Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.

2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.

3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.
Structural Similarity on XML Data and Its Applications

by

NG Kar Leung

A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

October 2006
CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

NG Kar Leung
Abstract

This dissertation addresses issues of detecting the structural similarity of XML (eXtensible Markup Language) documents from heterogeneous sources, and its applications to the areas of querying applications and web mining. This topic has brought much attention and a number of similarity measures have been proposed in recent years. Unlike most distance metrics which are based on the direct transformation between documents, a successful similarity measure should be able to assign higher scores to documents of similar types. To address the problem, we detect and analyze the document conformity against a schema which governs the document structure. Therefore, the goal of our study is to investigate issues involved in defining the structural measure which is supporting the detection of documents of similar types. (1) We first present a formal framework in defining the structural similarity of a document against a schema. We illustrate that the choice of schema languages, DTD or XML Schema, do not constitute major difference in the framework. (2) We extend the framework to compare documents without the prerequisite of a schema.

Structural similarity has a wide variety of applications in automatic document processing. In the second half of the dissertation, we demonstrate its applicability to XML indexing, proximity querying and group detection using the clustering technique. We first propose
RRSi, a novel structural index designed for structure-based query lookup on heterogeneous sources of XML documents supporting proximate query answers. The index successfully avoids the redundant processing of structurally irrelevant candidates that might show good content relevance. An optimized version, oRRSi, of the index is also developed to further reduce in both space and computational complexity. To the best of our knowledge, the structural indexes are the first work supporting proximity twig queries on XML documents. The experiment results show that the RRSi and oRRSi based query processing significantly outperforms previously proposed techniques in the XML repositories with structural heterogeneity.

Then we examine the applicability of structural similarity in the area of web mining. A sitemap is a convenient navigation link system reflecting the true key website structure, and have become a standard website feature. Although website owners may choose to present their services or information in a variety of different ways, a certain level of similarity in web structure and content are often observed for websites in the same domain since they typically follow some evolved community standard. Clustering sitemaps by structure helps to detect groups of websites in identical domains and is complimentary to the link based ranking algorithmic function. We examine in this dissertation how to cluster sitemaps as tree structured documents. We introduce a new similarity measure between sitemaps, which reflects their key characteristics in the scoring. Moreover, the measure supports a centroid-based clustering algorithm avoiding pair-wise comparisons that achieves a significant gain in efficiency. We implemented the proposed clustering algorithm and ran extensive experiments on real and synthetic datasets showing their effectiveness and
efficiency over other clustering algorithms, which were based on previous similarity metrics.
Publications


Acknowledgements

I thank God for encouraging me, giving me wisdom and strength, and making my Ph. D. dream a reality.

I wish to express my appreciation to my advisor, Prof. Vincent Ng, for his guidance and direction. He continually stimulated my analytical thinking and his valuable feedback contributed greatly to this dissertation.

Finally, and most importantly, I would like to express my deepest appreciation to my wife, Winnie Kam, for her sacrifice, unyielding devotion and love. Her support and encouragement had carried me on through difficult times, especially when I wanted to give up. Nothing in a simple paragraph can express my love for her.
Table of Contents

Abstract........................................................................................................................................ 1

Publications .................................................................................................................................. iv

Acknowledgements ................................................................................................................... v

Table of Contents ..................................................................................................................... vi

List of Figures ............................................................................................................................ x

List of Tables .............................................................................................................................. xiii

1 Introduction ............................................................................................................................... 1

2 XML Structural Similarity - A Literature Review ............................................................... 8

2.1 XML Basics ......................................................................................................................... 8

2.1.1 XML Model ............................................................................................................... 9

2.1.2 Data Type Definition ................................................................................................. 11

2.1.3 XML Schema ........................................................................................................... 13

2.2 XML Structural Similarity ............................................................................................... 15

2.2.1 Tree-edit Distance ...................................................................................................... 17

2.2.2 Fourier Transform ..................................................................................................... 18

2.2.3 Tag Similarity ........................................................................................................... 20

2.2.4 Restricted Tree-edit Distance .................................................................................... 22

2.2.5 PMC Similarity ......................................................................................................... 23

2.3 Motivating Example .......................................................................................................... 26

2.4 Conclusion .......................................................................................................................... 29
3 XML Structural Similarity against Schema

3.1 Introduction

3.2 Node Similarity

3.2.1 Partial Conformance

3.2.2 Abstract Schema

3.2.3 Approximate Regular Expression Matching

3.2.4 Distance Normalization

3.3 Document Similarity

3.3.1 Artificial Root

3.3.2 Node Weighting

3.3.3 Structural Similarity Score

3.4 Extendibility to XML Schemas

3.5 Experiment Results

3.5.1 Evaluation Metrics

3.5.2 Synthetic Data

3.5.3 Real Data

3.6 Conclusion

4 Similarity Measure for XML Structures

4.1 Introduction

4.2 Implied Schema Generation

4.3 XML Structural Similarity

4.4 Experimental Evaluation

4.4.1 Evaluation Metrics

4.4.2 Synthetic Data

4.4.3 Real Data

4.5 Extendibility to Structure and Content Comparisons

4.6 Conclusion

5 Application – RRSi: Indexing XML Data for Proximity Twig Queries

5.1 Introduction

5.2 Motivating Example
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Structural Heterogeneity of XML</td>
<td>80</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Proximate Twig Query Answers</td>
<td>82</td>
</tr>
<tr>
<td>5.3</td>
<td>Embedding Similarity</td>
<td>85</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Relaxed Representative Structure</td>
<td>85</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Embedding Similarity</td>
<td>89</td>
</tr>
<tr>
<td>5.4</td>
<td>RRSi</td>
<td>92</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Structure of RRSi</td>
<td>92</td>
</tr>
<tr>
<td>5.4.2</td>
<td>RRSi Construction and Updating</td>
<td>94</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Optimized Implementation of RRSi</td>
<td>98</td>
</tr>
<tr>
<td>5.5</td>
<td>Twig Query Processing</td>
<td>101</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Proximity Twig Query Match</td>
<td>101</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Query Lookup Algorithm</td>
<td>106</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Analysis of Lookup Algorithm</td>
<td>107</td>
</tr>
<tr>
<td>5.6</td>
<td>Experimental Evaluation</td>
<td>110</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Experimental Framework</td>
<td>110</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Experimental Results and Analysis</td>
<td>113</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>115</td>
</tr>
<tr>
<td>6</td>
<td>Application – Clustering Sitemaps for Group Detection</td>
<td>117</td>
</tr>
<tr>
<td>6.1</td>
<td>SI-GRAPH</td>
<td>121</td>
</tr>
<tr>
<td>6.2</td>
<td>Motivating Example</td>
<td>122</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Comparability of Sitemaps</td>
<td>124</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Node Label Similarity</td>
<td>125</td>
</tr>
<tr>
<td>6.3</td>
<td>Link Inclusion Similarity</td>
<td>125</td>
</tr>
<tr>
<td>6.4</td>
<td>Detecting Sitemap Clusters</td>
<td>129</td>
</tr>
<tr>
<td>6.5</td>
<td>Experimental Evaluation</td>
<td>133</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Real Data</td>
<td>134</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Synthetic Data</td>
<td>135</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusion</td>
<td>137</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion and Future Work</td>
<td>138</td>
</tr>
<tr>
<td>7.1</td>
<td>Conclusions</td>
<td>138</td>
</tr>
</tbody>
</table>
7.2 Future Work .................................................................................................................. 139
  7.2.1 XML Clustering by Document Structure and Content Vector ......................... 139
  7.2.2 Correlating XML Data Streams ....................................................................... 140
  7.2.3 Detecting XML Duplicates .............................................................................. 141

References ....................................................................................................................... 142
List of Figures

Figure 2.1 Example of an XML Document ............................................................... 10
Figure 2.2 An XML Document in Tree Representation ....................................... 10
Figure 2.3 Example of a DTD ............................................................................ 12
Figure 2.4 Example of an XML Schema .............................................................. 13
Figure 2.5 Example of Tree-edit Operations: Insertion, Deletion and Substitution 17
Figure 2.6 Restricted Tree-edit Operations: Insertion of Repeating Sub-Branches .... 22
Figure 2.7 Examples of XML Document and DTD as Trees .............................. 23
Figure 2.8 Examples of Element Level and Weighting Scheme .......................... 25
Figure 2.9 Examples of XML Documents (and DTDs) of Structural Heterogeneity 27

Figure 3.1 Example of DTD (top left), XML document (top right) and the tree representation of the XML document (bottom) ..................................................... 34
Figure 3.2 Examples of DTD (left) and Abstract Schema (right) ......................... 37
Figure 3.3 Element Weights (left) and the weight distribution (right) ................. 44
Figure 3.4 Evaluation of Similarity Function ...................................................... 46
Figure 3.5 Evaluation Algorithm SSIMCALC for the smaSim/ Function ............... 48
Figure 3.6 An Example of XML Schema ............................................................ 49
Figure 3.7 Score Distribution for Our Proposed smaSim Measure ...................... 54
Figure 3.8 Score Distribution for tagSim Measure .............................................. 55
Figure 3.9 Score Distribution for PMC Measure ................................................ 55
Figure 3.10 Comparisons of Similarity Score Distributions against Generated DTD (GEN) ............................................................................................................................................................................................. 57

Figure 4.1 Generalized Regular Expressions with Different mcrc ......................................................... 61
Figure 4.2 Example of Schema Generation Process ...................................................................................... 62
Figure 4.3 Two sample XML documents with their implied schemas ......................................................... 64
Figure 4.4 Evaluation Schemas .................................................................................................................. 67
Figure 4.5 Comparison Results of xmlSim(), PMC and Jagadish Measures for Synthetic
Data (X-axis = Similarity Scores, Y-axis = % of Documents) ................................................................. 68
Figure 4.6 Comparison results of xmlSim(), PMC and Jagadish measures for real data (X-axis = similarity scores, Y-axis = % of documents) .......................................................................................................................... 71
Figure 4.7 Comparison XML Documents with Contents ............................................................................... 73
Figure 4.8 Example of Implied Content Schema Generation Process ......................................................... 74
Figure 5.1 Two XML documents describing music albums with different document
structures ................................................................................................................................................. 81
Figure 5.2 An XML document describing photo album ................................................................................ 82
Figure 5.3 An example of twig query tree .................................................................................................. 82
Figure 5.4 Document structure of Doc1 .................................................................................................... 83
Figure 5.5 Structural matching of Q1 against Doc1, Doc2 and Doc3 respectively .................................................. 85
Figure 5.6 An XML document and its tree representation ........................................................................... 86
Figure 5.7 An example of rrs construction ..................................................................................................... 88
Figure 5.8 An illustration of a RRSi .................................................................................................................. 94
Figure 5.9 An illustration of an oRRSi ............................................................................................................. 99
Figure 5.10 A twig query and its tree representation ....................................................................................... 102
Figure 5.11 Query processing flow in RRSi .................................................................................................. 107
Figure 5.12 Comparison diagrams of indexes and query sets for (a) QER (b) Number of Elements Scanned (c) Execution Time ................................................................. 112

Figure 5.13 Comparisons of index sizes against data set sizes ......................................... 114

Figure 5.14 Comparisons of execution times against data set sizes .................................. 115

Figure 6.1 Similarity between two University Sitemaps .................................................. 119

Figure 6.2 Newspaper and Bookstore Sitemaps ............................................................ 123

Figure 6.3 SICLUST Algorithm ..................................................................................... 129

Figure 6.4 Initial Clusters ............................................................................................. 131

Figure 6.5 si-graphs sig_A1 and sig_A3 with Weighting and Link Similarity Figures... 131

Figure 6.6 Clustering Iterations ..................................................................................... 133

Figure 6.7 Accuracy of Clustering Results for (a) Synthetic Datasets (b) Number of Clusters ............................................................................................................... 136
List of Tables

Table 2.1 Comparisons of Similarity Measures using Examples from Figure 2.9 .......... 28
Table 2.2 Summary of Pros & Cons for Similarity Measures from Section 2.2 ............. 30
Table 3.1 Similarity Scores for smaSim Measure (Synthetic Data) ............................... 53
Table 3.2 Similarity Scores for tagSim Measure (Synthetic Data) ............................... 53
Table 3.3 Similarity Scores for PMC Measure (Synthetic Data) ................................. 54
Table 3.4 Comparisons of Similarity Scores against the DTDs: ORG and GEN ........... 57
Table 4.1 xmlSim[] vs idealSim() using examples from Figure 2.5 ......................... 65
Table 6.1 Clustering Results for Real Dataset ............................................................ 134
CHAPTER 1

Introduction

XML is emerging as a standard for data representation on the Web and a large amount of information has been stored in the XML data format. There is an increasing need to be able to automatically process those XML documents for information retrieval, document indexing and similarity clustering and querying applications. The majority of previous works [AS99, AI99, HK04] in document processing has focused on document content while much useful structural information in the structurally rich XML documents has not yet been fully explored. This structural information is an important clue as to the meaning of documents and can be applied to identify documents of related interests that might have significant content differences.

XML documents of heterogeneous sources can contain information in similar categories while they are often developed independently and adopt different formats. Notwithstanding this, these relevant documents have higher chance to share common data structures, for example, the document nodes, <author>, <title>, <publisher> often appear in documents describing books. Identifying structurally similar XML documents and quantifying the level of similarity among them plays a crucial role in supporting automatic document processing. Structural similarity measure is a quantitative method to compare documents to
determine their structural relevance. A successful structural similarity measure should be able to assign higher scores to documents representing similar kinds of information that captures the characteristics of XML documents.

There have been a number of studies [B04, BGM04, FMM+05, HK04, JN02, KTY05] on structural similarity of XML documents in recent years that apply various techniques to compare documents as tree-structured data. However, most of these techniques [FMM+05, HK04, KTY05] calculate the distances between two XML data trees without considering their relevance to their underlying explicit or implicit schemas. These tree-distance based measures, in many cases, fail to tell the document similarity in terms of their representing information categories. For example, there are two document instances generated from the same schema describing two bookstores holding hundreds and thousands of books respectively, a desired measure should be able to determine a very high similarity score between them given that they are both describing the bookstores following the same schema format. However, tree-distance based measures can detect great ‘unwanted’ distances between the two documents due to their significant differences in document sizes.

When the structure of an XML document is governed by the XML schema; two XML documents that conform to a schema will have compatible structures describing the same category of information, and should give maximum or close-to-maximum similarity scores. Although there is an approach in [JN02] assigning lower costs of tree-edit operations to additional but repeating sub-documents, it still cannot prevent the scenario of significant differences detected for documents generated from the same schema.
In the first part of this thesis, we first present a structural similarity measure [NN03] of an XML document against a schema that quantifies the corresponding level of conformity. Since schema is representing the data category of the conforming XML documents, this can be served as a measure on how close the XML data to a data category. We abstract a schema as a set of node declarations using regular expressions and define the node level similarity as the normalized string edit distance between its child node string and its corresponding declaration as regular expression. By assigning node weightings based on sub-document sizes, the document level similarity measure is defined as the weighted aggregation of node level scores. The measure successfully captures the characteristics of XML documents including the node level, sibling order and indifference of repeating nodes. Moreover, it is able to identify pattern matches across different document tree levels. This measure represents the level of similarity against the schema and always gives the value of ‘total match’ for documents generated from the same schema. We further define a structural similarity measure [NN05, NN07] which compares XML documents directly. The idea is to generalize from documents the implied schemas and apply the weighted average of the schema-based similarity scores. This successfully eliminates the prerequisite of a schema in determining the structural similarity while the associated well-properties of schema-based measure are being inherent.

Structural similarity has been an important topic and serves the core metric in many proposals [AKM+05, CLMY04, CJ01, GLP04] to support automatic document processing of structural heterogeneity. We illustrate, in the second part of thesis, how our proposed approach [NN03, NN05] of determining XML structural similarity is applied to document indexing and querying. We propose RRSi [NN06b], a novel structural index designed for
structure-based query lookup on heterogeneous sources of XML documents supporting proximate query answers. The index successfully avoids the redundant processing of structurally irrelevant candidates that might show good content relevance. An optimized version, oRRSi, of the index is also developed to further reduce in both space and computational complexity.

Another application of our structural similarity approach is in the area of web mining for detecting web communities [AC05, ACG+04, AIKT03, BG04, GL03, KMSY02], which are collections of websites sharing common interests on specific topics. A web community is detected by having more inter-site links within the community than outside the community. However, this result of approach can be easily disturbed by those links not reflecting the interest of the websites such as the spam links [CCC+06, DW05]. On the other hand, we believe that the intra-site links contain valuable information about the interest and domain of a websites. A sitemap is a special kind of web page links within a website in hierarchical structure, as all the backward links have been removed. A proper clustering of sitemaps by the hierarchical structures can detect the belonging domains of websites and is very useful in searching of services available in the Internet. By viewing sitemaps as a special type of semi-structured documents (like the XML documents), we examine how to cluster sitemaps as tree structured documents. We introduce a new similarity measure [NN06a] between sitemaps, which reflects their key characteristics in the scoring. Moreover, the measure supports a centroid-based clustering algorithm avoiding pair-wise comparisons that achieve a significant gain in efficiency. Our objective is to cluster the intra-site links, which are not affected by spam links, providing complementary information to increase the quality of detected groups.
More specifically the contributions we make in this thesis are given below.

- **XML Structural Similarity**

  We propose $smaSim()$ capturing the structural similarity of an XML document against an abstract schema, which can represent the document structure aspects of both DTD and XML Schema. The measure successfully assigns higher scores to higher level node matching, and takes into account of node sibling orders in the scoring. Moreover, it can always detect a ‘total match’ if an XML document is generated by the comparing schema. We further propose $xmlSim()$ to detect structural similarity among XML documents extended from $smaSim()$ while eliminated the prerequisite of a schema. The measure possesses the wellness properties inherent from $smaSim()$. We implemented the proposed similarity measures and ran extensive experiments with real and synthetic datasets. The results show that the proposed measures attain better effectiveness comparing previously proposed techniques.

- **Indexing XML Data for Proximity Twig Queries**

  We introduce $RRSi$, a structure index to support XML queries that can quickly locate the structurally relevant documents from heterogeneous sources of XML data. This avoids the redundant processing of structurally irrelevant candidates that might show good content relevance. We define $emSim()$ score capturing the structural similarity among XML documents and queries. The metric successfully assigns higher similarity scores to higher level query nodes matched against the querying documents. This score
allows the proximate query processing associated with $RRSi$. We propose an optimized index version of $RRSi$, $oRRSi$, which stores index nodes more effectively with trade-off of the accuracy of query results. $RRSi$ and $oRRSi$ are analyzed and $oRRSi$ achieves a significant space and performance advantage. We implemented all the proposed structure indexes and lookup strategies. We ran extensive experiments with real and synthetic datasets and queries while the results show that $RRSi$ and $oRRSi$ consistently outperform previously proposed techniques, in terms of result accuracy and query processing efficiency.

- **Clustering Sitemaps for Group Detection**

We propose $si$-$graph$ to represent the summary structure of sitemaps. We define the $si$-$similarity$ metric measuring the structural similarity between $si$-$graphs$, and apply it to sitemap clustering to provide complementary information in detecting community groups in the area of web mining. The similarity measure reflects their structural characteristics in the scoring and supports a centroid-based clustering algorithm avoiding pair-wise comparisons that achieves a significant gain in efficiency. We implemented the proposed clustering algorithm and ran extensive experiments on real and synthetic datasets showing their effectiveness and efficiency over the clustering algorithm, which was based on previous similarity metrics.

This thesis contains 7 chapters. Chapter 2 surveys background material on the problem of identifying structurally similar XML documents, presenting several existing algorithms used to detecting structural similarity. Chapter 3 presents the foundation of our approach,
the structural similarity of XML documents against schema. Chapter 4 extends further the
definition to a measure directly comparing XML documents. Chapter 5 demonstrates the
application of structural similarity in the document indexing and proximate querying.
Chapter 6 illustrates another application of structural similarity in clustering sitemaps, as
semi-structured documents, for group detection in the field of web mining. Chapter 7
concludes the thesis and presents directions for future research.
CHAPTER 2

XML Structural Similarity - A Literature Review

Numerous research studies [B04, BGM04, FMM+05, HK04, JN02, KTY05] have been conducted on the structural similarity for heterogeneous XML documents. The main interest is to quantify the similarity of XML documents falling in the same information category for effective document classification and querying. In this chapter, we present the state of the art in structural similarity for XML documents, and describe the key ideas and discuss advantages and disadvantages of proposed solutions.

2.1 XML Basics

XML (eXtensible Markup Language) [W3C98a] is the emerging standard format for structured documents and data on the Internet. It is a self-descriptive markup language recommended by the World Wide Web Consortium, capable of describing a wide variety of data on the Web and elsewhere. XML is designed to facilitate the sharing of data across different systems and across different platforms. Unlike the HTML (HyperText Markup Language), XML is not limited by fixed set of tags and is extensible by creating new tags when needed. It is a meta-language capable for defining customized markup languages specific to an information or an industrial category, such as the FpML (Financial Products
Markup Language) [FPML04] for the e-commerce activities in the field of financial derivatives, and the PMXML (Project Management Extensible Markup Language) [PMEM04] for the project planning and management. XML is derived from SGML (Standard Generalized Markup Language), the international standard meta-language for text markup systems (ISO 8879), without some of the more esoteric features of SGML. In this Chapter, we briefly present an overview of XML model and the schema languages.

### 2.1.1 XML Model

XML is a textual representation of the hierarchical data model consisting of nested elements, the main building blocks. The basic syntax for one element is:

```
<name attribute='value'>content</name>
```

The XML document in Figure 2.1 contains nested elements holding a CD’s information; purely information wrapped in XML tags. Separate software is necessary to transmit or display it. An XML document typically consists of two major components – *document structure* and *data content*. The document structure specifies hierarchical relationship of self-descriptive XML tags, for example, a `<name>` tag under a `<song>` tag contains a song name rather than a book author name. Figure 2.2 illustrates the previous document example in tree representation.
Document structure is governed by the schema specified in one of the schema languages. There are a number of schema languages proposed for various requirements of applications including DTD (Document Type Definition), XML-Schema, SOX (Schema for Object-Oriented XML), Schematron, DSD (Document Structure Description) and RELAX.
(Regular Language Description for XML), etc. Among them, DTD and XML Schema are the most popular schema languages for XML documents. DTD has a longer history and contains simple syntax making use of the well known regular expression constructs. However, DTD lacks of sufficient expressive power to satisfy the need for automated document processing. XML Schema language is proposed and recommended by the World Wide Web Consortium that is more expressive and usable by a wider variety of applications. The specification is mature and stable and has been adopted by the commercial and research communities. Therefore, in the subsequent sections, we describe the basics of DTD and the XML Schema and also explain their major differences.

### 2.1.2 Data Type Definition

DTD is a formal description in XML Declaration Syntax of a particular type of document. It specifies the structures for XML documents; what names are to be used for different types of elements, where they may occur, and how they all fit together. Any document, which conforms to the specification in DTD, is called valid with respect to the DTD. DTD contains markup declarations that provide a grammar for a class of XML documents. Elements and attributes are defined by the keywords `<!ELEMENT>` and `<!ATTLIST>` respectively, specified by the following syntax:

```
<!ELEMENT> <element_name> <element_declaration>
<!ATTLIST> <attr_name> <attr_type> <attr_option>
```

The element declaration specifies the logical structure of the element contents based on the regular expressions. The operators “?” , “*” , “+” are used to represent the allowed instances
of “zero or one”, “zero or many” and “one or many” respectively. Figure 2.3 shows an example of DTD for CD sound tracks. The element <CD> contains only one instance of sub-element <title>, one or many instances of sub-element <artist> and <song>, and zero or one sub-element <year>. Moreover, there is one attribute disc-length associated with the element <CD>.

```xml
<!DOCTYPE CD [ 
  <!ELEMENT CD      (artist+,title,year?,song+)>
  <!ATTLIST CD disc-length CDATA>
  <!ELEMENT song    (title)> 
  <!ELEMENT artist  (#PCDATA)>
  <!ELEMENT title   (#PCDATA)>
  <!ELEMENT year    (#PCDATA)>
 ]>
```

**Figure 2.3** Example of a DTD

Despite the importance of DTD, XML documents can be generated by application software without an explicit DTD. Although DTD is not a mandatory requirement, generated documents still follow some implicit structure rules. There has been a number of schema extraction methods proposed for semi-structured data addressed in [AMN98, FS97, GW97]. The state-of-the-art methodology is the XTRACT system [GGR+00, GGR+03] for inferring a DTD from a repository of XML documents. It employs a sequence of sophisticated steps that involve: (1) generating candidate declarations, (2) factoring candidate declarations using optimization rules, and (3) applying the Minimum Description Length (MDL) principle to select the best declarations and form an optimal DTD.
2.1.3 XML Schema

XML Schema [W3C04] is designed to be more expressive than the DTD for describing the document structure that the XML Schema language is also referred to as XML Schema Definition (XSD). XML Schema is XML-based supporting data typing and namespaces which can be more usable by a wider variety of applications. It is specified in terms of constraints on the structure and content of documents that results in describing a category of XML documents. An example of an XML Schema is shown in the Figure 2.4.

```xml
<xsd:element name="CD">
  <xs:attribute name="disc_length" type="xsd:time"/>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="artist" type="xsd:string" minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="year" type="xsd:integer" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="song" minOccurs="1" maxOccures="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

**Figure 2.4 Example of an XML Schema**
The `<CD>` and `<song>` elements are of complex types because they contain other elements while the other elements `<artist>`, `<title>`, `<year>` and `<name>` are of simple types. Comparing to DTD, the simple types support a much richer set of data types (`xs:string`, `xs:decimal`, `xs:integer`, `xs:Boolean`, `xs:date` and `xs:time`). Moreover, DTD can only specifies the element frequency using the regular expression operators “?” “+” and “*” while XML Schema can support any integral range of element frequency using the keywords “minOccurs” and “maxOccurs” specifying the lower and upper bounds.

To summarize, XML Schema possesses a number advantages over the DTD:

1) XML Schema is itself an XML document. It reduces the learning time while many existing XML tools can be applied directly, including the XML editor, format valuator, etc.

2) XML Schema supports over thirty different data types while DTD supports only a few primitive data types, making it sufficient to facilitate automated data exchange with other application components.

3) XML Schema language has more expressive power over the DTD language. In particular, XML Schema can easily specify unordered elements using the `<xsd:all>` keyword while it needs to specify all element sequence combinations operated by the “|” operator.

4) Namespace is supported in XML Schema but not in DTD.
2.2 XML Structural Similarity

XML Documents are governed by schema (e.g. DTD or XML Schema). Schema specifies the valid or allowable document structures which constitute the document type or category. There have been more and more industries or domains setting up their own standards of schemas, e.g. the “Extensible Business Reporting Language (XBRL)” documents utilized for financial reporting and the “Web Services Description Language (WSDL)” documents for Web Services.

When we are interested in the similarity between two XML documents in the view of their underlying document categories, we compare their document structures. One approach is to compare XML documents modeled as document trees by applying a number of tree similarity techniques. However, this fails to detect the structural similarity from the document category’s point of view. For example, we can have two XBRL documents specifying the financial statements of two different size companies. One multinational company might have two thousands product lines while the other medium size company might have on 10 products. Then the corresponding two business reports can vary significantly in terms of document size and document tree structures. On the other hand, documents generated by heterogeneous sources can describe same or similar types of information and have similar schemas.

Therefore, the key interest in this thesis is to detect and qualify the structural similarity among XML documents in terms of the underlying explicit or implicit schemas. Detecting
and qualifying similarity is a hard topic. Our objective to determine the structural similarity of XML documents according to the underlying document categories. First of all, the measure should compare document structures (element hierarchy) instead of document data. It also should not solely compare the “absolute” structural similarity as document trees as undesired dissimilarity can be misled by the repeating substructures generated by the schema rules. On the other hand, a good measure should be able to detect high or full similarity when comparing documents generated from the same schema, make the indifference of repeating document substructures generated by schema rules, and assign higher similarity weighting to higher level nodes which describe higher level categories for documents.

XML document is a tree-structured document and the associated study of structural similarity has a long history. An early approach makes use of the tree-edit distance to detect and measure the document difference by counting the minimal set of edit operations. Since it is expensive to compute the distance, a recent method approximates the tree-edit distance by Fourier Transform technique to reduce the computational effort. More recent results focus on detecting the types of documents by examining the document structures. There have been significant efforts to adapt document schemas specific to XML documents, either explicitly or implicitly, in detecting the structural similarity. A review of these methods is described in the following sections.
2.2.1 Tree-edit Distance

Tree-edit distance involves a widely adopted method of comparing tree-structured data that can be applied to the tree representations of XML documents. It is the edit distance between two ordered labeled trees, which is a measure of the structural similarity among trees. There are three types of tree operations, which can transform a tree into any other tree. The operations are insertion, deletion and substitution. The tree-edit distance between two trees is defined as the minimum number of tree-edit operations to transform one tree into another. The problem was analyzed in [SZ89], in which a simple fast algorithm was proposed. The algorithm calculates the tree distance by recurrence, with a careful evaluation sequence to optimize the complexity.

![Example of Tree-edit Operations: Insertion, Deletion and Substitution](image)

**Figure 2.5** Example of Tree-edit Operations: Insertion, Deletion and Substitution
XML documents can be represented as ordered labeled trees, in which tree distance algorithm can be directly applied to measure the structural similarity of XML documents. It practically compares the structural differences (by the minimal number of tree-edit operations) of tree representations of documents without taking into consideration of the types of information that the documents are describing. For example, two XML documents of bookstores holding tens and hundreds of books respectively give large tree-edit distances even if they have the similar document structures. Recently, [CDS+04] attempts to solve this problem by determining the tree-edit distance between the tree structural summaries, instead of directly between the document trees, tree structural summary is extracted by reducing nested and repeated nodes. The smaller tree summaries can also attain efficiency advantage in the tree-edit distance computation. However, nesting reduction that relocates and merges sub-trees of nested nodes is actually a destructive action in terms of hierarchical information.

While tree-edit distance supports comparison of document structures, these distances between different pairs of documents are not comparable. For example, a distance of ten can mean large structural difference between two small documents while it can also imply high structural similarity between two huge documents.

### 2.2.2 Fourier Transform

The Fourier Transform Technique of measuring document similarity is introduced [FMM+02, FMM+05]. It encodes the XML document structure into a sequence of numbers as a time series and converts the data into a set of frequencies using Fourier Transform. The
The structural difference between two documents is calculated from the difference of the magnitudes of the two signals.

The transformation process consists of tag and document level encoding. Tag encoding is an injection mapping assigning a unique positive real number to each different named start tag while assigning the same but negative number to the corresponding negative tag. Therefore, for two comparing documents, the same tag mapping must be applied to both documents. Document encoding is the mapping process from an XML document structure into a time series. In [FMM+02, FMM+05], a multilevel encoding is adopted, which captures the document structure with higher weights given to elements at higher levels of document tree.

The similarity measure for two comparing documents $d_1$ and $d_2$ is defined as the approximation of the difference of the magnitudes of the two signals using Fourier Transform:

$$\text{dist}(d_1,d_2) = \left( \sum_{k=1}^{M/2} \left( \|FT(\text{enc}(d_1))(k)\| - \|FT(\text{enc}(d_2))(k)\| \right)^2 \right)^{1/2}$$

where $FT$ is the interpolation of the Fourier Transform corresponding to the frequencies occurring in both encodings of $d_1$ and $d_2$, and $M$ is the total number of points occurring in the interpolation.
A deficiency of this method is the need to align a consistent tag encoding among all the documents to be compared. The need to pre-compute the maximum depth among the whole document population implies that any change of the comparing document set can lead to a substantial preparation rework. While this Fourier method addresses the problem of detecting document similarity differently from the tree edit distance approach, it still suffers from the same deficiency since its goal is to compute the absolute document similarity rather than the document category.

### 2.2.3 Tag Similarity

Tag similarity [B04] is a simple metric to detect the structural similarity of XML documents. It measures the number of common tags between two documents out of the set of the all tags from the comparing documents. The higher proportion for common tags, the higher the structural similarity is. The reason is that the set of allowable tags can be appeared in an XML document is governed by the schema. Therefore, documents sharing more common tags are likely have similar schema which specifies the document structure.

For two documents $d_1$ and $d_2$, the tag similarity is defined

$$tagdist(d_1, d_2) = \frac{|TAG(d_1) \cap TAG(d_2)|}{|TAG(d_1) \cup TAG(d_2)|}$$

where $TAG(d)$ is the set of distinct tags from document $d$. It measures the ratio of intersection of tag sets over the union of tag sets. This definition has the advantage over the other distance metrics, such tree-edit distance, Fourier Transform, etc., which do not
consider the degree of similarity to a similar schema, and hence fail to detect documents of the similar category.

However, the tag similarity is computed using the set of distinct tags without considering the number of tag occurrences. Intuitively, a common tag having the same number of occurrences in the comparing documents should contribute a higher similarity score than the case that one document has much higher number of occurrences. Therefore the weighted tag similarity measure is defined to extend the simple tag similarity.

\[
\text{wtagdist}(d_1, d_2) = \frac{2 \times |\text{TAG}^C(d_1) \cap \text{TAG}^C(d_2)|}{|\text{TAG}^C(d_1) \cup \text{TAG}^C(d_2)|}
\]

where \( \text{TAG}^C(d) \) is the tag collection from document \( d \) where repeating tags are retained in the collection. It effectively measures the weighted ratio of common tags over total number of tags collected from the comparing documents.

Nevertheless, the weighted tag similarity still cannot cater an important factor of the document structure; that is the hierarchical relationship of tags. For example, a \(<\text{title}>\) tag under the \(<\text{song}>\) parent tag mean differently to that under the \(<\text{book}>\) parent tag. So, this tag similarity has low accuracy for documents with limited tag labels while the document structure varies widely.
2.2.4 Restricted Tree-edit Distance

In [JN02], a similarity measure is introduced based on tree edit distance with restricted edit operations. It focuses on assigning lower distance to documents conforming to the similar schema. A schema allows repeatable elements in the document instances, for example, the “*” operator from a DTD. Different repetition choices for a repeatable element can lead to the occurrence of a number of sub-branches in the documents. Measuring the structural distance by counting the original insert, delete and substitute edit operations can lead to a great distance that is not desired. Therefore, the allowable sequences of restricted edit operations are proposed that permits the moving of a whole sub-branch as a single edit operation. A sequence of edit operations is defined as allowable if it satisfies two conditions:

1) A sub-branch can be inserted or deleted if it also occurs in the comparing document; and
2) A sub-branch that has been inserted or deleted cannot have sub-elements inserted subsequently or deleted previously respectively.

Figure 2.6 Restricted Tree-edit Operations: Insertion of Repeating Sub-Branches
The restricted edit operations attempt to assign relatively lower movement costs to similar repeating elements; however it still contributes non-zero distances for repeating elements in documents generated from the same schema that can interference the detection of structurally similar documents.

### 2.2.5 PMC Similarity

Based on the idea of the *PMC (plus, minus and common)* element counts, in [BGM04, BGMT01], a similarity measure is defined between an XML document and a Document Type Definition (DTD), which specifies the structures for XML documents. The measure is calculated as the weighted sum of the aggregated *PMC* values, at each document tree level, against the closest instance DTD. A brief definition is given in this section.

![Figure 2.7 Examples of XML Document and DTD as Trees](image-url)
Figure 2.7 present an XML document and a DTD as trees. An element from the XML document is called *common or plus* depending on whether the element appears in the DTD or not respectively. On the other hand, an element appearing in the DTD while not in the XML document is called *minus*. Intuitively, to achieve the best similarity, plus and minus elements should be minimized while common elements should be maximized. In order to make the numbers of *plus*, *minus* and *common* elements well-defined, an instance DTD is introduced by selecting the repetition choices for every repeatable element (operated by *, +, ?) in the DTD which gives the optimal *PMC* value. For the elements from the same level in both the document and an instant DTD, the *PMC* value can be computed. Then the Resultant PMC value is defined as the weighted sum of PMC values from each document level, with higher weightings assigned to higher level elements.
Finally the structural similarity measure is defined by the evaluation function:

$$\epsilon((p, m, c)) = \frac{c}{p + c + m}$$

In Figure 2.8, the Resultant PMC value is computed as $8 \times (0, 0, 1) + 4 \times (1, 1, 2) + 2 \times (1, 2, 2) + 1 \times (0, 1, 1) = (6, 9, 21)$, giving a similarity value of $21 / (6 + 9 + 21) = 0.58$.

The PMC similarity aims to determine the similarity of XML documents against the DTD that can represent of similarity in term of information category of documents. However, it
assumes that document elements are unordered which invalidates the XML document property. Moreover, the $PMC$ values are computed for elements of the same level, implying that any sub-document under an unmatched node is ignored, even if these sub-documents from different levels match very well.

2.3 Motivating Example

In this section, we present a set of XML documents (and the corresponding DTDs) in Figure 2.9, namely $XML_a$, $XML_b$, $XML_c$ and $XML_d$. They possess different document structures while, intuitively, with various degrees of structural similarities. We first we discuss the how an reference measure would behave in order to determine the structural similarity with respect to the document category.

Take $XML_a$ as the comparison base and compare it against the other documents. For Document $XML_b$, it has highly similar structure with $XML_a$ while $XML_b$ contains eight additional document branches of the $<song>$ tabs. Both $XML_a$ and $XML_b$ are generated from $DTD_a$, and hence they fall into the same document category. Suppose $refSim()$ is the “reference” similarity measure that we would like to have to determine the structural similarity by document category, $refSim(XML_a, XML_b)$ should give an indication of “perfectly matched”. For Document $XML_c$, it is also has highly similar structure with $XML_a$ while $XML_c$ has an additional tab $<songs>$ collecting all the $<song>$ tabs. Therefore, we expect that $refSim(XML_a, XML_c)$ should give an indication of “highly matched”. Lastly, $XML_d$ describes a movie while $XML_a$ describes an album that they intuitively belong to
different document categories. Thus we expect that \( \text{refSim}(XML_a, XML_d) \) should give an indication of “rarely matched” event both documents hold a number of overlapping tags, including \(<\text{name}>\), \(<\text{artist}>\), \(<\text{procedure}>\), \(<\text{title}>\) and \(<\text{duration}>\).

\[
\begin{align*}
\text{XML}_a & \quad \text{DTD}_a \\
\text{XML}_b & \quad \text{DTD}_b \\
\text{XML}_c & \quad \text{DTD}_c \\
\text{XML}_d & \quad \text{DTD}_d
\end{align*}
\]

**Figure 2.9** Examples of XML Documents (and DTDs) of Structural Heterogeneity

Now, we apply the previous measures introduced in Section 2.2 to determine the structural similarity between \(XML_a\) against \(XML_b, XML_c\) and \(XML_d\) respectively in Figure 2.9, with comparison to the reference results shown in Table 2.1. With the exception for PMC
measure from Section 2.2.5 that can only compare XML document to DTD so we do a similar comparison of $DTD_a$ against $XML_b$, $XML_c$ and $XML_d$ respectively.

<table>
<thead>
<tr>
<th></th>
<th>$XML_b$ vs $XML_a$</th>
<th>$XML_c$ vs $XML_a$</th>
<th>$XML_d$ vs $XML_a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree-edit Distance</td>
<td>24</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Fourier Transform</td>
<td>0.511</td>
<td>0.787</td>
<td>0.049</td>
</tr>
<tr>
<td>Weighted Tag Similarity</td>
<td>1</td>
<td>0.95</td>
<td>0.7</td>
</tr>
<tr>
<td>(between 0 and 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted Tree Distance</td>
<td>8</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>PMC Similarity</td>
<td></td>
<td>0.51</td>
<td>0</td>
</tr>
<tr>
<td>(between 0 and 1)</td>
<td>($XML_b$ vs DTD$_a$)</td>
<td>($XML_c$ vs DTD$_a$)</td>
<td>($XML_d$ vs DTD$_a$)</td>
</tr>
<tr>
<td>refSim()</td>
<td>Perfectly Matched</td>
<td>Highly Matched</td>
<td>Rarely Matched</td>
</tr>
</tbody>
</table>

Table 2.1 Comparisons of Similarity Measures using Examples from Figure 2.9

The results in Table 2.1 should different aspects of deficiencies of each previous method in the capability to determine the structural similarity.

- **Document Tree Comparison**: The *Tree-edit distance* [SZ89] and *Fourier Transform* [FMM+02, FMM+05] techniques compare XML documents as tree-structured data, and hence fail to detect documents of similar types while having significant differences as document trees.

- **Element Hierarchy**: The *Tag Similarity* [B04] method compares the common tags from documents. While this holds the information on the document types, it ignores the important piece of information about the ancestor-descendant relationship of elements.

- **Comparison by Level**: The technique in [BGM04, BGMT01] computes the $PMC$ values only for elements of the same level in the document trees. This can cause any
sub-branch under an unmatched element being skipped from the matching processing although it can match very well with other sub-branches from different levels.

- **Non-zero Movement Cost:** The Restricted Tree-edit Distance [JN02] attempts to identify repeating sub-branches, probably generated from the schema. However, it can still contribute undesired while significant distance.

These observations demonstrate the need and motivate us to define a new measure to determine structural similarity without the above deficiencies.

## 2.4 Conclusion

Document similarity lies at the heart of automated processing for XML documents for similarity clustering and proximate querying. Previous techniques by comparing the document contents are insufficient when applying to XML documents. XML document is of semi-structured nature where the document structure contains semantic information. Moreover, XML documents with the same content might not describe the same real world objects, such as, two <TITLE> elements with text "Harry Potter” nested under elements <BOOK> and <MOVIE> from two documents respectively. Therefore, there is a need to incorporate the structural semantic in the analysis of the document similarity.

In this chapter, we review the previous proposals of structural similarity measures for XML documents, including *Tree-edit distance* [SZ89], *Fourier Transform* [FMM+02, FMM+05],
Tag Similarity [B04], PMC [BGM04, BGMT01] and Restricted Tree-edit Distance [JN02].

Our focus is on their capabilities in detecting documents of the similar types, however, a number of deficiencies are observed in the reviewed techniques, summarized in Table 2.2.

<table>
<thead>
<tr>
<th>Similarity Measure</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree-edit Distance</td>
<td>Cater hierarchical</td>
<td>Slow</td>
</tr>
<tr>
<td></td>
<td>Directly applicable to XML documents</td>
<td>“Absolute” document tree comparison</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distances not comparable</td>
</tr>
<tr>
<td>Fourier Transform</td>
<td>Fast</td>
<td>Preprocessing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Absolute” document tree comparison</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distances not comparable</td>
</tr>
<tr>
<td>Tag Similarity</td>
<td>Simple and Fast</td>
<td>Not cater hierarchical relationship</td>
</tr>
<tr>
<td></td>
<td>Similarity comparable</td>
<td></td>
</tr>
<tr>
<td>Restricted Tree Distance</td>
<td>Lower movement costs to repeating substructures</td>
<td>Slow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distances not comparable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-zero costs to repeating substructures</td>
</tr>
<tr>
<td>PMC Similarity</td>
<td>Fast</td>
<td>Not direct XML comparison</td>
</tr>
<tr>
<td></td>
<td>Cater document category</td>
<td>Comparison by same levels only</td>
</tr>
<tr>
<td></td>
<td>Cater level of importance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Similarity comparable</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 Summary of Pros & Cons for Similarity Measures from Section 2.2

While there are a number of alternative techniques proposed [CLMY04, GK03, GLM+05, HK04, KTY05], they still fail to give good solution to some of the aforementioned problems.

Our claim is that new research solutions are needed for the problem of detecting structural similarity for XML documents. In the subsequent chapters, we introduce new structural similarity measures and evaluation algorithms to detect XML documents of similar types,
and demonstrate its qualitative advantages as well as its efficiency. We also present the applications of these similarity measures and algorithms to document indexing for proximate querying and to the similarity clustering.
CHAPTER 3

XML Structural Similarity against Schema

3.1 Introduction

In this chapter, we define the node level similarity of elements against a schema using the edit distance. The measure is firstly defined at node level against the corresponding declaration in the schema using the normalized edit distance. We transform the problem of node conformance checking against schema into the string edit distance calculation problem between strings and regular expressions. We propose a fast algorithm of linear complexity which is an enhanced version of the algorithm applied in the regular expression matching problem. Then, we define the document similarity by aggregating the node similarity using a weighting scheme based on nodes’ sub-document sizes. We illustrate that the choice of schema languages, DTD or XML Schema, do not constitute major difference in the definition. We also extend the definition to XML schemas and propose a structural similarity measure between XML documents and XML schemas. This approach takes into consideration the element order and the element similarity irrespective of the level. The idea is natural, easy to understand and quickly to calculate, which makes it useful in both document indexing and clustering.
3.2 Node Similarity

Our idea is originated from the observation; if two composite objects are similar, they are also similar locally. In the context of structural similarity, we consider the similarity at element, which is local in nature.

3.2.1 Partial Conformance

The main purpose of DTD is to ensure the validity of the XML documents. What’s meant by a valid XML document? It means that every element (and attribute) in the XML document are present and conform to the structure of the hierarchical specification in the DTD. In order to make the discussion concise and clear, we focus on the elements only. The treatment of attribute is very similar so that we skip this part. Figure 3.1 shows an example of DTD, the corresponding valid XML document and the tree representation of the XML document.

Pick the element deal with id “WK0612” from the XML document as example, it has the following child element list (id, customer, maturity, schedule, schedule) which matches the corresponding declaration of element deal in the DTD: (id, customer?, maturity, (amount|schedule*)). Note that there is no ambiguity in which element declaration to refer to (if it ever exists) because each element is uniquely declared in the DTD. However, this does not imply that the child element, say schedule, will also conform to the DTD, which demands further validation against its declaration in DTD. It is necessary to go through the DTD validation process for every element appearing in the XML document.
Figure 3.1 Example of DTD (top left), XML document (top right) and the tree representation of the XML document (bottom)

Definition 3.1 If every element of an XML document $X$ conforms to a DTD $D$, then we denote it as $X$ matches $D$. Then it is obviously true if $X$ is a valid with respect to $D$. 
We can do label substitutions, say \( a=\text{deal}, \ b=\text{id}, \ c=\text{customer}, \ d=\text{maturity}, \ e=\text{schedule}, \ f=\text{amount} \), then checking of the id “WK0612” can be viewed as checking whether the string “bcdee” matches the regular expression “bc?d(f|e*)”. As such, matching the XML elements to the DTD can be viewed as matching the corresponding child element strings with their regular expressions in DTD.

In the previous discussion, we only consider whether an XML is valid or not. However, even a document does not conform to a DTD, we are still interested in how close it is to a valid document, i.e. the partial conformance. For any element \( e \) in a document \( X \), which is not necessarily valid with respect to a DTD \( D \), it will fall into one of the three cases: (1) \( e \) matches \( D \); (2) \( e \) exists in \( D \) but does not matches \( D \); (3) \( e \) does not exist in \( D \). If we can quantify the above cases, it can represent the node level structural similarity between \( X \) and \( D \), i.e. degree of matching at element level. Here, we will only consider the cases (1) and (2) above while case (3) will be tackled later.

In both cases (1) and (2), there exists regular expression \( r \) from \( D \) corresponding to each element \( e \) from \( X \). We can make use of the edit distance to determine the distance between \( s \) and \( r \). Edit distance is the metric to determine the distance between two strings; any string can be transformed into any other string by a combination of three basic operations; insertion, deletion and substitution. The edit distance is defined as the minimum number of basic operations for such a transformation. For the strings \( abcdeh \) and \( aaacde \), it can be shown that the three operations: (1) substitute \( b \) into \( a \) (\( aaacdeh \)); (2) insert \( a \) before \( c \) (\( aaacdeh \)); (3) delete \( h \) (\( aaacdeh \)), is a minimum transformation and hence the edit distance is 3. With this metric, we can easily adopt for strings and regular expressions.
**Definition 3.2** Suppose $EDIST[s_1, s_2]$ as the edit distance between two strings $s_1$ and $s_2$. Then define the edit distance between a string $s$ and a regular expression $r$, $EDIST[s, r]$, as $\min \{EDIST[s, s_r] : s_r \text{ matches } r\}$. If $e$ is an element in an XML document and there exists the corresponding declaration in the DTD, we denote $EDIST[e]$ as $EDIST[s_e, r_e]$ where $s_e$ is the child element string of $e$ and $r_e$ is the regular expression representing the declaration of $e$ in the DTD.

For example, $EDIST[aa, a*b] = \min\{EDIST[aa, b], EDIST[aa, ab], EDIST[aa, aab], EDIST[aa, aaab], \ldots\}$, which is equal to one, from either $EDIST[aa, ab]$ or $EDIST[aa, aab]$. Of course, the determination of edit distance becomes more complicated if the string and the regular expression are not simple.

**3.2.2 Abstract Schema**

We study, in previous section, the partial conformance of XML documents against DTD, which is considered as the specification of element declarations for the conforming XML documents. We observe that the following features of DTD are applied: 1) every element in the DTD is uniquely defined; 2) the unique declaration can be transformed into regular expression without ambiguity; 3) there is a unique “root” element resolvable into the data element, say #PCDATA, by recursive regular expression substitutions. We observe that these above structural features of DTD are not necessarily bound in the DTD format. Therefore, it comes with the idea to extract and abstract these features and introduce an abstract specification sufficient to represent the structural aspects of documents.
Definition 3.3 For a finite set of elements $E$, we define an abstract schema as a set of declarations $\{R_i\}$ where $R_i$ is either in form of 1) $a \in E = \{\text{regular expression of elements } x \in E\}$ or 2) $a \in E = \#$, where $\#$ represents any data content. In addition, an abstract schema has the following properties: 1) the set of declaring elements is equal to $E$; 2) each declaring element is uniquely declared in $\{R_i\}$; 3) a unique “root” declaring element which does not appear on any regular expression in $\{R_i\}$; 4) starting from the “root” element, it can be resolvable into the $\#$ elements by a series of recursive regular expression submissions of using all the $\{R_i\}$.

$$a = bc+de? \quad \text{where}$$

\[
\begin{align*}
a &= "book" \\
b &= "title" \\
c &= "author" \\
d &= "publisher" \\
e &= "ISDN" \\
\# &= \text{content value}
\end{align*}
\]

Figure 3.2 Examples of DTD (left) and Abstract Schema (right)

In Figure 3.2, we see that the DTD can be transformed into an abstract schema. This observation is actually the general case, following from the definition. Hence it is natural to extend the definition of XML conformity to abstract schema in Definition 3.3.

Definition 3.3 If every element of a document $X$ is defined (uniquely) in an abstract schema $S$ and matches the corresponding declaration, in $S$, viewed as regular expression, we denote this as “$X$ conforms to or matches $S$”.
Following from Definition 3.3, we see that $X$ conforms to $S$ if and only if $X$ conforms to $D$ for all XML document $X$, and an abstract schema $S$ is transformed from a DTD $D$. This demonstrates that abstract schema is sufficient to represent the DTD, in terms of document structure aspects. We will further illustrate, in later section, that abstract schema is also sufficient to represent the structural aspects XML Schema in similar fashion. Therefore, without loss of generality, we make use of abstract schema in all the subsequent discussions of structural similarity. We also call an abstract schema simply a schema from now on if there is no ambiguity.

3.2.3 Approximate Regular Expression Matching

The problem of matching a string $s$ with a regular expression $r$ allowing errors has been widely studied, which is called the Approximate Regular Expression Matching problem [DH80, M92, MM89, MMW95, N01]. The problem statement is to find out all the substrings $s_i$ of $s$ such that $EDIST[s_i, r] \leq d$, for some given error $d$. The closest problem can be found in [MMW95] where it defines a function $E[]$ as the $\min \{EDIST[s_i, r] : s_i$ is a substring of $s$ ended at the last character of $s\}$. However, $E[]$ is different from $EDIST[]$ that $E[]$ includes all substrings of $s$ ended at the last character of $s$ in the string edit distance while $EDIST[]$ function is computed for the whole string $s$.

In this section, we present the recurrence formula with appropriate boundary conditions for the evaluation of $EDIST[]$ function, as an extension of the recurrence formula for the $E[]$ function from using the Thompson’s construction [MM89, MMW95]. For a regular expression $r$ and an automaton $M$, define
\[ Pr e(i) \] as the set of predecessors of the node \( i \) in \( M \)

\[ Pr e(i) \] as the subset of \( Pre(i) \) excluding back edges

where \( i \) is the topological numbering of nodes in \( M \).

**Definition 3.4** For any string \( s = s_1 s_2 \ldots s_n \), \( E[i, j] \) is defined as the minimum edit distance between all strings that can reach node \( i \) in \( M \) and all substrings of \( s \) ended at \( s_j \). If \( I \) is a set of nodes in \( M \), denote \( E[I, j] \) as \( \min \{ E[i, j] \text{ for all } i \in I \} \).

**Definition 3.5** \( E[i, j] \) must be equal to \( E[k, j-1] \) for some \( k \) (\( \leq i \)) plus the cost of editing a string \( w \) into \( s_j \) where \( w \) can range over all the strings on paths from \( k+1 \) to \( i \) in \( M \). Define \( E'[i, j] \) as the best edit distance over all strings \( w \) that do not take across a back edge in the automaton \( M \).

In [MMW95], it makes use of recurrence to calculate \( E[I] \). For any node \( i \) in \( M \), the last word \( w \) may contain nodes coming from later node through some back edges, which is still not yet evaluated at that moment. Therefore, it first defines and calculates \( E'[I] \), which assumes \( w \) will not pass any back edge. By considering different cases of recurrence, the recurrence formula below is adopted from [MMW95].
For $j \in [0,n]$, $E[\theta, j] = 0$

For $i \neq \theta$, $E[i, 0] = \begin{cases} 
\min E[\text{Pre}(i), 0] + 1 & \text{if } i \text{ is a non- } \varepsilon \text{ node} \\
\min E[\text{Pre}(i), 0] & \text{if } i \text{ is an } \varepsilon \text{ node}
\end{cases}$

For $j \in [1,n]$ and $i \neq \theta$,

\[
E'[i, j] = \begin{cases} 
\min \left( E[i, j-1] + 1, E[\text{Pre}(i), j-1] \right) + \begin{cases} 
0 & \text{if } r_i = s_j \\
1 & \text{otherwise}
\end{cases} \\
E[\text{Pre}(i), j]
\end{cases}
\]

\[
E[i, j] = \min \left( E'[i, j], \min \left( E'[\text{Pre}(i), j-1], E[\text{Pre}(i), j-1] \right) + \begin{cases} 
0 & \text{if } i \text{ is a non- } \varepsilon \text{ node} \\
1 & \text{if } i \text{ is an } \varepsilon \text{ node}
\end{cases} \right)
\]

where $\theta$ and $\varepsilon$ are the initial and the transition states in the automation $M$ respectively.

Remember that the difference between $E[]$ and $EDIST[]$ is whether we consider substrings in the distance definition. However, the major arguments in [MMW95] did not make any difference if we consider the whole string rather than that all the substrings ended at the last character. At a closer look at the formula above, the only difference comes from the initial condition; in $E[]$ case, $E[\theta, j] = 0$ for all $j$ as arisen from empty substring. For our $EDIST[]$ case, $EDIST[\theta, j] = j$ for $j$ between 0 and $n$.

This constitutes the recurrence formula for our $EDIST[]$ function, which runs at the same efficiency as the original algorithm.
3.2.4 Distance Normalization

With the distance function $EDIST[]$, we can determine how far an element in an XML document is from its declaration in the schema. However, the edit distances for different elements in the same document may not be comparable; for example, we have two elements $a, b$ having child strings “c” and “efghijklmn” and their corresponding regular expressions “$y*$”; “$z*$” respectively from a schema.

Obviously, $EDIST[a] = EDIST[c, y^*] = EDIST[c, empty string] = 1$ while $EDIST[b] = EDIST[efghijklmn, z^*] = EDIST[efghijklmn, empty string] = 10$. Intuitively, both elements $a$ and $b$ are simply totally different from the corresponding declarations in the schema but the edit distances show very different numbers. That arises the need to normalize the edit distances making the distances comparable.

For a string $s$ and a regular expression $r$, we denote $len(s)$ as the length of string $s$ and $minlen(r)$ as $\min\{len(s_r): s_r \text{ matches } r\}$. Then the maximum possible $EDIST[s, r]$ is $\max\{\text{len}(s), \text{minlen}(r)\}$ because $s$ can always be converted into a string $s_r$ matching $r$ by inserting (or deleting) the extra length of the longer side and substituting the characters of the common length. We then normalize the edit distances by using this maximum possible distance as the denominator.

**Definition 3.6** Define $smaSim[s, r]$, the structural similarity function between a string $s$ and a regular expression $r$, as $1 - \left( \frac{EDIST[s, r]}{\max(len(s), \text{minlen}(r))} \right)$. 
If \( s \) represents a child element string of an element \( e \) in an XML document while \( r \) represents the corresponding declaration in schema, then we define \( smaSim[e] \) as \( smaSim[s, r] \). Obviously, \( smaSim[e] \) is a number between zero and one inclusively. Intuitively, similarity of score one means totally match while score zero means totally mismatch.

The \( smaSim[] \) function provides a mean to quantify the node level structural similarity for the elements in an XML document with a schema. In Section 3.2.1, we have not yet tackled the case (3) of those elements \( e \) existing in the document while not in the schema. Intuitively, it is simply a total mismatch with the schema and we can assign a value zero to the \( smaSim[] \) function making the function well-defined in all cases. Then it is obvious that \( smaSim[e] = 1 \) for all \( e \) if \( X \) is a valid document.

### 3.3 Document Similarity

Our goal is to consider the structural similarity between XML documents and different schemas. We discuss in this section on how to promote the node similarity into document similarity.

#### 3.3.1 Artificial Root

In the previous section, it stated that “\( smaSim[e] = 1 \) for all \( e \) if \( X \) is a valid document”; however, the sufficient condition statement is not true. Consider the following XML
document against the schema (DTD) described in Figure 3.1. All the elements in the
document match the declarations in the schema.

```xml
<schedule>
  <date>16OCT2002</date>
  <amount>150000</amount>
</schedule>
```

However, the document does not conform to the schema because a valid document should have the root element `<portfolio>` instead of `<schedule>`. But why did not the matching check (by `smaSim[]`) identify that? It is because the root element `<schedule>` has no parent element so that it did not fall into any matching check. In order to force the root element into the checking, we can add an artificial root, say $R$, to both the document and the schema with the original roots as their child nodes. In other words, we effectively add an element $R$ having the child element string `<schedule>`, which corresponds to the regular expression `<portfolio>`. Then we get $smaSim[R] = 0$ because the `<schedule>` is totally different from `<portfolio>`, which successfully detected the invalidity.

We called such XML documents and the schemas as rooted, it is obvious that the artificial root will not affect either the values of the `smaSim[]` function or the validity of the original documents. That is, a rooted XML document $X$ is valid with respect to a rooted schema if and only if $smaSim[e] = 1$ for all elements $e$ in $X$. From now on, all the XML documents and schemas assume rooted unless otherwise specified.
3.3.2 Node Weighting

Intuitively, different elements should have different level of importance in an XML document. In other words, the bigger the sub-tree of an element in a document tree, the greater the importance the element should be.

**Definition 3.7** For any element $e$ in an XML document, $\text{Weight}[e]$ is defined as the size of the sub-tree rooted at $e$, excluding $e$ itself. As a result, all the leaf nodes (i.e. actual data) have zero weights.

Figure 3.3 shows the element weights for the XML example in Figure 3.1, which has total weight (for all elements) of 99.

![Figure 3.3 Element Weights (left) and the weight distribution (right)](image)

We can make use of the element weights to define a weighting scheme. As a consequence, the greater the substructure of an element, the greater the weighting percentage. Then we
can gather node level similarity information to give a document level view, which takes care of the semantic meaning for XML documents.

3.3.3 Structural Similarity Score

Now, we can give the document level definition of structural similarity.

**Definition 3.8** We define the structural similarity of an XML document $X$ against a schema $S$ as the weighted sum of node similarity.

$$smaSim[X, S] = \frac{\sum_{e \in X} Weight[e] \times smaSim[e]}{\sum_{e \in X} Weight[e]}$$

From Definition 3.8, we see that $X$ conforms $S$ if and only if $smaSim[X, S] = 1$. On the other extreme case, say $X$ has no common element with $S$, $smaSim[X, S] = 0$. Figure 3.4 shows an example of an XML document and a schema, and illustrates the similarity function evaluation.
From Figure 3.4, we can calculate the similarity as

\[
\text{smaSim}[XML2, SMA2] = \frac{8|x_a| + 7\times \frac{2}{3} + 2 \times 0 + 1 \times 1_k + 1 \times 1_k + 1 \times 1_k}{8|x_a| + 7 + 2 + 1 + 1 + 1} = 0.78
\]

The definition of the similarity function is natural and hence the evaluation is correspondingly straightforward. The following are the main steps for the calculation: (1) Calculate the weights of every element in the document (hence the total weight); (2) Calculate edit distances and the maximum possible edit distances of every element with their declarations in the schema; (3) Calculate the weighted sum.

Step (1) can be done by traversing the XML document tree once according to the postorder numbering. It is due to the property that the postorder numbering always goes through all
the child nodes before parent nodes for every sub-tree in a tree. The weights can be obtained by accumulating the weights of child nodes in every sub-tree.

For step (2), we can pre-build the automata for every declaration in the schema indexed by the declaring elements, as the same automata maybe referred by many instances of the elements in the document. Then the edit distances can be calculated by the recurrence formula shown in the previous section. While the maximum possible edit distances can be much easier to calculate by counting both the lengths of the strings and the regular expressions.

By gathering the results in step (1) and (2), step (3) can be computed accordingly. The time complexity in steps (2) dominates the other two steps. The time complexity of the overall algorithm is $O(R_SN_X)$, where $R_S$ is longest $\text{minlen}(r)$ among all $r$ in $S$ and $N_X$ is the size of $X$, which is fast in practice. The algorithm, called $\text{SSIMCALC}$, is shown in Figure 3.5.
Figure 3.5 Evaluation Algorithm SSIMCALS for the smaSim[] Function

3.4 Extendibility to XML Schemas

In previous sections, we focus on the structural comparison of XML documents against the abstract schemas, which can be generated from the DTDs. XML Schema is designed as the succession of DTD to describe the XML document structures, it is itself an XML document specified by the XML Schema Definition (XSD) which is both richer and more powerful than DTD. The major differences of XML Schema over the DTD are: 1) XML Schema is itself an XML document; 2) XML Schema supports many more data types; 3) XML Schema language can be more flexible in specifying allowed number of occurrences for elements; and 4) Namespace is supported in XML Schema. While there are a number of differences between the two schema languages, only point (3) above is related to the
structural aspect. In the analysis of structural similarity, we focus on the structural part of schemas and XML documents only. Other differences do not related to the structural part are not of concern. To broaden the applicability of the similarity measure, we see in this section that the structural similarity definition can be extended to the XML schema.

Although XML schema can contain richer schema declarations comparing to DTD, they share and specify a similar subset of structural information. We believe that the abstract schema defined previously has been sufficient to capture the structural information from the XML Schema. Therefore, the idea is to map an XML schema into an abstract schema and then all the structural similarity definitions follow immediately.

```xml
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"
                   minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="publisher" type="xsd:string"/>
      <xsd:element name="ISDN" type="xsd:string"
                   minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

**Figure 3.6 An Example of XML Schema**

Figure 3.6 is an example of XML schema. It specifies XML document structures compatible to the DTD and also the abstract schema shown in Figures 3.2. The `<book>` element contains ordered sequence of the elements `<title>`, `<author>`, `<publisher>` and
The `<title>` and `<publisher>` elements are mandatory and must occur one time. The `<author>` and `<ISDN>` can occur more than one time (minOccurs="1", maxOccurs="unbounded") and any time (minOccurs="0", maxOccurs="unbounded") respectively. Intuitively, they are the same effect as the “+” and “*” regular operators respectively. Therefore, we can write the declaration of the `<book>` element as a regular expression: \(ab+cd*\) where \(a = \text{<title>}, b = \text{<author>}, c = \text{<publisher> and } d = \text{<ISDN>}\).

In general, the combination of minOccurs and maxOccurs are more flexible than the regular operators “?”, “+” and “*”. We can apply the following logical rules to map the value pair combinations into the regular operators, and we can successfully transform an XML schema into an abstract schema.

1) No operator is appended for elements with minOccurs = “1” and maxOccurs = “1”
2) Operator “+” is appended for elements with minOccurs = “1” and maxOccurs > “1”
3) Operator “?” is appended for elements with minOccurs = “0” and maxOccurs = “1”
4) Operator “*” is appended for elements with minOccurs ≠ “1” and maxOccurs > “1”

Although the rules narrow down the all combinations of minOccurs and maxOccurs into a few regular operators, Alternatively, we can extend the regular expressions and hence the abstract schema by defining new regular operators “\(\{\text{min}, \text{max}\}\)” representing the corresponding minOccurs and maxOccurs counts defined in the XML schema. It can be shown that the definitions and arguments of structural similarity can be applied to this generalized version of abstract schema.
3.5 Experiment Results

In this section, we evaluate the correctness of the proposed smaSim measure using both synthetic and real data. We also demonstrate that smaSim outperforms the previously proposed techniques: the Tag Similarity (tagSim) and the PMC measure [BGM04]. All the experiments were run on a 3GHz Pentium IV processor with 4GB memory running Linux. The algorithms were written in Java using the DOM libraries [TA01, W3C98b].

3.5.1 Evaluation Metrics

A key objective of the experiments is to demonstrate the correctness and effectiveness of the measures. A similarity measure assigns a score from a predefined real number range to quantify the structural conformity. Without loss of generality, we assume (or transform if necessary) that the scores falling into the range of zero and one for comparison purpose. Ideally, a document conforming to a DTD should always give a score of one while a document has no common tag (and hence no common structure) against a DTD should always give a zero score. In addition, the effectiveness can be measured as the capability to spread the intermediate conformity cases evenly between zero and one (not inclusive). For example, when two documents generated from the same DTD using an XML generator [XM01b] with the error-rate parameter set as 30% and 80% respectively, the score for the former document should be higher than the latter one. The error-rate specifies the amount of controlled structural differences for the randomly generated XML documents from the DTD. The structural differences come in two aspects: 1) elements appearing in the
document but not in the DTD; 2) the elements’ ancestor-descendant relationship is different from that specified in the DTD. In the experiments, we evaluate the accuracy of the measures by how well their scorings can segregate different expected level of structural conformity.

Tag Similarity (tagSim) is a simple measure defined as the percentage of tags from a document X appearing in the comparing DTD D.

$$tagSim(X, D) = \frac{|X \cap D|}{|X|}$$

where the $\cap$ is taken the intersection of elements in X and D, and the $||$ is taken as the number of elements.

To my best knowledge, PMC algorithm is the only previously proposed measure comparing the structural similarity of XML documents against DTD. It is computed based on the (plus, minus and common) node counts (see Section 2.2.5). The original PMC algorithm [BGM04] assumes a simplified scenario that elements do not appear in the DTD more than once so that matching against XML nodes can be decided locally. However, this assumption is too restrictive that cannot be applied to the both the experimental setup using either the synthetic and real DTDs. Therefore, we generalize the algorithm by escalating all possible combinations to decide the optimal PMC at upper level nodes.
3.5.2 Synthetic Data

Synthetic data is generated by an XML document generator [XM01b], which generates valid documents for a chosen DTD. An error rate can be configured so that generated documents will not fully conform but closely conform to the DTD, depending on the error rate specified. Four sets of 500 documents were generated with different error rates: (A) 0%, (B) 20%, (C) 40% and (D) 70%. Similarity measures of the four datasets are run against the chosen DTD and shown Tables 3.1, 3.2 and 3.3 respectively.

![Table 3.1 Similarity Scores for smaSim Measure (Synthetic Data)]

<table>
<thead>
<tr>
<th>Similarity Range</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 &lt;= smaSim &lt; 0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>86</td>
</tr>
<tr>
<td>0.2 &lt;= smaSim &lt; 0.4</td>
<td>0</td>
<td>8</td>
<td>64</td>
<td>234</td>
</tr>
<tr>
<td>0.4 &lt;= smaSim &lt; 0.6</td>
<td>0</td>
<td>16</td>
<td>131</td>
<td>166</td>
</tr>
<tr>
<td>0.6 &lt;= smaSim &lt; 0.8</td>
<td>0</td>
<td>98</td>
<td>280</td>
<td>14</td>
</tr>
<tr>
<td>0.8 &lt;= smaSim &lt; 1.0</td>
<td>0</td>
<td>386</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>smaSim = 1.0</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.1 Similarity Scores for smaSim Measure (Synthetic Data)

![Table 3.2 Similarity Scores for tagSim Measure (Synthetic Data)]

<table>
<thead>
<tr>
<th>Similarity Range</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 &lt;= smaSim &lt; 0.2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>0.2 &lt;= smaSim &lt; 0.4</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>168</td>
</tr>
<tr>
<td>0.4 &lt;= smaSim &lt; 0.6</td>
<td>0</td>
<td>12</td>
<td>37</td>
<td>184</td>
</tr>
<tr>
<td>0.6 &lt;= smaSim &lt; 0.8</td>
<td>0</td>
<td>42</td>
<td>173</td>
<td>132</td>
</tr>
<tr>
<td>0.8 &lt;= smaSim &lt; 1.0</td>
<td>0</td>
<td>190</td>
<td>260</td>
<td>0</td>
</tr>
<tr>
<td>smaSim = 1.0</td>
<td>500</td>
<td>256</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.2 Similarity Scores for tagSim Measure (Synthetic Data)
<table>
<thead>
<tr>
<th>Similarity Range</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 &lt;= smaSim &lt; 0.2</td>
<td>0</td>
<td>2</td>
<td>19</td>
<td>243</td>
</tr>
<tr>
<td>0.2 &lt;= smaSim &lt; 0.4</td>
<td>0</td>
<td>5</td>
<td>153</td>
<td>234</td>
</tr>
<tr>
<td>0.4 &lt;= smaSim &lt; 0.6</td>
<td>0</td>
<td>155</td>
<td>192</td>
<td>23</td>
</tr>
<tr>
<td>0.6 &lt;= smaSim &lt; 0.8</td>
<td>0</td>
<td>215</td>
<td>113</td>
<td>0</td>
</tr>
<tr>
<td>0.8 &lt;= smaSim &lt; 1.0</td>
<td>0</td>
<td>123</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>smaSim = 1.0</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

Table 3.3 Similarity Scores for PMC Measure (Synthetic Data)

We also plot the results into Figure 3.7, 3.8 and 3.9 respectively so that we can easily see the score distributions.

![Figure 3.7 Score Distribution for Our Proposed smaSim Measure](image)

In Figure 3.7, it verified that the $smaSim$ measure always has the value one for valid documents ($A$), while it is less than one for invalid documents ($B$, $C$ and $D$). Moreover, the
majority zones for data sets $A$, $B$, $C$ and $D$ are clearly segregated that demonstrates the correctness of the $SSIMCALC$ algorithm.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig38.png}
\caption{Score Distribution for $tagSim$ Measure}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig39.png}
\caption{Score Distribution for $PMC$ Measure}
\end{figure}
In Figure 3.8, some non-conforming documents from data sets (B) and (C) also have a score of one. It is because these documents contain all elements from the DTD while having different ancestor-descendant relationship that cannot be detected by the \textit{tagSim} measure. This causes the score distribution shifted towards the higher score side and the significant overlapping of the majority zones for data sets (A), (B) and (C) which implies that inaccuracy of the \textit{tagSim} measure in these scenarios.

In Figure 3.9, the \textit{PMC} measure can accurately identify valid documents as shown in the majority zone for data set (A). However, there is overlapping of majority zones between data sets (B) and (C), due to the fact that \textit{PMC} measure is, by design, only compare elements of the same levels in the document and the DTD instance trees. This causes similar sub-branches under unmatched elements are ignored in the comparison and hence the similarity scores are lower than intuitively expected.

\textbf{3.5.3 Real Data}

Real data are downloaded from the Internet to evaluate the effectiveness of the measure. 600 XML documents are retrieved from a web site [ACM01], generated by a common DTD. Twenty documents are selected randomly from the population to generate a DTD based on the XTRACT method [GGR+00]. It happens that the generated DTD contains all the tag labels from the original DTD with minor structural differences. We compare our proposed \textit{smaSim} measure to the \textit{tagSim} and \textit{PMC} measures by computing their similarity scores of all the 600 documents against both the original (\textit{ORG}) and generated (\textit{GEN}) DTD. Table 3.4 illustrates the corresponding similarity scores.
Table 3.4 Comparisons of Similarity Scores against the DTDs: \textit{ORG} and \textit{GEN}

For the original DTD (\textit{ORG}), all the three evaluating measures successfully detect the validity of documents by giving a similarity score of one. For the generated DTD (\textit{GEN}), they show different distributions of similarity scores in Figure 3.10.

![Figure 3.10 Comparisons of Similarity Score Distributions against Generated DTD (\textit{GEN})](image-url)
First, tagSim measure fails to identify the structural differences of nearly half of the evaluating documents, which do not conform to the DTD GEN, and gives a score of one to every document because GEN contains all tag labels. Comparing the smaSim measure to the PMC measure, they both detect the correct subset of valid documents against the DTD GEN. However, for invalid but structurally similar documents, the smaSim measure demonstrates a condensed distribution of scores towards to higher score range (0.8 to 1.0) while the PMC measure shows more diverse distribution with majority falling into the range of (0.6 to 1.0). It is caused by the design deficiency of PMC measure that elements are compared level-by-level between the document and the DTD instance trees.

### 3.6 Conclusion

In this chapter, we propose the structural similarity measure smaSim to compare XML documents against schemas, which is applicable without loss of generality to DTDs and XML Schema. The measure gives the indication of how much a document conforms to the types of document set that the comparing represented by a schema. They are easy to understand and able to give the value of one, indicating the ‘total match’, for documents generated from the schema. We also present the SSIMCALC algorithm, which has good computational performance. With carefully chosen schema among an arbitrary set of XML documents, this metric can be used for XML clustering, classification and indexing, which has laid the foundation of our work.
CHAPTER 4

Similarity Measure for XML Structures

4.1 Introduction

In this chapter, we propose a structural similarity measure comparing XML documents directly. The idea is to generalize from documents the implied schemas and apply the XML-to-Schema measure defined in Chapter 3 to the documents against the implied schemas. The XML measure carries forward the associated well-properties of the XML-to-Schema measure while eliminated the prerequisite of a schema. We demonstrate the effectiveness of the measure by means of experiments based on both real and synthetic data. We also discuss the idea to extend the similarity measure to compare both document structure and content.

4.2 Implied Schema Generation

The previous chapter defined a way to measure the structural similarity between XML documents and schema. However, when we collect a set of XML documents, say from the Internet, it is not easy to get or identify any “good” reference schema as the documents may
come from very different sources. Here, we release the dependency on any reference schema by introducing a mechanism to generate based on the comparing XML documents. For a document node, the occurrence of ordered child nodes associated can be viewed as a string describing the parent nodes. A typical schema extraction algorithm takes this representative child string, through a series of steps, including the generalization, factorization and optimization, to generate a containing regular expression:

*Generalization*: given a set of candidate strings $S$, they can be generalized, such as $abbb \rightarrow ab^*$, by adopting various heuristics into a generalized string set $S_G$.

*Factorization*: various subsets of $(S \cup S_G)$ are then chosen to factorize, such as $(ab^*)(cb^*) \rightarrow (a|c)b^*$, into the candidate schemas.

*Optimization*: the “optimal” schema is then chosen using optimization rules, such as MDL used in [GGR+03].

Note that factorization only makes a regular expression more compact and does not change its generated string set, and hence this does not affect the edit distance computation in the $smaSim[]$ function and consequently no accuracy is being tradeoff. With a set of generalized expressions, we concatenate them using the “|” operators giving regular expressions where no optimization is done. Without factorization and optimization steps, the representation of the generated regular expressions may not be compact but it preserves the necessary accuracy in $smaSim[]$ function evaluation. Since a generated schema is only used as intermediate basis in the similarity score calculation, this simple but accurate
mechanism successfully reduces unnecessary time complexity in temporary schema generation.

For generalization, we adopt a simple approach as follows: for every node in an XML document tree, we look from their child node strings for some repeated patterns in which the repetition count is no less than a presumed *minimum contiguous repetition count (mcrc)*. Patterns are identified and generated by recursively scanning of the child strings while “+” operation is assumed given that at least one occurrence is observed. Figure 7 shows the resulting regular expressions with different mcrc accordingly.

\[
\text{mcrc}
\[
\begin{array}{c}
\text{abbcddddd} \\
\text{ab+c+d+} \\
\text{abbc}d+ \\
\text{abbcc}d+
\end{array}
\]

\textbf{Figure 4.1} Generalized Regular Expressions with Different mcrc

This method can be applied recursively to nested cases; it needs only to replace the observed patterns from previous iteration by new labels giving a new string for the next level of generation. An empirical recommendation to the mcrc value is two due to the observation that most repeating elements in an XML documents are generated from the regular expression operators “+” and “*” in the schema. For example, with a repetition count of two, we have

"bcddbcddddd"

→ "bcDbcD" where "D=d+"
→ "E+" where "E=bcD" and "D=d+
→ "(bcd)+"

It is normal that the same node label appears more than one times in a document while the corresponding declarations in schema are unique. The “|” operation is applied among all the generalized regular expressions for the same node label giving a unique declaration for the node label in the generated schema.

We define the *Implied Schema*, denoted by $Sma[X]$, as the unique schema generated by the following generalization mechanism:

1) Represent each parent node as the child strings or equal to “#” if the parent node contains content value.

2) Generalize each child string into a regular expression.

3) Concatenate all the regular expressions for the same parent node making the definition unique.

Given that X always conforms to $Sma[X]$, $smaSim[X, Sma[X]]$ is always equal to one.

---

**Figure 4.2** Example of Schema Generation Process
For a string $s$, the time complexity of string generalization is determined by the number of
times of scanning $s$. In the worst case scenario, the time complexity is $O(len(s)^3)$. For the
implied schema generation from an XML document $X$, we need to generalize every child
node strings and hence the time complexity is $O\left(\sum_{\text{child string } s_i \text{ in } X} (\text{len}(s_i)^3)\right) \leq O(|X|^3)$.

### 4.3 XML Structural Similarity

Given two XML documents $X$ and $Y$, we are interested in whether they belong to the same
document class, if not, how much they close to each other. We can generate the implied
schemas of $X$ and $Y$, which can be viewed as the document classes with $X$ and $Y$ as class
representatives. When document $X$ is compared against the implied schema of $Y$ using the
similarity function $smaSim[]$, it effectively compares how much the document $X$ closes to
the document class with representative document $Y$, given by the score $smaSim[X, Sma[Y]]$.
However, this score is not symmetric between $X$ and $Y$, and hence the reverse comparison
is done. Hence, we can define the structural similarity in Definition 4.2.

**Definition 4.2** Define the structural similarity function $xmlSim[X, Y]$ as $(smaSim[X, Sma[Y]] + smaSim[Y, Sma[X]]) / 2$ for any two XML documents $X$ and $Y$. The similarity score is always a number between zero and one inclusively. Intuitively, the higher the similarity score is, the higher the structural similarity they share. As $xmlSim[X, Sma[X]] = 1$ for any XML document $X$, $xmlSim[X, X] = 1$. That is, a “total match” can always be detected by the $xmlSim[]$ function for identical documents.
Figure 4.3 shows two XML documents (XML$_1$ and XML$_2$) and their implied schemas (SMA$_1$ and SMA$_2$). Following the calculation flow in Chapter 3, it can be shown that $\text{smaSim}[\text{XML}_1, \text{SMA}_2] = 0.76$ and $\text{smaSim}[\text{XML}_2, \text{SMA}_1] = 0.81$, and hence $\text{xmlSim}[\text{XML}_1, \text{XML}_2] = \frac{0.76 + 0.81}{2} = 0.785$.

![Figure 4.3 Two sample XML documents with their implied schemas](image)

Similarity issue is a hard topic in the sense that each proposed measure has tailored for particular application(s) and would not be good for some others. In this research, our aim is to quantify the structural similarity on XML data per document category, in which documents in the same or close document category refer to documents generated from the same or similar schemas. With this objective in mind, we have described in Section 2.3 the expected similarity results represented by the “pseudo” $\text{refSim}()$ measure.

In order to demonstrate the effectiveness of the $\text{xmlSim}()$ measure, we make use the XML documents $\text{XML}_a, \text{XML}_b, \text{XML}_c$ and $\text{XML}_d$ from Figure 2.9 (which were used to show the deficiencies of previously proposed measures in Table 2.1) and calculate the structural similarities of $\text{XML}_a$ with the other three documents respectively. We also compare the
results against the reference results discussed in Section 2.3. In Table 4.1, we see that the
\( \text{xmlSim}[\text{XML}_b, \text{XML}_a] = 1.00 \), \( \text{xmlSim}[\text{XML}_c, \text{XML}_a] = 0.81 \) and \( \text{xmlSim}[\text{XML}_d, \text{XML}_a] = 0.00 \) which successfully determine the ideal results of “perfectly matched”, “highly matched” and “rarely matched” respectively.

<table>
<thead>
<tr>
<th>xmlSim[]</th>
<th>XML(_b) vs XML(_a)</th>
<th>XML(_c) vs XML(_a)</th>
<th>XML(_d) vs XML(_a)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00</td>
<td>0.81</td>
<td>0.00</td>
</tr>
<tr>
<td>refSim()</td>
<td>Perfectly Matched</td>
<td>Highly Matched</td>
<td>Rarely Matched</td>
</tr>
</tbody>
</table>

Table 4.1 \( \text{xmlSim[]} \) vs \( \text{idealSim}() \) using examples from Figure 2.5

4.4 Experimental Evaluation

In our experiments, we compared the effectiveness and accuracy of \( \text{xmlSim()} \) against previously proposed measures: \( \text{PMC} \) measure [BGM04] and Jagadish measure [JN02]. We implemented all the algorithms using Java 1.5 using DOM libraries [W3C98b] and examined both synthetic and real XML documents. We ran experiments on a computer with a 3GHz Pentium IV processor with 4GB memory running Linux.

4.4.1 Evaluation Metrics

\( \text{PMC} \) algorithm is defined only between XML documents and schemas based on the (\( \text{plus}, \text{minus} \) and \( \text{common} \)) node count, so it cannot be used directly for comparison. By making use of the implied schema as proposed in this chapter, we extend the \( \text{PMC} \) algorithm to a measure to compare XML documents, i.e. \( \text{PmcSim}[X, Y] \) as \( (\text{PmcSim}[X, Sma[Y]] + \)
\( PmcSim[Y, Sm_X] / 2 \) for two documents \( X \) and \( Y \). Moreover, the original \( PMC \) algorithm assumes a simplified scenario that element labels do not appear in the schema more than once so that matching against XML nodes can be decided locally. However, this assumption is too restrictive and both our implied schema and many real schemas do not follow that. Therefore, we generalize the algorithm by escalating all possible combinations to decide the optimal PMC at upper level nodes.

Jagadish measure is defined to compare the tree edit distance among XML documents. It introduces the restricted edit operations to taking into account XML characteristics such as optional and repeated sub-documents, and is considered as an improved version of tree edit distance measure. However, the distance measure is not normalized and hence it gives no idea the level of similarity. Therefore we extend the Jagadish measure into a similarity measure with values in range \([0, 1]\) by normalizing it, i.e. \( JagadishSim[X, Y] \) as \( 1 - (JagadishDist[X, Y] / (|X| + |Y|)) \) for two documents \( X \) and \( Y \). The reason of using \(|X| + |Y|\) as denominator is due to the scenario that greatest possible distance can occur when two documents have non-overlapping node labels and one document with all descendent nodes being the children of the root node while another document with a single chain of nodes.

We evaluate the effectiveness of the measures by using multiple sets of documents with prior knowledge in the mutual structural relevance among them. The accuracy of the measures are determined by the extent of score distributions matched against the expected patterns that demonstrate the capacity of the measures to detect document categories.
4.4.2 Synthetic Data

Synthetic data valid for chosen schemas were generated using the XML Generator [XM01b]. Four schemas, in Figure 4.4, were selected to generate the testing documents.

<table>
<thead>
<tr>
<th>SCHEMA A</th>
<th>SCHEMA B</th>
<th>SCHEMA C</th>
<th>SCHEMA D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R = a^+ )</td>
<td>( R = k^+ )</td>
<td>( R = a^+ )</td>
<td>( X = x^+ )</td>
</tr>
<tr>
<td>( a = bc^*d^*e )</td>
<td>( k = bc^*d^*e )</td>
<td>( a = ed^*c^*b )</td>
<td>( a = # )</td>
</tr>
<tr>
<td>( b = f )</td>
<td>( b = f )</td>
<td>( b = d^+ )</td>
<td>( b = d^+ )</td>
</tr>
<tr>
<td>( c = g</td>
<td>h )</td>
<td>( c = g</td>
<td>h )</td>
</tr>
<tr>
<td>( d = # )</td>
<td>( d = # )</td>
<td>( d = # )</td>
<td>( d = # )</td>
</tr>
<tr>
<td>( e = # )</td>
<td>( e = # )</td>
<td>( e = # )</td>
<td>( w = # )</td>
</tr>
<tr>
<td>( f = # )</td>
<td>( f = # )</td>
<td>( f = # )</td>
<td>( x = a^*bc? (w</td>
</tr>
<tr>
<td>( g = # )</td>
<td>( g = # )</td>
<td>( g = # )</td>
<td>( y = z )</td>
</tr>
<tr>
<td>( h = # )</td>
<td>( h = # )</td>
<td>( h = # )</td>
<td>( z = # )</td>
</tr>
</tbody>
</table>

**Figure 4.4** Evaluation Schemas

With the Schema A as the reference schema, Schema B is chosen to have the same structure except that a high level node is different, namely nodes \( a \) and \( k \) respectively. Schema C is chosen to have a very similar structure as Schema A as well except that the element order of node \( a \) is reversed, namely \( bc^*d^*e \) and \( ed^*c^*b \). Schema D, instead, is chosen to be very different from Schema A where they have different nodes and structures. 1,200 documents were generated from the four schemas with each 300 per set. The structural similarity measures of the four sets are computed against the first set by using xmlSim(), PMC and Jagadish measures. The results are shown in the Figure 4.5 respectively.
Figure 4.5 Comparison Results of \textit{xmlSim()}, \textit{PMC} and Jagadish Measures for Synthetic Data (X-axis = Similarity Scores, Y-axis = % of Documents)

There are four sets of comparison results above, namely (a) AA, (b) AB, (c) AC and (d) AD. In (a), pairs of documents generated from the same schemas are compared; therefore very high similarity scores are expected intuitively. The scores of \textit{xmlSim()} have greater than 90% of comparisons falling into the top score range (0.8 – 1.0). The remaining ~10% of comparisons follow closely in the next lower score range (0.6 – 0.8) because some document instances do not hold some optional elements as a result of the “*” and “|” operators in Schema A. For example, when two documents pick different labels from the “|” operator, they will have different document structures. Then the implied schema generated from one document will not be valid for the comparing document. On the other
hand, only around 50% of the $PMC$ scores falling into the top score range implies that the $PMC$ measure cannot effectively recognize the highly similar documents. It is because the $PMC$ methodology ignores the entire sub-document when its parent node does not match against the implied schema. This deficiency becomes obvious in (b) when Schema B has only a single different node $k$ comparing to node $a$ in Schema A. Given the difference occurring in some high level in the structures, the documents generated from Schemas A and B are considered less similar and lower similarity scores are expected. Therefore $xmlSim()$ shows scores in $0.2 – 0.4$ ($\sim 60\%$) and $0.4 – 0.6$ ($\sim 40\%$) ranges. However, the $PMC$ measures show 100% zero scores and completely fail to recognize the underlying similarity of the two document sets. In (c), element order of node $a$ is revised in Schema C which defines a similar but different structure. This difference lowers the scores of $xmlSim()$; $0.8 – 1.0$ ($\sim 20\%$) and $0.6 – 0.8$ ($\sim 80\%$). However, the definition of $PMC$ is not sensitive to the element order and hence the same score distribution is obtained as in (a).

For Jagadish measure, the result in (a) apparently identifies highly similar documents as what $xmlSim()$ does. However, we see indifferent patterns in (b) and (c) as in (a) and this implies that the measure is not sensitive to documents with a few different nodes at high levels and the node sibling orders respectively. The Jagadish measure is designed to give lower distance among documents generated from the same schema but it can still accumulate significant distances for documents if they have large differences in terms of optional and repeating elements. This associated distance can outweigh that caused by the introduced differences in the schemas used in (b) and (c). Moreover, allowable re-labeling action makes the difference in high level nodes, in (b), only a one-step edit operation. On the other hand, the edit distance hardly reaches value of denominator as even two
documents of the similar tree structure but no common labels can only attain half of that by applying the re-labeling actions. This pushes the Jagadish similarity in general to higher scores and explained the patterns in (c).

In (d), both \textit{xmlSim()} and \textit{PMC} measure can recognize the significant differences in document structures and show zero scores for all the comparisons. However, Jagadish measure fails to identify this scenario that is also due to the large denominator issue described before.

The above comparison results show that \textit{xmlSim()} gives reasonable scores according to the characteristics of the documents, and \textit{xmlSim()} has outperformed the \textit{PMC} and Jagadish measures in terms of effectiveness and accuracy.

### 4.4.3 Real Data

Real data were downloaded from the Web to evaluate the effectiveness of the measure. Three sets of each twenty XML documents were downloaded from three sites respectively: Set E from [D99a], Set F from [D99b] and Set G from [ACM01]. The Sets E and F describe the university courses having relatively similar document structures; they both start with common nodes \textit{root} $\rightarrow$ \textit{course} $\rightarrow$ \textit{title, place}, etc. On the other hand, they also show specific nodes such as \textit{credit}, \textit{times}, \textit{limit}, etc. in Set E and \textit{units, time}, etc. in Set F. The Set G obviously describes something very different. Similarity scores of Set E against Sets F and G are computed using both the \textit{xmlSim()} and \textit{PMC} measures and the results, (e) EF and (f) EG, are shown in Figure 4.6.
Figure 4.6 Comparison results of \textit{xmlSim()}, \textit{PMC} and Jagadish measures for real data (X-axis = similarity scores, Y-axis = \% of documents)

In (e), \textit{xmlSim()} appears in the 0.6 – 0.8 score range while the \textit{PMC} measure appears in the 0.2 – 0.4 score range. \textit{xmlSim()} successfully recognizes that the Sets E and F are describing objects of similar categories while the \textit{PMC} measure shows only minor similarity. Even worse, Jagadish measure fails to detect a narrow score range that is caused by the detraction
of the optional and repeating elements chosen in the document instances. In (f), both $xmlSim()$ and $PMC$ measure can distinguish some obvious differences in document structures. Again, the large denominator issue makes the Jagadish measure shifted out of the small score range and hence fails to identify this scenario. These experiment results demonstrate the effectiveness of $xmlSim()$ in document clustering and classification.

4.5 Extendibility to Structure and Content Comparisons

We have presented an effective measure $xmlSim()$, which is capable to detect structurally similar XML documents. In some circumstances, we need also to examine and compare document contents in addition to document structures, that has been applied to the areas of relevance ranking \cite{AKM+05, BMSW04}, de-duplication \cite{CD03, NF05, W05} and document filtering \cite{AF00, KRM+05}, etc. for XML documents. In this section, we briefly describe the idea to extend the definition of $xmlSim()$ to compare both structures and contents of XML documents.

We illustrate, in this section, an example of duplication detection of real-world objects described by XML documents. In Figure 4.7, there are four XML documents and we compare $Doc1$ against $Cmp1$, $Cmp2$ and $Cmp3$ respectively to detect any of them actually being a duplicate. $Doc1$ represents the movie *Romeo & Juliet* which a modernizing version published in 1996, with *Leonardo DiCaprio* and *Claire Danes* as the chief actor and actress respectively. $Cmp1$ appears to be describing the same movie given that it has the same title, chief actor and actress, although it does not hold the publication *year* but the *director*. 
Instead, Cmp2 represents another movie of the same title that, however, was published sixty years earlier and of course acted by different artists. Cmp3 is a book version of the story written by the original author Shakespeare, republished in 1996.

![Comparison XML Documents with Contents](image)

**Figure 4.7** Comparison XML Documents with Contents

We need a similarity measure capable to identify Cmp1, but not Cmp2 or Cmp3, as a highly potential duplicate of Doc1. It is not sufficient only to compare the document structures, for instance, we can show that

\[
\text{xmlSim}(\text{Doc1}, \text{Cmp2}) = 1 > \text{xmlSim}(\text{Doc1}, \text{Cmp1}) > \text{xmlSim}(\text{Doc1}, \text{Cmp3})
\]

which is undesirable. We recall that the structural similarity measure \( \text{xmlSim()} \) is defined as the arithmetic sum of the \( \text{smaSim()} \) scores between the first document and the implied
schema generated by the comparing document, and vice versa. During the implied schema
generation process, we use a generic # symbol to represent all content values. If we retain
and include the content information in the implied schema, it can facilitate the content
comparison.

Figure 4.8 Example of Implied Content Schema Generation Process

For an XML document \( X \), we define the Implied Content Schema, denoted by \( C_{\text{ma}}[X] \), in
the same way as that of \( S_{\text{ma}}[X] \) with the exception of leaf tags. For each leaf tag \( a \) with
content values consisting of a sequence of words and/or numbers “\( v_1 \ v_2 \ldots v_n \)”, the schema
rule “\( a = v_1 | v_2 | \ldots | v_n \)” is added. Then we concatenate (by the “|” operator) the generated
regular expressions for leaf tags of the same label while any duplicate value is removed to
reduce the expression size, as illustrated in Figure 4.8.

The leaf nodes in content schema are represented as regular expressions. Hence we can
follow the steps in Section 3 to define node and document similarities. We can first define
the normalized node similarity \( csmaSim() \) between a document node and a content schema.
For a leaf node “\( a = u_1 \ u_2 \ldots u_m \)” and the associated declaration (if any) \( R : “a = v_1 | v_2 | \ldots | v_n \)”
in the content schema, we define \( csmaSim[a,R] = \frac{\left| \{ u_j \in \{ v_i \} \} \right|}{m} \). The document similarity
$csmaSim()$ can be defined by aggregating node similarity by a weighting scheme. Nevertheless, we need to make adjustment to the weighting scheme in Section 3.2.2 based on sub-tree sizes of document nodes. It is because the weighting scheme always assigns the lowest node weighting to the leaf nodes which is responsible for content comparisons. Therefore, the content weighting factor is introduced as a constant multiplier to increase the weighting of leaf nodes. This factor serves the purpose of balancing the importance between structure and content comparisons (while the choices of this factor are not included in this preliminary discussion). Therefore, we are now able to define the similarity measure $cxmlSim()$ comparing XML documents for both structure and content:

$$
cxmlSim[X, Y] = \frac{csmaSim[X, Csma[Y]] + csmaSim[Y, Csma[X]]}{2}$$

for two XML documents $X$ and $Y$.

By applying the $cxmlSim()$ measure to the examples in Figure 4.7, we can show that

$$xmlSim[Doc1, Cmp1] > xmlSim[Doc1, Cmp2] > xmlSim[Doc1, Cmp3]$$

which demonstrates its applicability to identify duplicate documents based on both structure and content.

### 4.6 Conclusion

In this chapter, we propose a measure $xmlSim()$ detecting the structural similarity directly among XML documents. It preserves the well-properties of XML-to-Schema measure
defined in Chapter 3. These experiment results have led us to think of how \textit{xmlSim}() can be applied to document clustering and classification. We also briefly discuss the idea to extend \textit{xmlSim}() and introduce the similarity measure \textit{cxmlSim}() to compare both document structure and content.
CHAPTER 5

Application – RRSi: Indexing XML Data for Proximity Twig Queries

5.1 Introduction

Indexing and querying XML documents using twig patterns has recently received a considerable amount of interest and a number of indexing techniques [BKS02, CLL05, JLWX03, KP05, MR04] are proposed to speed up the twig query processing of XML documents. In [BKS02], the proposed TwigStackXB algorithm is a stack-based pattern matching algorithm. It makes use of XB-trees to speed up twig query processing. In [MR04], the system PRIX for indexing XML documents is introduced to support twig query processing. It indexes XML documents using transformed Prüfer sequences to allow twig pattern matching. However, these two methods and most of the others that have been developed only support exact query answers.

Previous studies on proximate query answers have focused on querying both structure and content by ranking results to indicate relevance. In classical information retrieval methods, term frequency and inverse document frequency are used to calculate the ranks of query
results based on content matching. This approach has been extended to include structural information [BMSW04, CLZ04, MS02]. However, methods with this approach often generate a very large intermediate result set of which many results can turn out to be structurally irrelevant, especially when there are heterogeneous sources for XML documents. Although recent proposals attempt to incorporate structural information into content-based indexing and ranking [BMSW04, CLZ04, CMS02, FPWY03, LM01], none of them fully takes advantage of structural indexes to eliminate the structurally irrelevant data from proximity query processing.

In [GLP04], the TreeSketch synopsis has been proposed to capture the tree structure of underlying XML documents in order to produce approximate twig query answers. The idea is to allow a rapid ‘preview’ of query answers for the trade-off of result accuracy. In contrast, in our work, the RRSi facilitates fast proximate results with the aim of returning more complete ‘interesting’ documents without sacrificing result quality. Recently, the technique of query relaxation [AKM+05, CJ01, S02, SN00] is adopted in proximity query processing. Queries are relaxed and transformed into multiple proximate queries for exact query pattern matching. Ranks are determined from the proximity of the matching queries against the original query. However, none of the approaches suggest any indexing scheme to speed up the query processing.

In this chapter, we propose the RRSi, a novel structural index designed for twig query lookup supporting proximate query answers. For many heterogeneous sources of XML documents, it is particularly important to eliminate the structurally irrelevant answers which could be returned from content-based matching. This can take up significant
processing time and space before the answers can be pruned. In our design, the \textit{RRSi} is a hierarchical index on the \textit{rrs}, \textit{Relaxed Representative Structure}, which captures the key characteristics of the document structures representing the underlying structurally similarity of the documents. It returns the structurally similar documents for the submitted queries. Query searching involving the complete index branch of documents with significant structural differences is avoided and determined at the branch root. By analyzing the \textit{RRSi} structure, we propose an optimized version of the \textit{RRSi}, \textit{oRRSi}, which improves significantly in terms of both space and computational complexities. To the best of our knowledge, ours is the first work to propose the structural indexes on XML documents supporting proximity twig queries.

The rest of this chapter is organized as follows. Section 5.2 presents the motivating example. In Section 5.3, we define \textit{rrs} and \textit{emSim()} embedding structural similarity. Section 5.4 describes the index structures of the \textit{RRSi} and \textit{oRRSi} in detail. In Section 5.5, we demonstrate the effectiveness of the \textit{RRSi} and \textit{oRRSi} in supporting the proximity search through twig query patterns. Extensive experimental results are reported in Section 5.6. We conclude this chapter in Section 5.7.
5.2 Motivating Example

5.2.1 Structural Heterogeneity of XML

XML documents generated from heterogeneous sources often have different focuses; hence they will have data contents with different structures. As a result, documents describing similar information do not necessarily follow the same document structure. In Figure 5.1, the labeled trees represent two XML documents that store music album information including the album name, album singer/artist, and album song information. In Doc2, \(<\text{song}>\) nodes are grouped as child nodes under the \(<\text{tracks}>\) node which is then a child node of the root node \(<\text{album}>\); however in Doc1, \(<\text{song}>\) nodes are directly attached as child nodes under the root node \(<\text{album}>\). Moreover, the \(<\text{artist}>\) node in Doc1 is represented by the \(<\text{singer}>\) node in Doc2. We see that Doc1 and Doc2 have minor document structural differences while they describe essentially the same information about the music albums.
In another case, Figure 5.2 shows an XML document describing a photo album (Doc3). In comparison with Doc1 (and Doc2), there are a number of structural differences. For example, Doc1 contains the nodes about <artist>, <song>, <composer>, etc. while Doc3 contains nodes <photos>, <photo>, <photographer>, etc. Moreover, for their commonly labeled node, <date>, these nodes are attached to different parent nodes, <album> in Doc1 and <photo> in Doc3, as they describes publishing date of the music album and the date of the photo taken, respectively. While there are major structural differences between Doc1 and Doc3, both documents still have overlapping structure as they both contain the node <album> and child node <name>. If we are interested in XML documents about music albums, we would accept documents with minor structural variations while avoiding false positive returned of documents with major structural differences.
5.2.2 Proximate Twig Query Answers

Twig query is an important part of many XML query processing systems. It can be modeled as a labeled tree whose edges represent either parent-child “/” (a single line edge) or ancestor-descendant “//” (a double line edge) hierarchical relationships.
Figure 5.3 shows an example of twig query $Q_1$ that searches for music albums by the <artist>'s <lastname> and the <song>'s <title>. When the query $Q_1$ is issued and compared against the document set \{ Doc1, Doc2, Doc3 \}, one would intuitively expect Doc1 and Doc2 returned but not Doc3. Obviously, Doc1 is correctly returned with a perfect match against $Q_1$. This simulates the optimal scenario when a query writer fully knows the structure of an album document. However, Doc2 will not be returned if only perfectly matched documents are returned. This motivates the need for proximity searching for twig query while we need to avoid the return of false positive documents returned like Doc3. Our suggested approach is to quantify the structural similarity of $Q_1$ against XML document structures.

One observation which should be made when $Q_1$ is matched against Doc1 is the importance of the existence of node labels and their hierarchical relationships within the document; while the number of occurrences of the same labeled nodes and the node ordering amongst the siblings are not much of a concern. Therefore we can summarize the
document structure of Doc1 as shown in Figure 5.4 and compare this against every node from Q1. For instance the twig query node <album> expects a child node <artist> and a descendant node <song>. We see that the <album> node contains query expected child and descendant nodes. Similarly, the complete query pattern matching process with Doc1 is shown below:

1) Node <album> matches and further search for child node <artist> and descendant node <song>;
2) Node <artist> matches and further search for child node <lastname>;
3) Node <lastname> matches;
4) Node <song> matches and further search for child node <title>;
5) Node <title> matches.

Figure 5.5 shows the query matching results when comparing Q1 against Doc1, Doc2 and Doc3 respectively. A node is partially matched when only some of its child nodes are matched. In the case of Q1 vs. Doc1, we see that all query nodes are matched and hence we conclude that Q1 matches Doc1. When we compare Q1 against Doc2 and Doc3, they do not match perfectly as Doc1 does, Doc2 shows a better matching result than Doc3, as Doc2 has the nodes <album>, <song> and <title> while Doc3 only matches in the node <album>. This result is expected because Doc2 is more structurally similar to Q1 than Doc3. This observation motivates the idea to quantify the twig node level embedding similarity scores and aggregate them into the twig query level to determine the document’s relevance.
5.3 Embedding Similarity

With the idea discussed in the previously, in this section we introduce the *rrs* for XML documents. We then define the embedding structural similarity constituting the metric used in the structure index, which will be introduced below.

5.3.1 Relaxed Representative Structure

We model an XML document as an ordered labeled tree; a tree node represents a document element, while a tree edge corresponds to an element hierarchical relationship.
**Definition 5.1** An XML document tree can be represented as an ordered pair \((N, E)\) where \(N\) is the set of tree nodes while \(E\) is the set of directed edges in the form of \((n_1, n_2)\) where \(n_1\) points to \(n_2\), and \(n_1, n_2 \in N\). The function \(\text{label}(\cdot)\) returns the node label of an element, and we denote \(\text{label}(N)\) as \(\{\text{label}(n) : n \in N\}\) and \(\text{label}(E)\) as \(\{(\text{label}(n_1), \text{label}(n_2)) : (n_1, n_2) \in E\}\). Moreover, we define the function \(\text{fanout}(n \in N)\) as \(|\{(n, n_2) \in E\}|\), that is the number of edges starting from \(n\).

For example, Figure 5.6 shows an XML document and its corresponding representative tree.

![Figure 5.6 An XML document and its tree representation](image)

Next, we define *Relaxed Representative Structure (rrs)* which summarizes the element label and hierarchical relationships of an XML tree.

**Definition 5.2** The *Relaxed Representative Structure*, \(rrs\), of an XML document \((N, E)\) is the ordered pair \((M, F)\) where \(M\) is a set of nodes corresponding to the distinct set of node labels \(\text{label}(N)\) with a surjection \(n \in N \rightarrow m \in M\) such that \(\text{label}(n) = \text{label}(m)\) while \(F\) is a
set of edges equal to \( \{(m_a, m_b) : \text{label}(m_a) = \text{label}(n_a) \text{ and } \text{label}(m_b) = \text{label}(n_b) \text{ for some } (n_a, n_b) \in E\} \).

Intuitively, it is more useful in knowing the category of an XML document by, for example, the fact that it is describing a book with author information rather than the exact number of authors. The \( \text{rrs} \) captures the key characteristics of the XML document structure; it includes the existence of element labels and their hierarchical relationships while dropping the other structural information such as the number of occurrences of element labels and element ordering. The \( \text{rrs} \) can be of a much smaller size when compared to the original document. Another key advantage of \( \text{rrs} \) is its summarization capability for not only a single document but a set of documents, as shown in the modified definition in Definition 5.3.

**Definition 5.3** We define the \( \text{rrs} \) of a set of XML documents \( \{(N_i, E_i)\} \) as the ordered pair \( (M, F) \) where \( M \) is a set of nodes corresponding to the distinct union set of node labels \( \{\text{label}(n) : n \in N_i \forall i\} \) with surjections \( n \in N_i \rightarrow m \in M \) such that \( \text{label}(n) = \text{label}(m) \forall i \).

\[
F = \{(m_a, m_b) : \text{label}(m_a) = \text{label}(n_a) \text{ and } \text{label}(m_b) = \text{label}(n_b) \text{ for some } (n_a, n_b) \in E_i \text{ and } i\}.
\]
Figure 5.7 An example of rrs construction

Lemma 5.1 For two rrs $R_1$ and $R_2$ generated from two sets of XML documents $S_1 \subseteq S_2$, $R_1$ is a subgraph of $R_2$.

Proof Let $S_1 = \{(N_{1i}, E_{1i})\}$, $S_2 = \{(N_{2j}, E_{2j})\}$ and $R_1 = (M_1, F_1)$, $R_2 = (M_2, F_2)$. We have $M_1 = \bigcup_{l} \text{label}(N_{1i}) \subseteq \bigcup_{l} \text{label}(N_{2j}) = M_2$. For $(m_a, m_b)$ in $F_1$, it follows from the definition of rrs that there exist $(e_a, e_b)$ in $E_{1_k}$ for some $k$ such that $\text{label}(m_a) = \text{label}(e_a)$ and $\text{label}(m_b) = \text{label}(e_b)$. As $S_1 \subseteq S_2$, $E_{1_k} \in \{E_{2j}\}$, it implies that $(e_a, e_b) \in E_{2_l}$ for some $l$ and hence $(m_a, m_b) \in F_2$. This shows that $F_1 \subseteq F_2$ and Lemma 5.1 follows.

Figure 5.7 shows two XML documents of similar structures are summarized into a rrs representing the summarized document structure. The rrs consists of the set of labels and the associated edges from the generating documents. We can see rrs as the definition of labels $l = \text{label}(m \in M)$ with the structural hierarchical relationships of labels $(l, \text{label}(m'))$ where $(m, m') \in F$. rrs can be considered as a special case of the abstract schema defined (in Section 3.2.2) which specifies the structure of documents. For an abstract schema, a label is uniquely defined as a regular expression of other labels. When we release the requirement of label ordering occurrences (specified by the operators “*”, “?”,”+”), it essentially consists of the set of defining labels. For example,

$$b^*c+d?bbccddd \Rightarrow b^*bcc+ccd?dd \Rightarrow \{b,c,d\}$$
With the *rrs*, we can compare the structure of an XML document against a set of documents using this “summarized” structure, which is much more efficient than one-by-one comparison. This motivates us to propose the structural similarity of an XML document against the *rrs*.

### 5.3.2 Embedding Similarity

Suppose we have an XML document $X$ as $(N, E)$ and a *rrs* $R$ as $(M, F)$. Here, we define the node-level embedding similarity as follows.

**Definition 5.4** For $n \in N$, the embedding cardinality of $n$ against $R$, $\text{emCard}(n, R)$ is $|\{(m_1, m_2) \in F : \text{label}(m_1) = \text{label}(n) \text{ and } \text{label}(m_2) = \text{label}(n') \text{ for some } (n, n') \in E\}|$. Also, the embedding similarity of $n$ against $R$, $\text{emSim}(n, R)$ is $\text{emCard}(n, R) / \text{fanout}(n)$. For simplicity, we also denote as $\text{emCard}(n)$ and $\text{emSim}(n)$ respectively if $R$ does not change.

$\text{emSim}()$ measures the ratio of its outgoing edges falling in the corresponding label definition in the *rrs*. If the node and all its edges are found defined, it has a similarity value one and the node is embedded in the *rrs*. At the other extreme, $\text{emSim}()$ has a similarity value zero if the node label or all its edges are not defined. This embedding similarity is not sufficient because we are interested in the document-level similarity. Therefore, we define a weighting scheme to aggregate and promote the node-level similarity in the document-level.

**Lemma 5.2** For $n \in N$ and two *rrs* $R_1 \subseteq R_2$, we have $\text{emSim}(n, R_1) \leq \text{emSim}(n, R_2)$.
**Proof** Let $R_1 = (M_1, F_1)$ and $R_2 = (M_2, F_2)$. For any $(m_1, m_2) \in F_1$ with $\text{label}(m_1) = \text{label}(n)$ and $\text{label}(m_2) = \text{label}(n')$ for some $(n, n')$ in $E$, we have $(m_1, m_2) \in F_1 \subseteq F_2$ as $R_1 \subseteq R_2$. Therefore $|\{(m_1, m_2) \in F_1 : \text{label}(m_1) = \text{label}(n) \text{ and } \text{label}(m_2) = \text{label}(n') \text{ for some } (n, n') \text{ in } E\}| \leq |\{(m_1, m_2) \in F_2 : \text{label}(m_1) = \text{label}(n) \text{ and } \text{label}(m_2) = \text{label}(n') \text{ for some } (n, n') \text{ in } E\}|$ and that implies $\text{emSim}(n, R_1) = \text{emSim}(n, R_1)$.

**Definition 5.5** The weight of a node $n \in N$ is $w(n) = 1 + \sum_{(n, ni) \in E} w(ni)$. If $n$ is a leaf node, $w(n) = 1$. The weight percentage of $n$, $wp(n)$, is defined as $w(n) / \sum_{ni \in N} w(ni)$.

One idea is to determine if an element in the document is more important in term of structural semantics is that the corresponding node is the root of a large sub-tree. Following from this, we define the document-level embedding similarity accordingly as follows.

**Definition 5.6** The document-level embedding similarity of an XML document $X$ against an rrs $R$ is defined as $\text{emSim}(X, R) = \sum_{ni \in N} (wp(ni) * \text{emSim}(ni, R))$. For simplicity, we also denote as $\text{emSim}(X)$ if $R$ does not change.

The document-level embedding similarity quantifies to what extent a document is embedded in the structure specified in the rrs. The definition of embedding similarity $\text{emSim}(\cdot)$ is originated from the similarity measure $\text{smaSim}(\cdot)$ defined in Section 3. They both first define the normalized node similarity scores and then aggregate them into the document-level similarity score using an equivalent weighting scheme. We will show in a later section that this constitutes the key metric in supporting the exact and proximity index searching. From the above definition, we can separate it into components with different
dependencies. The document-level embedding similarity of $X$ against $R$ can be re-written as
\[ \text{emSim}(X, R) = \sum_{n_i \in N} (\text{emCard}(n_i, R) \ast (wp(n) / \text{fanout}(n))), \]
where the component $wp(n) / \text{fanout}(n)$ depends on $X$ but not on $R$. Given a document $X$, we can pre-compute this component and cache it, and this allows quicker computation as only the $\text{emCard}()$ needs to be worked out in relation to different rrs. In this case, a good linear complexity of $O(|R|)$ is achieved.

For an XML document $X$, we have the following lemmas implied from the definitions.

**Lemma 5.3** If rrs $R$ is generated from a document set $S$, we have $\text{emSim}(X, R) = 1$ for all $X \in S$.

**Proof** It is sufficient to prove that $\text{emSim}(n, R) = 1$ for all $n$ in $X$. For every edge $(n, n')$ in $X$, since $R$ is generated from $S$ and $X \not\in S$, it follows from the definition of rrs that there exists an edge $(m_a, m_b)$ in $R$ such that $\text{label}(m_a) = \text{label}(n_a)$ and $\text{label}(m_b) = \text{label}(n_b)$. We have $\text{emCard}(n, R) = |\text{all edges } (n, n') \text{ in } X| = \text{fanout}(n)$. Hence $\text{emSim}(n, R) = \text{emCard}(n, R) / \text{fanout}(n) = 1$.

**Lemma 5.4** For two rrs $R_1$ and $R_2$ generated from two sets of XML documents $S_1 \subset S_2$, we have $\text{emSim}(X, R_1) \leq \text{emSim}(X, R_2)$.

**Proof** For every $n$ in $X$, Lemmas 5.1 and 5.2 together imply that $\text{emCard}(X, R_1) \leq \text{emCard}(X, R_2)$. Given a fixed $X$, we have seen previously that $\text{emSim}(X, R)$ is an arithmetic sum of $\text{emCard}(n, R)$ for all $n_i$ in $X$. It follows that $\text{emSim}(X, R_1) \leq \text{emSim}(X, R_2)$.
Lemma 5.3 simply verifies that the generated \textit{rrs} actually captures the structure semantics of the original document. In Lemma 5.4, we notice that if \( X \) is not similar enough to \( R_2 \), it will never be more similar against \( R_1 \) which represents a document subset. This monotonic property helps to demonstrate that our development of the hierarchical index structure and query processing is working, as demonstrated in later sections.

**5.4 RRSi**

In this section, we present the \textit{RRSi}, a structure index designed to support both exact and proximity searches by structure. We first give the basic and conceptual structure, followed by an optimized implementation. Finally, we examine the index construction as well as updating considerations.

**5.4.1 Structure of RRSi**

Let \( DOC \) be the XML document set to be indexed.

**Definition 5.7** An \textit{RRSi} for \( DOC \) is a balanced bi-level tree with each node \( n \) associated with an \textit{rrs}. The \textit{rrs} of node \( n \) is denoted as \( n.rrs \), which is generated by a document subset, denoted by \( n.docs \subseteq DOC \), such that

1) An internal node \( n \) points a local B+-tree indexed on \( M \) with \( n.rrs = (M, F) \), while each \( m \in M \) points to the set of edges \( \{(m, m_1) \in F : m_1 \in M\} \).
2) An internal node \( n \) with child internal nodes \( n_1, \ldots, n_k \) satisfies the property \( n.docs = \bigcup_{i=1}^{k} (n_i.docs) \).

3) Any two leaf nodes \( n_1 \) and \( n_2 \) have disjointed underlying document subsets, i.e. \( n_1.docs \cap n_2.docs = \emptyset \). Moreover, \( \bigcup \forall \text{ leaf nodes } n \ (n.docs) = DOC \).

4) A leaf node \( n \) is associated with a pointer \( n.docp \) pointing to the document subset \( n.docs \).

Note that \( n.docs \) is not part of the index structure, but represents the underlying document subsets.

An RRSi is essentially a B+-tree for the indexing of structurally similar documents with second level B+-trees associated with every node indexing on the nodes’ rrs.

**Lemma 5.5** If \( na \) is the ancestor node of another node \( nd \), we have \( na.docs \supseteq nd.docs \).

**Proof** It is sufficient to prove that \( na.docs \supseteq nc.docs \) for each child node \( nc \) of \( na \). Let \( n_1, \ldots, n_k \) be the child nodes of \( na \). It follows from Property 2 in Definition 5.7 that \( na.docs = \bigcup_{i=1}^{k} (n_i.docs) \supseteq n_i.docs \) for \( i = 1, \ldots, k \).

We see that the root node \( n_{root} \) has \( n_{root}.docs = DOC \), as followed by Properties 2 and 3.
Figure 5.8 An illustration of a RRSi.

In Figure 5.8, we see that the RRSi forms a hierarchy of structural semantics represented by the rrs. The generating document subsets for the rrs also form a nice hierarchy starting from the DOC at the root node and splitting at each level into disjoint subsets of structurally similar documents until reaching the leaf nodes. Then further lower level nodes would represent more specific structural semantics. On the one hand, this can be extended so that each leaf node contains a single document. However, this increases the index size significantly while it does not take advantage of the opportunity to group documents of highly similar or identical structure. On the other hand, if the leaf nodes’ document sets are too rough to hold less structurally similar documents, we lose the capacity to locate relevant documents efficiently in query processing.

5.4.2 RRSi Construction and Updating

In this section, we present an algorithm in constructing the RRSi. Determining and grouping structurally similar documents into same leaf nodes are keys to the effectiveness of RRSi. As we use rrs to capture the key structural information of a document or a
document set, we first define the structural similarity between rrs as extended from \(emSim()\).

### 5.4.2.1 RRS-Similarity

As a recap, \(emSim()\) is defined as the weighted sum of node-level embedding similarity. If we replace XML document by rrs in the definition to compare against another rrs, we need only to revise the recursive definition of the weighting scheme, \(w()\) and \(wp()\), which does not support potential cyclic edges in rrs with a proper exit criterion. Therefore, we extend the weighting definition by adding an additional exit criterion of not counting a node occurring the second times.

**Definition 5.8** Let \((M, F)\) be a rrs. The weight of a node \(m \in M\) is \(w(m) = 1 + \sum_{(m, m_i) \in F} w(m_i)\) for \(m_i\) not appeared before in the recursion. If \(m\) is a leaf node, \(w(m) = 1\). The weight percentage of \(m\), \(wp(m)\), is defined as \(w(m) / \sum_{m_i \in M} w(m_i)\).

With the new definition, our objective is to compare bilateral structural similarity rather than the unilateral embedding similarity. We define the rrs-similarity as follows.

**Definition 5.9** Let \(R_1\) and \(R_2\) be two rrs. The RRS-Similarity, denoted by \(rrsim(R_1, R_2)\) is \((emSim(R_1, R_2) + emSim(R_2, R_1)) / 2\).

It immediately follows that \(rrsim(R_1, R_2) = rrsim(R_2, R_1)\).
5.4.2.2 Construction and Updating

Constructing an RRSi is similar to building a B+-tree; starting from a single document index, we insert documents into the leaf node \( n \) which points to the \( n.docs \) of most similar structures. If \( |n.docs| \) grows larger than a configurable threshold, \( n \) is split into two leaf nodes \( n_1 \) and \( n_2 \), pointing to each half of the original documents \( n1.docs \cup n2.docs \) (\( = n.docs \)) clustered using the \( rrsim() \) metric. In addition, a new intermediate node is inserted to hold the split leaf nodes. For all the ancestor nodes, the \( rrs \) are extended by merging the additional label nodes and label edges contributed from the inserted document. The insertion algorithm is presented below.

---

**RRSi Insertion Algorithm**

Input: new XML document \((X)\), RRSi \((R)\), threshold \((\theta)\)

Output: Revised RRSi with \( X \) inserted

\[
\text{InsertNode}(X, R, \theta) \{
\text{Set } n_c = \text{the root node of } R
\text{While } (n_c \text{ is not leaf node}) \{
\text{Let } \{n_i\} \text{ be the set of child nodes of } n_c
\text{Pick } n_k \text{ from } \{n_i\} \text{ with largest } \text{emSim}(X, n_k.rrs)
\text{\hspace{1cm} /* if more than one largest \text{emSim}(),}
\text{\hspace{1cm} pick the first largest } \{n_i\} \text{ */}
\text{n_k.rrs = rrsGen(X, n_k.rrs) \& n_k.docs = n_k.docs \cup X}
\text{\hspace{1cm} /* rrsGen() generates new rrs from}
\text{\hspace{1cm} the arguments rrs and/or documents */}
\text{Set } n_c = n_k
\}\}
\]
\[ n_c.rrs = rrsGen(X, n_c.rrs) \& n_c.docs = n_c.docs \cup X \]

If \( |n_c.docs| \geq \theta \) {
Create new child nodes \( n_1 \& n_2 \) of \( n_c \)
\((docs_1, docs_2) = rrsClust2(n_c.docs)\)

/* \( rrsClust2() \) clusters \( n_c.docs \) into 2 clusters */
Set \( n_j.rrs = rrsGen(docs_j) \& n_j.docs = docs_j \) for \( j=1,2 \)
Create pointers \( n_j.docp \) pointing to \( docs_j \) for \( j=1,2 \)
\}
\}

---

Input: XML document \( (X) \), \( RRSi \) (\( R \) - optional),
Output: Generated \( RRSi \)

\[ \text{rrsGen}(R, X) \{ \]
\[ \text{If} \ (R \text{ is not null } ) \{ \text{Set } rtn_{rrs} = R \} \]
\[ \text{Else} \{ \text{Set } rtn_{rrs} \text{ to an empty } rrs \} \]
For each \( x \) in \( X \) {
Let \( C_x \) be the set of child nodes of \( x \)
\[ \text{If} \ (x \text{ is not in } rtn_{rrs} ) \{ \]
\[ \text{Append } "x = C_x" \text{ to } rtn_{rrs} \]
\[ \text{Else} \{ \]
\[ /* \text{Let } "x = L_x" \text{ in } rtn_{rrs} */ \]
\[ \text{Replace } "x = C_x \cup L_x" \text{ to } rtn_{rrs} \]
\[ \} \]
\[ \} \]

Deleting a node in the \( RRSi \) is also similar to deleting a B+-tree; deleting a document from a leaf node \( n \) will update the \( n.docs \) and \( n.rrs \) for \( n \) and all its ancestors accordingly. Node merging and re-distribution occur if \( |n.docs| \) is less than the configurable threshold.
5.4.3 Optimized Implementation of RRSi

An RRSi is basically a master B+-tree with many secondary B+-trees associated with each tree node in the master tree. From the discussion in the previous section, it is clear that the root index node represents the whole document population \( DOC \), while a parent/ancestor node’s representing document set is always a superset of that of its child/descendant nodes. Hence, an \( rrs \) edge that appears in a leaf node also appears in all its ancestors. This would introduce a space complexity issue and resulting in an efficiency problem in the implementation. This characteristic motivates us to find a way to reduce data duplication in RRSi. Our solution is to store only the occurrences of \( rrs \) edges from the leaf nodes while working out the corresponding ancestors’ occurrences on the fly. The approach requires a convenient numbering scheme, defined as follows.

**Definition 5.10** We introduce a node numbering scheme on the RRSi where each node \( n \) is assigned a node id, denoted as \( n.nid \), in the form of \( a_1.a_2....a_k \). If \( n \) is the root node, then \( n.nid = 1 \). If \( p \) is the parent node of \( n \) with \( p_1, p_2, ..., p_j \) as ordered child nodes and \( n = p_i \), \( n.nid \) is defined as the \( p.nid \) with ".i" appended at the end. Then \( k \) is the level position of \( n \) in the tree.

Take the document in Figure 5.6 as example. We have \( a.nid = 1, b.nid = 1.1, c.nid = 1.2 \) and \( d.nid = 1.3 \). For two RRSi nodes \( m \) and \( n \) with \( n.nid = m.nid \) with “.i+1,i+2,...,i_k” appended at the end, we have \( m \) as an ancestor of \( n \) from the definition of \( nid \). Then it follows from Lemma 5.5 that \( n.docs \subseteq m.docs \) and this gives the expected capability to
work out the ancestor node id from the descendent node id. Now, we are ready to give the
definition of an optimized $RRSi$.

**Definition 5.11** Let $R = (M, F)$ be the $rrs$ generated from $DOC$. An Optimized $RRSi$, denoted as $oRRSi$, is a bi-level B+-tree indexed on $F$ such that each edge $f \in F$ is associated with an array of node id $f.nid[]$ where $f$ appears in the leaf node ids in the original (conceptual) $RRSi$ (see Figure 5.9).

![Figure 5.9 An illustration of an oRRSi](image)

**Theorem 5.1** The original $RRSi$ can be constructed from the $oRRSi$.

**Proof** For a root node $n_r$ in an $RRSi$ $R$, both the $n_r.rrs$ and its node’s B+-tree are the same as the only B+-tree in the optimized $RRSi$ denoted as $oR$. For an internal node $n$ in $R$, we can obtain the $n.rrs$ from $oR$ by retrieving the $rrs$ edges from $oR$ while filtering out the edges without the $n.nid$ as prefix $nid$. 

---

Page 99
In term of space requirements, the \( oRRSi \) requires much less space than the \( RRSi \), as a result of its structure. The \( oRRSi \) indexes each node label from the document population only once while the \( RRSi \) can index the same node label many times in different local document clusters. Theorem 5.2 demonstrates that the \( oRRSi \) achieves space efficiency of one order less than that required by the \( RRSi \).

**Theorem 5.2** The space complexity of the \( RRSi \) and \( oRRSi \) are \( O(D*P^2) \) and \( O(D*P) \) respectively where \( P \) is the number of indexing XML documents and \( D \) is the maximal number of nodes of the largest document among all the indexed documents.

**Proof** In general, a B+-tree has a space complexity of \( O(\Lambda) \) where \( \Lambda \) is the number of the indexing leaf nodes. In the \( RRSi \), each node contains a local B+-tree on an \( rrs \) generated by a subset of indexed documents. Therefore, its size is locally bounded by maximal size of the \( rrs \), which is in turn bounded by the total number of edges among all the indexed documents, i.e. \( D*P \). At the index tree level, the leaf nodes point to disjoint subsets of documents and hence the total number of leaf nodes is always less than \( P \). Therefore, the overall space complexity of the \( RRSi \) is \( O(D*P*P) \), i.e. \( O(D*P^2) \). On the other hand, the \( oRRSi \) is a B+-tree with leaf nodes pointing the lists of document subset numbering. For similar reasons as we outlined for the \( RRSi \), the size of the former is bounded by \( O(D*P) \). For the latter, since the documents are partitioned into disjoint subsets according to the document structures, it is realistic to assume that each indexing edge can appear in at most \( c \) document subsets for some constant \( c \). In this case, the overall space complexity of the \( oRRSi \) is \( O(D*P*c) \), i.e. \( O(D*P) \).
5.5 Twig Query Processing

This section describes twig query processing. We extend the embedding similarity measure \( emSim() \) and the \( RSSi \) to handle general twig queries with both parent-child and ancestor-descendant edges. We demonstrate an efficient algorithm for searching the \( RSSi \) index for effective proximity twig query evaluation, with comparative advantage in huge XML repositories with structural heterogeneity. We show that the \( RSSi \) can quickly locate document subsets with embedded and proximity twig queries. Moreover, it can reject false positive documents of irrelevant query topics which cannot be efficiently rejected using common content-based querying techniques.

5.5.1 Proximity Twig Query Match

Querying XML documents by twig pattern matching is an important part of XML query processing. A twig query is a labeled tree with edges representing either the \( parent-child \) “/” or \( ancestor-descendant \) “//” relationship. For example, the twig query “album[/artist/lastname and //song/title]” looks for an “album” with a “song” as descendant node and a “artist” as child node. Moreover, a “title” child node of “song” and a “lastname” child node of “artist” node are expected respectively. Figure 5.10 shows the tree representation of the twig query example. In general case, we model a twig query as below.
**Twig Query**

/album[
  /artist/lastname
  and //song/title
]

**Tree Representation**

![Tree Diagram]

\[N = \{a, b, c, d, e\} \]
\[E = \{e_{ab}, e_{ad}, e_{bc}, e_{de}\}\]

\[\text{label}(a) = \text{"album"}\]
\[\text{label}(b) = \text{"artist"}\]
\[\text{label}(c) = \text{"lastname"}\]
\[\text{label}(d) = \text{"song"}\]
\[\text{label}(e) = \text{"title"}\]

\[e_{ab}.pc = \text{true}\]
\[e_{ad}.pc = \text{false}\]
\[e_{bc}.pc = \text{true}\]
\[e_{de}.pc = \text{true}\]

**Figure 5.10** A twig query and its tree representation

**Definition 5.12** A twig query \(Q\) is an unordered labeled tree denoted as an ordered pair \((N_Q, E_Q)\) of nodes and edges respectively. \(n_q \in N_Q\) represents the twig node with the node label denoted as \(\text{label}(n_q)\), and \(e_q \in E_Q\) corresponds to the node hierarchical relationships. A parent-child relationship flag \(e_q.pc\) is associated with each edge indicating whether it expects an ancestor-descendant (\(e_q.pc = false,\) denoted ‘//’) or a parent-child relationship (\(e_q.pc = true,\) denoted ‘/’).

Below, we provide a definition of twig query matching against an XML document.
Definition 5.13 A twig query $Q = (N_Q, E_Q)$ is said to \textit{s-match} a document $X = (N_X, E_X)$ if there exists an injection $\zeta : N_Q \rightarrow N_X$ such that $\text{label}(n_a) = \text{label}(\zeta(n_a))$ and $\text{label}(n_b) = \text{label}(\zeta(n_b))$ and

1) $(\zeta(n_a), \zeta(n_b)) \in E_X$ if $(n_a, n_b) \in E_Q$ for $(n_a, n_b).pc = \text{true}$ or
2) $(\zeta(n_a), n_{X1}), (n_{X1}, n_{X2}), \ldots, (n_{Xk}, \zeta(n_b)) \in E_X$ if $(n_a, n_b) \in E_Q$ for $(n_a, n_b).pc = \text{false}$.

Intuitively, $Q$ s-matches $X$ if $Q$ can be embedded as a sub-tree in $X$ following the parent-child and ancestor-descendant hierarchical relationships. In other words, $X$ contains the exact sub-structure that is queried by $Q$. Suppose in a particular case that all edges of a twig query are parent-child edges; then we can regard $Q$ as a small XML document structure and the \textit{emSim()} can apply to calculating the embedding similarity against the rrs. It is easy to see that this \textit{s-match} definition of twig queries can be regarded as a special case of the \textit{emSim()} function. Hence, we have Lemma 5.6 next.

Lemma 5.6 Suppose $Q$ is a twig query with only parent-child edges. If $Q$ s-matches $X$ and $R$ is the rrs generated by $X$, we have $\text{emSim}(Q, R) = 1$.

\textbf{Proof} Similar to Lemma 5.3, it is sufficient to prove that $\text{emSim}(n, R) = 1$ for all $n$ in $Q$. For every edge $(n, n')$ in $Q$, it follows from Definition 14 for the parent-child case that there exist an edge $(m, m')$ in $X$ such the $\text{label}(n) = \text{label}(m)$ and $\text{label}(n') = \text{label}(m')$. On the other hand, since $R$ is generated from $X$, it follows from the definition of the rrs that there exists an edge $\text{label}(m) = \text{label}(q)$ and $\text{label}(m') = \text{label}(q')$We have $\text{emCard}(n, R) = |\text{all edges (n, n') in Q}| = \text{fanout}(n)$. Hence $\text{emSim}(n, R) = \text{emCard}(n, R) / \text{fanout}(n) = 1$. 


As in Lemma 5.6, the definition of the \( \text{emSim()} \) function can be extended to handle general twig queries with ancestor-descendent edges.

**Definition 5.14** Let \( Q = (N, E) \) be a twig query and \( R \) be a rrs \( = (M, F) \).

1) The “parent-child” embedding cardinality, \( \text{pc-emCard}(n, R) \), is of the value \(|\{m \in M : \exists m_1 \in M \text{ such that } \text{label}(m_1) = \text{label}(n), \text{ and } m \text{ is a child node of } m_1 \text{ in } R\}|\).

2) The “ancestor-descendant” embedding cardinality, \( \text{ad-emCard}(n, R) \), is of the value \(|\{m \in M : \exists m_1 \in M \text{ such that } \text{label}(m_1) = \text{label}(n), \text{ and } m \text{ is a descendant node of } m_1 \text{ in } R\}|\).

3) Then the embedding similarity of \( n \) against \( R \), \( \text{emSim}(n, R) \), is extended and redefined as \( [\alpha \times \text{pc-emCard}(n, R) + (1-\alpha) \times \text{ad-emCard}(n, R)] / \text{fanout}(n) \), where \( \alpha \) is a fixed number \( \in [0, 1] \). For simplicity, we also denote as \( \text{emCard}(n) \) and \( \text{emSim}(n) \) respectively if \( R \) does not change.

The extended version of the \( \text{emSim()} \) function is to count the embedding cardinality of child nodes and descendant nodes separately and add their corresponding contributions with a chosen weighting of \( \alpha \). If \( \alpha \) approaches one, more weighting is given to “child matches” and less weighting is given to “descendant matches”. It can be used as the preference parameter of the query processor. For illustration purpose, we assume in subsequent discussion that \( \alpha = 0.5 \), that is giving equal importance to “child and descendant matches”. Moreover, the backward compatibility of \( \text{emSim()} \) is obvious when we have a twig query.
with no ancestor-descendant edge. Now, the $emSim()$ on a twig query is defined in Definition 16 which extends the previous definition in Definition 5.6.

**Definition 5.15** The query-level embedding similarity of a twig query $Q = (N, E)$ against an rrs $R$ is defined as $emSim(Q, R) = \sum_{n \in N} (wp(n) * emSim(n, R))$. For simplicity, it is also denoted as $emSim(Q)$ when $R$ is clear in the discussion.

Our primary objective is to support proximity twig query processing. Due to the heterogeneity of many proprietary XML schemas and documents, only returning only documents that exactly match a twig query would miss many useful documents with only minor structural differences. However, users often expect structurally close documents to be returned as well. In the example in Section 5.2, the twig query $Q1$ looks for music album(s) among $Doc1$, $Doc2$ and $Doc3$ in Figures 5.1 and 5.2. It is expecting $Doc1$ and $Doc2$ to be returned but not $Doc3$. Now, this expected relevance can be quantified by using the embedding similarity function $emSim()$ and the results are $emSim(Q1, R1) = 1$, $emSim(Q1, R2) = 0.72$ and $emSim(Q1, R3) = 0.45$ where $R1$, $R2$ and $R3$ are the rrs generated by $Doc1$, $Doc2$ and $Doc3$, respectively. The proximate query result can be controlled or filtered by setting an appropriate similarity threshold of 0.7, and therefore we see that the embedding similarity can effectively support the proximity search by structure. The aforementioned similarity threshold should be chosen in a way which includes only the structurally relevant or interesting documents. Setting the threshold too low or too high leads to too many irrelevant results or missing relevant results respectively. Users can specify this threshold according to their needs.
5.5.2 Query Lookup Algorithm

Structurally similar documents have been grouped into index nodes in the RRSi. Given a twig query, we present an index lookup algorithm for the RRSi which can quickly locate structurally similar documents. Note that the algorithm also applies to the oRRSi index as we have shown in Theorem 5.1 that it shares a conceptual indexing structure with the RRSi.

---

**RRSi Query Lookup Algorithm**

Input: a twig query \( Q \), a RRSi node \( n_c \) – initially set as the root node, a threshold \( \theta \)

Output: Document subsets as query result

\[
\text{QueryRRSi}(Q, n_c, \theta) \{
\text{Set } \text{rtn}\_\text{docs} = \text{empty}
\]

\[
\text{If } \text{emSim}(Q,n_c.rrs) < \theta \{
\text{Return } \text{rtn}\_\text{docs}
\}
\]

\[
\text{Else } \{
\text{If } (n_c \text{ is not a leaf node}) 
\text{Let } \{n_i\} \text{ be the set of child nodes of } n_c
\text{for all } (n_i) \{ \text{rtn}\_\text{docs} += \text{MergeList} \text{QueryRRSi}(Q, n_i, \theta) \}
\text{Return } \text{rtn}\_\text{docs}
\}
\]

\[
\text{Else } \{
\text{Return } n_c.\text{docs}
\}
\]
The algorithm first checks the twig query against the root $rrs$ of the $RRSi$ and navigates down the index tree only if the embedding similarity is greater than the preset threshold. Similarly, a decision is made on each child node based on its embedding similarity. The procedures are repeated towards the leaf level as shown in Figure 5.11.

$$\text{emSim}(Q, rrs)$$

**Figure 5.11** Query processing flow in $RRSi$

This pruning approach works because the index supports $n_{\text{parent.docs}} \supseteq n_{\text{child.docs}}$ and hence is followed by Lemma 5.4. This nice property allows fast pruning of structurally irrelevant documents at the early stages of lookup. In other words, this effectively eliminates false positive results. Traditional content-based searching techniques fail to detect documents containing the query keywords but describing unrelated topics reflected in the differences in their document structures. This fails to eliminate these false positive results until in the later steps, and substantially increases the query processing time.

### 5.5.3 Analysis of Lookup Algorithm

Next, we show the accuracy of the query lookup algorithm using the $RRSi$, as well as its complexity.
**Theorem 5.3** All documents with embedding similarity $\geq \theta$ are returned in the $RRSi$ query lookup.

**Proof** Let $X$ be a document with embedding similarity $\geq \theta$. That is $emSim(Q, R_X) \geq \theta$ where $R_X$ is the $rrs$ generated from $X$. Then $X \in n.docs$ for some leaf node $n$. Hence $\{X\} \subseteq n.docs$ and we have $emSim(Q, n.rrs) \geq emSim(Q, R_X) \geq \theta$ by Lemma 5.4. Therefore, $n$ is a return node.

**Definition 5.16** For a twig query $Q$, the query result is complete if every queried XML document $X$ where $Q$ s-matches $X$ is returned.

**Theorem 5.4** The twig query result returned from the $RRSi$ lookup is complete.

**Proof** Let $Q$ be a twig query $Q$ and $X$ be a queried XML document such that $Q$ s-matches $X$. Let $R$ be the $rrs$ generated by $X$, Lemma 5.6 implies that $emSim(Q, R) = 1$ which is greater than or equal to any similarity threshold set $\theta$ for a lookup. Therefore, the completeness follows from Theorem 3.

While the correctness holds for both the $RRSi$ and $oRRSi$, we show that the IO complexity for the $oRRSi$ is much better than that of the $RRSi$ in Theorem 5.5.

**Theorem 5.5** The IO complexity of the $RRSi$ and $oRRSi$ are $O(Q*\log(D)*\log(P))$ and $O(Q*(\log(D)+\log(P)))$ respectively where $P$ is the number of indexing XML documents, $D$
is the maximal number of nodes of the largest document among all the indexed documents, and $Q$ is the number of nodes in the twig query.

**Proof** For a twig query of size $Q$, every query node is matched against the index. In the $RRSi$, a query node is searched through the bi-level B+-tree such that a local B+-tree has the number of contributing XML documents bounded by the fixed index threshold $\theta$ in the query lookup algorithm. The number of leaf nodes in the local B+-tree is bounded by $\theta*D$, and hence it has a local IO complexity of $O(\log(D))$, as $\theta$ is a constant. In a similar way, the outer B+-tree has the number of leaf nodes bounded by $P/\theta$; hence, the corresponding IO complexity is $O(\log(P))$. Therefore, the overall IO complexity for searching the $RRSi$ is $O(Q*\log(D)*\log(P))$. On the other hand, a query node is only searched in the $oRRSi$, a single level B+-tree, to obtain all the node ids in the conceptual $RRSi$ index for further memory lookup. As the number of leaf nodes in the B+-tree is bounded by the maximal number of different document labels $D*P$ in the document population, the IO complexity of searching the $oRRSi$ is bounded such that $O(Q*(\log(D)+\log(P)))$.

In the worst case scenario, we can still come up with a full lookup of the $RRSi$, if either the query structure or terms are too general to appear in all the indexed documents or all the indexed documents have the same structure. In the former case, we argue that most indexing method cannot return meaningful results to users when users’ queries are too general. The latter case can happen when all documents are generated from the same schema. Note that, firstly, documents generated even from the same schema often result in different structures. Secondly, it is unlikely that this scenario will occur when heterogeneous XML data sources are being queried. Although both the IO and space
complexities increase with the number of different document labels in the population, a relatively small increment ratio to the document population is often attained in large XML repositories because documents generated from the same sources or describing closely related information can have many common document labels, making the \textit{RRSi} and \textit{oRRSi} scalable in practice.

5.6 Experimental Evaluation

In this section, we present the experimental results of our novel structural similarity measure and indexing schemes on real and synthetic datasets. The results have verified the effectiveness and efficiency of the \textit{RRSi} and \textit{oRRSi} as XML indexes for fast and accurate proximity query processing, and demonstrated the advantages over previous techniques.

5.6.1 Experimental Framework

In our experiments, we compared the effectiveness and efficiency of XML indexes using the \textit{RRSi}, \textit{oRRSi}, \textit{TwigStackXB} [BKS02] and \textit{PRIX} [MR04]. We implemented all the algorithms using \textit{Java 1.5}. \textit{TwigStackXB} and \textit{PRIX} have been proposed as XML indexing techniques to support twig querying. \textit{TwigStackXB} uses XB-trees to skip nodes on input lists in order to speed up the processing. \textit{PRIX} transforms twig patterns into \textit{Prüfer} sequences for holistic twig matching. Since the original proposals focused on exact query answers, we have revised the algorithms for proximity query processing using query relaxation [AKM+05]. Twig scoring in [AKM+05] makes use of the query relaxation
technique and \( tf*idf \) to score and rank query answers. Although twig scoring is decided to score both structure and content, it can also be applied to structure-only queries. We ran experiments on a computer with a 2.6GHz Pentium IV processor with 4GB memory running Solaris 8. For each of the compared algorithms, there are parameters to be configured including the similarity thresholds and index node thresholds. Preliminary experiments have been conducted on each index independently to identify the optimal parameters to be used for comparisons in the subsequent experiments.

5.6.1.1 Data Sets and Queries

We selected four real world (1)-(4) and one synthetic (5) XML datasets in our experiment: (1) **TPC-H** - decision support benchmark, (2) **SwissProt** - protein sequences, (3) **TreeBank** - English sentences tagged with parts of speech, (4) **DBLP** - bibliographic information on major computer science journals and proceedings, (5) **XMark** [XM01a] - synthetic data set that models auction site data. The real-life data was retrieved from the UW XML Repository [UW05]. All five data sets were grouped into a large single XML repository, simulating the heterogeneous scenario. For each underlying data set, we randomly selected sample documents to generate twig queries in a controlled manner. One hundred queries were generated from each data set containing both parent-child and ancestor-descendant edges with 5 to 12 twig nodes. Hence in total, there were 5*100 queries which were called **QTPCH**, **QSPROT**, **QTBank**, **QDBLP** and **QXMARK** respectively.
Figure 5.12 Comparison diagrams of indexes and query sets for (a) QER (b) Number of Elements Scanned (c) Execution Time

5.6.1.2 Evaluation Metrics

We evaluate the indexes from several perspectives: accuracy, computational performance and scalability. The accuracy of proximate twig query answers is measured by the query error rate ($QER$) as the ratio of false positive answers to the total number of query answers returned. Given a twig query $Q$ and the corresponding set of proximate documents returned, the result relevance is examined against the submitting query to identify any false positive answer. In general, the lower the $QER$, the higher the accuracy of proximity twig query results. Secondly, for computational performance evaluation, we adopt the metrics of the
number of element scanned and the query execution time to compare the performance of twig query processing. Lastly, the scalability is measured as the growth in index sizes and query execution times over the growth of indexing document sizes.

5.6.3 Experimental Results and Analysis

In the first set of experiments, we compared our \textit{RRSi} and \textit{oRRSi} against the \textit{TwigStackXB} and \textit{PRIX} indexes. For each of five twig query sets, we examined the correctness of the returned document sets and calculated the average \textit{QERs} to compare the accuracy of the indexes. In Figure 5.12(a), we see that both the \textit{RRSi} and \textit{oRRSi} attain a low degree of error rates, of less than 20% because the structural differences of irrelevant documents are accurately pruned by our novel structural similarity measure \textit{emSim()}. Instead, the \textit{QERs} for \textit{TwigStackXB} and \textit{PRIX} are consistently higher than 20%. This demonstrates the accuracy and effectiveness of our novel similarity measure function. Meanwhile, the \textit{RRSi}/\textit{oRRSi} and \textit{TwigStackXB}/\textit{PRIX} respectively always attain the same \textit{QER} because they adopt the same similarity measures \textit{emSim()} and twig scoring in the index structure while their difference lies in the way in which the index is structured and queried. This difference is reflected in the computational performance and scalability evaluations in the following paragraphs.

With respect to the number of elements scanned, Figure 5.12(b) shows that both the \textit{RRSi} and \textit{oRRSi} successfully identify irrelevant documents at an early stage of query lookup, avoiding the processing overheads of \textit{TwigStackXB} and \textit{PRIX}. The \textit{RRSi} can save the IO by 11% to 53%. As the \textit{oRRSi} adopts an optimized structure to further speed up the index
lookup, it can attain IO savings from 43% to 72%. To verify the overall performance gain of our proposed indexes, the average query execution times for different indexes are compared using the 5 query sets. Figure 5.12(c) shows that the $RRSi$ and $oRRSi$ consistently process queries faster than TwigStackXB and PRIX. The $RRSi$ can attain a 7% to 39% improvement in efficiency in terms of running time. The $oRRSi$ can further improve the running time from 47% to 77% compared to TwigStackXB and PRIX.

**Figure 5.13** Comparisons of index sizes against data set sizes
Lastly, we tested the scalability of the comparing indexes by utilizing synthetic data sets generated using XMark of different sizes in the ratio of 1:2:4:8:16, namely XMARK1, XMARK2, XMARK4, XMARK8 and XMARK16. The twig query set QXMARK was run against the indexes built on each data set to compare the corresponding index sizes and execution times. Figure 5.13 and Figure 5.14 show that scalability is achieved for all the indexes while the RRSi and oRRSi achieve better space efficiency with smaller index sizes.

5.7 Conclusion

In this chapter, we have proposed the RRSi, a structure index intended to support XML queries for the rapid location of structurally relevant documents. This reduces the redundant processing of structurally irrelevant candidates that might show good content relevance. We have also introduced the rrs as the relaxed representative structure of XML
documents and used this for index nodes in the \textit{RRSi}. We have also developed a novel structural similarity measure, \textit{emSim()}, among twig queries and \textit{rrs} allowing proximity query processing using \textit{RRSi has been developed}. After careful analysis of the \textit{RRSi} index structure, an optimized index version \textit{oRRSi} which further achieves significant space and performance improvements while maintaining quality of the query results is proposed. The experimental results show that our \textit{RRSi} and \textit{oRRSi} significantly outperform previous techniques in terms of the accuracy of proximate query answers and the processing efficiency while maintaining the index scalability. The advantage is due to the fact that the documents with irrelevant structures are successfully pruned at an early stage of index lookup so as to avoid the overheads required in unnecessarily scanning irrelevant documents.
CHAPTER 6

Application – Clustering Sitemaps for Group Detection

In this chapter, we study how to cluster sitemaps as tree structured documents. We introduce a new similarity measure between sitemaps which reflects their key characteristics in the scoring. Moreover, the measure supports centroid-based clustering algorithm avoiding pair-wise comparisons that achieve significant gain in efficiency. We implemented the proposed clustering algorithm and ran extensive experiments on real and synthetic datasets and showed their effectiveness and efficiency over the clustering algorithm based on previous similarity metric. To the best of our knowledge, this is the first work to supporting the group detection by sitemaps clustering.

A major problem in the web mining field is identifying groups of websites in similar domains that have common topical interests [AC05, AIKT03, BG04, BMPW98, CNNS01, GKR98, GL03, K99, KMSY02]. Link-based ranking algorithms such as PageRank [BMPW98] and HITS [K99] were proposed and have been widely adopted in many web search engines. Their successes assume that a link to a page represents a vote from a user sustaining the quality of the targeted page. Recently, however, the volume of spam links has increased so dramatically that the value of link-based ranking results has been
significantly diminished. This is largely due to the sheer number of online organizations attempting to boost their rankings in order to attract more visitors to their websites. Although a number of proposals [CCC+06, DW05] have been made to eliminate the spam links, they are all merely heuristics presuming a predefined scenario of spam links while new and as yet unhandled artificial spam links keep appearing in the real world.

Instead, we observe that the intra-site links contain valuable information about the interests and domains of websites. A sitemap is a convenient navigation link system (with all the backward links removed). Sitemaps are provided by websites and are especially beneficial when users cannot reach all areas of a website through their browser interface. A sitemap reflects the true and key website structure while not being cluttered by the inter-site spam links. Sitemaps have been gaining in popularity and have become a standard website feature [GS05]. Although the website owners may choose to present their services or information in a variety of different ways, a certain level of similarity in web structure and content are often observed for websites in the same domain since they typically follow some evolved community standard. For example, similarities are observed when comparing the sitemaps from Princeton University [PU05] and UCLA [UC05]. Clustering sitemaps by structure helps to detect groups of websites in identical domains and its complementary to the link based ranking algorithmic function.
There have been numerous studies on group detection in the field of web mining. Considerable research [BMPW98, GKR98, K99] has been driven by the two link-based ranking algorithms: PageRank [BMPW98] and HITS [K99]. PageRank ranks a web page based on the number and quality of the pages that link to it; it computes the eigenvectors of the linking matrix through an iterative algorithm. HITS adopts a similar approach introducing the idea of hubs and authorities while computing the eigenvectors of the product of the linking matrix and its transpose. In spite of their success, the recent dramatic increase in artificial spam links has led these link analysis algorithms to provide incorrect conclusions about the quality of web pages. Although new methods [CCC+06, DW05] are proposed to address a subset of the cases, the huge profit implications associated with higher ranks and hit rates from search engines has sufficiently driven an entire new industry [GG05] specializing in creating spam information. Our focus is to cluster the intra-site links, which are not affected by spam links, providing complementary information to the link analysis algorithms to increase the quality of detected groups.

Sitemap clustering is a unique research topic; an important related issue is the clustering of tree-structured data. Extensive algorithms [BGM04, HVW01] have been proposed to
cluster tree-structural data according to structural similarities based on tree edit distance. As described in Section 2.2.1, tree edit distance is to find the minimal number of tree edit operations that can transform one tree into another (see also Section 2.2.1). In [BGM04], edit operations that allow for relocating whole document sections with an objective to generate representing DTDs for each document cluster is introduced. In [KTY05], it proposed a distance function by transforming tree-structured data into approximate numerical multidimensional vectors that is a lower bound of the edit distance. Arguably, this distance function can be applied to cluster tree-structured data. However, pair-wise evaluation of tree edit distances among documents is required in the clustering process. The computation costs of these approaches are too high for practical use. Instead, we introduce sitemap summaries with associated similarity measures, which are computed rapidly while supporting a centroid-based clustering algorithm, thereby achieving a significant gain in efficiency.

The remainder of this chapter is organized as follows. We introduce the si-graph to represent the summary of sitemaps in Section 6.1. An illustrating example with practical considerations is discussed in Section 6.2. In Section 6.3, we define the link inclusion similarity measure among si-graphs. Section 6.4 presents our sitemap clustering algorithm SICLUST. Experimental evaluation is demonstrated in Section 6.5. We conclude this chapter in Section 6.6.
6.1 SI-GRAPH

A website is a collection of web pages with a single entry web page called the homepage. Web pages are navigated via a series of intra-site directed links. A sitemap is a representation of the page links within a website – generally with all the backward links removed. A sitemap can be represented in a tree format in which nodes represent web pages while the edges correspond to intra-site links.

**Definition 6.1** A sitemap tree is defined as an ordered pair \((P, L)\) where \(P\) is the set of web pages while \(L\) is the set of directed intra-site links in the form of \((p_1, p_2)\) where \(p_1, p_2 \in P\). The function \(\text{label}()\) returns the page label of a web page.

With the objective of clustering the sitemaps, a significant efficiency gain can be attained if we can determine cluster representatives during the clustering iterations thereby avoiding tedious pair-wise comparisons. For a collection of sitemaps, we can abstract and summarize their intra-link relationships in order to generate a representative sitemap graph.

**Definition 6.2** Let \(\{(P_i, L_i)\}\) be a set of sitemaps. We define the si-graph as the ordered pair \((P', L')\) where \(P'\) is a set of nodes corresponding to the union set of page labels \(\{\text{label}(p) : p \in P_i \forall i\}\) with onto mappings \(p \in P_i \rightarrow p' \in P'\) such that \(\text{label}(p) = \text{label}(p') \forall i\). \(L' = \{(p_1', p_2') : \text{label}(p_1') = \text{label}(p_1)\) and \(\text{label}(p_2') = \text{label}(p_2)\) for some \((p_1, p_2) \in P_i\) and \(i\}\).

The si-graph is small in size with respect to the original sitemaps. It effectively summarizes the collective website structures regarding page labels and page navigation flows, because
it only stores the common page labels and links once. This structural information is important in categorizing the summarizing websites.

**Property 6.1** A *si-graph* always has a unique set of page labels and links.

This uniqueness property affirms that the *si-graph* can be regarded as the summary definition of page labels. Each page label is associated with the set of the navigation flows of the pages that originated from it. Page labels from different sitemaps or websites will not be exactly the same, however, label similarity can be evaluated through the use of a Thesaurus [M95]. Terms of either morphological variants or synonyms can be replaced by selected representative terms before going through the similarity evaluation process. Because of space constraints, specific details are not shown here.

### 6.2 Motivating Example

In this section, we discuss the advantage and demonstrate the capability in detecting groups of websites in similar domains, by web structure through a motivating example, with practical considerations.
In Figure 6.2, five sitemaps are obtained from five websites; sitemaps #1, #3, #5 represent the newspaper sites while sitemaps #2, #5 represent the bookstore sites. While content-based classifiers are popular nowadays, it is a known problem that content-based classifiers can be easily misled by the content noisy in the web pages. Intuitively, page content should reflect the interest of the websites unless the site owners maintain artificial contents in the web pages for some purposes. However, we observe that it is not the case as illustrated below. In the newspaper website #1 (with sitemap #1), it holds a business news about the online book selling market, and it also holds another entertainment news about the *Harry Potter Phenomenon* after the highly popular Harry Potter books and movies. A content-
based classifier might not be able to distinguish it from bookstore websites. Reversely, the bookstore website #2 (with sitemap #2) introduces the Business Week magazine as “…analyzes weekly business news …”. Again, this can mislead a content-based classifier comparing to other news delivering websites. Instead, we observe from the example in Figure 6.1 that the site structures contain useful information about the site groupings. Sitemaps are getting popular and reflect the true site structure with “backward links” eliminated. Structural similarity can be defined between two si-graphs as the representative graph of a group of sitemaps. This can prove to be able to support an efficient centroid-based clustering process for group detection of websites, as demonstrated in the subsequent sections. To implement this sitemap clustering process, a number of practicality considerations are discussed below.

6.2.1 Comparability of Sitemaps

While sitemap is popular and becomes standard feature of websites, their format and representation have not yet been standardized that introduces difficulty in the extraction. However, this situation will be greatly improved in the coming years given the huge collective benefit in standardizing the sitemap protocol that facilitates the web searching capability. This is consistent with the recent announcement from the three biggest web search engine providers, Google, Yahoo and Microsoft, to standardize the sitemap protocol [K06]. It proposes to make use of common XML representation [SX07] to hold standardized sitemaps and hence serve to track updates to pages. Then extensive collection and building of sitemap structures repository becomes feasible.
6.2.2 Node Label Similarity

When we compare two sitemaps or *si-graphs* by matching the node labels, it faces the problem of dealing with vocabulary discrepancies in the label names. While sitemap nodes allow free-text page description on web pages, they are mostly represented by keywords or phrases rather than sentences. Two labels are matched if they are identical or synonyms relying on a given Thesaurus. Common representation can be chosen and used to replace matched labels in subsequent matching and in the merged *si-graphs*. Realistically, this words/phrase standardization can be achieved by an ontology preprocessing consisting of a sequence of operations, such as stemming, stop words removal, phrase recognition and synonym labeling.

6.3 Link Inclusion Similarity

We define the similarity measure, *link inclusion similarity*, of a *si-graph* embedded in another *si-graph* as the representative of the underlying sitemaps. It measures the ratio of its outgoing links matched against the corresponding page label outgoing links in the *si-graph* of a webpage node.

**Definition 6.3** Given a *si-graph* $S$ as $(P_S, L_S)$ and a *si-graph* $G$ as $(P_G, L_G)$, we define $\text{incca}(p, G)$, the *link inclusion cardinality* of $p_s$ against $G$ for $p_s \in P_S$, as $|\{(p_s, p_{s2}) \in L_S : \exists (p_1, p_2) \in L_G \text{ s.t. } \text{label}(p_1) = \text{label}(p_s) \text{ and } \text{label}(p_2) = \text{label}(p_{s2})\}|$. Then we define $\text{incsim}(p, G)$, the *link inclusion similarity* of $p_s$ against $G$, as $\frac{\text{incca}(p_s, G)}{|\{(p_s, p_{s2}) \in L_S\}|}$. 
For a page label $p$, if the label itself and all the outgoing links are matched against a si-graph $G$, the page or its local page structure is said to be included in $G$. In this case, we have $incsim(p, G) = 1$. Conversely, when neither the page label nor any of its outgoing links are matched against $G$, we have $incsim(p, G) = 0$. Note that Definition 6.3 defines only the link inclusion similarities of pages in a si-graph, not the entire si-graph itself. The idea is to consolidate the local similarity values appropriately giving the inclusion similarity of a si-graph. Thus a weighting scheme is used which should give greater weightings to pages with more descendant page because higher level sitemap pages are intuitively more important in reflecting the interest of websites.

**Definition 6.4** For a si-graph $S = (P_S, L_S)$, we define recursively (1) the weight of a page $p \in P_S$ and (2) the associated weighting percentage of $p$ respectively as follows.

\[
(1) \quad pgwgt(p) = \begin{cases} 
1 + \sum_{(p_i, p) \in L_S} pgwgt(p_i) & \text{if } p \text{ is an internal page} \\
1 & \text{if } p \text{ is leaf page}
\end{cases}
\]

\[
(2) \quad pgwgt\% (m) = \frac{pgwgt(m)}{\sum_{p \in P_S} pgwgt(p)}
\]

We now define the inclusion similarity measure of a si-graph against another si-graph, which measures the extent that a si-graph is included in the navigation structure specified in the si-graph.

**Definition 6.5** For a si-graph $S = (P_S, L_S)$ and a si-graph $G$, define the link inclusion similarity as
\[ \text{incsim}(S, G) = \sum_{p_i \in PS} \left( \text{incsim}(p_i, G) \cdot \text{pgwgt}(p_i) \right) \]

The definition immediately implies the Properties 6.2 and 6.3.

**Property 6.2** For two *si-graphs* \( G_1 \) and \( G_2 \) generated by two sets of sitemaps \( \text{SET}_1 \subseteq \text{SET}_2 \) respectively, we have \( \text{incsim}(G_1, G_2) = 1 \).

Property 6.2 demonstrates the correctness with which the generated *si-graph* preserves the link structure semantics of the original sitemaps. This is consistent with the original idea for defining the measure.

**Property 6.3** For a *si-graph* \( S \) and two *si-graphs* \( G_1 \) and \( G_2 \) generated from two sets of sitemaps \( \text{SET}_1 \subseteq \text{SET}_2 \) respectively, we have \( \text{incsim}(S, G_1) \leq \text{incsim}(S, G_2) \).

Property 6.3 is an important observation in that if \( S \) is not similar enough against \( G_2 \), it will never be more similar against \( G_1 \) when representing a subset of sitemaps. This reinforces the representative power of *si-graphs* for the underlying sitemaps. When examining the definition of the \( \text{incsim}(\cdot) \) function, we can re-write its definition, for *si-graphs* \( S = (PS, LS) \) and \( G \), as

\[ \text{incsim}(S, G) = \sum_{p_i \in PS} \left( \frac{\text{pgwgt}(p_i)}{\|\{p_\beta, p_\delta \in LS \}} \cdot \text{incca}(p_i, G) \right) \]

We see that the part \( \text{pgwgt}(p_i) / \|\{p_\beta, p_\delta \in LS \} \) depends only on \( S \) (but not \( G \)). Given a *si-graph* \( S \), we can compute its dependent part only once when comparing it to different *si-
graphs $G$. This effectively accelerates the computational efficiency in this scenario as demonstrated in Theorem 6.1.

**Theorem 6.1** Given a si-graph $S$, we have a linear computational complexity of $\text{incsim}(S, G)$ as $O(|G|)$ for any si-graph $G$.

**Proof** With a cached weighting of $S$, we only need to compute $\text{incca}(p_s, G)$ by matching si-graph links of $p_s$ against $G$ in linear time.

In order to apply si-graphs as centroids in the sitemap clustering process, we need a similarity measure to compare two si-graphs and hence we extend the $\text{incsim}()$ into a symmetric measure.

**Definition 6.6** We define the si-similarity, $\text{sisim}(G_1, G_2)$, as $(\text{incsim}(G_1, G_2) + \text{incsim}(G_2, G_1)) / 2$ where $G_1$ and $G_2$ are two si-graphs.

The complexity argument in Theorem 6.1 generally applies to $\text{sisim}()$ as a function of the $\text{incsim}()$. Given a set of si-graphs $\{G_i\} = \{(P_{G_i}, L_{G_i})\}$, we can pre-compute for each page $p_{G_i}$ the independent weighting component $\text{incsim}(p_{G_i}, G_i) / |\{(p_{G_i}, p) \in L_{G_i}\}|$ and cache it. During runtime, we only need to count the number of common links between two si-graphs that achieve a linear time complexity.

**Theorem 6.2** $\text{sisim}(G_1, G_2)$ attains a linear computational complexity of $O(|G_1| + |G_2|)$ for two fixed si-graphs $G_1, G_2$. 
6.4 Detecting Sitemap Clusters

We employ an agglomerative hierarchical clustering algorithm SICLUST for sitemaps that require no prior knowledge of the number of clusters. Starting with each sitemap as a separate cluster, the process is to merge the nearest clusters until there are only $k$ clusters left. *Si-graphs*, as cluster centroids, are generated on each iteration. *Si-similarity* scores are computed among them measuring the *closeness* among clusters on the centroid-level that avoids the time-consuming pair-wise comparison. Other exit criteria, such as: the number of clusters, the minimum cluster size, and the minimum similarity score among clusters, can also be adopted in the clustering process.

```
Input: a set of sitemaps ($S_i$)
Output: Clusters of sitemap $\{C_i\}$
1) $\{C_i\} = \{\{S_i\}\}$ /* started with each sitemap as standalone cluster */
2) $\{G_i\} = \text{generate_si_graphs}(\{C_i\})$
3) $(sisim_{ij}) = \text{compute_sisim_matrix}(\{G_i\})$
4) do
5) $(k, l) = \text{find_nearest_clusters}(\{sisim_{ij}\})$
6) $\{C_i\} += \{C_k \cup C_l\} \backslash \{C_k, C_l\}$
7) $\{G_i\} = \text{generate_si_graphs}(\{C_i\})$
8) $(sisim_{ij}) = \text{compute_sisim_matrix}(\{G_i\})$
9) $\text{record_cluster_attributes}(\{C_i\})$
10) while $|\{C_i\}| > k$
```

*Figure 6.3 SICLUST Algorithm*

In the SICLUST algorithm, $\{C_i\}$ represents the current clusters and, in Line 2, is initialized by each sitemap as an individual cluster. In Lines 2 and 7, the *generate_si_graphs*
procedure generates the set of \( \text{si-graphs} \ \{G_i\} \) from the input cluster set. The \texttt{compute_sisim_matrix} procedure compute, in Lines 3 and 8, the inter-cluster si-similarity scores matrix \( \{sisim_{ij}\} \) as the parameter used to determine the nearest clusters in the \texttt{find_nearest_clusters} procedure in Lines 5. Line 6 merges the nearest clusters forming new current clusters. The \( \{G_i\} \) and \( \{sisim_{ij}\} \) are re-computed accordingly using the new current clusters. The same process goes on until only \( k \) clusters are left (Line 10). Note that the \texttt{record_cluster_attributes} procedure records the cluster information that can allow other choices of exit criteria. We borrow the idea from \textit{ROCK} [GRS99] to improve the both the efficiency and the effectiveness. For instance, we include consideration of merging in clusters when two clusters are not close enough to each other but share a large number of neighboring clusters. This illustrates that this method can be more effective on categorical data to which our sitemaps or \textit{si-graphs} belong to. Moreover, we can speed up the \texttt{find_nearest_clusters} procedure by caching along with each cluster and its si-similarities against all the other clusters in descending order.

We illustrate the \textit{SICLUST} algorithm using the example in Section 6.2. We start with each sitemap as separate clusters and we generate the \textit{si-graphs} from each cluster. Then we calculate the \textit{si-similarity} between each pairs of \textit{si-graphs} and then merge the closest clusters and go to the next iteration until we meet the exit criteria. The clustering iterations are shown below.

\textbf{Iteration #1}
We start with each sitemap as separate cluster and generate the *si-graphs* (sig_A#) per cluster. *sig_A1* and *sig_A3* generated from Sitemaps #1 and #3 in Clusters A1 and A3 respectively are shown in Figure 6.5. As there is no repeating node in the sitemaps and it is the only one sitemap in each cluster, the generated *si-graphs* are the same as the tree representations of the corresponding sitemaps.

Then we calculate and compare the *si-similarity* for *si-graph* pairs and choose the closest clusters to merge. In Figure 6.5, the node weightings (pgwgt) and link inclusion similarity (incsim) of *sig_A1* against *sig_A3* and vice versa respectively are shown. From that we calculate the
\[
\text{incsim}(\text{sig}_A1, \text{sig}_A3) = \frac{10 \times \frac{2}{2} + 6 \times \frac{2}{3} + 3 \times \frac{1}{2} + 1 + 3 \times \frac{1}{2} + 0 + 0 + 1 + 0 + 1 + 1 + 1}{10 + 6 + 3 + 1 + 3 + 1 + 1 + 1 + 1 + 1} = 0.71
\]

\[
\text{incsim}(\text{sig}_A3, \text{sig}_A1) = \frac{11 \times \frac{2}{2} + 6 \times \frac{2}{3} + 0 + 1 + 1 + 3 \times \frac{1}{2} + 3 \times \frac{1}{2} + 1 + 0 + 1 + 0}{11 + 6 + 4 + 1 + 1 + 3 + 3 + 1 + 1 + 1 + 1} = 0.67
\]

Hence, \(\text{sisim}(\text{sig}_A1, \text{sig}_A3) = \frac{0.71 + 0.67}{2} = 0.69\)

We follow similar calculations for every si-graph pairs and we found that \(\text{sisim}(\text{sig}_A2, \text{sig}_A4) = 0.76\) attains the highest value. Hence we merge clusters A2 and A4 and go to the next iterations. The same merging iterations are repeated with the new set of clusters until even the nearest clusters are no longer close enough as specified by the pre-determined exit criteria, say a similarity threshold of 0.5 as illustrated in Figure 6.6.

**Iteration #2**

![Cluster B1, Cluster B2, Cluster B3, Cluster B4]

Closest clusters are Clusters B1 and B3 with \(\text{sisim}(\text{sig}_B1, \text{sig}_B3) \geq 0.5\)

**Iteration #3**

![Cluster C1, Cluster C2, Cluster C3]

Closest clusters are Clusters C1 and C3 with \(\text{sisim}(\text{sig}_C1, \text{sig}_C3) \geq 0.5\)
6.5 Experimental Evaluation

In this section, we study the accuracy and effectiveness of our proposed SICLSUT algorithm on both real and synthetic data. Experiments are carried out on a PC with an Intel 2 GHz processor with one GB memory running on Windows XP. We compared our proposed algorithm against two other algorithms, TEDIT_C and TEDIT_P, based on tree edit distance metric. In Algorithm TEDIT_C, we used the tree edit distance method as a similarity measure between si-graphs as centroids replacing the sisim() function in our proposed clustering algorithm in Figure 6.3. In Algorithm TEDIT_P, we also adopted the tree edit distance method and clustered data by a hierarchical clustering algorithm like algorithm TEDIT_C. However, we do not make use of the proposed si-graphs as centroids and instead we compare every pair of sitemaps respectively from comparing the clusters to determining the merging clusters for the next iterations.
6.5.1 Real Data

We chose three sets of sitemaps from three different domains of websites, including universities (U) [YAH06a], bookshops (B) [YAH06c], and newspapers (N) [YAH06b]. The sitemaps were converted into the sitemap trees. The dataset contains 100 websites from each domain.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>TEDIT_C</th>
<th>TEDIT_P</th>
<th>SICLUST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>U(57)</td>
<td>B(58)</td>
<td>N(46)</td>
</tr>
<tr>
<td></td>
<td>U(94)</td>
<td>B(96)</td>
<td>N(93)</td>
</tr>
<tr>
<td></td>
<td>U(75)</td>
<td>B(11)</td>
<td>N(20)</td>
</tr>
<tr>
<td>2</td>
<td>U(22)</td>
<td>B(23)</td>
<td>N(34)</td>
</tr>
<tr>
<td></td>
<td>U(6)</td>
<td>B(4)</td>
<td>N(7)</td>
</tr>
<tr>
<td></td>
<td>U(12)</td>
<td>B(69)</td>
<td>N(9)</td>
</tr>
<tr>
<td>3</td>
<td>U(21)</td>
<td>B(19)</td>
<td>N(20)</td>
</tr>
<tr>
<td></td>
<td>U(0)</td>
<td>B(0)</td>
<td>N(0)</td>
</tr>
<tr>
<td></td>
<td>U(13)</td>
<td>B(20)</td>
<td>N(71)</td>
</tr>
</tbody>
</table>

Table 6.1 Clustering Results for Real Dataset

Table 6.1 shows the comparison results of using Algorithms TEDIT_C, TEDIT_P, and the proposed clustering Algorithm SICLUST. In Algorithm TEDIT_C, the majority of the sitemaps are in Cluster 1. There are 57%(U) 58%(B) 46%(N) in Cluster 1, 22%(U) 23%(B) 34%(N) in Cluster 2, and 21%(U) 19%(B) 20%(N) in Cluster 3. We observe that the tree edit distance method fails to identify sitemaps from the same domain that typically have different structures at lower level nodes. In Algorithm TEDIT_P, we obtain a cluster containing the most sitemaps: 94%(U) 96%(B) 93%(N). Cluster 2 contains only a few sitemaps while we have an empty Cluster 3. In Algorithm SICLUST, the Clusters 1, 2, and 3 contain the majority of 75%(U), 69%(B), and 71%(N) respectively; this significantly improves the quality of the clustering results.
6.5.2 Synthetic Data

We analyze the accuracy of the \textit{SICLUST} clustering algorithm using synthetic data. In order to simulate and generate sitemaps belonging to pre-determined domains, we treat sitemaps as XML documents. Using the XTRACT algorithm [GGR+00], we generated the three DTDs from the previous three real datasets of one hundred sitemaps which belong to the domains (U), (B) and (N) respectively. Then five sets of sitemaps are generated from the three DTDs (by limiting to zero or one instance for the “*” operator). Mixed from the three domains, the five datasets, S1, S2, S3, S4 and S5, contain 1000, 2000, 3000, 4000 and 5000 sitemaps respectively. The depth of these sitemap trees range from two to seven.

We measure the accuracy of the cluster results by function of the number of correct classified sitemaps over the total number of sitemaps. This could be determined without difficulty because we knew each sitemap and their correct corresponding clusters. We evaluate the accuracy of the clustering algorithms with respect to (a) the synthetic datasets and (b) the number of clusters. Figure 6.7(a) and 6.7(b) demonstrates the improved accuracy of the \textit{SICLUST} algorithm over the \textit{TEDIT\_C} and the \textit{TEDIT\_P} algorithms. Consequently, this concludes that \textit{SICLUST} generates clusters of a better quality in comparison to those of traditional clustering algorithms.
Figure 6.7 Accuracy of Clustering Results for (a) Synthetic Datasets (b) Number of Clusters
6.6 Conclusion

In this study, we proposed a framework for clustering sitemap data for detecting groups of websites in similar domain. We introduced the *si-graph* as the summarized structure reflecting the structural characteristics of the underlying sitemaps. We defined the *sisim()* measure to quantify the similarity between sitemaps. We made use of the *sisim()* measure and introduced an unsupervised clustering processing which is centroid-based and thereby avoids pair-wise element comparisons and ultimately speeds up the process.

To conclude, this work successfully applied clustering techniques for classifying websites using the intra-site graph. The experimental results demonstrated the effectiveness and accuracy of the clustering process and the lookup algorithm over the algorithms based on traditional methods for detecting the structural similarity.
CHAPTER 7

Conclusion and Future Work

7.1 Conclusions

There is an increasing need to be able to determine the structural similarity among XML documents in the view of their underlying document categories, in order to support the automated processing XML documents for information retrieval. In this dissertation, we propose a measure to detect the structural similarity of an XML document against a schema. It successfully captures the structural characteristics of XML documents and always gives the value of ‘total match’ for documents generated from a given schema. We also present an algorithm to obtain the measure, which is quickly to run in practice. Then we further extend and define another measure detecting the structural similarity between XML documents.

From the application point of view, we apply the structural similarity to indexing and querying semi-structure data. Through the introduction of the structure indexes $RRSi$ and $oRRSi$, they support quick location of the structurally relevant documents for XML queries. Also, we observe that sitemaps contain valuable information regarding the categories of websites. We define the sitemap graph as the structural summary of sitemaps and propose
an effective framework for clustering sitemap data for detecting groups of websites in similar domain.

In conclusion, this dissertation introduces a new structural measure capable to determine the document categories. We demonstrate the applicability in the document indexing, the proximity query support and the web group detection. It is believed that this similarity measure can be applied to a number of additional problem domains, and this dissertation opens a new strategy and direction to drive the automated XML document processing.

7.2 Future Work

Our contributions in this dissertation open a number of avenues for further work. We present a few interesting research directions in the following sub-sections.

7.2.1 XML Clustering by Document Structure and Content Vector

The proposed *si-graph* and the structural similarity measure *sisim* in the Chapter 6 can be applied to XML clustering. While it works well in clustering documents according to the document types, it does not take into consideration for the content values of documents. In some scenarios that there are groups of documents with similar structures and contents, the current approach has not yet been able segregate these groups of documents with similar structures but different contents.
We need further extensions in order to handle structural similarity of document structures as well as the content values. It is our intention to propose an enhance similarity measure for XML document comparisons on both structures and contents. It is because content comparison might not be meaningful if they appear under different document structures. In particular, it does not make sense to compare a song name \(<\textit{song}>\textit{name}\> \ldots \textit{name}\)</song> against an author name \(<\textit{author}>\textit{name}\> \ldots \textit{name}\)</author>. Our initial idea is to extract the content values as keyword vectors attached to the proposed structural summary. The content similarity can be defined as a function of the weighted vector distance. In this fashion, we can maintain a centroid-based clustering algorithm that is essential for the efficiency.

### 7.2.2 Correlating XML Data Streams

The problem of correlating continuous streams of XML data through approximate matching is first studied in [GK03]. It proposes an online algorithm for correlation that the general tree-edit distance metric plays a key role in achieving approximate matching. However, it is the same reason that the metric suffers the same deficiency in identifying structurally similar documents of the same types while they might appear differently as document trees. As demonstrated, our proposed structural similarity metric is capable to detect documents of similar types comparing to the tree-edit distance metric, and hence we are planning to study its applicability to this problem. The main challenge is the space and efficiency consideration in the streaming environment. We will need to resolve the potential issue of the prerequisite in generating the implied schemas from both comparing documents which can increase the space requirement and affect the efficiency.
7.2.3 Detecting XML Duplicates

The problem of duplicate detection is the identification of different representations in data sources representing the same real-world entity, which has been addressed extensively for relational data [ACC02, EM97, FGS+01, HR01]. However, the duplicate detection on XML data is a recent research topic [CD03, NF05, W05], two major challenges of this problem is the object identification and structural heterogeneity. Object identification is to determine the data values from XML documents describing a real-world object. Structural heterogeneity refers to the challenge of XML documents with heterogeneous structure while describing the same kind of object. Although [NF05] attempts to consider the structural diversity by schema mapping, it relies on expert input is the candidate selection while no automated algorithm is proposed. We believe that our proposed structural measure \( schSim() \) can fill this gap and be adopted to resolve this problem effectively.
References


ACM SIGIR Conference on Research and Development in Information Retrieval, 88-95, Sheffield, United Kingdom.


Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery, 23-29, Tuscon, Arizona.


International Joint Conference on Artificial Intelligence, 1-6, Seattle, Washington, USA.


