to transform XML documents and XML queries into geometric sequences. The objective is to ensure correct (i.e. without semantic false) and fast (i.e. without the post-processing phase) evaluation of XPath queries, while at the same time guaranteeing the linear size complexity of the sequence. This approach enables us to achieve better storage and query performance than ViST and the conventional mechanism.

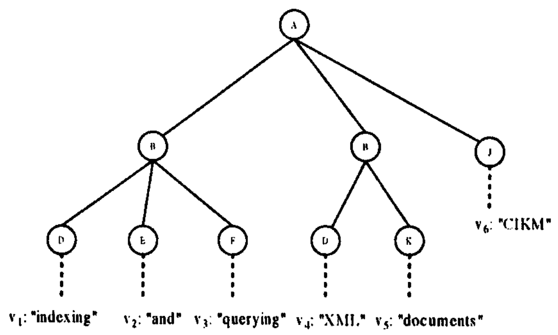


Figure 4.1: An Example of XML Document in Tree Structure

4.2 The Problems with ViST

As proposed in [WPFY03], a structure-encoded sequence is derived from a prefix traversal of an XML document, in format of a sequence of (*symbol, prefix*) pairs, (a_1, p_1), (a_2, p_2), ..., (a_n, p_n), where a_i represents a node in the XML document tree ($a_1 a_2 ... a_n$ is the pre-order sequence) and p_i is the encoded path from root to a_i. In the same spirit, XML queries are converted into structure-encoded query sequences in which "*" and "//" are explicitly encoded. Querying XML is equivalent to finding non-continuous subsequence matches in ViST. The corresponding structure-encoded sequence of the XML document example in Figure 4.1 is illustrated in Table 4.1. Let T_{Str} denote the structure encode sequence.

T_{Str} = $\underline{\mathbf{(A,\ \varepsilon)}}$ $\underline{\mathbf{(B,\ A)}}$ $\mathbf{(D,\ AB)}$ $\mathbf{(v_1,\ ABD)}$ $\underline{\mathbf{(E,\ AB)}}$ $\mathbf{(v_2,\ ABE)}$ $\mathbf{(F,\ AB)}$ $\mathbf{(v_3,\ ABF)}$ $\mathbf{(B,\ A)}$ $\mathbf{(D,\ AB)}$ $\mathbf{(v_4,\ ABD)}$ $\underline{\mathbf{(K,\ AB)}}$ $\mathbf{(v_5,\ ABK)}$ $\mathbf{(J,\ A)}$ $\mathbf{(v_6,\ AJ)}$

Table 4.1: Structure-Encoded Sequence of the XML Document in ViST Approach

The problem of false answers (a.k.a false alarms) arises immediately in ViST in which an XML document is represented by a structure-encoded sequence. For example, given a query Q2: $/A/B[./E][./K]$, its tree structure is shown in

Figure 4.3(b), and its corresponding structure-encoded query sequence is shown in Table 4.2. The underlined non-continuous subsequence in T_{Str} marks a result (matching). However, it is a false answer since the structure expressed in Q2 does not exist in the XML document example. This kind of queries is defined as *non-existence false* in this chapter.

$$Q2_{Str} = \textbf{(A, } \varepsilon\textbf{) (B, A) (E, AB) (K, AB)}$$

Table 4.2: Structure-Encoded Sequence of Q2

$$Q3_{Str} = \textbf{(A, } \varepsilon\textbf{) (B, A) (E, AB) (B, A) (K, AB)}$$

Table 4.3: Structure-Encoded Sequence of Q3

Consider, for another example, Q3 shown in Figure 4.3(c), its structure-encoded sequence is shown in Table 4.3. In ViST, Q2 and Q3 may return the same results because Q2 is a subsequence of Q3. This kind of query pairs is defined as *non-equivalence false* in this chapter. It implies that refinement phase or post-processing phase has to be called to eliminate the false answers in these two cases. However, the process may not be always trivial.

Moreover, ViST has a serious semantic flaw in transforming XPath queries into structure-encoded sequences. Suppose an XML fragment is presented:

$$<A><B><K><C></C></K></B></A>$$

and its corresponding structure-encoded sequence:

$$Frag_{Str} =< A, \varepsilon >< B, A >< K, AB >< C, ABK >$$

If Q: /A[./B//C][//K] is transformed into a structure-encoded query sequence and evaluated against this fragment:

$$Q_{Str} =< A, \varepsilon >< B, A >< C, AB// >< K, A// >$$

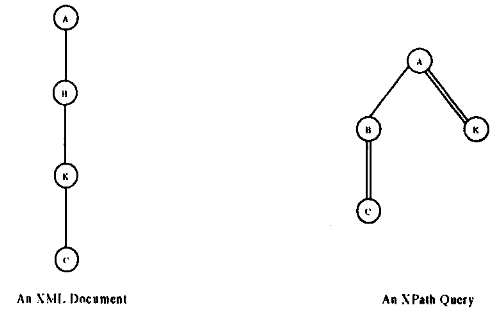


Figure 4.2: A Semantic False Query Evaluation in ViST

There is no such subsequence matching of Q_{Str} in $Frag_{str}$ since K appear after C in ViST, as shown in Figure 4.2. However, Q semantically matches the fragment. This flaw can hardly be fixed in ViST because the order among the items in a structure-encoded sequence is indispensable in ViST. This semantic flaw of ViST is defined as *semantic false* in this chapter.

4.3 Proposed Method

To overcome the shortcomings of ViST, in this section, a geometric-encoding mechanism, which transforms XML documents/queries into geometric data/query sequences, is proposed. Further enhancement to the proposed geometric encoding approach is also described.

4.3.1 Mapping XML Documents into Geometric Data Sequences

An XML document is firstly modelled as an ordered, node labelled, rooted tree. More formally, consider a graph $\mathbf{T} = (\mathbf{V_G}, \mathbf{V_T}, \mathbf{v_r}, \mathbf{E_G}, \mathbf{label_{node}}, \mathbf{nid}, \sum_\mathbf{T})$. V_G is the set of element nodes and V_T is the set of text nodes. $\forall v \in V_T$, v has no outgoing edge. v_r is the root of the XML data tree, where there exists a path from v_r to v, $\forall v \in V_G \cup V_T$. Moreover, it implies that v_r has no incoming edge.

ABDv_1DBEv_2EBFv_3FBABDv_4DBKv_5KBAJv_6JA

Table 4.4: A Sequence Representation of the Example XML Document

Each node $v \in V_G \cup V_T$ is labelled through the function $label_{node}$ over the set of terms, $\sum_T$. The label of a node $v \in V_G$ is referred to as the tag name. The label of $v \in V_T$ is referred to as a distinct keyword contained in the corresponding text. Quotation mark is used in future figures to distinguish the label in V_T.

Each edge e, $e \in E_G$, is a parent-child edge, denoting the parent-child relationship. The parent node is denoted as v_{e_p}, and the child node is denoted as v_{e_c}. A path is a sequence of edges starting from the node v_i to the node v_j, denoted as e_i, e_{i+1}, ..., e_j. A node v_i is ancestor of v_j *iff* a path to v_j goes through v_i. The order among the sibling nodes is distinguished. Each node is assigned a unique nid number for indexing and querying purpose. T_{v_i} refers to the subtree induced by node v_i. Figure 4.1 shows an example of the data model. The solid edges represent E_G. The dashed edge denotes a edge e, $v_{e_p} \in V_G$, and $v_{e_c} \in V_T$. The quoted string represents a label of a node $v \in V_T$.

An XML document is then transformed into a sequence by pre-order traversing the above XML data tree, recording a node's parent when backtracking. For the example in Figure 4.1, its sequence representation is shown in Table 4.4.

To clearly represent a sequence, the above sequence is slightly modified to indicate the start (s), intermediate(i), end (e) positions of a specific node which appears multiple times in the sequence. The modified sequence representation is shown in Table 4.5. Let T_{Geo} denote the modified sequence, and **f**: T $\rightarrow$ T_{Geo}. It is easy to prove that **f** is a bijection between T_{Geo} and T. In the rest of the chapter, the modified sequence is defined as *geometric sequence.* Later in Section 4.4 it is proved that those extra $symbol_i$ and $symbol_e$ only require trivial processing in

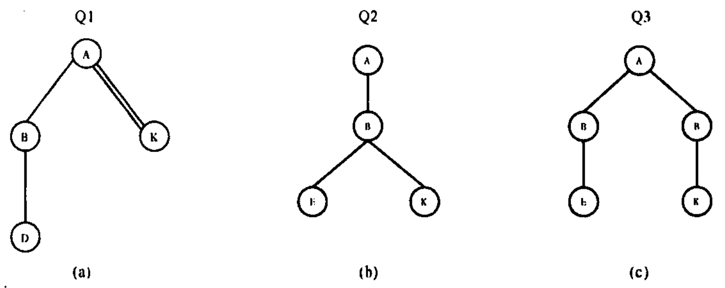


Figure 4.3: Example of Query Sequences in Tree Form

both indexing and querying process.

$T_{Geo}=\underline{\mathbf{A}_s\mathbf{B}_s\mathbf{D}_s}\mathrm{v}_1\underline{\mathbf{D}_e}\mathbf{B}_i\mathbf{E}_s\mathrm{v}_2\mathbf{E}_e\mathbf{B}_i\mathbf{F}_s\mathrm{v}_3\mathbf{F}_e\underline{\mathbf{B}_e\mathbf{A}_i}\mathbf{B}_s\mathbf{D}_s\mathrm{v}_4\mathbf{D}_e\mathbf{B}_i\underline{\mathbf{K}_s}\mathrm{v}_5\underline{\mathbf{K}_e}\mathbf{B}_e\mathbf{A}_i\mathbf{J}_s\mathrm{v}_6\mathbf{J}_e\underline{A_e}$

Table 4.5: A Geometric Sequence Representation of the Example XML Document

4.3.2 Transforming XPath Query into Geometric Query Sequence

A query compatible to XPath is modelled as a tree, as shown in Figure 4.3. The core of evaluating an XPath query at an XML document is finding all the answers of such a twig pattern matching the constraints (axes, nested structure, terms etc.) of the query. Moreover, a query can be complicated when wildcards "*" and self-or-descendent axis("//") are presented. When transforming an XPath query into a geometric query sequence in a similar way of mapping XML documents into geometric sequences, all the information in the XPath query is preserved. We show this by using example queries Q1, Q2, and Q3 in Table 4.6. Their tree structures are shown in Figure 4.3.

Consider the example query Q2: /A/B[./E][./K], its tree structure is shown in Figure 4.3(b). When it is transformed into a geometric query sequence, information needing to be preserved are: (1) A is parent of B, and (2) B is parent of both E and K. Using the proposed method, Q2 is transformed into a geometric query

Path Expression	Geometric Query Sequence
Q1: /A[B/D][//K]	$Q1_{Geo}$: A_s B_s D_s D_e^p B_e^p A_i^u K_s K_e A_e
Q2: /A/B[./E][./K]	$Q2_{Geo}$: A_s B_s E_s E_e^p B_i K_s K_e^p B_e^p A_e
Q3: /A/B[E]/following-sibling::B/K	$Q3_{Geo}$: A_s B_s E_s E_e^p B_e^p A_i B_s K_s K_e^p B_e^p A_e

Table 4.6: List of Q1, Q2, and Q3 in Geometric Query Sequences

sequence: A_s B_s E_s E_e^p B_i K_s K_e^p B_e^p A_e, where p implies that the upcoming item is *parent* of the current item. Any internal node is followed by its $parent_e$ or $parent_i$ in the geometric sequence. However, a E_e may be followed by B_i in real data sequence not B_e. This issue can be easily solve by defining B_i *equals to* B_e when determining the parent relationship. If p is not explicitly stated, the relationship is ancestor-descendant ("//") by default.

Similarly, for query Q1: /A[B/D][//K], its tree structure is shown in Figure 4.3(a). When it is transformed into geometric sequence, information needing to be preserved are: (1) D is a child node of B which, in turn, a child node of A and (2) K is a descendant of A. As stated in previous section, ViST may incur semantic false when transforming Q1 into structure-encoded query sequence since there is no explicit information of the relationship between K and B (D) stated. In this case, "u" is added to a specific node which has at least two child nodes and meanwhile "//" is involved. Q1 is transformed into a geometric sequence: A_s B_s D_s D_e^p B_e^p A_i^u K_s K_e A_e as shown in Table 4.6, where u signifies that semantic uncertainty may occur in the upcoming item.

After an XPath Query is transformed into a geometric query sequence, querying XML documents is equivalent to finding (under the guidance of flag 'p' and/or 'u') non-contiguous subsequence matches in the corresponding geometric data sequences. For query Q1, the underlined non-contiguous subsequence matching in

Table 4.5 marks a correct matching (i.e. the example document satisfies the query).

Consider now how the proposed geometric encoding mechanism avoids the semantic false problem presented by ViST. The geometric data/query sequence of the XML fragment and the query (Figure 4.2)is as follows:

$$Frag_{Geo} = A_s B_s K_s C_s C_e K_e B_e A_e$$

$$Q_{Geo} = A_s B_s C_s C_e^p B_e^p A_i^u K_s K_e A_e$$

To match Q_{Geo} against $Frag_{Geo}$, when evaluating A_i^u, the range information of A_s is resumed. It implies that K_e is searched within the range of A_s instead of B_e in $Frag_{Geo}$, starting with which, K_s, K_e and A_e are founed. Section 4.4 will introduce an elegant stack mechanism to implement the method.

4.3.3 Numbered Geometric Sequence

Furthermore, consider the fact that in an XML document, the same element names may appear several times. Given the data tree in Figure 4.1 and query Q2 in Figure 4.3(b), Q2 should return no result. However, in Table 4.6, $Q2_{Geo}$ does not provide enough information to eliminate the second B_e, which implies that a result would be returned if the second B_e is included.

$T_{Geonum} = \mathbf{A}_{1s}\mathbf{B}_{1s}\mathbf{D}_{1s}\mathbf{v}_1\mathbf{D}_{1e}\mathbf{B}_{1i}\mathbf{E}_{1s}\mathbf{v}_2\mathbf{E}_{1e}\mathbf{B}_{1i}\mathbf{F}_{1s}\mathbf{v}_3\mathbf{F}_{1e}\mathbf{B}_{1e}\mathbf{A}_{1i}\mathbf{B}_{2s}\mathbf{D}_{2s}\mathbf{v}_4\mathbf{D}_{2e}\mathbf{B}_{2i}\mathbf{K}_{1s}$ $\mathbf{v}_5\mathbf{K}_{1e}\mathbf{B}_{2e}\mathbf{A}_{1i}\mathbf{J}_{1s}\mathbf{v}_6\mathbf{J}_{1e}\mathbf{A}_{1e}$

Table 4.7: Numbered Geometric Sequence Representation of XML Document

To tackle this problem, the basic geometric-encoding data sequence is enhanced by numbering each (repeated) item, so that a geometric sequence is sequence of $symbol_{number_{(s|i|e)}}$. Table 4.7 gives a numbered geometric data sequence.

Note that the geometric query sequence is not numbered. There is then no such subsequence matching in $T_{Geo_{num}}$. Additionally, for each query sequence having $symbol_i$, only the first one in T_{Geo} is chosen on the basis of the fact that the rest $symbol_i$ is redundant in querying process. Moreover, for queries having the same child nodes in branches, it is equal to find all the non-decreasing subsequence matching in geometric sequence for all the nodes with the same names. "*" is handled as a range query as the same to ViST. If p is not explicitly stated in geometric sequence model, "//" is then default and not instanced on the basis of the fact that "//" only represents ancestor-descendant relationship. By contrasting to ViST's instance step, resource-consuming prefix checking and range query steps connected with "//" are eliminated in the geometric sequence model.

The correctness of querying XML through numbered geometric data/query sequence matching can be proved based on the following theorems.

Theorem 1. *Let D and $D_{Geo_{num}}$ denote an XML document and a geometric data sequence respectively. D can be uniquely transformed into $D_{Geo_{num}}$, and vice versa, denoted as $D_{Geo_{num}} \Leftrightarrow D$.*

Proof. $D \longrightarrow D_{Geo_{num}}$: Since pre-order traversal sequence is unique in a tree, D can be uniquely transformed into $D_{Geo_{num}}$.

$D_{Geo_{num}} \longrightarrow D$: (1) a text node, $v \in V_T$ in XML data tree is followed by its $parent_e$ in geometric sequence since a node has exactly one text node in an XML data tree; (2) each continuous subsequence between $symbol_{i_s}$ and $symbol_{i_e}$ uniquely denotes a subtree T_{symbol} in XML data tree since i is unique to a certain $symbol$; and (3) any internal node, $v \in V_G$, is followed by its $parent_i$ (respectively $parent_e$) since it have (respectively have no) sibling nodes. $\square$

Theorem 2. *Let Q and Q_{Geo} denote an XML document and a geometric data* *sequence respectively. Q can be uniquely transformed into Q_{Geo}, and vice versa, denoted as $Q_{Geo} \Leftrightarrow Q$.*

Proof. Structural bijection can be inducted from Theorem 1. Semantic bijection can be inducted from Section 3.2. $\square$

Theorem 3. *Let $G(\cdot,\cdot)$ denote the geometric sequence and $\equiv_c$ denote conforming-to-constraints relationship. $\forall v_i, v_j \in Q$, $G_Q(v_i, v_j) \equiv_c G_D(v_i, v_j)$ if $Q \equiv_c D$.*

Proof. $Q_{Geo} \Leftrightarrow$ Q and $D_{Geo_{num}} \Leftrightarrow$ D can be inducted from Theorem 2 and Theorem 3. Therefore, $G_Q(v_i, v_j) \equiv_c G_D(v_i, v_j)$ if $Q \equiv_c D$. $\square$

In the sequence data model of an XML data tree, the length of the sequence of an XML document in worst case is $2n$ - 1, where n denotes the number of nodes in the XML data tree.

Theorem 4. *The length of numbered geometric sequence is O(n).*

Proof. Since each node records its direct parent when backtracking, it means each node $v_i \in V_G$ with fan out FO_i appears $FO_i + 1$ times and each node $v_i \in V_T$ appears once since it has one fan in edge and zero fan out edge. Let the total number of edges be m, the total number of internal nodes be $|V_G|$, the total number of leaf nodes be $|V_T|$. The length of a geometric sequence is no longer than $\sum_{i=1}^{|V_G|}(FO_i + 1) + |V_T|$, where $\sum_{i=1}^{|V_G|}(FO_i + 1)$ is the number of internal nodes appearing in the sequence, $|V_T|$ is the number of leaf nodes appearing in the sequence. It is known that $\sum_{i=1}^{|V_G|} FO_i$ = m = n - 1 in a tree, and $|V_G| + |V_t|$ = n. The length of the sequence is thus no longer than $\sum_{i=1}^{|V_G|} FO_i + |V_G| + |V_T|$ = 2n - 1. $\square$

4.4 Holistic Sequence Matching

To accelerate XPath evaluation, the challenge of the geometric model is to (i) avoid the semantic false problem, (ii) eliminate the false answers without refinement or post-processing phases, and (iii) provide a linear storage complexity mechanism to reduce the size of index. In section 4.3, the total size of numbered geometric sequence is proved to be O(n). In this section, we demonstrate the subsequence matching can find all the correct answers without refinement or post-processing phase which is inevitable in ViST.

4.4.1 Index Structure

A hierarchical indexing structure similar to ViST with some modifications is used. Each item in a geometric data sequence is in form of $(symbol_{number_{(s|i|e)}})$. Items in a geometric sequence are first put into a trie-like structure. Then each node in the trie is assigned two extra elements "preorder" and "size", where "preoder" is the pre-order traversal position of the node in the data tree, and "size" is used for dynamic scope allocation purpose, whose detail study can be found in the next chapter. To build the index structure, each node in the trie, in format of $(symbol_{number_{(s|i|e)}}$, preorder, size), is firstly inserted into a sequence B+Tree index (i.e. SB-Index) using its $symbol_{(s|i|e)}$ as the key. For all the nodes with the same $symbol_{(s|i|e)}$, they are inserted into a position B+Tree (i.e. PB-Index) using its preorder as the key. Figure 4.4 illustrates the index structure used.

4.4.2 Holistic Geometric Sequence Matching Algorithm

To accelerate XPath evaluation steps with SB-Index and PB-Index, an elegant stack mechanism, which includes two different sets of stacks: a number of *se-*

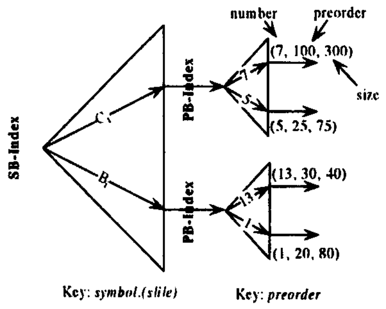


Figure 4.4: Index Structure: SB-Index and PB-Index

quence stacks and a *status stack* is proposed. Sequence stacks maintain a list of candidate items from a geometric data sequence that may contribute to the matching of a geometric query sequence. Status stack traces the identifiers of all the sequence stacks which have accommodated the candidate items found from the data sequence. The top value of the status stack gives the identifier of the current active sequence stack, denoted using *activeId*. For simplicity, $Stack_{id}$ is used to denote the sequence stack of identifier *id*, and *seqnum* to represent the identifier of the last occupied sequence stack so far, where initially $seqnum = 0$.

The matching of query sequence against the geometric data sequence is based on the fact that a subsequence between $symbol_s$ and $symbol_e$ denotes a subtree. Suppose a geometric query sequence $q_{1_{v_1}}^{l_1}\ q_{2_{v_2}}^{l_2}\ \dots q_{m_{v_m}}^{l_m}$ and a geometric data sequence $d_{1_{num_1,V_1}} d_{2_{num_2,V_2}} \dots d_{n_{num_n,V_n}}$, where $(m \leq n)$, $\forall x(1 \leq x \leq m)$ $(v_x = s|i|e) \wedge (l_x = p|u|\ )$, and $\forall y(1 \leq y \leq n)$ $(V_y = s|i|e) \wedge num_y \in N$ are presented. Starting with the empty stacks, the two sequences are scanned across from left to right. When equal symbols are encountered (i.e., $q_x = d_y$ and $v_x = V_y$) in $q_{x_{v_x}}^{l_x}$ and $d_{y_{num_y,V_y}}$, the following situations are considered.

[Case 1] $\underline{(v_x = V_y = s)}$

seqnum is increased by 1 to prepare a new sequence stack for storing can-

```
input: SB-Index: index of symbol names; PB-Index: index of (preorder, size) labels; $\mathbf{Q_{Geo}} = Q_{Geo_1}, ..., Q_{Geo_{len}}$: XML query in geometric sequence format; j: the jth point in $Q_{Geo}$; range: in format of (preorder, size); status: status stack. len: length of XPath query sequence.
output: all the matchings of $Q_{Geo}$ in the XML data
if $j \le len$ then
    if $s$ is in $Q_{Geo_j}$ then
        seqnum ← seqnum+1;
        status.push(seqnum);
    if $u$ is in $Q_{Geo_j}$ then
        resume range of corresponding $symbol_s$, say (n', size');
        $GeoMatching(n', size', j+1, status)$;
    else
        T ← All the matchings of $Q_{Geo_j}$ in SB-Index;
        R ← All the matchings of T in PB-Index satisfying range;
        active = status.top();
        for each $r_k \in R$ do
            if $stack_{active}.isempty()$ or $s$ is in $Q_{Geo_j}$ then
                $stack_{active}.push(r_k)$;
            else
                if $r_k.number = stack_{active}.top().number$ then
                    $stack_{active}.push(r_k)$;
            if $r_k = stack_{active}.top()$ then
                Assume range of $r_k$ is (n', size');
                if $size' \ge len - j$ then
                    $status_{new}$ = status;
                    if $p$ is in $Q_{Geo_j}$ and satisfy parent constraint then
                        $status_{new}$.pop();
                        if $i$ is in $Q_{Geo_{j+1}}$ then
                            $GeoMatching(n', size', j+1, status_{new})$ //pc;
                        else
                            $GeoMatching(n', size', j+1, status_{new})$ //ad;
                    else
                        if $e$ is in $Q_{Geo_j}$ then
                            $status_{new}$.pop();
                        $GeoMatching(n', size', j+1, status_{new})$;
                        if $i$ or $e$ is in $Q_{Geo_j}$ then
                            skip to $r_h$, where $r_h$.n $\ge$ ($r_k.n$ + $r_k.size$)
                    $stack_{active}$.pop();
else
    output a matching of $Q_{Geo}$;
```

Algorithm 4.1: GeoMatching

didate item(s). $d_{y_{num_y},s}$ is pushed into the sequence stack $Stack_{seqnum}$, and then push *seqnum* into the status stack, indicating that the corresponding sequence stack accommodates a candidate matched item. It is the current active sequence stack.

[Case 2] $\underline{(v_x = V_y = e) \text{ or } (v_x = V_y = i)}$

We check whether $d_{y_{num_y},e}$ has the same num_y as the candidate items in the sequence stack $Stack_{\text{activeId}}$. Note that *activeId* is the top value from the status stack.

If they are matched (i.e., the same), $d_{y_{num_y},V_y}$ is pushed into $Stack_{\text{activeId}}$, and pop the top item from the status stack when $(v_x = V_y = e)$, implying that a subtree whose starting/ending nodes specified in this active $Stack_{\text{activeId}}$ is found.

When coming across $q^u_{x_i}$ in the query sequence, the search pointer in the data sequence is shifted backward to $d_{y_{num_y},s}$ to avoid the semantic false problem (as specified in Section 4.3).

If there is a mismatch between the two numbers, the recursive backtracking procedure is triggered. The idea is to go back to the subtree-starting node (i.e., the node with a subscript s flag) in the current active stack, and re-start the search of the data sequence for a similar symbol with subscript s but with a larger *num* than before. To do this, all the candidate items detected before will be popped out of the corresponding sequence stacks, signifying an unsuccessful match. Meanwhile, the status stack and *seqnum* will be resumed to the states when to re-start the matching of the subtree-starting node in the data sequence.

For ease of explanation, the evaluation of query Q_2 over the data tree in

Figure 4.1 is exemplified . Detailed algorithmic description can be found in the Algorithm 4.1[1]. Figure 4.5 gives a snapshot of the stacks used during the evaluation.

Firstly, A_{1s}, B_{1s}, and E_{1s} from the geometric data sequence are pushed into sequence stack $Stack_1$, $Stack_2$, and $Stack_3$ (Step 1, 2, and 3, as shown in Figure 4.5(a)). The status stack contains (1, 2, 3) after E_{1s} is pushed into $Stack_3$. When meeting E^p_e, E_{1e} is pushed into $Stack_3$ since *activeId*=3, and then pop 3 out of the status stack as a subtree between E_{1s} and E_{1e} is found in the data tree (Step 4). Since p in E^p_e implies that the next matching item B_i should be its parent, according to Table 4.7 B_{1i} in the data sequence satisfies this constraint, and thus push it into $Stack_2$ (according to the top value in the status stack which is 2 = *activeId*). Similarly K_{1s} and K_{1e} are pushed into $Stack_4$ (according to *seqnum*=4) to match K_s and K^p_e in the query (Step 6 and 7). Now the status stack contains (1 2). To look further for the parent of K^p_e, i.e., B_e, B_{2e} is the only possibility in the data sequence. However, its number 2 does not conform to the number 1 in $Stack_2$ (Step 8, as shown in Figure 4.5(b)). Hence, a mismatch happens, and backtracks the matching process to B_{1s} in $Stack_2$. Note that B_{2i} is not pushed into $Stack_2$ due to the same reason, as shown in Figure 4.5(c). After B_{1s} is popped out of $Stack_2$, the next and only item that satisfies B_s in the query is B_{2s} (Step 10, as shown in Figure 4.5(d)). However, there does not exist any E_s in the data sequence after B_{2s}. Therefore, the matching process returns no result for Q_2.

The stack mechanism answers the previous question why additional $symbol_i$ and $symbol_e$ (Case 2) do not require extra computing efforts during the query process [2].

[1] *pc* denotes parent-child relationship and *ad* denotes ancestor-descendant relationship.

[2] Parent-child relationship is determined without level information since each $symbol_e$ is

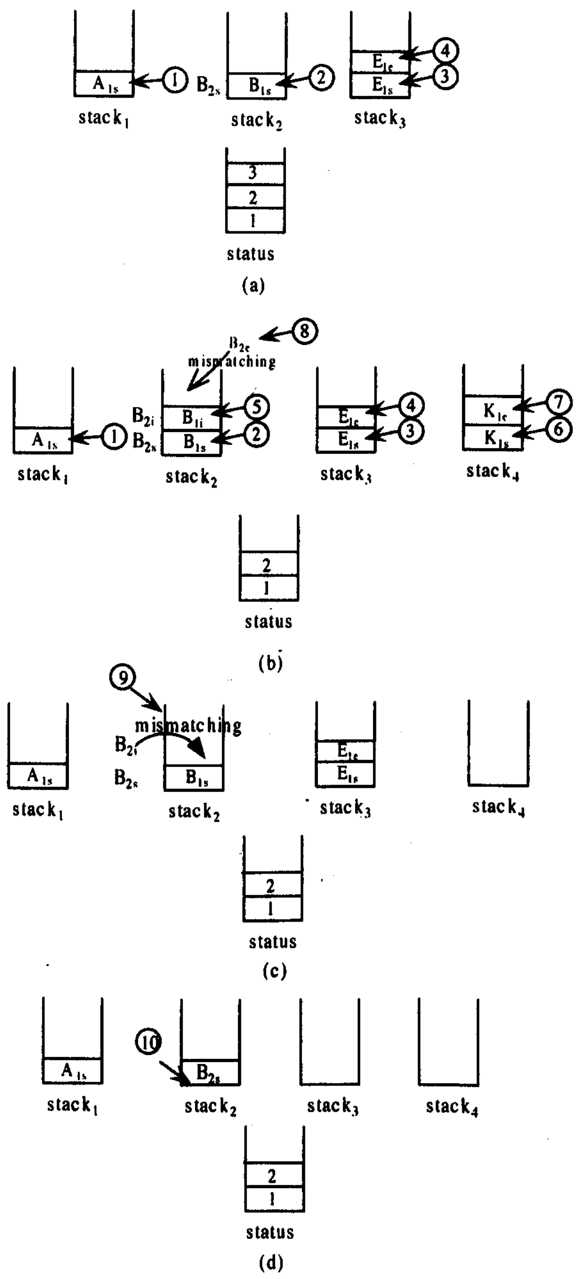


Figure 4.5: An Example of Stack Status Avoiding Non-existence Query

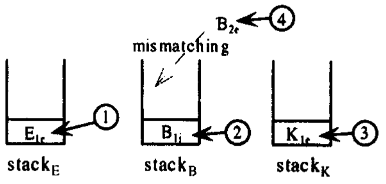


Figure 4.6: Stack Status Avoiding Non-existence Query in OptGeoMatching

Path Expression	Geometric Sequence
Q1: /A[B/D][//K]	$Q1_{OptGeo}$: A_s $D_e{}^p$ $B_e{}^p$ $A_i{}^u$ K_e A_e
Q2: /A/B[./E][./K]	$Q2_{OptGeo}$: $E_e{}^p$ B_i $K_e{}^p$ $B_e{}^p$ A_e
Q3: /A/B[E]/following-sibling::B/K	$Q3_{OptGeo}$: $E_e{}^p$ $B_e{}^p$ A_i B_s $K_e{}^p$ $B_e{}^p$ A_e

Table 4.8: List of Q1, Q2, and Q3 in Optimised Geometric Query Sequences

4.4.3 Optimisation

Observing that the performance of evaluating XPath queries over XML documents is significantly affected by the lengths of geometric query sequences, the subsequence matching algorithm is further improved on the basis of optimised geometric query sequence transformation. The rationale behind is that instead of keeping pairs of nodes like B_s and B_e in a query sequence, one of them can be removed without loss of semantics while performing subsequence matching.

Here, a geometric query sequence transformation rule with an aim to minimise the length of the query sequence is proposed. That is: *removing all the $symbol_s$ unless it connects with a $symbol_i^u$.* [3] Examples of the optimised XPath query sequences after transformation are listed in Table 4.8.

Interestingly, the transformed query subsequences enable us to perform query evaluation in a bottom-up manner, since the subsequence matching starts from

followed by its $parent_e$ or $parent_i$ in the geometric sequence model.

[3]Recall in Section 4, $symbol_s^u$ signifies that the range information of $symbol_s$ is *resumed* so as to cope with the semantic false problem.

a $symbol_e$. For example, given the query $Q_2 : /A/B/[./E][./K]$ in Figure 4.3 and its optimised geometric query sequence: $E_e^p\ B_i\ K_e^p\ B_e^p\ A_e$, the evaluation process starts from E_e instead of A_s. In comparison, the matching algorithm described in the previous subsection exhibits the top-down flavour.

To facilitate the optimised geometric subsequence matching, the stack mechanism is improved accordingly, where only one set of stacks called **symbol** stacks are involved. $Stack_{symbol}$ is used to denote the stack which accommodates items having *symbol*.

Given an optimised geometric query sequence $q_{1_{v_1}}^{l_1}\ q_{2_{v_2}}^{l_2}\ \dots\ q_{m_{v_m}}^{l_m}$ and a geometric data sequence $d_{1_{num_1,V_1}} d_{2_{num_2,V_2}} \dots d_{n_{num_n,V_n}}$, where $(m \leq n)$, $\forall x(1 \leq x \leq m)\ (v_x = s|i|e)\ \wedge\ (l_x = u|p|\ )$, and $\forall y(1 \leq y \leq n)\ (V_y = s|i|e)$. Starting with the empty stacks, the two sequences are scanned across from left to right. When two equal symbols encounter (i.e., $q_x = d_y$ and $v_x = V_y$) in $q_{x_{v_x}}^{l_x}$ and $d_{y_{num_y,V_y}}$, the following situations are considered.

[**Case 1**] $\underline{(v_x = V_y = s)}$

$d_{y_{num_y,s}}$ is pushed into the symbol stack $Stack_{d_y}$.

[**Case 2**] $\underline{(v_x = V_y = e)}$

There exist two possibilities. 1) When the top item of $Stack_{d_y}$ has a subscript *intermediate* flag i, we check whether $d_{y_{num_y,i}}$ has the same num_y as this top item. If they are the same, $d_{y_{num_y,i}}$ is pushed into $Stack_{d_y}$; otherwise a mismatch happens and backtracks the process. That is, all those candidate items, which lie between $d_{y_{num_y,e}}$ and the top item in $Stack_{d_y}$, are popped out of the corresponding symbol stacks including this top item, and the process continues to re-search these candidate items in the data sequence.

2) When the top item of $Stack_{d_y}$ has a subscript *end* flag e, we check whether $d_{y_{num_y},i}$ has the same num_y as this top item. If they are not the same, $d_{y_{num_y},i}$ is pushed into $Stack_{d_y}$; otherwise a mismatch happens and triggers the above backtracking process.

[Case 3] $\underline{(v_x = V_y = i)}$

We check whether $d_{y_{num_y},i}$ has the same num_y as the top item in $Stack_{d_y}$. If they are the same, it is pushed into $Stack_{d_y}$; otherwise, a mismatch happens, and triggers backtracking process.

Note that when encountering $q^u_{x_i}$ in the query sequence, the search pointer in the data sequence is shifting backward to $d_{y_{num_y},s}$ to avoid the semantic false problem (as specified in Section 4.3).

To illustrate the optimised geometric subsequence matching procedure, let's take query Q_2 as the example. A snapshot of the symbol stacks is given in Figure 4.6. Detailed algorithmic description can be found in the Algorithm 4.2. Firstly, E_{1e} is pushed into $Stack_E$ (Step 1). Since p is in E^p_e in the query sequence, the only item in Table 4.7 that satisfies the parent-child constraint is B_{1i}, and is thus pushed into $Stack_B$ (Step 2). K_{1e} is further pushed into $Stack_K$ (Step 3). As p is in K^p_e, B_{2e} is the only possible parent item. However, its number 2 does not conform to the number 1 of the top item B_{1i} in $Stack_B$ (Step 4). Thus B_{2e} cannot be pushed into $Stack_B$, and a mismatch happens. The algorithm backtracks to B_i and re-starts the search of the data sequence from B_{2i}, returning no satisfactory query answer in the end.

```
input: SB-Index: index of symbol names; PB-Index: index of
(preorder, size) labels; $Q_{Geo} = Q_{Geo_1}, \ldots, Q_{Geo_{len}}$: XML query in
geometric sequence format; j: the jth point in $Q_{Geo}$; range: in format of
(preorder, size); len: length of XPath query sequence.
output: all the matchings of $Q_{Geo}$ in the XML data
if $j \leq len$ then
    if $u$ is in $Q_{Geo_j}$ then
        resume range of corresponding $symbol_s$, say (n', size');
        $OptGeoMatching(n', size', j + 1)$;
    else
        T ← All the matchings of $Q_{Geo_j}$ in SB-Index;
        R ← All the matchings of T in PB-Index satisfying range;
        for each $r_k \in R$ do
            if $stack_{symbol}.isempty()$ or $s$ is in $Q_{Geo_j}$ then
                $stack_{symbol}.push(r_k)$;
            else
                if $r_k.number = stack_{symbol}.top().number$ and $i$ is in $Q_{Geo_j}$
                then
                    $stack_{symbol}.push(r_k)$;
                if $r_k.number = stack_{symbol}.top().number$ and $e$ is in $Q_{Geo_j}$
                and $i$ is in $stack_{symbol}.top()$ then
                    $stack_{symbol}.push(r_k)$;
                if $r_k.number \mathrel{!}= stack_{symbol}.top().number$ and $e$ is in $Q_{Geo_j}$
                and $e$ is in $stack_{symbol}.top()$ then
                    $stack_{symbol}.push(r_k)$;
            if $r_k = stack_{symbol}.top()$ then
                Assume range of $r_k$ is (n', size');
                if $size' \geq len - j$ then
                    if $p$ is in $Q_{Geo_j}$ and parent constraint is satisfied then
                        if $i$ is in $Q_{Geo_{j+1}}$ then
                            $OptGeoMatching(n', size', j + 1)$ //pc;
                        else
                            $OptGeoMatching(n', size', j + 1)$ //ad;
                    else
                        $OptGeoMatching(n', size', j + 1)$;
                        if $i$ or $e$ is in $Q_{Geo_j}$ then
                            skip to $r_h$, where $r_h.n \geq (r_k.n + r_k.size)$
                    $stack_{symbol}$.pop();
else
    output a matching of $Q_{Geo}$;
```

Algorithm 4.2: OptGeoMatching

T: title; A: article; AU: author; I: inproceedings; N: namerica; P: payment;

PE: personref; PER: person; O: open_auction; C: closed_auctions;

CA: closed_auction; B: bidder; BU: buyer;

XPath Queries	Data Sets
Q1: //T[text()="On views and XML"]	DBLP
Q2: //A[./AU[text()="Dan Suciu"]][./AU[text()="Tan"]]	DBLP
Q3: /*//I/AU[text()="Peter Buneman"]/following-sibling::AU	DBLP
Q4: //N/*/P[text()="Cash"]	XMARK
Q5: //*/O[./B/PE[@PER="person0"]][./B/PE[@PER="person23"]]	XMARK
Q6: //C/CA/BU[@PER="person11"]/following-sibling::BU	XMARK

Table 4.9: List of XPath Queries

4.5 Experimental Results

The proposed sequence matching mechanism, *OptGeoMatching*, is implemented in C++. ViST, and a classical indexing and querying mechanism, XISS [LM01], is also implemented for comparison purpose. XISS breaks down the queries into binary twigs and "stitches" them together to obtain the final results. ViST treats both XML documents and XML queries as sequences and obtains the final results by using subsequence matching phase to get preliminary results and post-processing phase to eliminate false answers. The strings are encoded as they are in ViST and use substring matching algorithm to detect the prefix matching.

The B+Tree library in Berkeley DB provided by Sleepycat software was used. All the experiments are carried out on a Pentium III 750MHZ machine with 512MB main memory. Disk pages of 8k is used for Berkeley B+Tree index. To evaluate both the efficiency and scalability of the proposed method, the experiments are performed on both real-world datasets and synthetic datasets.

4.5.1 Experiments on Real-World Datasets

Data Sets

For the experiments, public XML databases DBLP [Ley] and the XML benchmark XMARK [BCF+] are used.

- DBLP is popularly used in benchmarking XML indexing methods. In the version used in this study, it has 3,332,130 elements and 404,276 attributes, totally 130,726KB data. The maximum depth of DBLP is 6. The average length of geometric sequence is 39.
- XMARK is widely used in benchmarking XML indexing mechanism with complex nesting structure. In the version used in this study, it has 1,666,315 elements and 381,878 attributes, totally 115,775KB. The maximum depth of XMARK is 12.

Performance of Query Processing

6 queries are evaluated on the DBLP and XMARK, and their results are compared with ViST and XISS. Table 4.9 lists 6 different queries for DBLP and XMARK, respectively. The experimental results of using the proposed method, ViST and XISS are shown in Table 4.10.

Q1 is a simple query, "find all the titles with 'On views and XML'". Tur geometric sequence model performs slightly better that ViST because there is no instantiation step in geometric sequence model which is inevitable in ViST.

Q2 and Q3 are relatively complex queries, respectively, "find all the articles written by 'Dan Suciu' and 'Tan'" and "find the authors co-writting inproceeding papers with 'Peter Buneman'". This time, the geometric sequence model outperforms ViST because (1) there is no need to perform substring matching

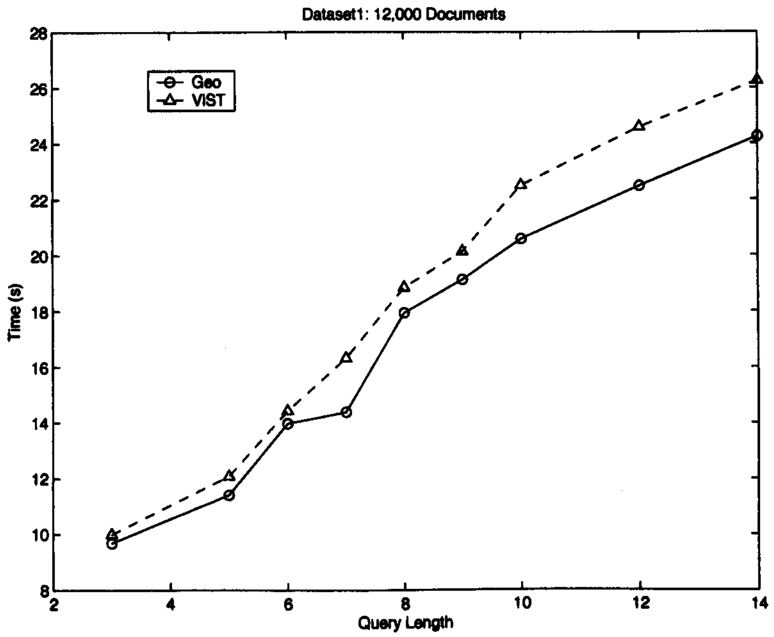


Figure 4.7: Queries over Synthetic Data: *Dataset*$_1$

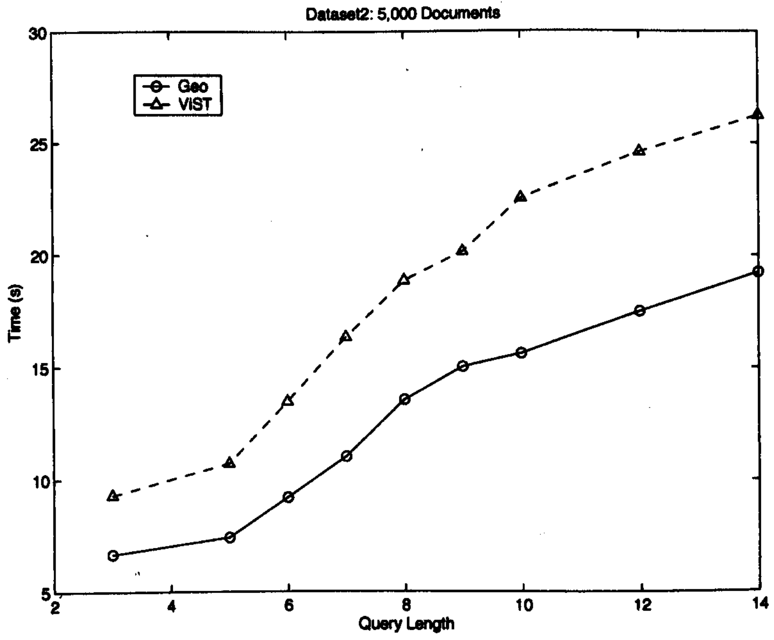


Figure 4.8: Queries over Synthetic Data: *Dataset*$_2$

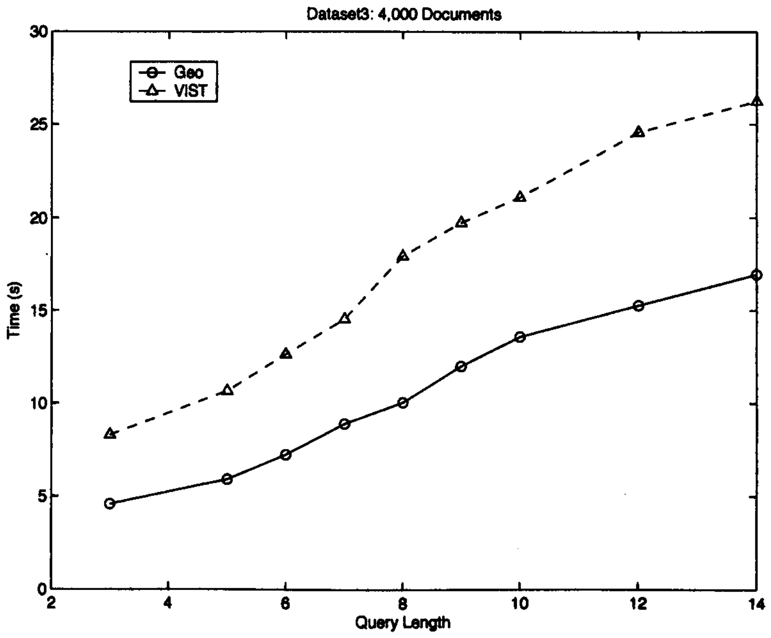


Figure 4.9: Queries over Synthetic Data: *Dataset*$_3$

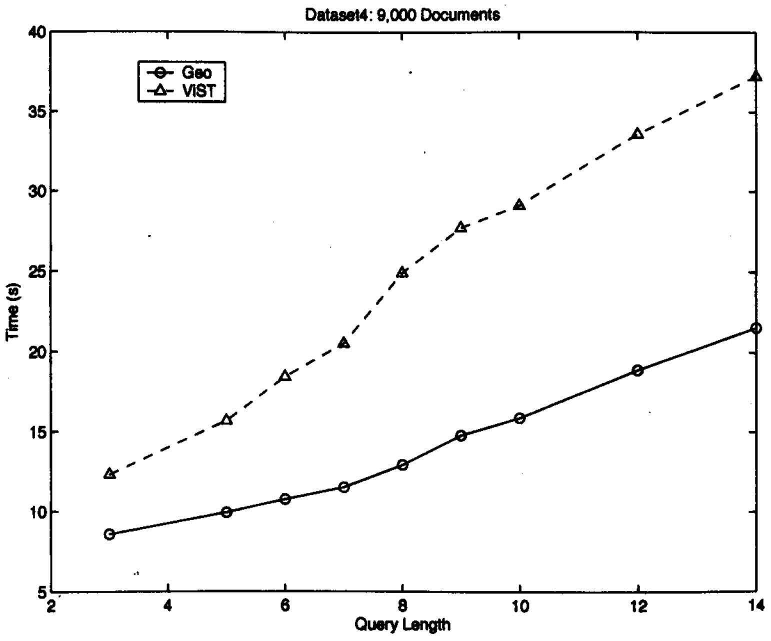


Figure 4.10: Queries over Synthetic Data: *Dataset*$_4$

in validating and instancing structure-encoded query sequences. The substring matching increases the disk I/O since enormous data is retrieved from the index; (2) there exists no post-processing phase in the proposed method; (3) most importantly, OptGeoMatching performs bottom-up query evaluation strategy. Since the number of the nodes with specific authors' names are comparatively small and their ranges are narrow, it can therefore achieve significant increase in XPath evaluation performance.

Q6 is a query which should return no result since there exists only one buyer in one closed auction. The structure expressed by Q6 is a kind of *false alarm*. Again, without exception, the geometric sequence model is significantly faster than ViST because there is no answer during the subsequence matching in the proposed method. It is safe to conclude that there is no such structure existing in XMARK file, while time-consuming refinement phase has to be called by ViST to eliminate enormous false answers.

4.5.2 Experiments on Synthetic Data

Datasets

To evaluate the extensibility of the proposed method, several synthetic datasets are generated. In the experimental environment, there are totally 30,000 documents with 20 different symbols. The maximum depth of the datasets is 16, and maximum fan-out of a node is set to 4. 8KB disk page for B+Tree index and 8-byte integer for pre-order number are still used. Geometric sequences are directly generated instead of documents.

Performance of Query Processing

The length of queries are set to 3, 5, 6, 7, 8, 9, 10, 12, and 14 respectively. All the queries are non-existence queries. To focus on the impact of refinement

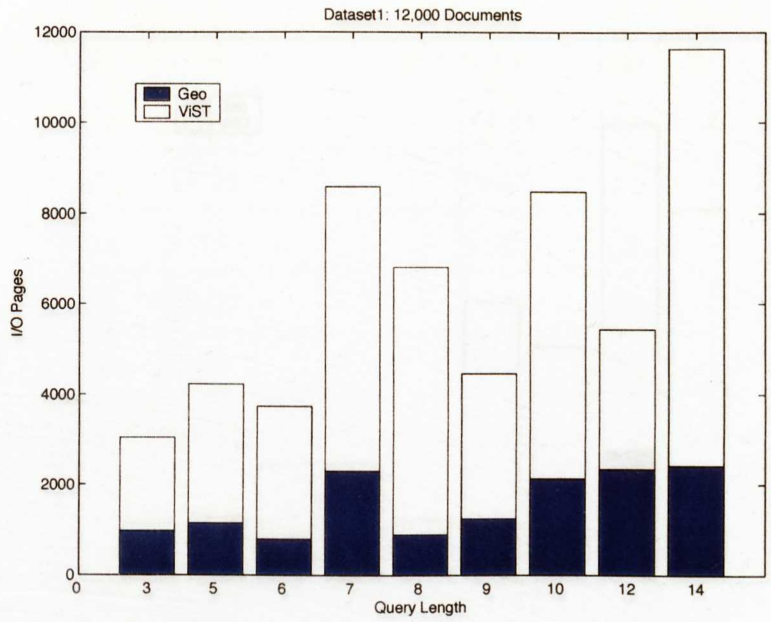


Figure 4.11: I/O Performance: Geo vs. ViST: $Dataset_1$

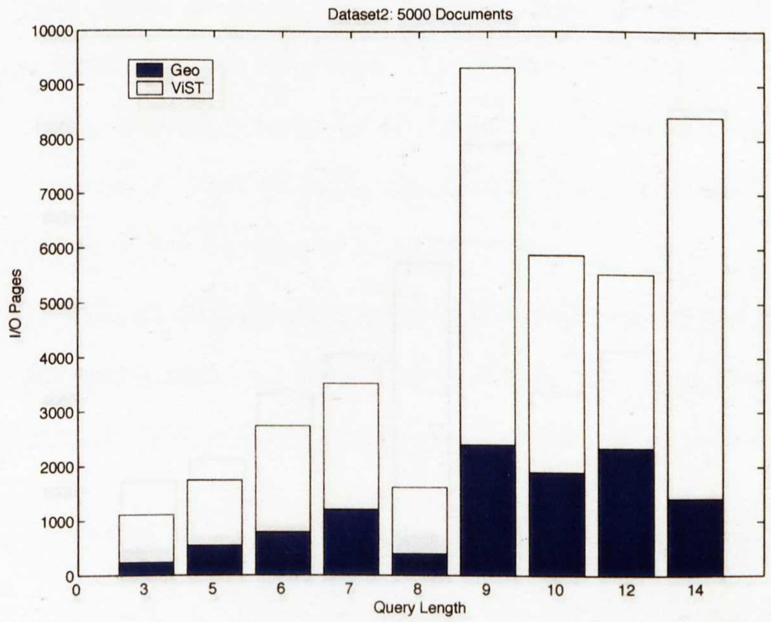


Figure 4.12: I/O Performance: Geo vs. ViST: $Dataset_2$

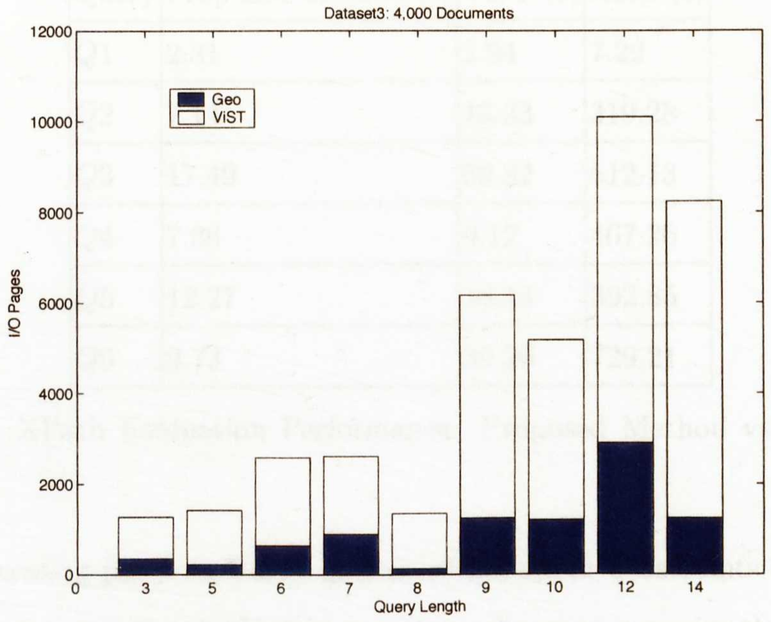


Figure 4.13: I/O Performance: Geo vs. ViST: $Dataset_3$

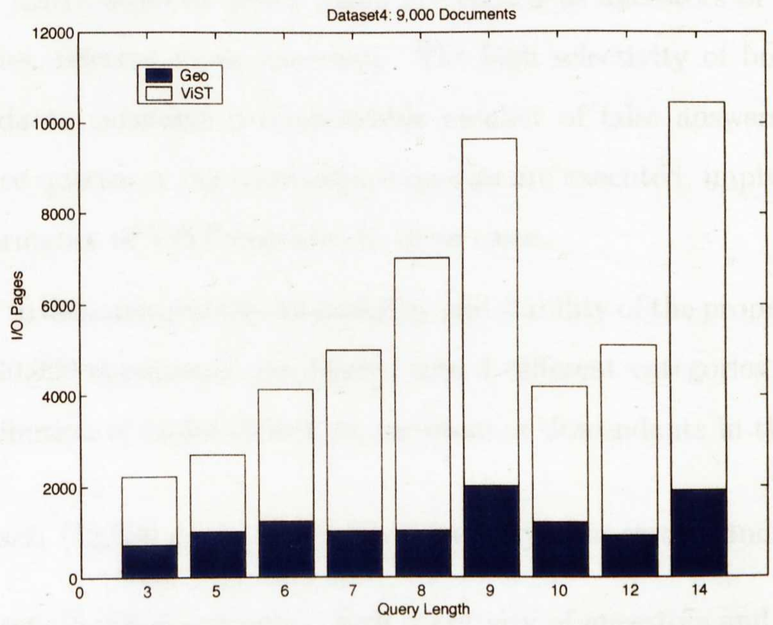


Figure 4.14: I/O Performance: Geo vs. ViST: $Dataset_4$

Query	Proposed Method (s)	ViST (s)	XISS (s)
$Q1$	2.81	2.94	7.22
$Q2$	7.14	13.33	319.28
$Q3$	17.49	69.82	612.13
$Q4$	7.86	9.12	467.26
$Q5$	12.27	18.13	392.85
$Q6$	9.73	39.20	729.21

Table 4.10: XPath Evaluation Performance: Proposed Method vs. ViST and XISS

or post-processing phase in ViST, queries with content constraints are not used since the bottom-up OptGeoMatching is naturally more superior than top-down ViST. Queries related to semantic false are also not used since ViST can not handle these queries at all. In the scalability test, the performance of ViST depends on distribution of nodes which are chosen as ancestors or descendants in the queries, referred to as *selectivity*. The high selectivity of both ancestors and descendants generates a considerable number of false answers in ViST if non-existence queries or non-equivalence queries are executed, implying that the query performance of ViST degrades in these cases.

In order to demonstrate the extensibility and stability of the proposed method, the above 30,000 documents are divided into 4 different categories on the basis of the distribution of nodes chosen as ancestors or descendants in the queries.

- $Dataset_1$ (12,000 documents): low selectivity of ancestors and descendants
- $Dataset_2$ (5,000 documents): high selectivity of ancestors and low selectivity of descendants

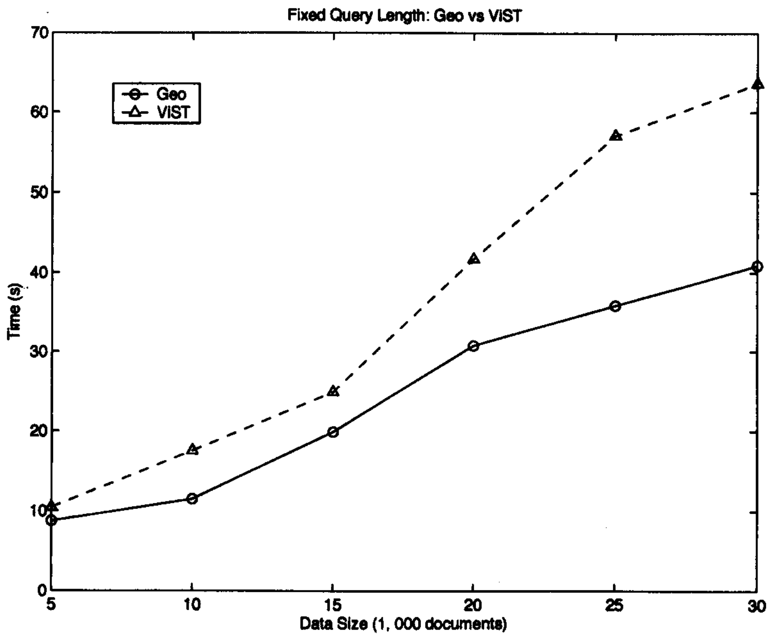


Figure 4.15: Fixed Query Length Performance: Geo vs. ViST

- *Dataset*$_3$ (4,000 documents): low selectivity of ancestors and high selectivity of descendants
- *Dataset*$_4$ (9,000 documents): high selectivity of ancestors and descendants

The results are shown from Figure 4.7 and Figure 4.14. The proposed method performs slightly better than ViST in *Dataset*$_1$ because the post-processing phase is trivial in dataset1. However, for the rest of the three datasets, the proposed method performs significantly better than ViST since refinement phase requires enormous efforts to eliminate the false answer. Contrasting to ViST, the proposed method performs stably in these three datasets. Even content constraint is not involved in the synthetic data experiments, OptGeoMatching demonstrates significant disk I/O performance improvement comparing with ViST since top-down ViST is uncertain of its descendants and has to search its full *range* for correct answers. In contrasting to top-down ViST, OptGeoMatching performs a bottom-up subsequence matching and only needs to search a more specific *range* where an ancestor node may exist. Finally, to demonstrate the scalability of both geometric model and ViST, dataset1, ..., dataset4 are mixed and the query length is fixed to 7. The result is shown in Figure 4.15.

4.6 Summary

In this chapter, an efficient mechanism for accelerating XPath evaluation steps based on the proposed geometric sequence is reported. A top-down holistic subsequence matching algorithm and a bottom-up holistic subsequence matching algorithm are proposed on the basis of a novel geometric sequence model for XML documents. According to the experiments, the proposed mechanism can significantly improve the current best approach ViST, finding all the correct answers without refinement or post-processing phase with linear size complexity of geometric sequence and guaranteeing the completeness of XPath evaluation without semantic false.

CHAPTER 5

Self-Adaptive Labelling Scheme

The art of progress is to preserve order amid change and to preserve change amid order.

– Alfred North Whitehead

To fully evolve XML into a universal data representation and exchange format, the capability of modifying XML documents is indispensable. This chapter proposes a novel self-adaptive scope allocation scheme [SFSW04] for labelling dynamic XML documents. It is general, light-weight and can be built upon existing data retrieval mechanisms. Bayesian inference is used to compute the actual scope allocated for labelling a certain node based on both the prior information and the actual document. Through extensive experiments, the proposed Bayesian allocation model can practically and significantly improve the performance of the conventional fixed scope allocation models.

5.1 Introduction

It is increasingly expected that XML [ABS00] will become the de facto standard of the Web, ultimately replacing HTML. An XML document consists of data enclosed by user defined tags, and its nested tree structure is described by DTD. Figure 5.1 shows an example of an XML document and Figure 5.2 illustrates the corresponding DTD.

To allow efficient querying of XML data, each node in the XML data tree is typically given a unique label, such that given the labels of two nodes, whether one node is an ancestor of the other can be determined. Till now, many indexing structures have been proposed to process structural queries efficiently based on certain coding schemes [LM01, NND02, WPFY03]. [LM01] "stitches" the binary twigs to obtain final results based on range labelling scheme. [NND02] improves [LM01] by using a stack-based algorithm to efficiently join root-to-leaf paths. [WPFY03] treats both document and query as sequences, and matches the query as a whole unit when querying.

The capability of modifying XML documents arises another important aspect to XML indexing: how to support dynamic data insertion, deletion, and update with corresponding index structure. Cohen et al. [CKM02] firstly proposed a dynamic labelling scheme for XML documents to support updating operations. The children of a node v have the label concatenated with the string s attached to their incoming edge. Given s(1)=0, to obtain s(i+1), the binary number represented by s(i) is increased 1 and if the representation of s(i)+1 consists of all ones, its length is doubled. The problem with this approach is that the size of labelling will grow fast with the increase of degree of the nodes because of its dependency on the fan-out of a node.

State-of-the-art research efforts [LM01, WPFY03, WJLY03] have been proposed to cope with the problem of allocating scope for dynamic XML documents. [LM01, WJLY03] considered to give some extra scope while allocating the labels. But as for how much to allocate, they did not address it. [WPFY03] considered to allocate scope for each node on the basis of the probability of its occurrences from statistical information. The deficiency is that the probability of allocating

```
<prices>
 <book>
  <title>Algorithms</title>
  <information>
   <source>bstore2.example.com</source>
   <price>31.99</price>
  </information>
 </book>
 <book>
  <title>Data on the Web</title>
  <information>
   <source>bstore2.example.com</source>
   <price>34.95</price>
  </information>
  <author>Serge Abiteboul</author>
  <author>Peter Buneman</author>
  <author>Dan Suciu</author>
 </book>
 <book>
  <title>TCP/IP Illustrated</title>
  <information>
   <source>bstore2.example.com</source>
   <price>65.95</price>
  </information>
  <author>W. Richard Stevens</author>
 </book>
</prices>
```

Figure 5.1: An Example of an XML Document

```
<!ELEMENT prices (book*)>
<!ELEMENT book (title?, information+, author*)>
<!ELEMENT information (source, price)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Figure 5.2: An Example of DTD

space for a certain node is fixed and is considered as a constant. To summarise, these methods are not self-adaptive to real document environment where the probability of a node's occurrence varies in difference XML documents.

In this chapter, the conventional scope allocation scheme is significantly improved by using Bayesian inference technique. Combining the prior information (i.e DTD and statistical information) and the actual documents, better performance in scope allocation for dynamic XML documents is achieved. Bayesian Allocation Model (BAM), a general, self-adaptive scope allocation model is proposed to address the above important challenges for dynamic XML data. It is general, light-weight and can be adapted to existing data retrieval mechanisms [CTZZ02, NND02, KYU01, ZND+01, Gru02, WPFY03, LM01]. The scope allocated for each node depends on the probability it will be updated. Thus a better performance than the traditional allocation methods when updating can be guaranteed.

5.2 Bayesian Allocation Model

Notations: Let u be a node in a DTD, which has t child nodes. Let $nodetype_i$ denote the type of the ith child node ($1 \leq i \leq t$), which occurs x_i times under the node u in corresponding XML document. $\sum_{i=1}^{t} x_i$ = z, where z equals to the total number of children under node u. Assume all sibling nodes of different *nodetype*$_i$ occur independently with probability θ_i. Let $\bar{\theta}_i$ denote estimators of θ_i, which can be obtained from semantic of the XML document or the statistics of a sample dataset. Let n denote the *range* scope allocated for the z nodes under node u. In the chapter, n = c × z, where c denotes the *range enlarging* factor.

5.2.1 Scope Allocation Overview

The scope allocation scheme works as follows; 1) parse DTD to obtain prior information for each name type; then during the breadth first traversal of an XML document, embed the tree into a complete K-ary tree, 2) root node of an XML document is put in level 0 and labelled 1 in the complete K-ary tree; 3) for each non-leaf node, calculate the number of children, denoted as z, and the types of its children, denoted as t; 4) allocate scope for each child node using Bayesian allocation model in 1*th* level below its parent, satisfying $l \geq \lceil \log_k(c \times z) \rceil$, c denotes a range enlarging factor; 5) repeat 3) and 4) till breadth-first traversal finishes.

5.2.2 Bayesian Allocation Model

Self-adaptive Bayesian allocation model is proposed to allocate scope for dynamic XML documents on the basis of K-ary tree and Breadth-First Traversal. Pre-allocating scope for dynamic XML data is a natural solution. The core of Bayesian allocation model is on efficiently allocating scope for each node in actual dynamic XML documents in a self-adaptive manner.

Bayesian allocation model The core of the work is on estimating probability θ_i. In ViST [WPFY03], Haixun Wang et al. calculate θ_i only from available DTD or statistics from sample set of XML documents, and consider θ_i as a constant, which is fixed without considering the actual documents. The objective is self-adaptively allocating dynamic scope for each node according to its actual probability in each document, not just using a fixed constant probability, which is simply calculated from DTD or sample set of datasets.

The proposed Bayesian allocation model considers θ_i as a random variable not a constant, and chooses a proper prior distribution for θ_i, which reflects the observer's beliefs about the proper value θ_i may take, and then updates these beliefs on the basis of the actual data observed. Thus estimators of θ_i can accord with the real world data in a self-adaptive manner. To summarise, the heuristics guiding the allocation model is that the more child nodes of *nodetype*$_i$ a node u has, the more likely for these child nodes being updated.

Given a node u with t children in a DTD. Each of them occurs x_i times, i = (1, ..., t), in the corresponding XML document. x_i may be zero in an actual XML document if a certain node is optional in DTD. Assume 1) all sibling nodes of different *nodetype*$_i$ occur independently with probability θ_i (i=1, ..., t) and 2) the probability of data insertion/deletion on these nodes occurs according to the same probability. Thus, given scope range n for the z nodes in an XML document, if θ_i (i = 1, ..., t) for the z nodes is known, a natural idea is allocating scope for each node type according to probability θ_i. For instance, if all the sibling nodes with *nodetype*$_i$ occur x_i times under a node u, and update probability is θ_i, then we allocate $\frac{n\theta_i}{x_i}$ for each node with *nodetype*$_i$.

In general the Bayesian allocation model is based on the following hypothesis below:

- Sibling nodes of different *nodetype*$_i$ occur independently with probability θ_i (i=1, 2, ..., t), where t denotes the node name types in the correspond DTD. And all data insertion/deletion/update on the nodes of different *nodetype*$_i$

occurs independently with the same probability θ_i.

- θ_i is a random variable, $\pi(\theta_i)$ is used to denotes prior distribution of θ_i , which reflects the beliefs about what values parameters θ_i may take. It is assumed that θ_i is a $beta(\alpha_i, \beta_i)$ distribution

Whether or not using prior distribution in statistical inference is not only a problem of mathematical technique but a problem of philosophy [Ber85] as well. Thus the necessity of using prior distribution here is not discussed, using it by the way of Bayesian inference theory. However, how to choose prior distribution is another problem. Beta distribution is chosen as prior distribution because: 1) density curve of beta distribution is plane curve when $\alpha \gg 1, \beta > 1$, and $0 \leq$ value of beta distribution ≤ 1, matching the definition of probability. Thus $\bar{\theta}_i$, prior information of θ_i, is considered as mean value of beta distribution, which means that the probability of $node_i$ occurs around $\bar{\theta}_i$ is greater than in other zone, matching the hypothesis; 2) from lemma 1 posterior distribution $\theta_i|x_i$ is also a beta distribution which is convenient to compute the posterior estimators of θ_i. Using other prior like norm distribution will result in complicated monte carlo simulations and the computational complexity. In fact, using beta distribution as prior distribution of parameter θ_i in binomial distribution is very common in practice [Ber85].

Theoretical proof In this model sample information of θ_i is considered as occurrence times x_i under a certain node u since it is assumed that a node with $nodetype_i$ occurs with same probability θ_i in the hypotheses. From the Hypotheses x_i observes binomial distribution, denoted as $b(z, \theta_i)$. The updating procedure is performed using Bayesian theorem, which states that the posterior distribution $\theta_i|x_i$, representing the beliefs on the observed data, is proportional to the product of the prior distribution and the likelihood function.

The following two lemmas are proved for the correctness of the Bayesian allocation model:

Lemma 1. *Assume $\pi(\theta_i)$ is beta(α_i, β_i) $(\alpha_i \gg 1, \beta_i > 1)$, and sample information variable $x_i \sim b(z, \theta_i)$ (binomial distribution) with parameter θ_i. Thus, the posterior distribution function $p(\theta_i|x_i)$ is also a beta(α_i^*, β_i^*) distribution, and beta(α_i, β_i) is called conjugate prior distribution. The following result is obtained:*

$$E(\theta_i|x_i) = \lambda_i E(x_i|\theta_i) + (1 - \lambda_i)\frac{x_i}{z}$$

where

$$E(x_i|\theta_i) = \frac{\alpha_i}{\alpha_i + \beta_i}$$

Proof: Given

$$\pi(\theta_i) \sim beta(\alpha_i, \beta_i)$$

and its density function

$$\pi(\theta_i) = \frac{\Gamma(\alpha_i + \beta_i)}{\Gamma(\alpha_i)\Gamma(\beta_i)}\theta_i^{\alpha_i - 1}(1 - \theta_i)^{\beta_i - 1}$$

because x_i is binomial distribution, density function of x_i given parameter θ_i is:

$$f(x_i|\theta_i) = C_z^{x_i}\theta_i^{x_i}(1 - \theta_i)^{z - x_i}, where 0 \le \theta_i \le 1$$

Thus, according to Bayesian theorem posterior distribution of θ_i is:

$$\begin{aligned} p(\theta_i|x_i) &= \frac{f(x_i|\theta_i)\pi(\theta_i)}{\int_0^1 f(x_i|\theta_i)\pi(\theta_i)d\theta_i} \\ &= \frac{\theta_i^{\alpha_i + x_i - 1}(1 - \theta_i)^{z + \beta_i - x_i - 1}}{\int_0^1 \theta_i^{\alpha_i + x_i - 1}(1 - \theta_i)^{z + \beta_i - x_i - 1}d\theta_i} \\ &= k_i\theta_i^{x_i + \alpha_i - 1}(1 - \theta_i)^{z + \beta_i - x_i - 1} \end{aligned}$$

where

$$k_i = \frac{\Gamma(\alpha_i + x_i)\Gamma(\beta_i + z - x_i)}{\Gamma(\alpha_i + \beta_i + z)}$$

hence:

$$p(\theta_i|x_i) \sim beta(\alpha_i + x_i, \beta_i + z - x_i)$$

and:

$$\begin{aligned} E(\theta_i|x_i) &= \frac{\alpha_i + x_i}{\alpha_i + \beta_i + z} = \frac{\alpha_i}{\alpha_i + \beta_i}\lambda_i + (1 - \lambda_i)\frac{x_i}{z} \\ &= E(\theta_i)\lambda_i + (1 - \lambda_i)\frac{x_i}{z} \end{aligned}$$

where

$$\lambda_i = \frac{\alpha_i + \beta_i}{\alpha_i + \beta_i + z}$$

λ_i reflects on the importance balance between prior information and sample information.

Lemma 2. *Assume prior distribution $\pi(\theta_i)$, sample information x_i and square loss function $L(\delta_i, \theta_i)$ is given. The Bayesian estimators of θ_i, $\delta^\pi(x_i)$, is the expectation (or mean value) of posterior distribution $\pi(\theta_i|x_i)$ which is written as:*

$$\delta^\pi(x_i) = E(\theta_i|x_i)$$

Proof: The posterior risk of any decision function $\delta_i = \delta(x_i)$ given square loss function is :

$$E((\delta_i - \theta_i)^2|x_i) = \delta_i^2 - 2\delta_i E(\theta_i|x_i) + E(\theta_i^2|x_i)$$

obviously the function value is minimised *iff*:

$$\delta_i = E(\theta_i|x_i)$$

From Lemma 1

$$\delta_i = \frac{\alpha_i + x_i}{\alpha_i + \beta_i + z} = E(\theta_i)\lambda_i + (1 - \lambda_i)\frac{x_i}{z}$$

In this model it is assumed that $E(\theta_i) = \bar{\theta}_i$ which means that the prior probability of node i occurs around $\bar{\theta}_i$ is greater than in other zone. And from lemma 1 posterior expectation of θ_i is obtained. $E(\theta_i|x_i)$ is then used as estimators of θ_i according to Lemma 2.

Let $\lambda_i = 0.5$, which implies that importance of prior information is the same as that of sample information. Let $\theta_i^{'}$ denote estimators of θ_i given prior information $\bar{\theta}_i$. From Lemma 1 and Lemma 2, we have:

$$\theta_i^{'} = \frac{1}{2}\bar{\theta}_i + \frac{1}{2} * \frac{x_i}{z} \tag{5.1}$$

Eq(5.1) proves that the prior information and sample information both contribute to the final probability a node will be updated, which is better than ViST allocation method which only utilises prior information only.

5.3 Algorithms

The above process of allocating scope in a complete K-ary tree with Bayesian allocation model is called *Bayesianization* in this chapter. After a brief description of how to compute the prior information from DTD, Algorithm 5.1 and Algorithm 5.2 which describe the labelling process that combines with the Bayesian theorem are presented in detail. Figure 5.3 gives an example of the proposed Bayesian allocation model. How to compute these p_i (i=1, 2, 3, ...) is shown in the following section.

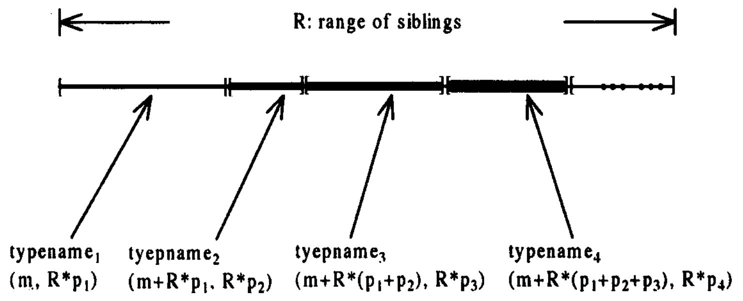


Figure 5.3: An Example of Bayesian Allocation Model

5.3.1 Prior Information

Prior information about the occurrence probability of all the children below a node u in DTD, denoted by $\bar{p}_{DTD_u}$, is defined as follows:

$$\bar{p}_{DTD_u} = (\bar{p}_{c_1}, \bar{p}_{c_2}, ..., \bar{p}_{c_t})$$

where t is the number of different child nodes u has. Each $\bar{p}_{c_i}$, i = 1, 2, ...,t, can be computed from DTD Table 5.1, which defines the proportion of the scope among the different cardinality based on the authors' experience. However, Table 5.1 can also be generated based on the actual users' observation or experience.

For example, the value 0.08 at (?, +) in Table 1 specifies the proportion relationship 0.08:1 between child node type with "?" and child node type with "+". Also the proportion 1.25:1 between child node type with "+" and the one with "*". Thus we have 0.08:1:0.8 having three child nodes with type "?", "+" and "*". Notice that the data in Table 5.1 can be calculated from the statistic information of the sample XML documents.

Actually, Table 5.1 reflects the belief on prior information. (a|b) is transformed into (a,b) to minimise the computational complexity. Consider, for example, given <!ELEMENT book (title?, information+, author*)>, the proportion among these three nodetypes is 0.08:1:0.8, to normalise, $\bar{p}_{title?} = \frac{0.08}{0.08+1+0.8} =$

	*	+	?
*	1	0.8	10
+	1.25	1	12.5
?	0.1	0.08	1

Table 5.1: Prior Information on the Cardinality Proportion

0.043, $\bar{p}_{information+} = \frac{1}{0.08+1+0.8} = 0.532$ and $\bar{p}_{author*} = \frac{0.8}{0.08+1+0.8} = 0.425$. Therefore, $\bar{p}_{DTD_{book}} = (0.043, 0.532, 0.425)$.

5.3.2 Algorithms

As in Algorithm 5.1 and Algorithm 5.2, the specific Bayesian allocation model is light-weight and self-adaptive for each node in the XML document tree. The time complexity of the algorithms is O(n), depending on the number of the nodes in a tree, and the space complexity is linear as well. It implies that the algorithm guarantees both time and space complexity efficiency while allocating self-adaptive scope for each node, which is not provided by the previous methods. The performance results are shown in section 5.4.

Consider, for example, given <!ELEMENT book (title?, information+, author *)>, we get $p_{DTD_{book}} = (0.043, 0.532, 0.425)$. If in an actual XML document, a node named "book" has 1 "title" child node, 2 "information" child nodes, however, 10 "author" child nodes. Suppose the range enlarging factor is 100, thus these 13 child nodes are allocated a scope 13*100 = 1300 (n = c × z, section 2). Thus their actual probability should be <0.0599 = $((0.043 + \frac{1}{13})/2)$, 0.4198 = $((0.532 + \frac{2}{13})/2)$, 0.5971 = $((0.425 + \frac{10}{13})/2)$ >, and the scope allocated for each node are 0.0599*1300 = 77 for "title" node, 0.4198 * 1300 = 545 for "information" nodes, and 0.5971 * 1300 = 776 for "author" nodes. Notice that the allocation

```
input : T: Data tree of XML document; Queue: queue of
        nodes; $p_{DTD}$: the DTD prior distribution generated
output: BAM Allocated Document
while !Queue.empty() do
    u ← Queue.fetch();
    list ← listofchildren(u);
    z ← numofchildren(list);
    t ← typeofchildren(list);
    BayesInference($\bar{p}_{DTD_u}$, z, t, list);
    Queue.insert(list);
```

Algorithm 5.1: Bayesianization

```
input : $\bar{p}_{DTD_u}$: prior information of node u; z: number of
        child nodes; t: number of child node types, list: list
        of nodes.
output: BAM allocated scope of u
level = $\lceil \log_k(c \times z) \rceil$;
n = $k^{level}$;
for $i$ ← 1 to t do
    $p_i \leftarrow \frac{\bar{p}_{DTD_{u_i}} + x_i/z}{2}$;
    $subrange_i$ ← n × $p_i$;
    for $j$ ← 1 to z in list do
        $subrange_{node_j}$ ← $subrange_i$ / $t_i$;
        $seqnum_{node_j}$ ← ($\sum_1^{i-1} subrange_i$ +
        $\sum_1^{j-1} subrange_{node_j}$);
```

Algorithm 5.2: BayesInference

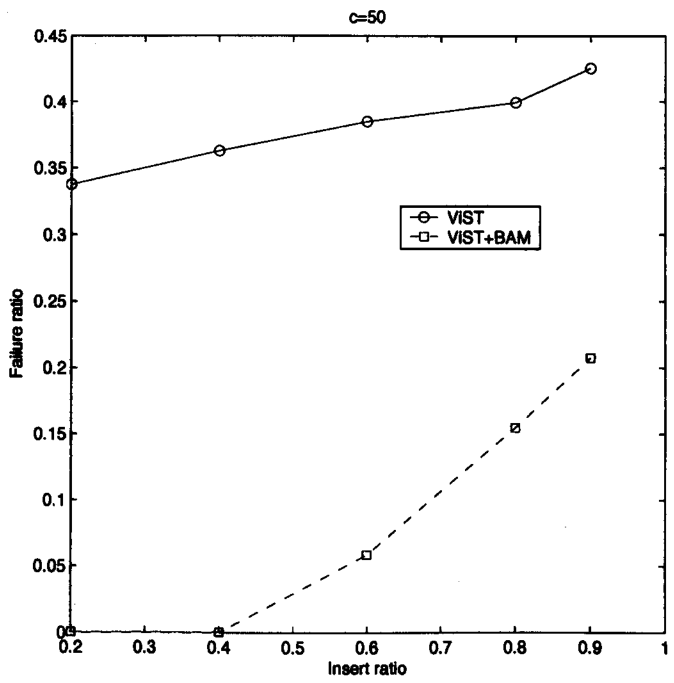


Figure 5.4: Insert ratio. vs. Failure ratioc: C=50

scopes of these three nodes accord to their actual occurrence probability.

5.4 Performance Experiments

5.4.1 Experiments

The experiments were conducted on a PC with Pentium III CPU 1.2GHZ and 512 MB main memory. The proposed method Bayesian allocation model and the conventional fixed allocation scheme for comparison purpose are implemented. The synthetic XML documents are generated using Sun XML instance generator [Sun03] utilising various DTDs, i.e. ACM Sigmod Record DTD [ACM], and public XML benchmark database XMARK [BCF$^+$].

Update performance

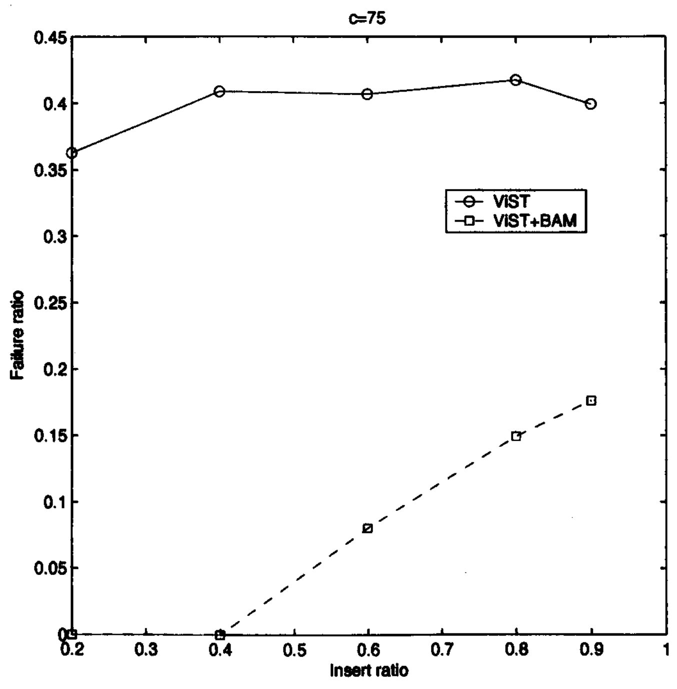


Figure 5.5: Insert ratio vs. Failure ratio: C=75

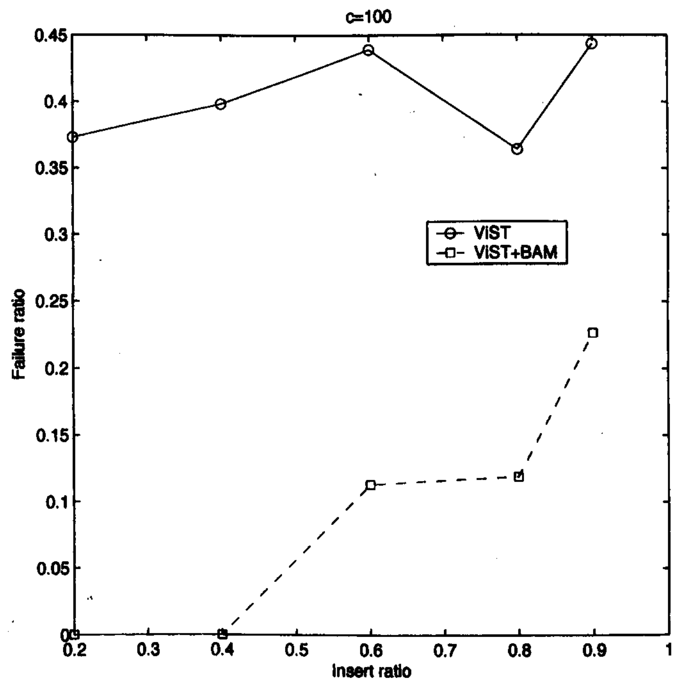


Figure 5.6: Insert ratio vs. Failure ratio: C=100

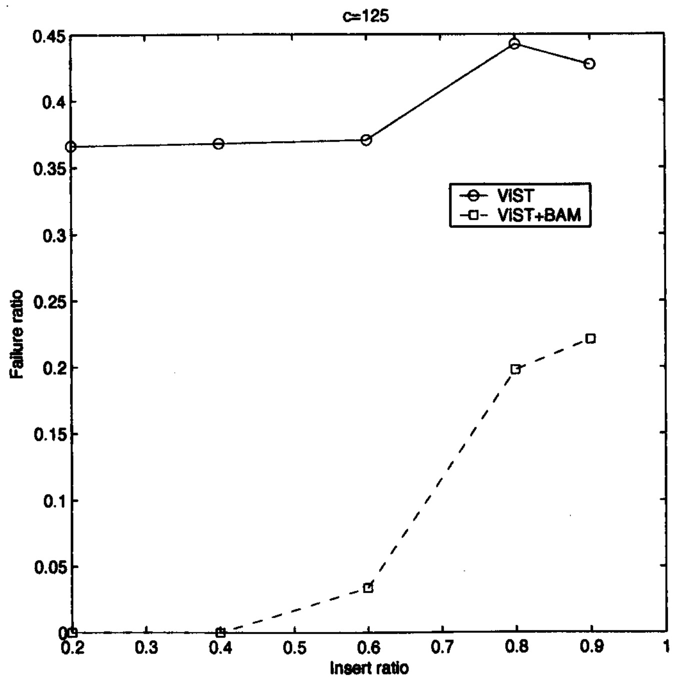


Figure 5.7: Insert ratio vs. Failure ratio: : C=125

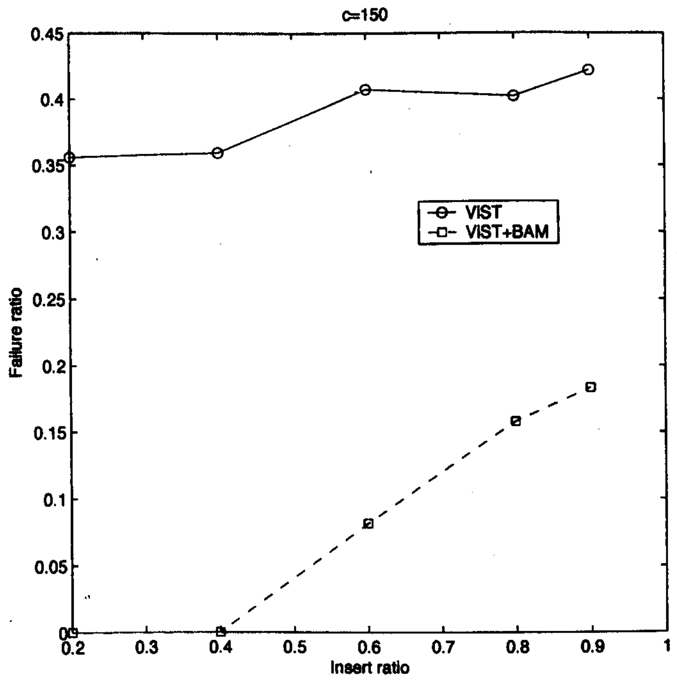


Figure 5.8: Insert ratio vs. Failure ratio: C=150

Th focus is on studying the update performance in dynamic XML documents. 200 XML documents with maximum depth 8 for experiments are generated. Five different experiments are performed with different range enlarging factors. In the experiments, respectively setting the range enlarging factor c = 50, 75, 100, 125 and 150 for these documents in the set of experiments. An allocation scheme similar to ViST and the proposed allocation scheme tailored to ViST are implemented. In ViST, it allocates scope for each type of child nodes directly from DTD without constructing trie-like tree. However, for comparison purpose, the trie-like tree is physically constructed, and then BAM is applied to allocate scope for each node in the trie-like tree.

In the experiments, m nodes are randomly chosen from the generated datasets. Suppose each $node_i$ (i = 1, ..., m) has t different node name types. Firstly the prior information for each $node_i$ from the corresponding DTD is obtained, denoted as $\bar{p}_{DTD_{node_i}} = (\bar{p}_1, \bar{p}_2, ..., \bar{p}_t)$. Then t independent beta distribution are used to generate t random numbers, denoted as $(r_1, r_2, ..., r_t)$. It is proved that $0 \leq r_1, r_2, ..., r_t < 1$. Thirdly the insertion/deletion probability for each node type are generated: $p_i = \frac{r_i}{s}$, where s = $\sum_{i=1}^{t} r_i$, which obeys the hypotheses and is fair to both ViST and ViST with BAM when the probability of insertion/deletion is concerned. Finally the position a node should be inserted/deleted are randomly generated.

It is defined that a "failure" occurs when a position has been allocated during inserting, and an "overflow" when pre-allocated space for a certain node is used up. During the experiments, the "failure" times are recorded. Eq(5.2) and eq(5.3) are presented to clarify the experimental results shown in Figure 5.8.

$$failure_{ratio} = \begin{cases} 1, & \text{when "overflow" occurs,} \\ \frac{times_{failure}}{times_{insert}}, & \text{otherwise.} \end{cases} \qquad (5.2)$$

$$insert_{ratio} = \frac{times_{insert}}{Space_{free}} \qquad (5.3)$$

Notice that BAM improves at least 49.34% comparing to the conventional ViST method, for the scope allocated for each node accords to the probability it would be updated, which further depends not only on the prior information, i.e. statistical information from sample datasets, but combining the actual probability of its occurrence as well. Especially, when the probability of inserting/deleting a node in an XML document is much greater than the average prior information, BAM performs much more better than ViST method.

5.5 Summary

In the chapter, a general self-adaptive labelling scheme for dynamic XML documents is studied. Bayesian allocation model can easily be adapted to the state-of-the-art data retrieval methods to provide support for dynamic XML documents. Through the experiments, it is demonstrated that the proposed model can efficiently support updating for the dynamic XML documents, at least 49.34% better than the conventional methods, while not affecting their original query speed.

CHAPTER 6

Conclusion

What we call the beginning is often the end. And to make an end is to make a beginning. The end is where we start from.

– *Thomas Stearns Eliot*

Previous chapters have discussed in detail the fundamental mechanisms of accelerating data retrieval steps in XML documents. In this chapter, the work that has been done and the major theoretical results obtained are summarised. Several promising research directions are discussed afterwards.

6.1 Prototype

A prototype implementation has been developed in order to test some of these ideas, and to demonstrate the practical use of these three interconnected techniques. The prototype made use of the B+Tree library in Berkeley DB provided by Sleepycat software and the WordNet package. Part of the results are analysed in Matlab. One limitation of the prototype is that it needs some manual processing to transform XPath queries. Another limitation is that the optimised cut-off value is set by the user's knowledge.

The prototype architecture is shown in Figure 6.1. The data is firstly clustered (Chapter 3) and is divided into a certain number of clusters with a optimised cut-

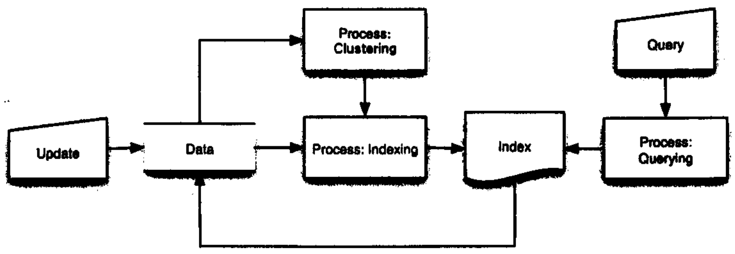
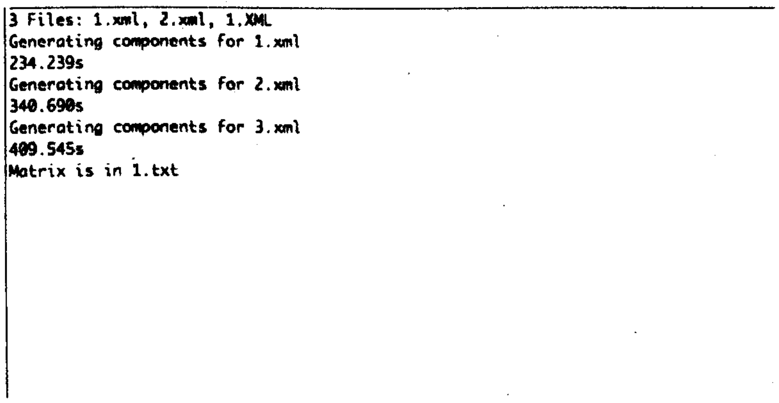
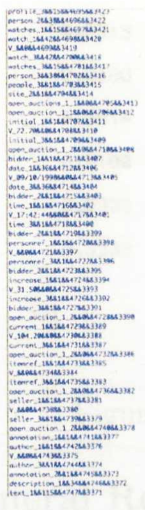


Figure 6.1: Prototype Architecture



```
3 Files: 1.xml, 2.xml, 1.XML
Generating components for 1.xml
234.239s
Generating components for 2.xml
340.690s
Generating components for 3.xml
409.545s
Matrix is in 1.txt
```

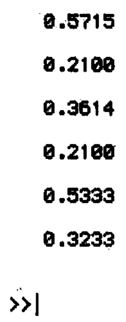
Figure 6.2: Prototype: Screenshot of Clustering



```
profile_3&&15&&4695&&3423
person_2&&38&&4696&&3422
watches_1&&15&&4697&&3421
watch_1&&42&&4698&&3420
V_&&0&&4699&&3419
watch_3&&42&&4700&&3418
watches_3&&15&&4701&&3417
person_3&&38&&4702&&3416
people_3&&1&&4703&&3415
site_2&&1&&4704&&3414
open_auctions_1_1&&0&&4705&&3413
open_auction_1_1&&0&&4706&&3412
initial_1&&1&&4707&&3411
V_72.70&&0&&4708&&3410
initial_3&&1&&4709&&3409
open_auction_1_2&&0&&4710&&3408
bidder_1&&1&&4711&&3407
date_1&&36&&4712&&3406
V_09/10/1999&&0&&4713&&3405
date_3&&36&&4714&&3404
bidder_2&&1&&4715&&3403
time_1&&1&&4716&&3402
V_17:42:44&&0&&4717&&3401
time_3&&1&&4718&&3400
bidder_2&&1&&4719&&3399
personref_1&&1&&4720&&3398
V_&&0&&4721&&3397
personref_3&&1&&4722&&3396
bidder_2&&1&&4723&&3395
increase_1&&1&&4724&&3394
V_31.50&&0&&4725&&3393
increase_3&&1&&4726&&3392
bidder_3&&1&&4727&&3391
open_auction_1_2&&0&&4728&&3390
current_1&&1&&4729&&3389
V_104.20&&0&&4730&&3388
current_3&&1&&4731&&3387
open_auction_1_2&&0&&4732&&3386
itemref_1&&1&&4733&&3385
V_&&0&&4734&&3384
itemref_3&&1&&4735&&3383
open_auction_1_2&&0&&4736&&3382
seller_1&&1&&4737&&3381
V_&&0&&4738&&3380
seller_3&&1&&4739&&3379
open_auction_1_2&&0&&4740&&3378
annotation_1&&1&&4741&&3377
author_1&&1&&4742&&3376
V_&&0&&4743&&3375
author_3&&1&&4744&&3374
annotation_2&&1&&4745&&3373
description_1&&34&&4746&&3372
text_1&&115&&4747&&3371
```

Figure 6.3: Prototype: Screenshot of Indexing

off value. Each cluster is then pumped into the indexing process. In certain case the data can be directly indexed without the clustering process. The indexing process (Chapter 4 and Chapter 5) indexes the clusters and generates the corresponding index for each cluster. When a query is issued, the querying process consults to index and find the answers. When the data is updated, the indexing process (Chapter 5) updates the index accordingly. Several figures show how the prototype works: Figure 6.2 shows the prototype is clustering three synthetic XML files and the similarity matrix is stored in a file called "1.txt". The clustering is processed in Matlab using the cut-off value 0.85. Figure 6.3 demos a screenshot of the trie index structure which is core of Chapter 4. Figure 6.4 shows a result of intensive updating operations using different enlarge constants; Odd lines show the failure ratio of the traditional method. Even lines show the failure ratio of the proposed BAM.



```
0.5715
0.2100
0.3614
0.2100
0.5333
0.3233

>>|
```

Figure 6.4: Prototype: Screenshot of Updating

6.2 General Review of Work and Conclusion

As the World Wide Web is gradually becoming one of the most important communication media, there is an exponential increase in the amount of electronic data which are in a web-compliant format such as HTML [RHJ99], SGML. The emergence of XML [W3C00] has remarkably facilitated the above electronic data publication by providing a simple syntax for data which is both machine- and human-understandable. Though introduced in a document community, XML is quickly gaining popularity in data representation and exchange on the Web [Suc98]. With such growing importance of XML in data representation and exchange in large-scale scientific and commercial applications, substantial research efforts have been made to provide efficient querying mechanisms to retrieve data from XML-formatted data, which is core of efficiently managing XML large volumes of XML data. However, due to its fundamentally different tree data model and query languages, the study of retrieving data from XML documents is inadequate.

This thesis has thus focuses on examining the problems of accelerating data retrieval steps in XML documents, which is an essential part of current XML research efforts. This thesis uses a divide-and-conquer strategy: Chapter 3 (divide step) discusses an accurate clustering mechanism [SW03] to group similar schemaless XML documents based on a new semantic component model. This technique uses only 30% information of an XML document and can guarantee almost 90% correctness of the clustering results according to the experiments. Chapter 4 (conquer step 1) proposes a novel and efficient querying method which evaluates a twig query as a whole unit without semantic false and false alarms based on a novel pre-processing mechanism for both XML documents and queries. This technique initiatively and theoretically transforms both an XML document and an XPath query into geometric document sequence and geometric query sequence with space complexity O(n) . According to the experiments, the performance of retrieving data from XML documents is increased 23.75% on average. Chapter 5 (conquer step 2) presents a robust allocation method [SFSW04] to support update functions in dynamic XML documents based on a new allocation model. This technique self-adaptively combines both prior information from either DTD or sample XML documents and the actual XML documents when pre-allocating space for a node. According to the experiments, this technique decreases the update failure ratio for at least 49.34% comparing to the conventional fixed allocation scheme.

6.2.1 Review of Chapter 3 and Conclusion

Data clustering has been widely recognised as a powerful data mining technique and contributes to a number of areas of research including data mining, statistics, machine learning, spatial database, biology and markets [Dun03]. Chapter 3 discusses the problem of how to accurately cluster schemaless XML documents based on their semantics. The focus in this thesis is accuracy. Accurately grouping similar schemaless XML documents is a preparation step (divide step) to efficiently index and query XML data in Chapter 4. A novel semantic component model is proposed to model the semantic implied in an XML document. According to the experiments, the number of the semantic components generated is less than 30% of the nodes of an XML document. However, it provides a subtle method to grasp the implicit semantic expressed in an XML document and minimises the loss of semantic information due to the trade-off between computational efficiency and semantical accuracy, enabling us to devise a set of heuristic algorithms to compute the degree of semantic similarity among a collection of schemaless XML documents. Algorithm *DocSimilarity* presents a heuristic mechanism to solve the problem of isomorphic tree and tree inclusion detection, which can not be achieved using the traditional edit distance algorithms. Regarding accuracy, a hierarchical clustering technique is utilised to construct final clusters based on the similarity matrix generated. We argue that "offline" clustering XML documents using a hierarchical clustering technique as a preparation step is acceptable in most systems because hierarchical clustering has been considered as the best quality clustering approach. According to the experiments, the semantic component model guarantees almost 90% correctness of the clustering results.

To summarise, this thesis demonstrates that the semantic component model and the corresponding heuristic algorithms can be utilised to accurately construct clusters for data-centric XML data.

6.2.2 Review of Chapter 4 and Conclusion

Efficient XML data retrieval is an essential part of current XML research and plays an important rôle in managing a large volume of XML documents as well as many other XML applications. In Chapter 4, the problem of retrieving data from XML documents by using XPath queries is studied. A novel mechanism called geometric sequence model to transform both XML documents and XPath queries into geometric sequences is proposed. In this thesis, it is theoretically proved in Theorem 4 that the geometric sequence model uniquely transforms an XML document into a numbered geometric document sequence, and strictly guarantees the linear size complexity (i.e. 2n - 1, where n stands for the number of nodes in a document tree) of the document sequence. This model also transforms an XPath query into a geometric query sequence (i.e. not numbered) and preserves all the essential axes information using additional markup characters (i.e. 'p' and 'u') with the same space complexity. The XPath evaluation problem is thus turned into a subsequence matching problem and theoretically proved in Theorem 1, 2 and 3. This new model enables us to devise two join- and false-free algorithms: GeoMatching and OptGeoMatching based on an index structure using the trie data structure, which is fully connected with the clusters generated in Chapter 3. GeoMatching maintains a status stack and a set of sequence stack during the process of top-down evaluating a geometric query sequence. OptGeoMatching considers to reduce the length of the geometric query sequence which is an important factor in evaluating an XPath query, and maintains a set of symbol stacks during the process of bottom-up evaluating a geometric query sequence. Regarding efficiency, while both XML documents and XPath queries are transformed into sequences, the algorithms can significantly accelerate the data retrieval steps in XML documents when XPath queries are either content-related or structure-centered. When XPath queries are related to the content, bottom-up OptGeoMatching can be utilised to evaluate because content constraints can greatly reduce the number of nodes being searched. Otherwise, top-down GeoMatching can be used if XPath queries are not concerned on the content. Comparing to the conventional methods, OptGeoMatching significantly improves the performance of data retrieval steps in both real world data and synthetic data at least 23.75% on average, and decreases the I/O requests at least 31.29% on average.

To summarise, this thesis demonstrates that the geometric sequence model and its two stack-based algorithms (top-down GeoMatching and bottom-up OptGeoMatching) can support XPath query evaluation in XML document more efficiently than the conventional methods.

6.2.3 Review of Chapter 5 and Conclusion

To fully evolve XML into a universal data representation and exchange format, the capability of modifying XML document is indispensable. In Chapter 5, the problem of supporting update operations in XML documents is addressed. A new allocation model called Bayesian Allocation Model for the index structure generated in Chapter 4 is introduced. The Bayesian allocation model is based on the complete k-ary tree data structure and Bayesian inference, a technique intensively used in Statistics and Artificial Intelligence. The focus is to self-adaptively support update functions in an XML document. Chapter 5 theoretically proves the correctness of the Bayesian allocation model with a corresponding labelling scheme on the basis of level traversal of a complete k-ary tree. It also theoretically demonstrates that the Bayesian allocation model finds the best balance between prior information (i.e. sample information of DTDs) and posterior information (i.e. real-world XML documents), and works self-adaptively since the space quota allocated for a certain node has been considered on the basis of both prior information and posterior information. According to the experiments, the proposed Bayesian allocation model decreases the update failure ratio for at least 49.34% comparing to the conventional fixed allocation scheme. Additionally, it can be applied as a general method for most of the index structures.

To summarise, this thesis demonstrates that the Bayesian allocation model is an efficient step in supporting update functions in XML documents and a well-designed supporting step in XML data retrieval process.

XML employs a tree data model and new query languages. These require new mechanisms to support efficient XML data retrieval purpose. This thesis thus presents advanced and novel solutions to the interconnected document clustering, query processing and document updating problems, which are core of current XML research, and achieves the original goal: accelerating data retrieval steps in XML documents.

6.3 Future Research Directions

In this section, several future research directions in the area of XML research are proposed. The end of this thesis marks a new beginning of some opportunities.

XML Applied in Sensor Network A sensor network consists of sensor nodes with a short-range radio and on-board processing capability. The purpose is to process some high-level sensing tasks in a collaborative fashion, and is periodically queried by an external source to report a summary of the sensed data (i.e. in XML format)/tasks. Sensor network poses the following challenges since standard DBMS assumptions about the reliability, availability, interface, and requirements of data sources do not apply to sensors. First of all, sensor network is dynamic and mobile. How to accelerate the performance of processing and querying streaming data (i.e. data in an XML format that is seen only once in a fixed order) generated by sensors in real-time remains open. We have some initial results in processing streaming data [DGGR02, MSHR02, FLBC02, PC03, GGR03, Gz03, ABB+04]. Secondly, sensors in a sensor network is physically distributed and connected through the network. To query distributed sensors is like the evaluation of an XPath query on a single XML document that has been fragmented across multiple sites [IO01, BKK$^+$01, Suc02, DNGS03]. Further improvements are still in need.

Security and Privacy in XML Documents Security and privacy is currently one of the biggest concerns in XML research because the original specifications of its underlying technologies did not even mention security which targets to guarantee availability, integrity, confidentiality, authentication, and accountability. Researches in the aspects of a secured XML environment are still in a preliminary stage. Therefore, many critical problems remain open. Several specification for XML data have been proposed [W3C01a, W3C02b, W3C02a, OAS04], and some research efforts have been carried out on the issues of managing secure XML documents [AF00, CAYLS02, CFGR02, FJ03, BFD$^+$04, FCG04]. First of all, there is no known index that support secure XML documents. If such an index were available, an XPath query could be efficiently processed by probing the index. However, to achieve a secure and encrypted environment, further work is in need to support such purpose. Secondly, if such an index could be achieved, how to fully exploit XPath filtering technique or other advance techniques to enhance the performance of evaluating both static and streaming XML data remain open.

XML Evolution and Versioning Web is a quickly evolving information source and makes the data accessing a difficult task. Researches in accessing dynamic XML data in a dynamically evolving Web environment are inadequate. Some preliminary research efforts have been proposed to tackle these problems [TIHW01, CKM02, CTZZ01, CTZZ02, CVZ+02, CTZ02]. However, there are still issues waiting for better solutions. First of all, there is no ideal storage solution for multi-version XML management system to avoid duplicate storage of shared fragments. To ensure consistency (i.e. link consistency, content consistency etc.) in dynamic XML data is still open. Secondly, if there were solutions for consistency problem, advanced techniques for efficient data retrieval purpose should be developed (i.e. an advanced consistency-preserved indexing mechanism). The core of this problem is how to query multi-version XML documents of the same document. Thirdly, since data mining is an essential task for both academia and industry, how to support data mining in a multi-version XML document environment and maintain logical and physical correctness is still open.

REFERENCES

[ABB+04] Arvind Arasu, Brian Babcock, Shivnath Babu, Jon McAlister, and Jennifer Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Trans. Database Syst.*, 29(1):162–194, 2004.

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from Relations to Semistructured Data and XML.* Morgan Kaufmann Publisher Inc., 2000.

[ACM] ACM. Acm sigmod record dtd. http://www.dia.uniroma3.it/Araneus/Sigmod/.

[AF00] Mehmet Altinel and Michael J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *The VLDB Journal*, pages 53–64, 2000.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[AKJKP02] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, and Jignesh M. Patel. Structural joins: A primitive for efficient xml query pattern matching, 2002.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[AV97] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 122–133. ACM Press, 1997.

[BCF+] Ralph Busse, Mike Carey, Daniela Florescu, Martin Kersten, Ioana Manolescu, Albrecht Schmidt, and Florian Waas. Xmark: An xml benchmark project. http://monetdb.cwi.nl/xml/index.html.

[BDFS97] Peter Buneman, Susan B. Davidson, Mary F. Fernandez, and Dan Suciu. Adding structure to unstructured data. In *Proceedings of the 6th International Conference on Database Theory*, pages 336–350. Springer-Verlag, 1997.

[Ber85] James O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, 1985.

[BFD$^+$04] R. Brinkman, L. Feng, J.M. Doumen, P. Hartel, and W. Jonker. Efficient Tree Search in Encrypted Data. In *Proc. of the 2nd Intl. Workshop on Security in Information Systems*, pages 126–135, 2004.

[BFM98] Tim Bray, Charles Frankston, and Ashok Malhotra. Document content description for xml. http://www.w3.org/TR/NOTE-dcd, July 1998.

[BFS00] Peter Buneman, Mary Fernandez, and Dan Suciu. Unql: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9(1):76–110, 2000.

[BKK$^+$01] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. Objectglobe: Ubiquitous query processing on the internet. *The VLDB Journal*, 10(1):48–71, 2001.

[BYRNRN99] Ricardo Baeza-Yates, Berthier Ribiero-Neto, and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[Cat96] Rick Cattell. *The Object Database Standard: ODMG-93, Release 1.2.* Morgan Kaufmann Publishers, San Francisco, 1996.

[CAYLS02] SungRan Cho, Sihem Amer-Yahia, Laks V.S. Lakshmanan, and Divesh Srivastava. Optimizing the secure evaluation of twig queries. 2002.

[CCD$^+$99] Stefano Ceri, Sara Comai, Ernesto Damiani, Piero Fraternali, Stefano Paraboschi, and Letizia Tanca. XML-GL: A graphical language for querying and restructuring XML documents. In *Proceeding of the International World Wide Web Conference*, pages 151–165, 1999.

[CEM03] Charles E. Campbell, Andrew Eisenberg, and Jim Melton. Xml schema. *SIGMOD Rec.*, 32(2):96–101, 2003.

[CFGR02] Chee Yong Chan, Pascal Felber, Minos N. Garofalakis, and Rajeev Rastogi. Efficient filtering of XML documents with XPath expressions. In *ICDE*, 2002.

[CKM02] Edith Cohen, Haim Kaplan, and Tova Milo. Labeling dynamic xml trees. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 271–281. ACM Press, 2002.

[CM01] James Clark and MURATA Makoto. Relax ng specification. http://www.relaxng.org/spec-20011203.html, 2001.

[CMW03] Byron Choi, Malika Mahoui, and Derick Wood. On the optimality of the holistic twig join algorithms. 2003.

[CRF01] Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. volume 1997, 2001.

[CSF+01] Brian Cooper, Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, and Moshe Shadmon. A fast index for semistructured data. In *The VLDB Conference*, pages 341–350, 2001.

[CTZ02] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. Efficient schemes for managing multiversion xml documents. *The VLDB Journal*, 11(4):332–353, 2002.

[CTZZ01] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, and Donghui Zhang. Storing and querying multiversion XML documents using durable node numbers. In *WISE*, pages 232–241, 2001.

[CTZZ02] Shu-Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo, and Donghui Zhang. Efficient complex query support for multiversion XML documents. In *Extending Database Technology*, pages 161–178, 2002.

[CVZ+02] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassilis J. Tsotras, and Carlo Zaniolo. Efficient structural joins on indexed XML documents, 2002.

[DAM02] Antoine Doucet and Helena Ahonen-Myka. Naive clustering of a large xml document collection. In *Proceeding of the First Annual Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, 2002.

[DFF+99] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for xml. http://www.research.att.com/ mff/files/final.html, March 1999.

[DGGR02] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 61–72. ACM Press, 2002.

[Die82] Paul F. Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122–127. ACM Press, 1982.

[DNGS03] Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 503–514. ACM Press, 2003.

[Dun03] Margaret H. Dunham. *Data Mining Introductory and Advance Topics.* Prentice Hall, 2003.

[FCG04] Wenfei Fan, Chee-Yong Chan, and Minos Garofalakis. Secure xml querying with security views. In *Proceedings of SIGMOD.* ACM Press, 2004.

[Fel98] Christiane Fellbaum. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[FFLS97] Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for a web-site management system. *SIGMOD Rec.*, 26(3):4–11, 1997.

[FGM03] Francesco Francesca, Gianluca Gordano, and Giuseppe Manco. A general framework for xml document clustering. Technical report, ICAR-CNR, 2003.

[FJ03] L. Feng and W. Jonker. Efficient Processing of Secured XML Metadata. In *Proceedings of Intl. Workshop on Security for Metadata*, 2003.

[FK99] Daniela Florescu and Donald Kossmann. Storing and querying xml data using an rdmbs. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 1999.

[FLBC02] Leonidas Fegaras, David Levine, Sujoe Bose, and Vamsi Chaluvadi. Query processing of streamed xml data. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 126–133. ACM Press, 2002.

[FLSW03] Damien K. Fisher, Franky Lam, William M. Shui, and Raymond K. Wong. Efficient ordering for xml data. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 350–357. ACM Press, 2003.

[FMM$^+$02] Sergio Flesca, Giuseppe Manco, Elio Masciari, Luigi Pontieri, and Andrea Pugliese. Detecting structural similarities between xml documents. In *WebDB 2002*, pages 55 – 60, 2002.

[GGR03] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. Processing set expressions over continuous update streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 265–276. ACM Press, 2003.

[GM00] Damien Guillaume and Fionn Murtagh. Clustering of xml documents. *Computer Physics Communications*, 2000.

[GMW99] R. Goldman, J. McHugh, and J. Widom. From semistructured data to XML: Migrating the lore data model and query language. In *Workshop on the Web and Databases (WebDB '99)*, pages 25–30, 1999.

[Gru02] Torsten Grust. Accelerating xpath location steps. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 109–120. ACM Press, 2002.

[GW97] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the Twenty-Third International Conference on Very Larqe Data Bases*, pages 436 – 445. Springer-Verlag, 1997.

[Gz03] Lukasz Golab and M. Tamer &#214;zsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[HGMI$^+$95] Joachim Hammer, Hector Garcia-Molina, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, page 483. ACM Press, 1995.

[HHK95] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, 2000.

[IO01] Hiroshi Ishikawa and Manabu Ohta. Querying web distributed databases for xml-based e-businesses: Requirement analysis, design, and implementation. In *Proceedings of the 12th Australasian conference on Database technologies*, pages 60–67. IEEE Computer Society, 2001.

[ISO99] ISO. *Information Technology-Database Language SQL.* International Organization for Standardization (ISO), standard no. iso/iec 9075:1999 edition, 1999.

[JLW03] Haifeng Jiang, Hongjun Lu, and Wei Wang. Xr-tree: Indexing xml data for efficient structural joins. In *19th International Conference on Data Engineering*, pages 253–264, 2003.

[KYU01] Dao Dinh Kha, Masatoshi Yoshikawa, and Shunsuke Uemura. An XML indexing structure with relative region coordinate. In *ICDE*, pages 313–320, 2001.

[LCMY04] Wang Lian, David W. Cheung, Nikos Mamouis, and Siu-Ming Yiu. An efficient and scalable algorithm for clustering xml documents by structure. *IEEE Transactions on Knowledge and Data Engineering*, 16(3), 2004.

[Ley] Michael Ley. Dblp bibliography. http://www.informatik.uni-trier.de/ ley/db/.

[LLK01] Jung-Won Lee, Kiho Lee, and Won Kim. Preparations for semantics-based xml mining. In *Proc. of 2001 IEEE International Conference on Data Mining*, 2001.

[Llo84] J. W. Lloyd. *Foundations of logic programming.* Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[LM01] Quanzhong Li and Bongki Moon. Indexing and querying XML data for regular path expressions. In *The VLDB Journal*, pages 361–370, 2001.

[LP02] Laks V. S. Lakshmanan and Sailaja Parthasarathy. On efficient matching of streaming xml documents and queries. In *Proceedings of the 8th International Conference on Extending Database Technology*, pages 142–160. Springer-Verlag, 2002.

[LYHY02] Mong Li Lee, Liang Huai Yang, Wynne Hsu, and Xia Yang. Xclust: Clustering xml schemas for effective integration. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 292–299. ACM Press, 2002.

[MAG+97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallas Quass, and Jennifer Widom. Lore: a database management system for semistructured data. *SIGMOD Rec.*, 26(3):54–66, 1997.

[MBV03] Laurent Mignet, Denilson Barbosa, and Pierangelo Veltri. The xml web: a first study. In *Proceedings of the twelfth international conference on World Wide Web*, pages 500–510. ACM Press, 2003.

[MS99] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the 8th International Conference on Database Theory*, pages 277–295, 1999.

[MSHR02] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 49–60. ACM Press, 2002.

[Mur00] MURATA Murata. Hedge automata: a formal model for XML schemata. http://www.oasis-open.org/cover/hedge20000224.html, 2000.

[MW99] Jason McHugh and Jennifer Widom. Query optimization for XML. In *The VLDB Journal*, pages 315–326, 1999.

[MWA+98] Jason McHugh, Jennifer Widom, Serge Abiteboul, Qingshan Luo, and Anand Rajaraman. Indexing semistructured data. Technical report, Stanford University, 1998.

[NND02] N.Bruno, N.Koudas, and D.Srivastava. Holistic twig joins: Optimal xml pattern matching. In *ACM SIGMOD*, June 2002.

[NUWC97] Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *ICDE*, pages 79–90, 1997.

[OAS04] OASIS. Security assertion markup language (saml). http://www.oasis-open.org/committees/download.php/6837/sstc-saml-tech-overview-1.1-cd.pdf, 2004.

[PAGM96] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 413–424, 1996.

[PC03] Feng Peng and Sudarshan S. Chawathe. Xpath queries on streaming data. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 431–442. ACM Press, 2003.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.

[QWG$^+$96] Dallan Quass, Jennifer Widom, Roy Goldman, Kevin Haas, Qingshan Luo, Jason McHugh, Svetlozar Nestorov, Anand Rajaraman, Hugo Rivero, Serge Abiteboul, Jeff Ullman, and Janet Wiener. Lore: a lightweight object repository for semistructured data. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, page 549. ACM Press, 1996.

[RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. Html 4.01 specification (w3c recommendation). http://www.w3.org/TR/html4/, 1999.

[RLS99] Jonathan Robie, Joe Lapp, and David Schach. Xml query language (xql). http://www.w3.org/TandS/QL/QL98/pp/xql.html, March 1999.

[RM04] Praveen Rao and Bongki Moon. Prix: Indexing and querying xml using prüfer sequences. In *ICDE*, pages 288–300. IEEE Computer Society, 2004.

[SFSW04] Yun Shen, Ling Feng, Tao Shen, and Bing Wang. A self-adaptive scope allocation scheme for labeling dynamic xml documents. In Fernando Galindo, Makoto Takizawa, and Roland Traunmüller, editors, *DEXA*, volume 3180 of *Lecture Notes in Computer Science*, pages 811–821. Springer, 2004.

[STZ+99] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 302–314. Morgan Kaufmann, 1999.

[Suc98] Dan Suciu. Semistructured data and XML. In *Proceedings of 5th International Conference on Foundations of Data Organization*, 1998.

[Suc02] Dan Suciu. Distributed query evaluation on semistructured data. *ACM Trans. Database Syst.*, 27(1):1–62, 2002.

[Sun03] Sun. Sun xml instance generator. http://wwws.sun.com/software/xml/developers/instancegenerator/, 2003.

[SW03] Yun Shen and Bing Wang. Clustering schemaless xml documents. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *CoopIS/DOA/ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 767–784. Springer, 2003.

[SWG02] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–52. ACM Press, 2002.

[TIHW01] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Updating xml. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 413–424. ACM Press, 2001.

[TVB+02] Igor Tatarinov, Stratis D. Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 204–215. ACM Press, 2002.

[Via01] Victor Vianu. A web odyssey: from codd to xml. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–15. ACM Press, 2001.

[W3C99a] W3C. Document definition markup language (ddml) specification, version 1.0. http://www.w3.org/TR/NOTE-ddml, 1999.

[W3C99b] W3C. Schema for object-oriented xml 2.0. http://www.w3.org/TR/NOTE-SOX/, 1999.

[W3C99c] W3C. Xml path language (xpath) version 1.0. http://www.w3.org/TR/xpath, 1999.

[W3C00] W3C. extensible markup language (xml) 1.1. http://www.w3.org/TR/2004/REC-xml11-20040204/, 2000.

[W3C01a] W3C. Xml key management specification (xkms). http://www.w3.org/TR/xkms/, 2001.

[W3C01b] W3C. Xml schema. http://www.w3.org/XML/Schema, 2001.

[W3C02a] W3C. Xml encryption syntax and processing. http://www.w3.org/TR/xmlenc-core/, 2002.

[W3C02b] W3C. Xml-signature syntax and processing. http://www.w3.org/TR/xmldsig-core/, 2002.

[W3C03] W3C. Xquery 1.0: An xml query language. http://www.w3.org/TR/xquery/, 2003.

[W3C04] W3C. Resource description framework (rdf). http://www.w3.org/RDF/, 2004.

[WFS05] Bing Wang, Ling Feng, and Yun Shen. Efficient xpath evaluation. In *ADBIS*, 2005.

[WJLY03] Wei Wang, Haifeng Jiang, Hongjun Lu, and Jeffery Xu Yu. Pbi-tree coding and efficient processing of containment joins. In *Proc. of 19th ICDE Conference*, 2003.

[WM05] Haixun Wang and Xiaofeng Meng. On the sequencing of tree structures for xml indexing. In *ICDE*, pages 372–383. IEEE Computer Society, 2005.

[WPFY03] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. Vist: a dynamic index method for querying xml data by tree structures. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 110–121. ACM Press, 2003.

[WPJ03] Yuqing Wu, Jignesh M. Patel, and H. V. Jagadish. Structural join order selection for xml query optimization. In *Proc. of ICDE*, 2003.

[Xyl01] Lucie Xyleme. Xyleme: a dynamic warehouse for xml data of the web. *IEEE Data Engineering Bullet*, 24(2):40 - 47, 2001.

[YASU01] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. Xrel: a path-based approach to storage and retrieval of xml documents using relational databases. *ACM Trans. Inter. Tech.*, 1(1):110–141, 2001.

[YRCK01] Jong P. Yoon, Vijay Raghavan, Venu Chakilam, and Larry Kerschberg. Bitcube: A three-dimensional bitmap indexing for XML documents. *Journal of Intelligent Information Systems*, 17(2-3):241–254, 2001.

[ZND+01] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On supporting containment queries in relational database management systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 425–436. ACM Press, 2001.