Mining Evolution of Structure of Semi-Structured Web Data

Zhao Qiankun

School of Computer Engineering

A thesis submitted to the Nanyang Technological University in FULFILMENT of the requirement for the degree of Doctor of Philosophy

2007
Acknowledgments

First and foremost, I would like to thank my supervisor: Professor Sourav S Bhowmick. Doing research with him was one of the best academic experiences I had. I have learned a lot from him about doing good research, problem solving methodology, technical writing and speaking, and much more.

I am grateful to many other professors that I have learned a lot from during the last four years of my PhD life. I’d like to thank Professor Ng Wee-Kong, Professor Lim Ee-Peng, and Professor Sun Aixin for their advices and supports all the time.

During my research, I am glad to collaborate with many other professors. Thanks to Professor Sanjay Madria at University of Missouri-Rolla, Professor Le Gruenwald at the University of Oklahoma, and Professor Michael R. Lyu at The Chinese University of Hongkong.

I also benefited a lot from experiences of collaborating with many other researchers from the industry. Thanks to Dr. Mukesh Mohania of IBM India Research Lab, Dr. Tie-Yan Liu and Dr. Wei-Ying Ma of Microsoft Research Asia.

Also thanks to all my friends for the discussion we had about research and life. I am grateful to all of them, especially Dr. Steven C-H Hoi at The Chinese University of Hongkong, Mr. Tao Qin and Mr. Guang Feng at Tsinghua University, Mr. Bin Gao, Mr. Kai Yi, Mr. Huai-Yuan Yang, and Mr. Ming-Qiang Hou at Peking University, Dr. Xiaoyu Fu at The University of Texas at Dallas, Dr. Gui-Rong Xue and Mr. Feng Zhou at Shanghai Jiaotong University, Dr. Wensi Xi and Dr. Xin Zheng at Google, Dr. Jiantao Sun at Microsoft Research Asia, and Mr. Wujun Li at UST HK. Thanks to all the interns I met in MSRA.

Thanks to School of Computer Engineering and Nanyang Technological University for providing the general financial aid during my whole PhD candidature. Also thanks to Ms.
Lim-Tan Lay Choo and Mr. Lai Chee Keong in Centre for Advanced Information Systems for providing technical supports and the facilities.

Thanks to my fellow friends Dr. Binshou Xiong, Dr. Zhuowei Li, Dr. Zhao Li, Dr. Zehua Liu, Dr. Duan Liu, Mr. Jianshu Weng, Dr. Erwin Leonardi, Mr. Baoyao Zhou, and Dr. Wenyuan Li for encouraging discussions related to both research and life.

Last but not the least, I would like to thank my parents, sister, and brother for their supports throughout the years. I could not have come this far without their constant love and supports.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>i</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>Abstract</td>
<td>ix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Key Features of Web Data</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Novel Research Issues</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Limitation of State-of-the-art Web Mining Techniques</td>
<td>10</td>
</tr>
<tr>
<td>1.4 Overview</td>
<td>13</td>
</tr>
<tr>
<td>1.5 Contributions</td>
<td>20</td>
</tr>
<tr>
<td>1.6 Organization of the Dissertation</td>
<td>21</td>
</tr>
<tr>
<td><strong>2 Literature Review</strong></td>
<td>23</td>
</tr>
<tr>
<td>2.1 Mining Semi-Structured Data</td>
<td>23</td>
</tr>
<tr>
<td>2.2 Analyzing of Web Data Dynamics</td>
<td>47</td>
</tr>
<tr>
<td>2.3 Summary</td>
<td>56</td>
</tr>
<tr>
<td><strong>3 Mining XML Document Versions</strong></td>
<td>60</td>
</tr>
<tr>
<td>3.1 Motivation</td>
<td>62</td>
</tr>
<tr>
<td>3.2 Overview and Background</td>
<td>64</td>
</tr>
<tr>
<td>3.3 FCS Mining</td>
<td>67</td>
</tr>
<tr>
<td>3.4 Extraction of Interesting FCS</td>
<td>86</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>92</td>
</tr>
<tr>
<td><strong>4 Mining Historical XML Query Patterns</strong></td>
<td>93</td>
</tr>
<tr>
<td>4.1 Motivation</td>
<td>94</td>
</tr>
<tr>
<td>4.2 An Overview of Mining Historical XML Queries</td>
<td>98</td>
</tr>
<tr>
<td>4.3 Mining Conserved XML Query Paths</td>
<td>103</td>
</tr>
<tr>
<td>4.4 Dynamic-Conscious Cache Strategy</td>
<td>118</td>
</tr>
</tbody>
</table>
4.5 Performance Study .......................................................... 122
4.6 Summary ........................................................................... 132

5 Mining Evolution of Website Log Data .................................... 134
5.1 Web Usage Mining ............................................................... 135
5.2 Motivation ......................................................................... 139
5.3 WAM Mining ..................................................................... 140
5.4 CLEOPATRA: Evolutionary Pattern-based Clustering of WAS .... 152
5.5 Integrated Website Event Detection: iWED .......................... 162
5.6 Summary ........................................................................... 175

6 Building Time-Dependent Semantic Query Similarity Model .... 176
6.1 Background ......................................................................... 177
6.2 Motivations .......................................................................... 179
6.3 Time-Dependent Query Similarity Model ............................. 182
6.4 Empirical Evaluation .............................................................. 187
6.5 Summary ........................................................................... 195

7 Event Detection From Evolution of Click-Through Data ......... 196
7.1 Background of Event Detection Problem .............................. 197
7.2 Motivation ......................................................................... 198
7.3 Event Detection From Evolution of Click-Through Data ....... 200
7.4 Performance Evaluation ......................................................... 210
7.5 Summary ........................................................................... 216

8 Conclusions and Future Work .............................................. 217
8.1 Summary and Contributions .................................................. 217
8.2 Future Research Directions ................................................... 220

References .............................................................................. 223

A List of Patents and Publications ............................................ 239
List of Figures

1.1 Types of Web data ............................................. 3
1.2 Examples XML document versions .................................. 4
1.3 Example of Web search engine log dynamics ..................... 6
1.4 Web mining categorization ....................................... 11
1.5 The MONETA framework ....................................... 13

2.1 Example of graphs and trees ...................................... 24
2.2 Example of subgraphs and subtrees ................................ 26
2.3 Example of mining frequent bottom-up subtrees .................. 29
2.4 Number of different types of subtrees ............................ 31
2.5 Example of TreeMiner algorithm ................................. 32
2.6 Enumeration and occurrence list in FREQT algorithm ............ 33
2.7 Canonical representation of unordered tree ...................... 34
2.8 Candidate Generation in FreeTreeMiner algorithm ............... 36
2.9 Candidate Generation in FSC algorithm ........................ 39
2.10 DFS code tree in gSpan algorithm ................................ 40
2.11 Canonical Adjacency Matrix used in FFSSM algorithm .......... 42
2.12 Example of XMINE ............................................. 44
2.13 Example of TreeFinder .......................................... 45

3.1 Examples of XML document dynamics ............................. 61
3.2 Tree representation of XML fragment ............................ 65
3.3 Version 2 and version 3 of the XML fragment .................... 66
3.4 Part of the H-DOM Tree ......................................... 72
3.5 FCS Experiment Results ......................................... 84
3.6 An Example of Concept Hierarchy ................................ 87
3.7 DTD of the real Yahoo auction data .............................. 88
3.8 Efficiency and scalability of FASST .............................. 89
3.9 Effect of FASST on the thresholds of evolution metrics and results of compactness of H-DOM+ tree ......................... 90
3.10 Effect of depth of concept substructures and number of interesting FCS. . . . 90
3.11 Evolutionary features of various categories. ................................. 92

4.1 Example of QPTs and frequent QPTs. ........................................ 95
4.2 Support history of RQPs. ........................................................... 98
4.3 Representing historical XML queries. ......................................... 100
4.4 Example of QPG-tree and HQPG-tree. ...................................... 107
4.5 Dynamic-Conscious caching Strategy ........................................ 120
4.6 Experimental results (1). ............................................................ 124
4.7 Experimental results (2). ............................................................ 125
4.8 Experimental results (3). ............................................................ 128
4.9 Experimental results (4). ............................................................ 130
4.10 Effects of q value. ................................................................. 132

5.1 Types of Web usage/log data. .................................................... 135
5.2 WAS related examples .............................................................. 141
5.3 WAM experiment results. ........................................................ 148
5.4 Size of the H-WAS tree. .......................................................... 149
5.5 Support of WASs over a time period ........................................ 152
5.6 Web data representation .......................................................... 162
5.7 Web data representation .......................................................... 166

6.1 Bipartite graph representation of click-through data. ..................... 177
6.2 Incremental and interval-based methods for similarity measure ........ 180
6.3 Examples of query-page dynamic (1) ...................................... 190
6.4 Examples of query-page dynamic (2) ...................................... 191
6.5 Quality of the Time-dependent model (2) ................................. 194
6.6 Quality of the Time-dependent model (3) ................................. 195

7.1 The Framework for event detection from click-through data .......... 201
7.2 Example of vector-based and dual graphs. .................................. 205
7.3 Example of events detected from the real dataset. ....................... 214
List of Tables

1.1 Examples of XML query log. .............................................. 5
1.2 Examples of Web search engine log. ................................. 5
1.3 Examples of Web access sequences. ................................. 7
1.4 Historical support values for \textit{WASs} ............................. 12
1.5 Summary of the mining approaches. ................................. 15

2.1 Summary for the three types of subtrees. .......................... 27
2.2 Summary for the three types of subgraphs. ........................ 27
2.3 Sequential pattern mining .............................................. 49
2.4 Comparison of existing Web mining approaches with MONETA. 57

3.1 Examples of XML query dynamics .................................... 61
3.2 A summary of symbols ............................................... 65
3.3 Symbols, Descriptions, and Datasets ............................... 82
3.4 Datasets for evaluation of FASST. .................................. 88

4.1 Example QPTs in the DBLP and SSPlay Datasets .................. 123
4.2 Comparison of WCQP-Miner and RCQP-Miner results. ........... 128
4.3 Effects of \textit{q} in cost ratio ........................................ 130

5.1 Example of \textit{WASs} .................................................. 136
5.2 Synthetic datasets .................................................... 147
5.3 WAM experimental results. .......................................... 150
5.4 CLEOPATRA experimental results using DBSCAN. ............... 160
5.5 CLEOPATRA experimental results using Graph Cut. ............. 161
5.6 Event Detection Results ............................................. 173

6.1 Example of the click-through data. .................................. 177
6.2 Quality of the Time-dependent model (1) ........................... 193

7.1 Examples of events. .................................................. 212
7.2 The cluster-event contingency table. ........................................... 212
7.3 Event detection results. ............................................................ 214
Abstract

Web mining is a converging research area from several research communities, such as database, information retrieval, machine learning and natural language processing. In the literature, many research efforts have been directed to Web mining. However, we observed that existing Web mining approaches mainly focused on the semi-structured and the massiveness properties of Web data, whereas there is no systematic research that discovers knowledge by mining the dynamic property of Web data. We believe that knowledge hidden behind the dynamic nature of Web data is also important and useful in many applications such as dynamic-conscious XML cache strategy, intelligent Web advertisement placing scheme, and event detection from Web data.

Taking the dynamic property of Web data into account, we proposed a framework, called MONETA (Mining evOlutioN of sEmi-strucTured web dAta), to mine novel knowledge from the evolution of semi-structured Web data. The MONETA framework consists of three major components. Given a collection of historical semi-structured data, firstly, an efficient and compact representation is required to record both the structural and temporal information. Then, a set of dynamic metrics are designed to measure the dynamic behaviors of the data source. Lastly, a set of algorithms are proposed to extract novel knowledge that satisfies constraints in terms of the dynamic metrics.

In this dissertation, the MONETA framework is evaluated with both author-centric data and visitor-centric data such as XML document versions, XML queries, Website log data, and click-through data. The data sources are modelled as tree and graph structures. Also, we have designed sets of dynamic metrics to measure both the micro-patterns and macro-patterns. Based on the representation and dynamic metrics, we have proposed different mining techniques to extract the following four types of novel knowledge. Note that in this dissertation, we focus on Web structural data.

- Discovering substructures of Web data that have specific change patterns over time.
- Clustering substructures of Web data based on their change patterns over time.
- Modeling the change patterns of certain Web data substructures over time.
- Extracting semantics from the change patterns of Web data substructures over time.
Extensive experiments have been conducted using both synthetic dataset and real world datasets obtained from Yahoo! auction and MSN Web search engine. We evaluate both the efficiency and scalability of our proposed algorithms, quality of the mining results, and the usefulness of the discovered knowledge using some real applications. Our results showed that our MONETA framework is flexible and the proposed evolution mining techniques are efficient and can produce novel and useful knowledge for many applications.
Chapter 1

Introduction

The World Wide Web serves as a global information infrastructure that covers a huge amount of data from essentially every area such as news, business, entertainment, government, research, and many others. This fact makes Web data a rich source for various users such as Web surfer, business organization, government, etc. However, the massive amount of Web data does not imply that Web users can get whatever they want more easily. On the contrary, it overwhelmed their abilities to find the desired information. It has been claimed that 99% of the Web data is useless to 99% of the users [78]. Moreover, this problem is further complicated by the semi-structured [1] and dynamic nature of Web data. That is, Web data lacks of rigid structure and it is evolving over time. As a result, Web search engines based on only keyword-based search techniques return poor quality search results. Under such circumstances, Web mining techniques, which were proposed to automatically extract hidden knowledge from Web data and services [64], were utilized to improve the quality of keyword-based search results. For instance, link analysis techniques [131], part of the Web mining techniques, have been used by Google\(^1\) to rank the search results based on the quality of Web pages.

However, we observed that existing Web mining approaches mainly focused on the semi-structured and massiveness properties, whereas there is no systematic research that exploits the dynamic property of Web data to answer the richer variety of queries. In this dissertation, we present a general framework, to systematically discover novel knowledge from Web data.

\(^1\)http://www.google.com
by incorporating the dynamic and semi-structured properties of Web data. Specifically, we discuss the framework and techniques for extracting novel knowledge from the *structural evolution* of Web data.

In this chapter, we begin by describing the key features of Web data in Section 1.1. In Section 1.2, the challenging problems led to by the dynamic nature of Web data are discussed and the importance of solving these problems is illustrated with examples in the context of *author-centric* data and *visitor-centric* data. Then, in Section 1.3, we motivate the need for a new framework and mining techniques by summarizing existing techniques and highlight their inability to solve these challenging problems efficiently in this context. In Section 1.4, we present an overview of the dissertation. We begin by summarizing the objective of this dissertation. Then, the MONETA (Mining evOlutioN of sEmi-strucTured web dAta) framework is discussed by describing the architecture. We also discuss the types of novel knowledge that can be discovered under the MONETA framework and define the scope of research. Moreover, a list of applications is presented to illustrate the usefulness of the discovered knowledge as well. In Section 1.5, we summarized the contributions of this dissertation. Finally, Section 1.6 outlines the organization for the rest of this dissertation.

### 1.1 Key Features of Web Data

#### Size of Web Data

One of the key features of Web data is that the size of Web data is huge. It was predicted that majority of human information will be available on the Web in the very near future in the Asilomar Report [26]. At the same time, the amount of Web data available has increased dramatically in the last decade. For instance, more and more data, such as hard copy books and films, are emigrating from traditional data repository to the Web. Google book search \(^2\) is one of the examples that transfer the hard copy materials to the Web that can be accessed by any user. More importantly, massive volumes of new data are created everyday for different purposes such as personal blog, release of new products, and release of new products.

\(^2\)http://books.google.com/
CHAPTER 1. INTRODUCTION

Figure 1.1: Types of Web data.

events that are happening around the world and the corresponding comments, and Web log data generated with the usage of the Web.

**Semi-structured Nature**: The second key feature of Web data is that it is *semi-structured*. In [34], semi-structured data is defined as data that is neither raw data, nor very strictly typed as in conventional database systems. Basically, the structure of semi-structured data is irregular, implicit, and partial. Web data consists of two types of semi-structured data. They are HTML and XML. A typical example of semi-structured data type is the HTML file format, which has some structuring primitives such as tags and anchors. For instance, take the HTML format Web pages that store the shopping malls in Singapore as examples. For each shopping mall, there is an entry with information such as address, phone number, major products, and transportation. However, there is a large degree of irregularity in the structures since not all the shopping malls are treated in a uniform manner and the information is recorded as plain text, which may or may not follow the standard rigidity of certain data parser.

**Types of Data**: The third key feature of Web data is that it can be created by any user in different ways. As shown in Figure 1.1, Web data can be classified into *surface Web data* and *deep or hidden Web data*. *Surface Web data* refers to the set of Web pages reachable by purely following the hyperlinks, ignoring search forms and pages that require authorization or prior registration [104]. *Deep Web data* refers to a large portion of Web data that are
hidden behind search box and dynamically generated from the back end database in response to the submitted queries [24]. It has been estimated that deep Web is 500 times larger than the surface Web [25]. Moreover, both surface and deep Web data can be classified into author-centric data and visitor-centric data. Here, author-centric data refers to Web data that are created and stored for sharing and browsing purposes such as general Web page in the context of surface Web data and back end data in the context of hidden Web data. The author-centric data can be HTML documents or XML documents. Visitor-centric data refers to Web data that are created or generated by users for specific browsing and searching purposes such as Website usage log data in the context of surface Web data and the Web query log data (XML query log and Web search engine query log) in the context of hidden Web data. Note that author-centric data represent Web data that are created by the author of the Web pages, whereas visitor-centric data represent Web data that are created by visitors of the Web when they are interacting with Web services/sites.

Dynamic Nature: Additionally, dynamic feature of Web data is a key issue that must be considered. That is, Web data may evolve over time. New data may be created at any time and existing data may be updated in different ways. Let us elaborate on the dynamic nature of Web data in the context of both author-centric data and visitor-centric data.

Author-centric Data: Figure 1.2 shows three example versions of an XML document representing parts of the publication data of a researcher at three different time points. The
CHAPTER 1. INTRODUCTION

Table 1.1: Examples of XML query log.

<table>
<thead>
<tr>
<th>ID</th>
<th>XML query</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>/professor/bio/industry</td>
</tr>
<tr>
<td>102</td>
<td>//research/activity/conf</td>
</tr>
<tr>
<td>103</td>
<td>/professor/publication</td>
</tr>
<tr>
<td>104</td>
<td>//bio/edu/phd</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>201</td>
<td>/professor/bio/academic</td>
</tr>
<tr>
<td>202</td>
<td>//bio/edu/ms</td>
</tr>
<tr>
<td>203</td>
<td>//research/activity/conf</td>
</tr>
<tr>
<td>204</td>
<td>//research/activity/conf</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(a) The first week (b) The second week

Table 1.2: Examples of Web search engine log.

<table>
<thead>
<tr>
<th>IP address</th>
<th>Query</th>
<th>Page</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx.xxx.xxx</td>
<td>BMW</td>
<td><a href="http://www.bmw.com">http://www.bmw.com</a></td>
<td>14:02 02112005</td>
</tr>
<tr>
<td>xxx.xxx.xxx</td>
<td>SVM</td>
<td><a href="http://svm.first.gmd.de">http://svm.first.gmd.de</a></td>
<td>15:31 03252005</td>
</tr>
<tr>
<td>xxx.xxx.xxx</td>
<td>MSN</td>
<td><a href="http://www.MSN.com">http://www.MSN.com</a></td>
<td>21:14 02142005</td>
</tr>
</tbody>
</table>

publications are organized according to the areas they belong to. It can be observed that the XML document may change in different ways. For instance, there may be more and more papers inserted under /publication/DM, whereas the segment of /publication/XML does not change at all. Such changes may be contributed to various facts such as the researcher may be active in the DM area but not active in the XML area anymore. Or, the researcher may move to some new areas. Note that other than insertions in the above examples, there may other types of changes such as deletion and update for XML document.

Visitor-centric Data: To illustrate the dynamic feature of the visitor-centric data, we take the XML query log, Web search engine log, and Web usage log data as examples.

- **XML Query Log**: Table 1.1 presents the list of XML queries that were issued against the XML documents shown in Figure 1.2 in two weeks. It can be observed that there may be new XML queries emerging and old queries disappearing from time to time. Moreover, the number of times a specific XML query was issued may change from one week to another. For instance, query 201 is a new query that appeared in the second week, while query 101 disappeared in the second week. Queries 102 and 203 are the same, whereas it occurred more frequently in the second week than in the first week. The underlying reasons for such changes may be the content changes to the XML documents or the shift of user interests over time.
CHAPTER 1. INTRODUCTION

Figure 1.3: Example of Web search engine log dynamics.

- **Web Search Engine Log:** Web search engine log data records the queries issued by users against Web search engines and the corresponding Web pages being clicked. Table 1.2 presents a list of example query and page transactions in Web search engine log data. Note that the IP addresses are not shown in the table due to the privacy issue. Each transaction in the table records the co-occurrence of a query and a Web page. From the real data obtained from MSN Web search engine\(^3\) (http://search.msn.com/), we observed that not only the frequencies of queries being issued and Web pages being visited changed over time, but also the co-occurrences between queries and Web pages changed over time as well. For instance, Figure 1.3(a) shows how the frequency of query “Shania Twain” changed over time. Figure 1.3(b) shows how the frequency of a “Christmas” Web page being visited changed over time. Figure 1.3(c) shows how the co-occurrences relations between two query and page pairs, 911-\(P_1\) and 911-\(P_2\), changed over time. Note that 911 here is the query, while \(P_1\) and \(P_2\) are two Web pages that are related to the 911 event and the movie Fahrenheit 911, respectively. The reason for the dynamic of Web search engine log may be the evolution of users’ interests over time and occurrences of some real world events around the world.

- **Web Usage Log:** Web usage log data describes the patterns of usage for a Website or a set of Web pages such as IP addresses, page references, and the date and time of access. Web usage log is similar as Web search engine log shown in Table 1.2 except that there is no query. Usually the raw Web usage log is pre-processed into Web access

\(^3\)The author was an intern in Microsoft Research Asia from March to August 2005.
sequences for further analysis and mining tasks [37, 173]. Table 1.5 shows examples of Web access sequences (WASs). Here ID represents a sequence id. A WAS such as \((a, b, d, c, a, f, g)\) denotes a visiting sequence from Web page \(a\) to pages \(b, d, c, a, f\) and finally to page \(g\). It can be observed that the access patterns have changed in different ways. For instance, some new access patterns emerged in the second month such as access sequence \(201, 202\) and \(204\) whereas some obsolete Web access sequences such as \(101, 102,\) and \(104\) were not followed any more. Moreover, the frequency of the same WAS may change over time as well. The dynamic of Web usage log may reflect how the visitors' interests, navigation habits, and Web content changed over time.

### 1.2 Novel Research Issues

The features of Web data discussed in the preceding section raise several challenging problems in the context of Web data mining. These problems can be mainly classified into two categories: maintenance of Web mining results and discovering novel knowledge.

- **Maintenance of Web Mining Results:** As Web data may change at any time in any way, the Web mining results obtained at time point \(t_1\) may not be valid at time point \(t_2\). As a result, the Web mining results have to be updated constantly with respect to changes to the data sources. Since the size of the Web is huge, repeatedly applying these techniques to the entire or parts of the data sources are prohibitively expensive. Hence, efficient incremental Web mining techniques are required.

- **Discovering Novel Knowledge:** Historical Web data contains rich temporal information. While knowledge extracted from snapshot Web data is important, novel
knowledge that reflects dynamic behaviors of Web data may be important and useful for many applications as well. The underlying intuition is that the behaviors of specific Web data segments or Web users may follow certain change patterns, which can be extracted by integrating the temporal information. Such change patterns can be useful for various applications such as Web search engine optimization (personalization, time-dependent ranking, etc) and business intelligence (Web advertisement placing, adaptive Web, etc).

For the first category of problems, existing incremental mining techniques that were proposed for transaction database can be adopted [39, 123, 133]. In this dissertation, we shall focus on the second problem. That is, we present novel architecture to discover novel knowledge from the evolution of semi-structured Web data. Considering the dynamic nature of Web data and the second challenging problem, the following questions may be asked.

- Which parts of the Web data followed certain specific change patterns over time?
- Which parts of Web data are correlated in terms of change patterns over time?
- Can we model the change patterns of certain Web data segments over time?
- Can we extract semantics from the cluster of Web data segments that have similar change patterns over time?

We now illustrate different types of novel knowledge that can be extracted from the evolution of semi-structured Web data with respect to the above questions. Actually, each of the above questions represents a type of novel knowledge that can be extracted. Note that as we focus only on the evolution of structural Web data in this dissertation, the Web data segments referred in the above questions are actually substructures of Web data. However, these techniques can be extended and used in the context of textual data as well.

- **Substructures with specific evolution patterns:** Similar to existing Web mining approaches, which extract substructures that occurred frequently in the data collection, we propose to extract substructures with specific evolution patterns. These
specific evolution patterns can be patterns such as frequently changing patterns, increase/decrease changing patterns, and motif patterns. Note that different types of evolution patterns may be defined as useful in different applications or domains. In this dissertation, algorithms and dynamic metrics have been proposed to extract the three types of substructures. Frequently changing substructures refer to substructures that changed frequently and significantly in the history. Increase/decrease changing substructures refer to substructures that changed in increasing/decreasing manner in terms of frequency and/or significance over time. Motif substructures refer to substructures that never changed or did not change significantly most of the time (if not always) in the history [208, 209, 204, 206, 205].

- **Clusters of substructures based on evolution patterns**: While existing Web mining approaches focus on clustering Web data based on the content and snapshot structures, we propose to cluster Web data based on the evolution patterns of the corresponding substructures. The intuition is that if two substructures follow similar evolution patterns in the history, then they are expected to be similar to some extend. For example, in this dissertation, we propose to cluster Web access sequences based on their evolution patterns. Moreover, we propose to cluster Website data by integrating the evolution patterns of Web usage data with structure and content information [207, 210].

- **Evolution model for substructures**: Rather than identifying substructures with specific evolution patterns, we explore the issue of modeling the evolution patterns of substructures. In this dissertation, we propose to build a time-dependent model for the evolution patterns of the query similarity in Web search engine log using the marginalized kernel [212].

- **Semantics extracted from evolution patterns**: Beside extracting and monitoring evolution patterns of Web data, we move a step forward to explore the underlying
CHAPTER 1. INTRODUCTION

... semantics behind their evolutions such as real world events. Specifically, we proposed to detect real world events by analyzing the evolution patterns of Web access sequences in the Web usage log data and query and page co-occurrences in Web search engine log data [213, 210].

1.3 Limitation of State-of-the-art Web Mining Techniques

In the preceding sections, we highlighted different types of novel knowledge that can be extracted from the evolution of semi-structured Web data. In this section, we highlight the limitation of state-of-the-art Web mining techniques for analyzing the dynamic property of Web data. Firstly, we summarize existing efforts in Web mining. Then, we explain why these state-of-the-art approaches cannot be used to discover the above variety of knowledge.

1.3.1 Existing Web Mining Research - A Bird Eye View

The objectives of Web mining can be divided into three subtasks: resources discovery, finding unfamiliar Web documents and services on the web, information extraction, extracting specific information from newly discovered Web resources, and generalization, discovering general patterns from individual Web sites and across multiple sites [64].

In the literature, many research efforts have been directed to Web mining [15, 32, 52, 53, 61, 71, 72, 74, 102, 103, 149, 197]. Generally, Web mining can be classified into three categories based on the types of Web data the mining techniques are proposed for. They are Web content mining, Web structure mining and Web usage mining [71] as shown in Figure 1.3. Web content mining focuses on extracting knowledge from pure text of Web document such as HTML documents and XML documents, and multimedia contents such as image, video, and audio. The web content mining results can be used to summarize Web pages and aid search engines [15, 30, 102, 145, 197]. Web structure mining is to extract useful structure of the overall Web or sets of Web documents such as intra-structure of Web pages and inter-structure among Web pages. The Web structure mining results can be used to construct Web communities [72, 74, 103], facilitate Web site management [149],...
identify authoritative Web pages [61], rank the search result according to the connectivity and popularity [32]. Web usage mining is to discover information from Web log files such as Website usage log, user profiles, Web search engine log data, etc. The Web usage mining results can be used for personalizing and improve the quality of Web services such as navigation pattern based recommendation [52, 53] and personalization [151], prefetch Web documents [127, 186], improved Web search techniques [90, 138, 177].

1.3.2 Limitations

Most of existing Web mining techniques focused on designing alternative mining algorithms and apply them into specific applications, which have been proved useful. However, existing approaches only focused on the semi-structured and massiveness properties of Web data, while few work has been done concerning the dynamic nature of Web data. Specifically, we observed that existing Web mining techniques have the following limitations.

One of the key limitations is that Web data is considered as snapshot data in most existing approaches [52, 53, 61, 72, 74, 103, 149], while for each segment of Web data there is actually a timestamp and such temporal information can be useful in discovering novel knowledge. For example, Table 1.4 shows a set of Web access sequences with the corresponding supports during different time periods. Here, support of a Web access sequence refers to the number of times the Web access sequence appears in a specific time period [135]. It can be observed that the total support for each Web access sequence is the same, whereas some of them were frequently accessed during the first 3 days while others were frequently accessed during the
last 3 days. Given the Web access sequence shown in Table 1.4, existing Web data mining approaches may take all of them as equally important and return some of them as frequent Web access patterns based on the support values [135, 149]. However, in real applications such as recommendation system and service personalization [125, 149, 151], the occurrences of the same Web access sequence with different timestamps should be treated as different. For instance, recommend strategies of existing recommendation systems, which depend on the total support and ignore the timestamps of the occurrences, may not be able to produce accurate recommendations. The reason is that users’ interests may shift over time. As a result, Web access sequences that were accessed frequently recently should be assigned with high priorities in terms of recommendation. Considering the example in Table 1.4, Web pages \(b, c, d\) may have high priorities for recommendation. By incorporating the temporal and dynamic feature of the Web access sequences, an evolution-conscious recommend strategy can be more accurate.

Another limitation of existing Web mining approaches is the lack of systematic analysis of the evolution of Web data. In the literature, the dynamic nature of Web data is mainly exploited in two aspects. First, works have been done on incremental Web mining [39, 123, 133]. Second, works have been done about detecting changes between Web pages and XML documents [50, 57, 107, 108, 169]. Existing works focus on identifying the changes to the data sources and updating the Web mining results accordingly. However, they have not considered in discovering novel knowledge from the structural evolution of Web data. Let us reconsider the example of Web access sequence data in Table 1.4. Rather than updating the set of frequent Web access patterns [201, 38] and detecting changes to the mining results such as association rules [19, 21, 115], novel knowledge such as clusters of \(\mathcal{WAS}s\) with
similar evolution patterns can be discovered. In Table 1.4, the first three WASs will be in one cluster whereas the last two WASs will be in another cluster. It is obvious that such knowledge cannot be discovered using existing Web mining approaches either.

1.4 Overview

In this section, we present the overview of this dissertation by firstly describing the objective of research in this dissertation. Then, the MONETA (Mining evOlutioN of sEmi-strucTured web dAta) framework is discussed by describing the general architecture and the three main components. After that, a list of applications that can benefit from the knowledge discovered under the MONETA framework is presented. Lastly, the scope of research is defined.

The objective of this dissertation is to discover novel knowledge from the structural evolution of semi-structured Web data. Specifically, we propose a framework called MONETA to systematically mining evolution of Web data. Under this framework, various mining approaches are presented to discover different types of novel knowledge as listed in the preceding sections. Note that, in this dissertation, we focused on historical Web data that have been archived but not online and one-parse algorithms for stream data. Example applications will be presented to verify the usefulness of the discovered knowledge.
1.4.1 The MONETA Framework

The general architecture of the MONETA framework is shown in Figure 1.5, which consists of three main components: representation, measurement, and mining [211]. Basically, the input of the MONETA framework is a Web data collection of any form. Note that the most important property of the Web data collection is that there are timestamps attached to segments of the Web data. For example, the input can be a collection of Web access sequences, where for each Web access sequence the corresponding timestamp is included. The output of the MONETA framework will be different types of novel knowledge such as clusters of Web access sequences with similar evolution patterns. Here we elaborate on the three main components in turn.

The objective of the representation component is to efficiently model the historical Web data in a compact and expressive form. Given the Web data collection, firstly, each segment of the data source is represented as a tree structure. Then, based on the timestamps and calendar patterns that will be defined in the subsequent chapters, the set of trees are partitioned into a sequence of groups. Finally, the sequence of groups is merged into a single tree representation. Let us take a collection of Web access sequence as an example. Firstly, each Web access sequence is represented as a tree with the corresponding timestamps. Then, the set of trees or graphs are merged together based on the timestamps and calendar patterns. For instance, all Web access sequence that were visited during the same hour, day, week, or month, may be merged together into the same Web access tree groups. Then, each Web access sequence group is represented as a tree as well. Further, the sequence of tree representations of the Web access sequence groups is merged to form a single historical WAS tree representation. Note that the underlying intuition for the calendar pattern-based merge operation is as follows. In the original Web data source, we may get timestamps with resolutions as small as second. However, to analyze the evolution patterns, the time granularity is application dependent. To make our framework flexible, we allow users to specify any types of calendar patterns they are interested in depending on their domain knowledge and application requirements.
CHAPTER 1. INTRODUCTION

<table>
<thead>
<tr>
<th>Representation</th>
<th>Visitor-centric data</th>
<th>Author-centric data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data source</td>
<td>Data structure</td>
<td>Data source</td>
</tr>
<tr>
<td>Web usage log</td>
<td>H-HAS tree</td>
<td>XML document versions</td>
</tr>
<tr>
<td>XML query log</td>
<td>H-QPG-tree</td>
<td></td>
</tr>
<tr>
<td>Web usage log</td>
<td>Usage graph</td>
<td></td>
</tr>
<tr>
<td>Web search engine log</td>
<td>Bipartite graph</td>
<td></td>
</tr>
<tr>
<td>Web search engine log</td>
<td>Vector-based graph</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of pattern</th>
<th>Metric</th>
<th>Micro-patterns</th>
<th>Macro-patterns</th>
<th>Macro-patterns</th>
<th>Macro-patterns</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro-patterns</td>
<td>Degree of dynamic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Structure dynamic</td>
</tr>
<tr>
<td>Macro-patterns</td>
<td>Conservation rate</td>
<td>Micro-patterns</td>
<td>Macro-patterns</td>
<td>Macro-patterns</td>
<td>Macro-patterns</td>
<td>Version dynamic</td>
</tr>
<tr>
<td>Macro-patterns</td>
<td>Support range</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Degree of dynamic</td>
</tr>
<tr>
<td>Macro-patterns</td>
<td>Evolution-base similarity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macro-patterns</td>
<td>Structure dynamic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macro-patterns</td>
<td>Version dynamic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Novel knowledge</th>
<th>Algorithm</th>
<th>Novel knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAM-Miner</td>
<td>Web access motif</td>
<td>FCS</td>
<td>Frequently changing structure</td>
</tr>
<tr>
<td>CLEOPATRA</td>
<td>Web access cluster</td>
<td></td>
<td>Frequently changing semantic substructure</td>
</tr>
<tr>
<td>Conserved path mining</td>
<td>Conserved XML query</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iWED</td>
<td>Website-level events</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time-dependent model</td>
<td>Query similarity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event detection</td>
<td>Real world event</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FASST</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.5: Summary of the mining approaches.

The measurement component is used to model the dynamics of Web data by combining both the temporal information and the semi-structured property. The input to the measurement component is the sequence of trees/graphs. The objective of this component is to design a set of dynamic metrics that can model the evolution patterns of the sub-trees/subgraphs in the sequence. To reflect the evolution patterns, we should be able to not only model the micro-patterns but all the macro-patterns. Here, micro-patterns reflect the local change patterns such as change patterns between consecutive Web access tree groups, while macro-patterns refers to the global change patterns such as the overall trend and frequency of changes.

The objective of the mining component is to extract novel knowledge using data mining techniques based on the outputs of the above two components. That is, based on the metrics proposed in the measurement component, different types of evolution patterns can be defined and corresponding substructures can be extracted from the unified representation. Note that there are various definitions of novel knowledge and different mining approaches as well. For the mining component, the key issue is to design a general and scalable algorithm that can be used with different constraints. Table 1.5 shows the summary of the three main components with respect to the author-centric data and visitor-centric data.
1.4.2 Applications of MONETA

In this section, we present some representative applications that can benefit from the four types of novel knowledge that can be extracted under the MONETA framework.

Research Community Evolution Analysis

Considering the dynamic nature of XML data, works have been done to support complex temporal queries for multi-version XML documents [165, 166, 164]. The basic idea is to create the valid time and transaction time attributes for elements in the XML to support temporal join operations. However, such temporal queries have limited power to extract novel knowledge from multi-version XML documents in terms of the evolution patterns. By mining the multi-version XML documents, together with the ontology, the object level evolution patterns can be extracted, which may be difficult or impossible to be expressed in the form of any specific temporal query. For example, given the XML format DBLP publication archive\(^4\), which records the activities in the research community, novel knowledge about research objects such as emerging new research topics, rising research "stars", and hot conferences cannot be efficiently discovered using temporal queries as some interesting patterns are unknown and hard to define with only queries. However, such knowledge can be useful in the research community. By extracting substructures with specific evolution patterns such as increase/decrease change pattern, frequently changing pattern, and motif pattern, together with the corresponding domain knowledge, such knowledge can be efficiently discovered under our MONETA framework.

Intelligent XML Cache Strategy

Recently, different algorithms have been proposed to discover the frequent query patterns from historical XML queries for caching [183, 184, 185]. The idea is to cache the results for the frequently issued query patterns as they are expected to be issued frequently in the future. These caching strategies are solely based on statistics obtained by treating historical

\(^4\)DBLP in XML format: http://dblp.uni-trier.de/xml/
CHAPTER 1. INTRODUCTION

XML queries as snapshot data. That is, the frequent XML query patterns are based on only the number of occurrences of the query patterns in the history. Every occurrence of a query pattern contributes equally to the caching strategy regardless of when the query was issued. However, this may not always be true in real life applications, where the timestamps for different occurrences of an XML query in the history can affect the caching strategy significantly. For instance, the users may not be interested in queries that were issued long time ago as their interests have shifted. By taking the temporal information into account, we define the conserved query paths as paths in query pattern trees that never change or do not change significantly most of the time (if not always) in terms of their support values during a specific time period. By extracting and utilizing the set of conserved XML query paths, such as frequent and conserved query paths and infrequent and conserved query paths that shall be discussed in Chapter 4, more intelligent XML cache strategies can be constructed.

Accurate User Segmentation

User segmentation is to cluster web users based on the corresponding Web access sequences to provide personalized services [76, 89]. Existing works either use sequence-based distance or probability models to measure the distance between access sequences [76, 89]. However, none of these efforts has taken the dynamic nature of access sequences into account. For instance, two users may have the same list of access sequences that belong to two topics, $T_1$ and $T_2$, with the same support. Using existing segmentation techniques, the two users will be grouped into the same cluster. However, they may have different preferences. For example, the first user may be currently interested in $T_2$ as most of the access sequences about $T_1$ were accessed long time ago, while the second user may be currently interested in $T_1$ as most of the access sequences about $T_2$ were also accessed long time ago. By taking the temporal information, the user segmentation can be more accurate as users in the same group are not only expected to have similar access sequences but also evolutionary patterns of those access sequences are expected to be similar as well. In Chapter 5, we shall present the details of clustering the Web access sequence by taking into account the evolution patterns in the history [203, 207, 210].
CHAPTER 1. INTRODUCTION

Time-Dependent Query Expansion for Web Search Engines

A real dilemma for most of the existing Web search engines is that users request for accurate search results while they only provide queries of limited length, which is usually less than two on average according to [172]. Recently, some works have been done to expand the query with similar keywords to refine the search results [56, 91, 139, 161]. For example, in [56] the authors proposed to expand queries with similar terms that occur frequently in the corresponding documents being visited in the history. Existing query expansion techniques attempted to identify a unified correlation or similarity between any two queries that may not exist for every case and may change rapidly in different situations of real-world environment. That is, for some query term pairs, their correlations or similarities are time-dependent with respect to the dynamic nature of both the Web content and user behaviors. For example, from the real Web search engine log data collected from MSN search, we observed that the similarity between two queries "father" and "gift" changes from day to day and reaches its peak in the days around the Father's Day. As a result, the unified correlations and similarities may not always be accurate enough. In Chapter 6, we shall present a time-dependent model to monitor the similarities between queries, which has the potential of improving the query expansion for search engine [212].

Event Detection from Web Data

Previous efforts on event detection from the Web have focused primarily on the author-centric data such as Web content and structure ignoring the rich collection of visitor-centric data. Furthermore, the dynamic nature of Web data has not been fully exploited in existing event detection approaches. There are two motivating factors behind our approach of event detection from the evolution patterns of both author-centric and visitor-centric data. Firstly, Web data relevant to the same event are often expected to be created, updated, and modified in a similar pattern in the history. That is, the evolution patterns of the author-centric data are expected to reflect the evolution of real world events. For example, when a new product is released, a set of related Web pages may be created such as news, specification,
advertisements, and comments. Secondly, the evolution patterns of how the visitors accessed the Web data are also an indicator of the related events. That is, Web data related to the same event are often expected to have similar evolution patterns in terms of the usage data generated by Web users. In Chapter 5, we shall present an event detection approach by integrating the evolution patterns of both author-centric data and visitor-centric data. In Chapter 6, we shall discuss another event detection approach that only uses the visitor-centric data [213].

1.4.3 Scope of Research

The scope of research in this dissertation is as follows.

- We focus on the structural evolution of Web data. Here structural evolution refers to the evolution of Web data structures such as intra-structures within Web pages and inter-structures among Web pages. Specifically, XML documents and XML queries will be used as examples of intra-structures, whereas implicit structures in the Website log data and Web search engine log data will be used as examples of inter-structures. Note that although our framework is able to handle the evolution of any type of semi-structured data, we choose XML instead of HTML because that XML is increasingly used to represent data on the Web and it has been predicted more and more queries in XML format will be issued over the hidden Web [2].

- Both author-centric data and visitor-centric data are used to verify the proposed framework and mining techniques. Examples of author-centric data used in this dissertation are versions of XML documents and Website structures, whereas examples of visitor-centric data are the XML queries, Website log data, and Web search engine log data.

- This dissertation does not focus on discovering the frequency of changes of the data source. That is, we assume that the data source is available and complete, which means that all the historical changes are captured in the data archive. Issues of efficiently crawling and predicting the frequency of changes have been extensively studied [46, 47, 48, 86].
CHAPTER 1. INTRODUCTION

1.5 Contributions

The main contributions of this dissertation can be summarized as follows.

- **The MONETA framework for mining evolution of semi-structured Web data**: This is the first effort to systematically analyze the dynamic nature of semi-structured Web data. Three main components of the MONETA framework: representation, measurement, and mining are explained explicitly for each mining approach proposed under this framework.

- **Novel tree and graph representations for historical Web data**: Two sets of novel representation methods have been proposed to record the historical Web data collection and preserve the temporal information as well. As for the tree representation, the $H$-DOM tree, $H$-DOM$^+$ tree, $H$-QPG-tree, and $H$-WAS tree have been proposed to record the collection of XML documents, XML queries, and Web access sequences, respectively. As for the graph representation, the structure graph, content graph, usage graph, and vector-based graph have been proposed to represent the Website data and Web search engine log data.

- **A set of dynamic metrics**: To measure the dynamic nature of the Web data, a set of novel dynamic metrics have been proposed. Basically, there are two types of metrics: micro-pattern metrics and macro-pattern metrics. As shown in Table 1.5, the following metrics have been proposed and used in this dissertation: structure dynamic, version dynamic, degree of dynamic, conservation rate, support range, and evolution pattern-based similarity.

- **A set of algorithms for extracting novel knowledge**: In this dissertation, we proposed a set of algorithms to extract different types of novel knowledge based on the evolution of semi-structured Web data. With respect to the four types of novel knowledge we focused on, the following algorithms have been designed as shown in Table 1.5. To extract substructures with specific evolution patterns, the FCS, FASST,
CHAPTER 1. INTRODUCTION

WAM-Miner, and DCC algorithms were proposed. The CLEOPATRA and FM-WAP algorithms were used to cluster substructures with similar evolution patterns. The TERA algorithm was proposed to model the evolution of substructures. Lastly, the iWED and VOLT algorithms were designed to extract semantics from evolution of Web data.

- **Real applications:** Besides the framework and the mining techniques, real applications have been presented to shown how the extracted novel knowledge can be useful in improving existing Web data management and business intelligence applications. Specifically, the applications of intelligent XML query cache, event detection from Website data, time-dependent query expansion, research community evolution analysis, and event detection from Web search engine log data have been discussed in this dissertation.

1.6 Organization of the Dissertation

The rest of this dissertation is organized as follows.

- In Chapter 2, we discuss prior work in related areas. We focus on two general topics: mining semi-structured Web data and analyzing of Web data dynamics. More specifically, we reviewed existing works in mining tree and graph structured data, mining XML data, change detection for tree structured data, time series analysis, data stream mining, and detect and monitor changes to mining results. The comparison of our work and existing works is presented as well.

- In Chapter 3, we illustrate the MONETA framework by extracting substructures with specific evolution patterns from author-centric data. Specifically, the frequently changing substructures (FCS) and frequent changing semantic substructures (FASST) are extracted from versions of XML documents. Details of the representation, measurement, and mining components are discussed in turn. Moreover, the application of
research community evolution analysis is presented to demonstrate the usefulness of such knowledge.

- In Chapter 4, the approaches of extracting substructures with specific evolution patterns from visitor-centric data are discussed. Specifically, the conserved query paths are extracted from historical XML queries. After that, a dynamic-conscious cache (DCC) strategy for XML query is presented by ranking these conserved query paths.

- In Chapter 5, we address the issue of clustering substructures based on their evolution patterns. Specifically, the visitor-centric data (Website log data) is used. Firstly, we present two algorithms called FM-WAP and CLEOPATRA to cluster the Web access sequences in the Web usage log. Then, a Website level event detection algorithm, called iWED, is presented by integrating both the author-centric data and the evolution of visitor-centric data.

- In Chapter 6, we move a step forward to model the evolution patterns of visitor-centric data. The real Web search engine log data obtained from MSN Web search engine is used. Specifically, we present a marginalized kernel-based approach to construct a time-dependent model that monitors the similarities between queries over time in Web search engine log.

- In Chapter 7, the issue of extracting semantics from evolution of semi-structured data is illustrated using the visitor-centric data (MSN Web search engine log). By clustering the query and page pairs in the Web search engine log data, a graph cut based algorithm is proposed to detect real world events from the evolution of Web search engine log data.

- In Chapter 8, we conclude this research with a summary and an overview of future research direction.
Chapter 2

Literature Review

In this chapter, we review representative works in research areas that are related to this dissertation. Firstly, we survey traditional works on mining tree and graph structured data in the context of the Web. Then, research related to Web data dynamics such as change detection from semi-structured data, time series analysis, mining data stream, detect and monitor changes to mining results is reviewed. Finally, we summarize the existing works and compare them with our work.

2.1 Mining Semi-Structured Data

Due to the requirement of more expressive patterns for applications in various fields such as computer network, Web mining, bioinformatics, XML data mining, recently, increasing research efforts have focused on mining semi-structured data such as tree and graph structures to capture the complex relations among data entries [40, 170]. As a result, many studies have been conducted in mining semi-structured data, especially in the context of the Web data [75, 157, 16, 150, 179, 158, 129, 17]. Basically, most of the existing semi-structured data mining approaches borrow the techniques and ideas from the matured association rule mining community [10, 11, 132, 141]. The literature of mining semi-structured data in the context of the Web can be categorized into three groups: mining general tree structured data, mining general graph structured data, and mining XML data. We begin by presenting the necessary background knowledge and reviewing some basic concepts related to tree and
CHAPTER 2. LITERATURE REVIEW

Figure 2.1: Example of graphs and trees.

2.1.1 Background: Graph and Tree

Graph: A graph is defined as a triple \((V, E, f)\), where \(V\) is a set of arbitrary vertices, \(E\) is a subset of edges connecting some vertex pairs in \(V\), \(f\) is a mapping function: \(f : E \rightarrow V \times V\).

More generally speaking, \(V\) is a set of arbitrary vertices that can represent any objects such as Web pages in the Web graph, elements in XML documents, people in social network, etc. \(E\) is the set of edges that connect the vertices such as hyperlink in the Web graph, parent and child relations in XML documents, social connections between people, etc. An example of graph is presented in Figure 2.1(a).

Labeled Graph: In the semi-structured data mining area, most approaches focused on the labeled graph. Formally, a labeled graph is denoted as \(G = (V, E, f, \Sigma, L)\) that consists of a set of vertices \(V\), a set of edges \(E\), a mapping function: \(f : E \rightarrow V \times V\), an alphabet \(\Sigma\) for vertex and edge labels, and a labeling function \(L : V \cup E \rightarrow \Sigma\) that assigns labels to vertices and edges. An example of labeled graph is presented in Figure 2.1(b).

Directed and Undirected Graph: A graph is undirected if each edge connects an unordered pair of vertices; it is directed if each edge connects an ordered pair of vertices. It can be observed that the graph in Figure 2.1(b) is undirected while the graph in Figure 2.1(c) is directed.
CHAPTER 2. LITERATURE REVIEW

Path and Length: A path is a list of vertices of the graph such that each pair of neighboring vertices in the list corresponds to an edge in the graph. The length of the path is defined as the number of edges in the path.

Cycle and Acyclic: A cycle is a path of which the first and last vertices are the same. A graph is acyclic if the graph contains no cycle. It can be observed that the graph in Figure 2.1(b) contains cycles while the graph in Figure 2.1(c) is acyclic.

Connected and Disconnected Graph: An undirected graph is connected if there exists at least one path between any pair of vertices in the graph, disconnected otherwise. For instance the graph in Figure 2.1(b) is disconnected.

Also, there are many types of trees. Here we review three of them: unrooted unordered trees (free trees), rooted unordered trees, and rooted ordered trees.

Unrooted Unordered Tree (Free Tree): A free tree is an undirected graph that is connected and acyclic. Free trees have certain properties that graph may not have. For example, there is a single path between any pair of vertices in a free tree. An example of free tree is shown in Figure 2.1(d). Note that there is no rooted node in any free tree.

Rooted Unordered Tree: A rooted unordered tree is a directed acyclic graph that satisfies: (1) there is a distinguish vertices called root that has no entrance edges; (2) every vertex except root has exactly one entrance edge; and (3) there is a unique path from the root to every other vertex. In the rooted unordered tree, if a vertex $a$ is on the path from the root to the vertex $b$, then $a$ is the ancestor of $b$ and $b$ is the descendant of $a$. For any edge $(a, b) \in E$, $a$ is the parent of $b$ and $b$ is a child of $a$. Vertices that share the same parent are siblings. A vertex that has no descendant is called a leaf. The depth or level of a vertex is defined as the length of the path from the root to that vertex. An example of rooted unordered tree is shown in Figure 2.1(e).

Rooted Ordered Tree: A rooted ordered tree is a rooted unordered tree that has a predefined ordering among each set of siblings. The order is implied by the left-to-right order in the structure representation. For a rooted ordered tree, the left and right siblings of a vertex are defined. An example of rooted ordered tree is shown in Figure 2.1(f).
CHAPTER 2. LITERATURE REVIEW

(C) A bottom-up subtree (b) An induced subtree (c) An embedded subtree
(d) A general subgraph (e) An induced subgraph (f) A connected subgraph

Figure 2.2: Example of subgraphs and subtrees.

With the above preliminaries, different types of subtrees and subgraphs have been defined. Here we review the bottom-up subtree, induced subtree, embedded subtree, general subgraph, induced subgraph, and connected subgraph.

**Bottom-up Subtree:** Given two rooted trees $T$ and $T'$, $T'$ is the bottom-up subtree of $T$ if and only if: (1) $V' \subseteq V$ and $E' \subseteq E$; (2) the labeling of $V'$ and $E'$ is preserved in $T'$; (3) if $T$ is ordered, then the left-to-right order among siblings in $T$ should be preserved in $T'$; (4) for a vertex $v_m \in V$, if $v_m \in V'$ then all descendants of $v_m$ in $V$ must be in $V'$. For example, the subtree in Figure 2.2(a) is a bottom-up subtree of the tree in Figure 2.1(f).

**Induced Subtree:** Given two rooted trees $T$ and $T'$, $T'$ is the induced subtree of $T$ if and only if: (1) $V' \subseteq V$ and $E' \subseteq E$; (2) the labeling of $V'$ and $E'$ is preserved in $T'$; (3) if $T$ is ordered, then the left-to-right order among siblings in $T$ should be preserved in $T'$. For example, the subtree in Figure 2.2(b) is an induced subtree of the tree in Figure 2.1(f).

**Embedded Subtree:** Given two rooted trees $T$ and $T'$, $T'$ is the embedded subtree of $T$ if and only if: (1) $V' \subseteq V$; (2) the labeling of $V'$ is preserved in $T'$; (3) $v_m, v_n \in V'$, $v_m$ is the parent of $v_n$ in $T'$ if and only if $v_m$ is an ancestor of $v_n$ in $T$; (4) if $T$ is ordered, then the order among the preorder of the vertices should be preserved, where the preorder of a vertex is its index in the tree according to the preorder traversal. For example, the subtree in Figure 2.2(c) is an embedded subtree of the tree in Figure 2.1(f). In summary,
CHAPTER 2. LITERATURE REVIEW

### Types of subtrees

<table>
<thead>
<tr>
<th>Types of subtrees</th>
<th>Bottom-up subtree</th>
<th>Induced subtree</th>
<th>Embedded subtree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V' \subseteq V )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>( E' \subseteq E )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Labeling of ( V' ) is presented</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Labeling of ( E' )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sibling order is preserved</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>∀ ( v_m, v_n \in V ), all descendant of ( v_m \in V' )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ancestor and descendant relation is preserved</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Preorder of vertices is preserved</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.1: Summary for the three types of subtrees.

<table>
<thead>
<tr>
<th>Types of subgraphs</th>
<th>General subgraph</th>
<th>Induced subgraph</th>
<th>Connected subgraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V' \subseteq V )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>( E' \subseteq E )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>( \forall f(e_i) = (v_m, v_n) \in E', (v_m, v_n) \in V' )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>( \forall v_m, v_n \in V', f(e_i) = (v_m, v_n) \in E' )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>( \Leftrightarrow f(e_i) = (v_m, v_n) \in E )</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>( \forall v_m, v_n \in V ) are mutually reachable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.2: Summary for the three types of subgraphs.

the differences between the three definitions of subtrees are shown in Table 2.1 with respect to various constraints.

**General Subgraph:** Given two labeled graphs \( G \) and \( G' \), \( G' \) is the **general subgraph** of \( G \) if \( V' \subseteq V \), \( E' \subseteq E \) and \( \forall f(e_i) = (v_m, v_n) \in E' \), \( v_m, v_n \in V' \). Figure 2.2(d) is an example of the general subgraph of the graph in Figure 2.1(b).

**Induced Subgraph:** Given two labeled graphs \( G \) and \( G' \), \( G' \) is the **induced subgraph** of \( G \) if \( V' \subseteq V \), \( E' \subseteq E \) and \( \forall v_m, v_n \in V' \), \( f(e_i) = (v_m, v_n) \in E' \Leftrightarrow f(e_i) = (v_m, v_n) \in E \). Figure 2.2(e) is an example of the induced subgraph of the graph in Figure 2.1(b).

**Connected Subgraph:** Given two labeled graphs \( G \) and \( G' \), \( G' \) is the **connected subgraph** of \( G \) if \( V' \subseteq V \), \( E' \subseteq E \) and all vertices in \( V' \) are mutually reachable through some edges in \( E' \). Figure 2.2(f) is an example of the connected subgraph of the graph in Figure 2.1(b).

Table 2.3 summarizes the types of subgraphs.

### 2.1.2 Mining Tree Structured Data

In the literature, most of the existing tree structured data mining works focus on extracting **frequent subtrees** from the tree structured data. In order to define the **frequent subtree**, we begin by reviewing the definition of **support**.

**Support:** Let \( \delta_T(t) \) denote the number of occurrences (bottom-up, induced, or embedded, depending on the context) of the subtree \( t \) in a tree \( T \). Let \( d_T \) be an indicator variable,
where \( d_T(t) = 1 \) if \( \delta_T(t) > 0 \) and \( d_T(t) = 0 \) otherwise. Let \( \mathcal{D} \) denote a database of trees. The support of a subtree \( t \) in the database is defined as \( \sigma(t) = \sum_{T \in \mathcal{D}} d_T(t) \), i.e., the number of trees in \( \mathcal{D} \) that contain at least one occurrence of \( t \).

**Frequent Subtree:** Given a database of trees, denoted as \( \mathcal{D} \), with the minimum support threshold, denoted as \( \text{minsup} \), a subtree \( t \) is frequent if its support is more than or equal to the minimum support value. The size of a subtree \( t \) is denoted as \( |t| \), which is the number of vertices. We denote the set of all frequent subtrees of size \( k \) as \( F_k \).

Similar to the frequent itemset mining in transactional database [10, 11, 132, 141], the task of mining frequent subtrees can be decomposed into two main phases: candidate generation and support calculation.

**Candidate Generation:** Given a database of trees, firstly, we need to generate an initial set of candidate subtrees that have the possibility of being frequent, rather than enumerate all possible subtrees. Usually, the frequent \( k \)-subtrees are generated by extending the existing frequent \( k-1 \)-subtrees. For example, we can extract the frequent \( 1 \)-subtrees by calculating the support for each node in the database. Then, the frequent \( 2 \)-subtrees are generated by extending or joining the frequent \( 1 \)-subtrees. This candidate generation is based on the anti-monotone property of frequent subtrees. That is, all subtrees of a frequent subtrees should be frequent. We shall see how the candidates are generated in detail in different context later.

**Support Calculation:** The support value of a candidate subtree needs to be calculated in order to determine if it is frequent subtree. It can be calculated based on direct checking against the database: for each \( T \) in \( \mathcal{D} \), the support of all its subtrees is increased by one. Another method is to calculate the support using the vertical representation. That is, associate an occurrence list for each frequent subtree, i.e., a list of transaction IDs that contains the subtree. Then, the support of the candidate subtrees can be obtained by manipulating the occurrence lists. Details of how to calculate the supports for subtrees shall be discussed later in the context of individual algorithm.
CHAPTER 2. LITERATURE REVIEW

A database of 3 trees

Encode the bottom-up subtrees

Sort them

Support

Frequent bottom-up subtrees

Figure 2.3: Example of mining frequent bottom-up subtrees.

Based on the concepts introduced in the previous section, the existing works can be classified into three categories: frequent bottom-up subtree mining, frequent induced subtree mining, and frequent embedded subtree mining. At the same time, there are three types of tree structures: free tree, rooted unordered tree, and rooted ordered tree. We now elaborate the three types of mining approaches for different types of tree structures.

Frequent Bottom-up Subtree Mining

From the definition of bottom-up subtree, it is obvious that for a tree $T$, the number of bottom-up subtrees is the number of vertices of $T$. Note that these bottom-up subtrees may not be all distinct. As a result, a straightforward way of extracting the frequent bottom-up subtree is to enumerate all subtrees, sort them, and calculate their support values directly.

Figure 2.3 shows how the mining process works with an example. Firstly, all the subtrees in the database are encoded in a canonical representation. For instance, all the ordered bottom-up subtrees are represented as the corresponding depth-first pre-order string. That is, given a tree $T$, its string encoding denoted as $T$ is initiated as empty. Then, perform a depth-first preorder search starting from the root, adding the current node’s label $x$ to $T$. Whenever we backtrack from a child to its parent we add a unique symbol $-1$ to the string. After that, we initialize an array of pointers to each node in the database from the
roots of bottom-up subtrees. Then, the pointers are sorted by comparing their canonical representations. Lastly, scan the array to determine the support of each subtree and return the set of frequent bottom-up subtrees.

The key advantage of this straightforward frequent subtree mining approach is that the number of bottom-up subtrees for a rooted tree is bounded by $V_D$, which is the number of nodes in the database $D$. As shown in [117, 118, 40], this approach works efficiently for subtree matching and frequent bottom-up subtree mining with both rooted ordered trees and rooted unordered trees, as long as an appropriate canonical representation for the bottom-up subtree is provided. The canonical representations for rooted ordered trees and rooted unordered trees have been defined as the pre-order string and sorted pre-order string in [117, 118]. The time complexity of the above algorithm is $O(m|V_D|\log|V_D|)$, where $m$ the size of the largest tree in $D$ and $|V_D|$ is the number of nodes in $D$.

**Frequent Embedded Subtree Mining**

In the above frequent bottom-up subtree mining approach, the brute-force method works well. However, for the frequent embedded subtree or induced subtree, the above method becomes very inefficient. The reason is that the number embedded subtrees and induced subtrees grow exponentially with the size of $T$ as shown in Figure 2.4. In [193] and [194], the TreeMiner and SLEUTH algorithms were proposed by Zaki to mine frequent embedded subtree from ordered and unordered trees, respectively. We now elaborate on them with respect to the candidate generation and support calculation phases.

**TreeMiner-Candidate Generation Phase:** The TreeMiner algorithm [193] is proposed for ordered trees. In this algorithm, both ordered trees and subtrees are represented as a string using the depth-first preorder traversal. For example, the string encoding of a tree and the tree itself are shown in Figure 2.5(a). To generate the candidate frequent $k+1$-subtree, the frequent $k$-subtrees are extended with the last vertices in each of them. Two $k$-subtrees $X, Y$ are in the same equivalence class iff they share a common prefix up
to the $k$-th node. As we can see from the examples, Figure 2.5(c) shows two frequent 2-subtree, which belong to the 1-prefix equivalence class. Then, three candidate 3-subtrees are generated as shown in the figure.

**TreeMiner-Support Calculation Phase:** To efficiently calculate the *support* of each subtree, the author proposed a *scope* attribute for each vertex in the tree. That is, the scope of a vertex $v_i$ is given as the interval $[l, r]$, where $l$ and $r$ are the positions of vertex $v_i$ and the right-most vertex in the subtree rooted at $v_i$, respectively, in the depth-first preorder search method. An example of the scope of the vertex is shown in Figure 2.5(c). Based on the scope list, the *frequency* of each candidate subtree can be obtained efficiently. Figure 2.5(c) shows the join process that uses the scope list. The scope lists of the frequent 2-subtrees are joined to get the scope list of the resulting candidate 3-subtrees.

The candidate generation and support calculation processes iterate till all frequent subtrees are discovered. The complexity of TreeMiner algorithm is $O(ftlp^2m^{2p})$, where $f$ is the number of frequent pattern trees in the result, $l$ is the number of distinct labels in $\mathcal{D}$, $t$
CHAPTER 2. LITERATURE REVIEW

is the number of trees in the database $D$, $p$ is the number of nodes in the largest frequent subtree, $m$ is the number of nodes in the largest tree in $D$.

**SLEUTH Algorithm:** In [194], an algorithm called *SLEUTH* was proposed to mine frequent embedded subtrees from unordered trees. For the candidate frequent subtree generation, different approaches have been proposed such as prefix extension, canonical extension, and equivalence class-based extension. In the prefix extension, it is similar to the *TreeMiner* algorithm for ordered tree except there is a canonical form constraint on the candidate subtree. In the canonical extension approach, only canonical trees, the prefixes of which are in canonical form, are extended to form new subtree candidate. In the equivalence class-based extension, only known frequent elements from the same class are used for extending existing frequent subtrees that are in canonical form. Canonical extensions generate non-redundant candidates, but many of them turn out to be not frequent. At the same time, the equivalence class-based extensions generate redundant candidates, but only a small number of extensions is considered. In the experiments, the author showed that the equivalence class-based extension is the most efficient approach. Similar to the *TreeMiner* algorithm, in the *SLEUTH* algorithm, the scope list is used for fast support calculation for all the candidate frequent subtrees. The time complexity of *SLEUTH* and *TreeMiner* are the same, $O(|D| p^2 m^{2p})$. However, note that the numbers of frequent subtrees are different in the mining results for the two algorithms even with the same dataset and same $\text{minsup}$ threshold.
Chapter 2. Literature Review

Frequent Induced Subtree Mining

In the literature, frequent induced subtree mining has been applied to rooted ordered trees [16], rooted unordered trees [17, 129, 43, 168, 174], and free trees [42, 140, 43, 130]. We review them in turn.

Rooted Ordered Trees: The FREQT algorithm proposed by Asai et al. employed the extension-only approach to find all frequent induced subtrees in a database of rooted ordered trees [16].

FREQT-Candidate Generation: Similarly, the preorder traversal-based encodings are used as canonical representations of the ordered trees and subtrees. Then, the candidate generation process is guided by the procedure called rightmost expansion. That is, to get the candidate $k+1$-subtrees, the frequent $k$-subtrees are extended by connecting a new vertex with a frequent label to the rightmost path, which is the path from the root to the last vertex in the subtree in preorder traversal. Figure 2.6(a) shows the enumeration process with two frequent labels $A$ and $B$. By expanding frequent label $A$ with itself and $B$, two 2-subtree candidates are generated by appending $A$ and $B$ to the rightmost of $A$, respectively. Similarly, other candidates are generated.
**CHAPTER 2. LITERATURE REVIEW**

*Tree Tl T2 T3 T4 T5

<table>
<thead>
<tr>
<th>Tree</th>
<th>Encode</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>(0,A)(1,F)(1,B)(2,C)(2,E)(1,B)</td>
</tr>
<tr>
<td>T2</td>
<td>(0,A)(1,B)(1,F)(1,B)(2,C)(2,E)</td>
</tr>
<tr>
<td>T3</td>
<td>(0,A)(1,B)(2,C)(2,E)(1,F)(1,B)</td>
</tr>
<tr>
<td>T4</td>
<td>(0,A)(1,B)(2,C)(2,E)(1,B)(1,F)</td>
</tr>
<tr>
<td>T5</td>
<td>(0,A)(1,B)(2,C)(2,E)(1,F)(1,B)</td>
</tr>
</tbody>
</table>

...  

Figure 2.7: Canonical representation of unordered tree.

**FREQT-Support Calculation:** To determine the support of the subtrees, they proposed an occurrence list based approach. That is, for each subtree, all the occurrences of the subtree in the database are recorded by matching the rightmost vertex in the subtree. Based on the two trees in Figure 2.5, a part of the enumeration tree with the occurrence lists are given in Figure 2.6(b). The reason that only the occurrence list of the rightmost vertex is stored is that both candidate generation and frequency of subtree can be derived from this list.

Also, two pruning techniques called node-skip and edge-skip have been proposed based on the frequent labels and the frequent 2-subtrees. The running time of the algorithm is bounded by \( O(k^2 b L N) \), where \( k \) is the maximum size of the frequent patterns, \( b \) is the maximum branching factor of \( D \), \( L \) is the number of labels, and \( N \) is the sum of the lengths of the rightmost occurrences lists of frequent patterns.

**Rooted Unordered Trees:** By extending the FREQT algorithm, another two algorithms called Unot [17] and uFreqt [129] were proposed to extract frequent induced subtrees from rooted unordered trees, where duplicated siblings may exist.

**Candidate Generation:** For the candidate generation process, it is conceptually more difficult as some subtrees may be isomorphic. In both the Unot and uFreqt algorithms, the ordered trees are used as the canonical representation of the unordered trees. Basically, the ordered trees are encoded with the depth-first traversal. For example, Figure 2.7 shows two unordered trees and the corresponding ordered tree representations. Their encodes are also
shown in the table, where each vertex is represented as \((depth, label)\). Then, they define the order between these encode strings. As a result, the encode string with the lowest order among all the ordered tree representation of an unordered tree is used as the canonical representation. For instance, in this example, \(T_4\) is used as the canonical representation of the unordered tree. Then, the candidate generation process is guided by the following constraint. Only extension from the canonical form of a subtree that can result in the canonical form of another subtree is allowed. The extension is the same to the rightmost extension in the \(FREQT\) algorithm.

**Support Calculation:** To calculate the support of the subtrees, \(Unot\) employed the same method as we have described in the TreeMiner algorithm proposed by Zaki. The support calculation is an incremental computation of the occurrences based on the reverse search \([129]\). In the \(uFreqt\) approach, the supports are calculated using the occurrence list based strategy. However, the list of occurrences is stored for each node in the pattern tree. That is the size of the occurrence list is bounded by the product of the size of the database and the size of the pattern.

The complexity of the \(uNot\) algorithm is \(O(fb \cdot m^{b+1}p)\), where \(b\) is the largest numbers of trees that grow from a single tree. The complexity for the \(uFreqt\) algorithm is \(O(ft \cdot m^2p \cdot \sqrt{p})\). Note that the \(uFreqt\) algorithm is relatively more efficient in terms of time, but it needs more space than the \(uNot\) algorithm.

Based on the assumption that no two siblings share the same label, algorithms have been proposed to extract frequent subtrees by joining root paths for the rooted unordered trees \([168, 174]\). In \([174]\), an algorithm called \(PathJoin\) was proposed to enumerate all frequent subtrees by joining the frequent rooted paths. As the labels of siblings are unique, each unordered subtree can be defined as the join of its rooted paths. In the algorithm, a data structure called \(FST-Forest\) is used, of which each root path corresponds to a frequent path in the database and has a list of occurrences associated. The candidates are generated by traversal the \(FST-Forest\) in a breadth-first fashion. Similarly, the frequency of each candidate subtrees can be obtained by joining the list of occurrences. The complexity of
CHAPTER 2. LITERATURE REVIEW

Figure 2.8: Candidate Generation in FreeTreeMiner algorithm.

this algorithm is $O(t m^2 + c p |D|)$, where $c$ is the number of candidate subtree generated in the mining process.

**Free Trees:** In [42], the FreeTreeMiner algorithm was proposed by Chi et al. to discover all frequent induced subtrees from a database of labeled free trees.

**Candidate Generation:** For candidate generation, the downward closure constraint is used. That is, all the $k$-subtrees of the frequent $k+1$-subtree are frequent. To generate candidate $k+1$-subtree without redundancy, two frequent $k$-subtrees are joined only if: (1) they share the $k$-$1$-subtree, which is called the core of the pair of frequent $k$-subtrees; (2) the two labeled leaves must be the top 2 labels in the resulting candidate $k+1$-subtree. An example of candidate generation is shown in Figure 2.8, where the candidate 5-subtree is the joining result of tree(a) and tree(b). This method works well for trees with distinct vertex labels and changes have to be made for trees that have duplicated vertex labels [42].

**Support Calculation:** To calculate the support of the subtrees, the naive approach that check the candidate subtree $c$ against the database $D$ is used. They adopted the best bipartite matching tree inclusion algorithm proposed in [142]. The main idea is to first fix a root $r$ for each tree $t$ in $D$ (denoted by $t^r$), then test for each vertex $v$ of $c$ if the rooted tree $c^v$ is isomorphic to some subtrees of $t^r$. The test is done on in a postorder.

Chi et al. proposed an algorithm called HybridTreeMiner to extract frequent induced subtrees from both rooted unordered trees and free trees. In this approach both the joins and extensions are used to generate the candidate subtrees with the bread-first canonical form obtained using the level-order traversal. Given a set of frequent canonical ordered trees of size $k$ that share a common prefix of size $k-1$, the candidate subtrees are either joined by
two frequent subtrees or extended from an existing frequent subtree. Also, the frequency of each subtree is calculated by joining the occurrence list of the candidates. 

By modifying the *HybridTreeMiner*, another approach called *Gaston* was proposed by Nijssen *et al.* to extract frequent induced subtrees from unlabeled free trees using depth sequences. The advantage of this approach is that there is a constant time enumeration strategy for unlabeled free trees. This approach combines the join operation and extension operation to generate candidate subtrees. *Frequencies* of subtrees are calculated using the occurrence lists as well. Moreover, there is another algorithm called *FreeTreeMiner* [140] proposed by Ruckert *et al.*, which is different from Chi’s *FreeTreeMiner* algorithm [42]. The difference is that Ruckert’s approach searches for frequent free trees in a database of graphs.

The time complexities of Chi’s *FreeTreeMiner* candidate generation and support calculation are \( O(b f p^2) \) and \( O(c t p \sqrt{p \ m / \log P}) \), respectively. The complexity of the *HybridTreeMiner* is \( O(f t m^p (b m^p + m)) \), which is the same for the for the algorithm proposed by *Gaston*. The complexity of the *FreeTreeMiner* proposed by Ruckert *et al.* is \( O(f(b' p t + b t m^{p+1})) \), where \( b' \) is the maximal number of candidates generated and \( b \) is the number of candidates that turn out to be frequent (see [40] for more details).

**Maximal and Closed Frequent Subtrees:** Besides the different types of definitions for subtrees, similar to the frequent itemset mining approaches [29, 67, 167, 196], there are some approaches in frequent subtree mining try to reduce the number of trees in the result and preserve the useful information. That is, some closed and/or maximal frequent subtree mining approaches have been proposed [41, 44, 158, 168, 174]. The intuition is that the number of frequent induced subtrees and embedded subtrees may increase exponentially and the users may be overwhelmed. In [168, 174], the authors attempt to reduce the number of frequent induced subtrees by proposing the term *maximal frequent subtree*. A *maximal frequent subtree* is a frequent subtree that none of its proper supertrees are frequent. In [41, 44], the authors proposed an algorithm called *CMTreeMiner* to discover all *closed* frequent embedded subtrees and maximal frequent embedded subtrees. Here, a frequent subtree is *closed* if none of its proper supertrees has a larger frequency value than itself.
CHAPTER 2. LITERATURE REVIEW

2.1.3 Mining Graph Structured Data

Previously, works have been done in mining tree structured data. However, the expressive power of tree structure is still limited and cannot model the complex relations among different attributes or objects. Recently, many works have been done on mining graph structured data as graph is a more general and expressive structure. Specifically, most works focus on extracting frequent subgraphs from the graph structured database, which has a wide spectrum of applications from chemical applications, bioinformatics applications, to CAD (Computer Aided Design) applications [170].

Similar to the frequent subtree mining problem, the frequent subgraph mining problem is to extract the set of subgraphs that occur frequently enough in the graph database. That is, given the \( \text{minsup} \) threshold, the algorithms are expected to return all the subgraphs whose support values are no less than the \( \text{minsup} \). Note that the definition of \( \text{support} \) is defined in the same as it is defined for subtree in the previous section. The frequent subgraph mining approaches consist of two major phases: candidate subgraph generation and support calculation. In this section, we review the two types of frequent subgraph mining approaches: frequent connected subgraph mining and frequent induced subgraph mining, respectively.

**Graph Isomorphism:** Before we present the frequent subgraph mining approaches, we first introduce the concept of graph isomorphism. Generally, two graphs which contain the same number of graph vertices connected in the same way are said to be isomorphic. Formally, two graphs \( G_1 = (V_1, E_1) \) and \( G_2 = (V_2, E_2) \) are isomorphic iff there is bijection \( f: V_1 \rightarrow V_2 \) such that for all \( u, v \in V - 1: (u, v) \in E_1 \leftrightarrow (f(u), f(v)) \in E_2 \). The bijection, \( f \), is called an isomorphism between the two graphs. For example, the function \( f: \{-1, \cdots, -2n\} \rightarrow \{1, \cdots, 2n\} \), where \( f(k) = -2k \) if \( -n \leq k \leq -1 \) and \( f(k) = 2(2n + k) + 1 \) if \( -2n \leq k < -n \), is an isomorphism between the two graphs described above.

**Mining Frequent Connected Subgraph**

**SUBDUE:** Probably, the first and most well-known heuristic-based approach for mining frequent connected subgraph is the **SUBDUE** system [51]. The system was initially designed
to find patterns that can effectively compress the original input data based on the minimum description length principle, by substituting those patterns with single vertices. To extract the set of frequent substructures, the system starts with a subgraph that consists of a single vertex and grows the subgraph incrementally by absorbing vertices in its neighborhood one by one. Whenever a new vertex is expanded, the description length of the original graph $G$ is updated by replacing each instance of the subgraph as a single vertex. The subgraph keeps expanding until the minimal description length of the original graph is found. The \textit{SUBDUE} system uses a computationally-bounded inexact graph match that identifies similar, but not identical, instances of substructures. The \textit{heuristic beam search} approach makes the algorithm fail to extract some subgraphs that are frequent.

Similarly, a number of approaches have been proposed to find commonly occurred subgraphs in the context of inductive logic programming (ILP) system [126, 137]. A well-known ILP system \textit{WARMR} developed by Dehaspe and De Raedt [59] can find all frequently occurring subgraphs. However, this approach is not optimized for handling graphs, it does not employ any graph-based optimization, and hence it has high computational requirements.

\textbf{FSG:} Kuramochi and Karypis proposed an algorithm named \textit{FSG} to discover frequent connected subgraphs [100, 99]. The \textit{FSG} algorithm finds frequent subgraphs using the same level-by-level expansion used in Apriori [11]. Rather than expanding the subgraphs by adding new vertex, in the \textit{FSG} approach, subgraphs are expanded by adding edges one by one. Note
CHAPTER 2. LITERATURE REVIEW

![DFS code tree in gSpan algorithm.](image)

that this algorithm allows duplicated labels within individual graph.

**FSG-Candidate Generation:** In the candidate subgraph generation process, to avoid generating the same candidate many times, the authors proved that by only joining the primary $k$-subgraphs that share the common $k-1$ connected graph, all valid candidate $k+1$-subgraphs can be generated. Here primary graphs refer to the subgraphs that have the smallest and second smallest canonical label among variant subgraphs of the same size. There are two types of join operations to generate the candidate subgraphs as shown in Figure 2.9. In the first case, the difference between two $k$-subgraphs is a vertex with the same label as shown in Figure 2.9(a). As a result, the join operation will generate two distinct $k+1$-subgraphs. In the second case, the primary subgraphs may have multiple automorphisms and each of them may lead to a different $k+1$-subgraph candidate. For example, three candidates are generated as shown in Figure 2.9(b).

**FSG-Support Calculation:** To obtain the frequency of subgraphs efficiently, for each frequent subgraph, a list of transaction identifiers that include that subgraph is recorded. The frequency of the candidate $k+1$-subgraph is obtained by joining the lists of identifiers of the corresponding $k$-subgraphs. That is, the upper bound of the support for a $k+1$-subgraph is the intersection of the transaction identifiers for the two $k$-subgraphs. If the upper bound is less than the $\text{minsup}$, then this candidate is pruned. Otherwise, the graph isomorphism algorithm will be used to check the support against all the transactions in the intersection list.
**gSpan:** In [179], a new algorithm named gSpan was proposed to discover frequent connected subgraphs without candidate generation. The algorithm builds a new lexicographic order among graphs and maps each graph to a unique minimum DFS (depth-first search) code as its canonical label. As shown in Figure 2.10, each vertex in the DFS code tree represents a subgraph. For example, 0-edge vertices represent single nodes, while 1-edge vertices represent subgraph that contain 1 edge with two vertices. Note that, the n-edge vertices are subgraphs that are extended from their parent n-1-edge vertices by adding one more edge. Then, the frequent subgraph mining process is to search in the DFS tree build based on the lexicographic order, where all the candidate subgraphs can be derived. To calculate the frequency of the subgraphs, the data is scanned on the fly and projected. Note that the smaller the lexicographic label a node has, the earlier the node will be discovered. According to the empirical studies, gSpan and FSG are comparable on a chemical compound dataset, whereas gSpan outperformed FSG on synthetic datasets [179].

**FFSM:** In [81], Huan et al. proposed the FFSM algorithm to extract frequent connected subgraphs using the new canonical form with two efficient candidate generation operations. In their approach, each subgraph is represented as a canonical adjacency matrix, which is further encoded as a code of string that consists of the lower triangular entries of the matrix. For example, Figure 2.11 shows a graph and different adjacency matrix representations. Among them, according to the predefined order that \(a > b > x > y > 0\), the first matrix is defined as the canonical adjacency matrix (CAM) representation. This matrix can be encoded as a row-wise string. Then, using the standard lexicographic order on the total sequences, the order of two codes can be defined. Then, similar to the DFS tree used in gSpan, a CAM tree is constructed based on the FFSM-Join and FFSM-Extension operations. It has been proved that those two operations can produce the complete list of subgraphs. To calculate the frequencies of the subgraphs, the transaction identifiers are recorded and joined. Experimental results showed that FFSM outperforms gSpan by a factor of seven on the same chemical compound benchmark [81].
CHAPTER 2. LITERATURE REVIEW

(a) Graph $G$  
(b) Adjacency Matrix of $G$  

Figure 2.11: Canonical Adjacency Matric used in FFSM algorithm.

Mining Frequent Induced Subgraph

Besides the connected subgraphs, in the literature there is an algorithm called $AGM$ that was proposed to extract frequent induced subgraphs [84, 85]. Note that this algorithm can work for both directed and undirected graph with labels or without labels.

In the $AGM$ algorithm, the graphs are represented as their adjacency matrices as well. For undirected graph, the code of adjacency matrix is defined as the elements in the upper triangular part of the matrix; for directed graph, the code is defined in a similar way except that the diagonally symmetric element $m_{ii}$ is added after each $m_{ji}$ where $j=i-1$. Based on the normal adjacency matrix, the canonical adjacency matrix is defined as the minimal matrix in the same way it was defined in FFSM [81]. The difference is that there is no tree representation for the canonical forms of the subgraphs. The candidate generation process is based on the join operation, where two frequent $k$-subgraphs are joined to form the $k+1$-subgraph candidate if they share all the elements except the elements in the $k$-th row and the $k$-th column. After that the canonical form is used to represent identical subgraphs. The frequency of each candidate subgraph is counted by scanning the database. Basically, the mining process is extended from the Apriori-based association rule mining approaches [11, 132].

Besides mining frequent subgraphs, there are some other approaches that aim to reduce the number of extracted subgraphs as it can be very huge [82, 180]. In [180], the authors propose to reduce the number frequent subgraphs by mining only closed subgraphs. Here
a subgraph $g$ is \textit{closed} if there exist no proper super graph of $g$ that has the same frequency as $g$. In [82], the authors attempt to mining only \textit{maximal frequent subgraphs}. A subgraph $g$ is a \textit{maximal frequent subgraph} if there exist no proper super graph of $g$ that is frequent. More recently, the frequent subgraphs have been used to build index for graph database. In [182], Yan \textit{et al.} proposed an index for graph database called $gIndex$, which makes use of the frequent subgraphs as the basic indexing feature while previous works are path-based.

In summary, existing frequent subgraph mining approaches focused on improving the efficiency of their algorithms using different canonical representations and candidate generation methods. At the same time, different approaches focused on extracting different types of subgraphs such as connected subgraph, induced subgraph, and general subgraph with their own applications. Above all, the most expensive part of existing graph mining algorithms is the graph matching process. We observed that it is the graph isomorphism that makes both the candidate generation and support count calculation difficult.

\subsection*{2.1.4 Mining XML Data}

Recently, XML has become a popular way for storing and exchanging data in different domains from news feeds to DNA sequences as the semi-structured nature makes XML capable of modelling a variety of databases. As a result, there is a handful of literature about XML data mining. Basically, the literature of XML data mining techniques can be broadly classified into three categories: \textit{XML association rule mining}, \textit{frequent substructure mining}, \textit{XML classification and clustering}. We elaborate on the representatives of these approaches in turn.

\textbf{Mining Association Rule from XML}

The first native XML data mining approach is called \textit{XMINE}, which was proposed by Braga \textit{et al.} in [28, 27]. This approach is motivated by the \textit{mine rule} operators in the relational database extended from SQL specification [124]. In [28], the authors extended XQuery for mining XML association rule following the way \textit{mine rule} was created. For example,
Figure 2.12(a) shows an example to extract co-authors in the data source with user-defined support and confidence threshold, Figure 2.12(b) presents some of the extracted association rules. Later in [27], they proposed an architecture of the XMINE operator as a tool for extract association rules and extended it from XQuery to XPath. They also claimed that the XMINE operator can be used to express complex mining tasks on both the content and structure of the XML data. Similar approaches on mining association rules using XQuery have been proposed in [162, 163].

Rather than extending the existing XML query language, Zhang and Yao have proposed a guideline for designing XML algebras for data mining [120]. By comparing the existing XML algebras, they argued that by adding additional operators for mining tasks, the XML algebra may work well for mining the native XML documents. More recently, in [202], Zhang et al. proposed to transform the XML documents into the multi-relational database or the indexed content tree depending on the size of the XML document and the memory constraint of the hardware. Then, association rules are extracted based on the generalized meta-patterns, which are extracted based on the characteristics of the data source such as the density of instances for the targeted elements.
Frequent Substructure Mining

To extract frequent substructures from the XML data, the TreeFinder algorithm was first proposed by Termier et al. in [157]. TreeFinder [157] is an algorithm to find frequent trees that are approximately rather than exactly embedded in a collection of tree-structured data modelling XML documents. As shown in Figure 2.13, each labelled tree is described in relaxed relational description which maintains ancestor-descendant relationship of vertices. Then, the relational representation of input trees are clustered if their atoms of relaxed relational description occur together frequently enough using the Apriori frequent itemset mining algorithm. Then maximal common trees are found in each cluster by using algorithm of least general generalization (LGG). Finally, the LGGs are represented as trees. Recently, there is another line of work that employs the pattern-growth strategy to discover frequent subtrees, which has been reviewed in the previous section of mining tree structured data.

Besides mining frequent substructures from the XML documents, works have been done on mining frequent subtrees from the XML queries for efficient XML query processing and indexing [184, 183, 185, 114]. In [184, 183, 185], the authors proposed different algorithms for extracting frequent rooted query patterns for caching. The intuition is that some query patterns are issued more frequently and the results can be cached to improve the query performance. Two algorithms named FastXMIner and 2PXMiner were proposed. Those approaches are similar to the frequent subtree mining algorithms, but the wildcard and relative path in the queries make the matching process more expensive. More recently, in [114], the authors proposed to index the XML database using minimal infrequent structures (MISs), as those structures are expected to be very selective in the query processing.
process. According to the definition, MISs are structures that 1) exist in the data, 2) are not frequent with respect to a support threshold, and 3) all substructures of them are frequent.

**XML Classification and Clustering**

Similar to the frequent substructure mining of XML data, in the literature there are works on clustering and classification of XML documents and XML schemas.

The first XML document classification and clustering related work is the semantic similarity between XML documents proposed by Lee et al. [105]. They created an extended element vector for each element in an XML document and the similarity matrix for comparing extended element vectors. Then a sequential pattern mining algorithm is used to calculate the similarity between XML documents. To evaluate the role of XML tags in the clustering process, in [62], a naive k-means based clustering approach was used to with three types of features: tag features, text features, and combination of tag and text features. Experimental results showed that the element tags provide additional information that helps to improve the quality of clustering. Similarly, in [116], the XML documents are clustered by applying principle component analysis to the vectors derived from the ordered tree representations and clustering the vectors in the reduced dimensionality using cosine similarity. In [113], a new similarity measure was proposed for clustering XML documents based on the structure graph representation (s-graph) and overlaps between s-graphs, which can produce better clustering results compared to the tree-edit distance based approaches. In [106], Lee et al. proposed to cluster XML Document Type Definitions (DTDs). The goal is to develop scalable integration techniques for the growing number of XML data sources with different DTDs. They proposed an algorithm named XClust, a novel integration strategy that involves the clustering of DTDs. The matching algorithm is based on the semantics, immediate descendent and leaf-context similarity of DTD elements.

Classification of XML documents has also been addressed in [195]. Zaki proposed an algorithm called XRule to construct structural rules in order to classify XML documents. The basic idea is to relate the presence of a particular kind of structural pattern in an
CHAPTER 2. LITERATURE REVIEW

XML document to its likelihood of belonging to a particular class. In [159], the authors proposed to exploit the structure, annotation, and ontological knowledge of XML document for classifying XML documents. By combining the text terms in XML elements with the twigs and tag paths, together with the WordNet thesaurus, the extracted features improved the accuracy of automatic classification.

As a summary, the existing works on mining semi-structured data mostly focused on extracting different types of substructures for different application scenarios. Once the mining problem is well-defined, most of the alternative algorithms focus on improving the efficiency of the algorithm or the space overload. More or less, they mainly deal with the semi-structure property of the data source by using different representations such as canonical forms and other encoding methods.

2.2 Analyzing of Web Data Dynamics

With respect to the dynamic nature of Web data, works have been done on detecting and reasoning the changes to Web data [12, 18, 50, 57, 95, 134, 150, 169, 181]. Existing efforts on analyzing of Web data dynamics can be categorized into four classes: change detection for tree-structured data, time series mining, data stream mining, and detecting and modeling changes to mining results.

2.2.1 Change Detection for Tree Structured Data

Due to the dynamic and autonomous properties of XML data, recently, different techniques of detecting changes to XML documents have been proposed. These approaches can be classified into two groups. One is memory-based XML change detection approaches such as XML TreeDiff [57], Xy-Diff [50], and X-Diff [169]. The other is the relational database-based XML change detection approach [108, 107].

In [57], Curba and Epstein proposed the TreeDiff algorithm to compute the differences between two XML documents using DOMHash values. TreeDiff is a tool developed by IBM to detect changes to ordered XML documents. Using the hash values, TreeDiff can reduce
the size of the trees by filtering the identical subtrees. However, the results generated by TreeDiff may not be the optimal. XyDiff [50] is another algorithm for detecting changes to ordered XML documents. The algorithm computes a signature (i.e., hash value) and a weight (i.e., subtree size) for every node in both documents in a bottom-up fashion. Based on the signature and weight, subtrees with the largest weight are always compared first. If the signatures are equal, the two nodes are matched. The algorithm starts from finding a match between the heaviest nodes and heavier subtrees have higher priority to be chosen for comparison. Once a match is found, it will propagate to ancestors to get more matches. Insertions/Deletions and Moves will be computed after all exact matches are found. However, XyDiff cannot guarantee any form of optimal or near-optimal result because of the greedy rules used in the algorithm. X-Diff [169] is used to detect changes to unordered XML documents. In X-Diff algorithm, for each pair of nodes from the input documents the distance between their respective subtrees is obtained by finding the minimum cost mapping for matching children (by reduction to the minimum cost maximum flow problem). X-Diff generates more accurate results compared with XyDiff. The main strength of X-Diff algorithm is that it reduces the mapping space significantly and achieves polynomial time complexity. However, the change detection response time is slower than XyDiff and it cannot handle very large XML documents.

All these algorithms suffer from scalability problem as they fail to detect changes to large XML documents due to lack of memory. Consequently, a number of approaches [107, 108] have been proposed to address the scalability problem of XML change detection by using relational databases. In this approach, first, XML documents are stored in RDBMS. Then, the changes are detected by using a set of SQL queries. Experimental results show that this approach has better scalability, running time, and comparable result quality compared to existing change detection systems [50, 57, 169].

As a summary, the existing XML change detection approach mainly focused on the following three issues: the efficiency of the change detection algorithms, the scalability of change detection algorithms, and the quality of the change detection results. Experiments show that
Table 2.3: Sequential pattern mining

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Transaction sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{(a,b),(a,c),(a,d,e),(b,c,f)}</td>
</tr>
<tr>
<td>2</td>
<td>{(a,c),(b,c),(d,c,e),(b,c,a,d)}</td>
</tr>
<tr>
<td>3</td>
<td>{(a,c),(f,d),(f,a,e),(d,f,e)}</td>
</tr>
<tr>
<td>4</td>
<td>{(e,b),(c,b),(a,c,e),(b,c,f)}</td>
</tr>
<tr>
<td>5</td>
<td>{(d,a),(a,f,g),(b,f,e),(d,c,e)}</td>
</tr>
</tbody>
</table>

the memory-based approaches outperform the relational database-based approaches for small XML documents in terms of running time. While the relational database-based approaches are more scalable and efficient for large XML documents compared with memory-based approaches. As for the quality of results, both memory-based and relational database-based approaches produce change results with comparable quality.

2.2.2 Time Series Mining

In the last decade, time series mining has been a very active area that has attracted lots of research efforts with the availability of massive historical data [12, 18, 95, 134, 150]. Literally there are hundreds of papers that have introduced new algorithms to index, classify and cluster, and segment time series. As this dissertation focuses on mining Web data, in this section, we review two groups of the time series mining techniques that have been used for Web data mining.

Sequential Pattern Mining

With the rich temporal information embedded in the transactional data such as basket datasets, sequential pattern mining has been proposed to extract subsequences of itemsets that occurred frequently in the dataset in a fixed order [12]. For example, Table 2.3 shows a collection of basket data. Note that each row in the table represent a customer sequence, which represent the sequence of itemsets that were purchased by a specific customer ordered by the transaction time. It is different from the transaction representation used in association rule mining, where each row represents a set of items that are purchased by any customer. For instance, the first row in this table will be represented as four transactions in association rule mining.
Similar to the way of defining frequent itemset, given the sequential transaction representation of the basket dataset, a sequential pattern is defined as a sequence itemsets whose support value is no less than a user-defined minimum support threshold. The definition of support is different from frequent itemset mining. A sequence of itemsets \( \langle a_1, a_2, \ldots, a_n \rangle \), where \( a_i \) is a set of items, supports another sequence of itemsets \( \langle b_1, b_2, \ldots, b_m \rangle \) if \( b_1 \subseteq a_i \), \( b_2 \subseteq a_i \), \ldots, \( b_m \subseteq a_i \) and \( 1 \leq i_1 < i_2 < \cdots < i_m \leq n \). For example, customer_id 1 in Table 2.3 supports the itemsets sequence \( \langle (b), (c), (a, c), (c, f) \rangle \).

As a result, algorithms for mining frequent itemsets have been extended to extract sequential patterns. For instance, in [12], the first two sequential pattern mining algorithms called AprioriSome and AprioriAll is proposed based on the Apriori algorithm in association rule mining. In [150], they proposed a generalized framework for mining frequent sequential pattern by relaxing the transaction boundary by using the user-defined time-window and time constraints and incorporating the taxonomy as well. Under this framework, a more efficient and scalable algorithm named GSP was proposed. After that various algorithms such as SPADE, PrefixSpan, and SPAM have been proposed under this framework [136, 134, 133, 18]. All of them use the depth-first search strategy. SPADE proposed a vertical ID-list representation and performed candidate generation and frequency calculation by joining the ID-lists [133]. PrefixSpan uses projected databases to accelerate the mining process [136, 134]. SPAM uses vertical bitmap representation for candidate generation and frequency calculation [18]. Literature studies showed that PrefixSpan runs faster when the size of the database is small, SPAM outperforms PrefixSpan and SPADE for large datasets, while SPADE has relative small space consumption than SPAM [181, 134, 18].

Besides the issue of improving the efficiency of mining frequent subsequences, some other approaches have been proposed to mine more concise and representative subsequences based on the observation that the number of frequent subsequences increases exponentially for long sequences [181, 167]. In [181], Yan et al. proposed an algorithm named CloSpan to mine the closed frequent sequential patterns. Compared to previous works, they showed that the CloSpan algorithm can generate really long frequent subsequence that cannot be
CHAPTER 2. LITERATURE REVIEW

mined previously and outperforms the previous works. More recently, the BIDE algorithm is proposed to mine the frequent closed sequential patterns using the bi-direction extension approach [167]. Empirical results with real datasets showed that BIDE outperforms CloSpan. Moreover, approximation and incremented sequential pattern mining methods have been proposed as well [98, 123, 38]. In [98], an algorithm named ApporRxMap was proposed to extract consensus sequential pattern in two steps. Firstly, the sequences are clustered into clusters using the pattern-based similarity. Then, within each cluster, consensus sequential patterns are extracted using the multiple alignment. In [38], the authors classify the updates of sequential database into insertion and appending and they proposed the IncSpan algorithm to incrementally extract frequent sequential patterns from the appended sequential database.

Clustering and Classification of Time series

Clustering and classification problems have been actively researched for decades [95]. However because of the unique characteristics of time series such as high dimensionality, high feature correlation, and large amount of noises, most of the classic machine learning and data mining algorithms do not work well for time series. As a result, clustering and classification of time series have been explored with respect to their characteristics. Most of the contributions in the existing approaches focus on either proposing different representations or providing new similarity measures with classic classification and clustering algorithms.

In the literature, there are in total six different types of representation of the time series. They are the Discrete Fourier Transform (DFT) [9], Wavelet families (DWT) [33], Singular Value Decomposition (SVD) [97], Adaptive Piecewise Constant Approximation (APCA) [31], Inner Products [65], and Piecewise Aggregated Approximation (PAA) [191]. As for similarity measures, there are many such as Euclidean distance, edit distance, cosine Wavelet, and autocorrelation function [92]. Details of the representation, quality of those similarity measures, and error rates have been reviewed and extensively studied using experiments in [95]. Also there are many applications for classification and clustering time series data such as event detection, bioinformatics, finance, and music [77, 109].
CHAPTER 2. LITERATURE REVIEW

As a summary, existing time series mining approaches focused on representation and accurate similarity measures for time series data. They all focus only on traditional datasets such as transaction database and temporal database that are well-structured.

2.2.3 Data Stream Mining

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate such as the sensor network data, Web logs, and computer network traffic. Recently, a lot of works have been done on mining data stream [66]. Basically, the works can be classified into three categories: change detection from data stream, mining frequent itemsets from stream data, and classification and clustering of stream data.

Change Detection of Stream Data

Considering the dynamic nature of stream data, it is an important issue to maintain the models built from the data. In [70], the authors proposed a generic algorithm for model maintenance for stream data and mining on a temporal subset of the database. Also, a generic framework for change detection, that quantifies the difference between two datasets in terms of the data mining models they induce, was discussed. At the same time, in [3, 4], Aggarwal proposed a framework to diagnose changes in stream data. A new concept of velocity density estimation was introduced to understand, visualize and determine trends in the evolution of fast data streams. Using the velocity density estimation, he proposed to create both temporal velocity profiles and spatial velocity profiles at periodic instants in time. These profiles are then used in order to predict three kinds of data evolution: dissolution, coagulation and shift. Methods were also proposed to visualize the changing data trends. Later, in [96], the authors claimed a first step towards formalizing the detection and quantification of change in a data stream. In their approach, the data points are generated sequentially and independently by some underlying probability distribution. The objective of their framework is to detect when this distribution changes, and to quantify and describe this change. More recently, Yang et al. proposed to organize the data history into a history of concepts using a measure of conceptual equivalence [189]. They proposed to identify new
CONCEPTS AS WELL AS RE-APPEARING ONES, AND LEARN TRANSITION PATTERNS AMONG CONCEPTS TO HELP PREDICTION. DIFFERENT FROM CONVENTIONAL METHODOLOGY THAT PASSIVELY WAITS UNTIL THE CONCEPT CHANGES, THEIR APPROACH INCORPORATES PROACTIVE AND REACTIVE PREDICTIONS THAT CAN PREDICT THE CHANGES IN ADVANCE AND ADAPT A PREDICTION MODEL INSTANTLY.

MINING FREQUENT ITEMSETS

ANOTHER IMPORTANT ISSUE IS TO MINE FREQUENT ITEMSETS FROM THE STREAM OF TRANSACTIONS SUCH AS MARKET BASKET DATA [66]. Generally, the frequent itemset mining approaches can be classified into two categories: False-Positive and False-Negative. In the former approaches, some infrequent patterns are included in the final result, whereas some frequent patterns are missed in the later approaches.

In [73], Giannella et al. proposed a frequent itemset mining algorithm for stream data. They assume that users are interested in only the most recent transactions and propose to extract frequent itemsets from most recent transactions based on the concept of tilted windows. An algorithm named FP-stream was proposed to incrementally maintain the frequent itemsets using a tree data structure. Later some approximation based frequency count algorithms were proposed to extract the frequent itemset in stream data [122, 14]. Although the output is approximate, they have proved that the error is guaranteed not to exceed a user-specified parameter. More recently, Cormode and Muthukrishnan developed two algorithms for counting the frequency of itemset based on “group testing” [54, 55]. The algorithm proposed is able to maintain the hottest items with user-defined parameters while the data source can undergo both insertion and deletion.

While the above approach are False-Positive oriented, two False-Negative approaches have been proposed [192, 49]. In [192], they proposed an algorithm that can efficiently extract the frequent itemsets and items with a bound of memory consumption (based on the Chernoff bound) and the number of false negative itemsets can be controlled by a predefined parameter. In [49], they provided theoretical bound of the algorithms and analyzed the possibility of minimization of missing frequent items, in terms of two possibilities: impossibility and out-possibility. Here the former is about how a frequent item can possibly
pass the first pruning. The latter is about how long a frequent item can stay in memory while no occurrence of the item comes in the following data stream for a certain period. Experimental results showed that their algorithms are efficient and effective.

Classification and Clustering

For classification and clustering of stream data, one of the major issues is speed as the data may come and go very quickly. In [60], the authors proposed a new method for decision tree classification on spatial data streams using a data structure called Peano Count Tree (P-tree). Using P-tree structure, fast calculation of measurements, such as information gain, can be achieved. Their experimental results showed that the P-tree method is significantly faster than existing classification methods, making it the preferred method for mining on spatial data streams. In [7], the first classification approach that considered the data stream classification problem from the point of view of dynamic. An on-demand approach, in which simultaneous training and testing streams are used, was proposed to classify the data stream. The objective of this approach is to create a classification system in which the training model can adapt quickly to the changes of the underlying data stream. Note that the proposed classification process can dynamically select the appropriate window of past training data to build the classifier. Empirical results showed that the system maintains high classification accuracy in an evolving data stream.

In [5], the first framework for clustering data stream with respect to the application-centered requirements. The idea is to divide the clustering process into an online component and an offline component. The online component periodically stores the summarized statistics of the data stream, while the offline component uses only those stored summaries. As a result, the input to the clustering problem can be varied in order to fully understand the data stream. They proposed a pyramidal time frame with a micro-clustering approach. More recently, in [6, 8], considering the fact that data stream are high dimensional and previous projection method cannot be generalized to data stream, a new, high-dimensional, projected data stream clustering method, called HPStream was proposed. The method incorporates a fading cluster structure, and the projection based clustering methodology. It
CHAPTER 2. LITERATURE REVIEW

is incrementally updatable and is highly scalable on both the number of dimensions and the size of the data streams, and it achieves better clustering quality in comparison with the previous stream clustering methods. Empirical study with both real and synthetic data sets demonstrates that their framework is efficient and effective.

As a summary, the existing data stream mining approaches mainly focus on either updating or maintaining the data mining results such as the frequent itemsets, the clustering and classification results or detect the underlying changes to the data stream.

2.2.4 Detect and Model Changes to Mining Results

Considering the dynamic nature of Web data, there are a set of approaches that were proposed to detect and model the changes to the Web mining results.

In [68], Ganti et al. proposed a framework to measure the changes in data characteristics. They tried to quantify the difference between two datasets in terms of the models induced from them. The goal is to detect whether the underlying datasets have statistically significant differences in their characteristics. Further, in [69], they introduced a new dimension, called the data span dimension, which allows user-defined selections of a temporal subset of the database. Then, they proposed a generic algorithm that takes traditional incremental model maintenance algorithm and transforms it into an algorithm that allows restrictions on the data span dimension.

Later, in [115], Liu et al. proposed to detect changes of association rules from one time period to another. They presented a technique to highlight the small subset of fundamental changes, where a change is fundamental if it cannot be explained by some other changes. Similarly, in [20], the authors divided the KDD process into two phases. In the first phase, data from the first period is mined and interesting rules and patterns are identified. In the second phase, using the data from subsequent periods, statistics of these rules are extracted in order to decide whether or not they still hold.

In [19], Baron and Spiliopoulou presented a framework that can monitor the changes a rule undergoes and the emerging trends over rules when the dataset is updated. They
CHAPTER 2. LITERATURE REVIEW

propose a generic rule model that distinguishes between different types of pattern changes, and provide formal definitions for the changes. Later, in [21], they introduced PAM, an automated Pattern Monitor, to observe changes to the behavior of a web site's visitors. More recently, in [161, 45], the authors proposed an approach to extract semantic similarities between web search queries with respect to the evolution of query popularity.

However, most of the above approaches focus only on representing and detecting changes by analyzing two consecutive versions of rules extracted from Web data. Such approaches may be able to monitor the micro change pattern of the mining results. For instance, by monitoring the changes to the mined association rules, existing approaches can only model the changes to frequent itemsets and itemsets that changed from infrequent/frequent into frequent/infrequent. Whereas the evolution of other parts of data sources are ignored. More importantly, the evolution of the original data source is different from the evolution of the mining results. That is, the evolution of the mining results cannot accurately reflect the evolution of the source data.

2.3 Summary

We conclude this chapter by summarizing the features of existing Web mining and analyzing approaches. Basically, Web data have five major characteristics: semi-structured, massive, dynamic, temporal, and semantics. Note that dynamic refers to the changes to the data source as Web data may change from one version to another version. The temporal property refers to timestamps associated with the Web data. For example, a typical approach that addresses the dynamic property of Web data is the change detection approach, which compares two versions of Web data and returns the differences. However, the time series mining approaches mainly focused on the temporal property of the data source. That is to model and predict characteristics of Web data by analyzing historical data with the temporal information. Existing Web mining and analyzing approaches were designed to address one or more of them as shown in Table 2.4. Now we summarize the existing approaches and compare them with the MONETA framework we proposed.
CHAPTER 2. LITERATURE REVIEW

<table>
<thead>
<tr>
<th></th>
<th>Semi-structured</th>
<th>Massive</th>
<th>Dynamic</th>
<th>Temporal</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mining tree data</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mining graph data</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mining XML data</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Change detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Time series mining</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Data stream mining</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Monitor mining results</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>MONETA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2.4: Comparison of existing Web mining approaches with MONETA.

The first category of existing works is semi-structured data mining, which consists of mining tree data, mining graph data, and mining XML data. Semi-structured data mining approaches mainly focus on the structural characteristic of Web data. For example, different representation methods such as DOM tree and graph have been proposed and used. At the same time, the massive property of Web data is also addressed by proposing scalable algorithms. However, to the best of our knowledge, the dynamic, temporal, and semantic nature of Web data are ignored. For instance, the existing XML query mining algorithms and cache strategies ignored that fact that both the XML data source and the queries may evolve over time. Moreover, both the tree-structured and graph-structured data mining approaches return only the substructures rather than semantics. People may argue that semantics can be extracted by postprocessing the substructures with related domain knowledge, however, in our MONETA framework, semantics can be extracted without resorting to domain knowledge. For example, We shall present how to detect events from click-through data under the MONETA framework in Chapter 7.

The second category of existing works is analyzing Web dynamics, which consists of change detection, time series mining, data stream mining, and monitor mining results. The change detection approaches mainly focus on the dynamic, massive, and semi-structured properties. For instance, the XML change detection techniques focus on detecting changes from the tree representation of XML document versions. However, they ignore the temporal and semantic properties. That is, changes are detected only for two versions of XML documents, while the historical change patterns of XML documents are ignored. Moreover, none
of the existing change detection approaches has considered the semantics behind changes to the data.

Most of existing time series mining approaches focus on the temporal information associated with the data sources. However, time series mining approaches were not specifically designed for Web data. As a result, existing approaches ignore the semi-structured, dynamic, and semantic properties of Web data. That is, time series mining approaches were widely used for well-structured and high-dimensional data such as temporal database, spatial temporal database, and customer transaction database. Without the structural information, the semantics within Web data is lost in the time series mining approaches. For instance, in the sequential pattern mining approaches, the mining results are sequences of itemsets while the relations between items and itemsets are ignored.

The existing works on stream data mining focus on the dynamic nature of Web data by updating the mining results such as frequent itemsets and classification models over the data stream. However, existing approaches only focus on well-structured data, similar to time series mining. Similar to the time series mining, existing data stream mining approaches ignore the semi-structured and semantic properties of Web data. Moreover, stream data mining approaches use the order among the data source rather than the temporal information, which may produce more informative results.

Another line of works focus on monitoring the mining results. These approaches can detect the changes of mining results and monitor change patterns of the mining results. However, mining results cannot accurately reflect the data sources in terms of evolution. Moreover, the semi-structured and semantic properties are ignored in existing works about monitoring mining results. That is, without the structural information, the semantics within Web data is not fully exploited.

Our MONETA framework is proposed to address the above five issues together. We use the semi-structured data as examples and focus on extracting novel knowledge by incorporating the dynamic, temporal, and semantic properties of Web data. Such novel knowledge,
as explained in Chapter 1, cannot be extracted efficiently using existing Web mining and analyzing techniques.
Chapter 3

Mining XML Document Versions

Recently, XML (eXtensible Markup Language) has become the standard for representing and exchanging semi-structured data on the Web. With the self-describing feature, XML has been used for data integration and exchange in many different areas such as biology, multi-media, e-commerce, and personal blog. That is, there is a massive volume of XML data available. As a result, mining XML data has become an interesting and important research issue [50, 159, 106, 195, 169, 107, 113, 105]. However, it has been observed that none of the existing XML mining techniques has systematically exploited the dynamic nature of XML data.

As Web data is dynamic, both the author-centric and visitor-centric data are dynamic as well. For example, Figure 3.1 shows two versions of an XML document that describe a part of the faculty database. It can be observed that XML documents may change in different ways. That is, new elements such as publications may be inserted, obsolete elements may be deleted, and some attributes such as academic affiliation may be updated as well. Note that such changes can be attributed to different factors such as new publications, new activities, and other real world events.

Table 3.1 shows two groups of XML queries and their supports (number of times they are issued against the total number of queries have been issued) at two different time intervals. It can be observed that there are new queries emerging such as XML queries 201 and 202 in Table 3.1(b); some queries are not issued anymore such as XML queries 101 and 103 in Table 3.1(a); and some queries are issued more/less frequently than it was before.
CHAPTER 3. MINING XML DOCUMENT VERSIONS

(a) A piece of professor data version 1

(a) A piece of professor data version 2

Figure 3.1: Examples of XML document dynamics.

<table>
<thead>
<tr>
<th>ID</th>
<th>XML query</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>/professor/bio/industry</td>
<td>0.18</td>
</tr>
<tr>
<td>102</td>
<td>/research/activity/conf</td>
<td>0.10</td>
</tr>
<tr>
<td>103</td>
<td>/professor/publication</td>
<td>0.13</td>
</tr>
<tr>
<td>104</td>
<td>/bio/edu/phd</td>
<td>0.11</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(a) The first week

<table>
<thead>
<tr>
<th>ID</th>
<th>XML query</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>/professor/bio/academic</td>
<td>0.12</td>
</tr>
<tr>
<td>202</td>
<td>/bio/edu/ms</td>
<td>0.09</td>
</tr>
<tr>
<td>203</td>
<td>/research/activity/conf</td>
<td>0.32</td>
</tr>
<tr>
<td>204</td>
<td>/professor/research/project</td>
<td>0.03</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(b) The second week

Table 3.1: Examples of XML query dynamics.

For instance, queries 102 and 203 in Table 3.1 are the same. However, it occurred more frequently in the second group than in the first group.

In this chapter, we will focus on the problem of mining author-centric XML data. Specifically, historical XML document versions are used. In Section 3.1, we motivate our research by presenting the types of novel knowledge that can be discovered from the XML document versions. Also, example applications are used to illustrate the usefulness of such novel knowledge. In Section 3.2, we present the techniques for representing historical XML document versions. After that, two sets of algorithms called FCS mining (Frequently Changing Struc-
CHAPTER 3. MINING XML DOCUMENT VERSIONS

ture) and FASST mining (Frequent chAning Semantic STructures) are presented to extract the frequently changing structures and frequent changing semantic structures in Sections 3.3 and 3.4, respectively. Moreover, performances of our proposed algorithms are studied in detail as well. Lastly, Section 3.5 concludes this chapter.

3.1 Motivation

Existing works about mining XML data can be classified into three categories: XML association rule mining, frequent substructure mining, and XML classification/clustering. XML association rule mining approaches focus on either exploring the ability of XML query language for mining association rules [27, 28] or designing XML algebras for data mining [120, 202]. Frequent substructure mining algorithms for XML were proposed by adopting tree structured data mining algorithms and modelling XML data as trees [114, 157, 183, 184, 185]. Whereas, the XML classification/clustering approaches focused on proposing new similarity measures by utilizing or integrating the tree or graph structure representation of the XML data [105, 62, 116, 113, 106].

However, based on our discussion in Chapter 2, it can be observed that existing XML data mining approaches mainly focus on designing algorithms by exploring the semi-structure property and ignoring the dynamic property. Whereas, there are different types of novel knowledge can be discovered by exploiting the dynamic nature of the XML data. As for the author-centric XML document versions, the following types of novel knowledge can be discovered.

- *Frequently changing structures*: Different parts of the XML documents changed in different ways over time. Some parts of the XML document changed *frequently* and *significantly* in the history, which we refer to as the frequently changing structures. Here, *frequently* refers to the large number of times the corresponding parts changed, while *significantly* refers to the large percentage of nodes that have changed in the corresponding subtree representation.
CHAPTER 3. MINING XML DOCUMENT VERSIONS

- **Structure change association:** Similar to the transactional association rule, different parts of the XML data may be associated in terms of their changes over time. That is, whenever one part of the XML data changed, the other part of the XML data also changed with certain probability. Furthermore, knowledge such as whenever certain parts changed, the other parts will not change with certain probability can be extracted as well.

- **Change pattern:** Among these frequently changing structures, more specific change patterns can be extracted by analyzing both the frequency and significance of the changes over time. For example, some structures changed periodically on a weekly basis, while others do not follow any periodical patterns. Some structures change more and more frequently over time, while other structures become more and more stable.

Such novel knowledge, which cannot be extracted by existing XML mining techniques that ignore the temporal information, can be useful in many applications as we discussed in Chapter 1. Here, we present the details for two of the representative applications: **XML indexing** and **XML change detection**.

**XML indexing:** As we know that one of the key issue of XML indexing is to identify the ancestor and descendant relationship quickly. To this end, different numbering schemes have been proposed [110, 88]. Li and Moon proposed a numbering scheme in XISS (XML Indexing and Storage System) [110]. The XISS numbering scheme uses an extended preorder and a size. The extended preorder allows additional nodes to be inserted without reordering and the size determines the possible number of descendants. More recently, XR-Tree [88] was proposed to index XML data for efficient structural joins. Compared with the XR-tree [88], XISS numbering scheme is more flexible and can deal with dynamic updates of XML data more efficiently. However, Li and Moon did not highlight on how much extra space should be allocated. Allocating too small reserved space will lead to the ineffectiveness in maintaining the numbering scheme, whereas allocating too much extra space will lead to too
large numbers being assigned to nodes in a large XML document. Moreover, in the XISS approach, the gaps are equally allocated, while in practice different parts of the document change with different significance. Based on our mining results, the numbering scheme can be improved by allocating the gaps in a more intelligent manner. For example, for the parts of structure that change frequently and significantly, larger gaps are allocated while for frozen structures, smaller gaps can be reserved. By using this strategy, the numbering scheme should be more efficient in terms of both index maintenance and space allocation.

**XML Change Detection:** One of the major limitations of existing XML change detection systems [50, 169] is that they are not scalable for very large XML documents. Using the frequently changing structures and associations between changes extracted from the history, the scalability of XML change detection system can be improved. For instance, we can focus on detecting changes for the frequently changing structures. That is, only parts of the frequently changing substructures are parsed and compared. Note that, the association between structure changes can also be used. Suppose, there is a association which states that when part $A$ of the XML document changes most of the time part $B$ will change. Then we can load part $B$ of the document for change detection when part $A$ changes. Based on such knowledge, rather than loading the entire XML documents, only parts of the documents that are likely to change are loaded and compared. We believe that this will improve the efficiency and scalability of change detection process, especially for very large XML documents.

### 3.2 Overview and Background

As the frequently changing structures are the fundamental part for most of the other types of novel knowledge, in this chapter we focus on discovering the *frequently changing structures* from versions of XML document. More specifically, we present two sets of algorithms to extract the *frequently changing structures* and the *frequently changing semantic structures*. Before we discuss the details of the specific approaches, in this section, we present the background knowledge about representing the structure of XML documents and the *structural*
### Chapter 3. Mining XML Document Versions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X)</td>
<td>An XML document.</td>
</tr>
<tr>
<td>(T)</td>
<td>Tree representation of an XML document.</td>
</tr>
<tr>
<td>(t)</td>
<td>A subtree in XML tree (T).</td>
</tr>
<tr>
<td>(o_i)</td>
<td>A basic structural edit operation.</td>
</tr>
<tr>
<td>(\Delta_i(t))</td>
<td>Structural delta of subtree (t) from (i)-th to ((i+1))-th version.</td>
</tr>
<tr>
<td>(</td>
<td>\Delta_i(t)</td>
</tr>
<tr>
<td>(t_i \cup t_j)</td>
<td>Consolidate structure of subtree (t_i) and (t_j).</td>
</tr>
<tr>
<td>(V(t))</td>
<td>Version dynamic of subtree (t).</td>
</tr>
<tr>
<td>(N_i(t))</td>
<td>Structure dynamic of subtree (t) from version (i) to ((i+1)).</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>Threshold for structure dynamic.</td>
</tr>
<tr>
<td>(DoD(t, \alpha))</td>
<td>Degree of dynamic of subtree (t).</td>
</tr>
<tr>
<td>(\beta)</td>
<td>Threshold for version dynamic.</td>
</tr>
<tr>
<td>(\gamma)</td>
<td>Threshold for DoD.</td>
</tr>
<tr>
<td>(H)</td>
<td>The H-DOM tree.</td>
</tr>
<tr>
<td>(C_v(t))</td>
<td>Number of nodes.</td>
</tr>
<tr>
<td>(C_n(t))</td>
<td>Number of versions.</td>
</tr>
<tr>
<td>(PoC)</td>
<td>Percentage of changes.</td>
</tr>
<tr>
<td>(C)</td>
<td>A concept.</td>
</tr>
<tr>
<td>(t \approx C)</td>
<td>A substructure (t) provides the required information for concept (C).</td>
</tr>
</tbody>
</table>

| Table 3.2: A summary of symbols |

![Tree representation of XML fragment.](image)

\(\delta\) of XML document versions. Table 3.2 provides a summary of symbols used in the subsequent sections.

The structure of an XML document can be modelled as a tree according to the Document Object Model (DOM) specification. Specifically, we model the structures of XML documents as unordered, labeled, rooted tree structures. The reason that we use the unordered tree representation is that it has been argued that an unordered model (only ancestor relationships are significant) is more suitable for most database applications [169].

The structure of an XML document is denoted as \(T = (N, E, r)\), where \(N\) is the set of
CHAPTER 3. MINING XML DOCUMENT VERSIONS

Figure 3.3: Version 2 and version 3 of the XML fragment.

labeled nodes, \( E \) is the set of edges, \( r \in N \) is the root. Note that we do not distinguish between elements and attributes, both of them are mapped to the set of labeled nodes. Each edge, \( e = (x, y) \) is an ordered pair of nodes, where \( x \) is the parent of \( y \) in the DOM representation of XML. The size of the structure \( T \), denoted by \( |T| \), is the number of nodes in \( N \). For example, Figure 3.2 shows the unordered tree representation of the XML document fragment shown in Figure 3.1(a). Figure 3.3 shows the tree representation for part of the changes to the XML document in another two versions. Note that, the gray nodes represent elements that have been deleted, whereas the bold nodes represent elements that are newly inserted. In the tree representation, only insertions and deletions are considered since we focus on the structural changes in this dissertation.

Using the unordered tree representation, all changes to XML documents can be represented as five types of edit operations [169]. The first three are basic operations and the last two are composite operations that can be represented as a list of basic operations.

- **Insert** \((x(name, value), y)\): insert a node \( x \), with node name \( name \) and node value \( value \), as a leaf child node of node \( y \).
- **Delete** \((x)\): delete a leaf node \( x \).
- **Update** \((x, new.value)\): change the value of a leaf node \( x \) to \( new.value \). Note that only the value can be updated, but not its name.
- **Insert** \((T_x, y)\): insert a subtree \( T_x \), which is rooted at \( x \), to node \( y \).
CHAPTER 3. MINING XML DOCUMENT VERSIONS

- **Delete($T_x$):** delete a subtree $T_x$, which is rooted at node $x$.

Based on the edit operations, an *edit script* is defined as a sequence of edit operations that transform an XML document from one version to another [169]. However, not all the edit operations can change the structure of the XML documents. For example, the *Update* operation will not change the structure of a document. Corresponding to the structural changes, we define the *structural edit script* as a sequence of edit operations that convert one structure to another. It is similar to the *edit script* except that all *Update* operations are excluded in the *structural edit script*. To make it easier to locate the edit operation, an *affiliated node* is defined for each edit operation. For all the insertion operations $\text{Insert}(x(\text{name, value}), y)$, their affiliated nodes are nodes $y$; for all the deletion $\text{Delete}(x)$ and updating $\text{Update}(x, \text{new.value})$, their affiliated nodes are $x$.

**Definition 3.1 Structural Delta:** Let $T_i$ and $T_{i+1}$ be the tree representations of two XML documents $X_i$ and $X_{i+1}$. The structural delta from $X_i$ to $X_{i+1}$ is represented as $\Delta_i$, where $\Delta_i$ is a minimal structural edit script $\text{min}(\langle o_1, o_2, \cdots, o_m \rangle)$ that transforms $T_i$ into $T_{i+1}$, denoted as $T_i \xrightarrow{o_1, o_2, \cdots, o_m} T_{i+1}$, where $o_j$ ($1 \leq j \leq m$) is a basic edit operation.

The size of the structural delta, denoted as $|\Delta_i|$, is defined as the number of basic edit operations in the structural edit script. The structural delta is defined as the minimal structural edit script (minimal number of basic edit operation) between two version of XML documents. Consider the previous examples in Figure 3.3, the structural delta from the left version to right version is $\Delta_2=(\text{Delete}(C_1), \text{Delete}(J_1), \text{Insert}(C_3(3, CFP)), \text{Insert}(J_2(J_2), CFP))$ and the value of $|\Delta_2|$ is 4 since there are 4 basic edit operations.

### 3.3 FCS Mining

In this section, we present the novel approach for mining frequently changing structures from XML document versions. Specifically, we present the details of the three main components in the MONETA framework for mining frequently changing substructures. Also, performance
study and optimization techniques are presented as well. Note that, in this section, we
are interested in substructures at all possible granularities from the single node level to the
maximal subtrees that satisfy our proposed dynamic measures.

3.3.1 Measurement

In this section, we present a set of metrics to systematically measure the evolution pattern
of the XML document structures. Firstly, we introduce some preliminary concepts that will
be used to define the dynamic metrics. Then, three dynamic metrics will be presented (one
micro-pattern based metric and two macro-pattern based metrics). Note that, in the rest of
this chapter, whenever we use the term subtree, it refers to the induced subtree defined in
Chapter 2.1.

Definition 3.2 Structural Delta of Substructures: Let $\langle \Delta_1, \Delta_2, \cdots, \Delta_t \rangle$ be a se-
quene of structural delta for an XML document $X$, whose tree representation is $T_i$. Suppose
t = $(N_t, E_t, r_t)$ is a substructure of $T_i$. Then, the sequence of structural delta for $t$ is denoted
as $\langle \Delta_{t_1}, \Delta_{t_2}, \cdots, \Delta_{t_i} \rangle$, where for all operation $o_j \in \Delta_{t_i}$, $\Delta_{t_i} \subseteq \Delta_i$ and their affiliated nodes
should be in $N_t$.

Reconsider the examples in Figures 3.2 and 3.3. For the substructure rooted at node
Activity, the corresponding of structural delta is $\langle \Delta_1, \Delta_2 \rangle$ where $\Delta_1$ and $\Delta_2$ are the sets
of structural edit operations whose affiliated nodes are included in the substructure rooted
at node Activity.

Definition 3.3 Consolidate Structure: Given two structures $t_i$ and $t_j$, where $r_i = r_j$.
The consolidate structure of them is denoted as $t_i \uplus t_j$, where $i) N_{t_i \cup t_j} = N_{t_i} \cup N_{t_j}$, $ii)$
e = $(x, y) \in E_{t_i \cup t_j}$, if and only if $x$ is the parent of $y$ in $E_{t_i}$ or $E_{t_j}$.

Consider the structures in Figure 3.3. For the substructures rooted at node CFP in
version 2 and version 3, the consolidate structure is the structure rooted at node CFP in
version 3.
From the example in Figures 3.2 and 3.3, we observed that different substructures of
the XML document might change in different ways at different frequencies. To evaluate the
historical behavior of different substructures, we propose a set of dynamic metrics.

**Definition 3.4 Structure Dynamic:** Let \( \langle T_i, T_{i+1} \rangle \) be the tree representations of XML
documents \( \langle X_i, X_{i+1} \rangle \). Suppose \( t \leq T_i \). The structure dynamic of \( t \) from document \( X_i \) to
document \( X_{i+1} \), denoted by \( N_i(t) \), is defined as:
\[
N_i(t) = \frac{|\Delta_i|}{|t_i \cup t_{i+1}|},
\]
where \( N_i(t) \) is the structural dynamic of \( t \) from version \( i \) to \( i+1 \). By using the consolidation
structure, the total number of unique nodes in the two versions can be obtained as \( |t_i \cup t_{i+1}| \).
It includes not only nodes that are in version \( i+1 \) but also nodes that have been deleted in
version \( i \). \( N_i(t) \) is the percentage of nodes that have changed from \( X_i \) to \( X_{i+1} \) in \( t \) against
the number of nodes in its consolidation structure. For example, consider the two structures
shown in Figures 3.2 and 3.3. We calculate the structure dynamic value for the substructure
rooted at node \( DM \) from version 1 to version 2. Based on the definition, \( |\Delta_{DM} \rangle = 2, \)
\( |DM_1 \cup DM_2| = 19 \). Consequently, \( N_1(DM) = 0.11 \) (2/19). It also can be observed that
\( N_i(t) \in [0,1] \). If \( t \) is inserted or deleted, then the corresponding value of structure dynamic
is 1 since \( \Delta_i = t_i \cup t_{i+1} = t \). If \( t \) did not change from version \( i \) to version \( i+1 \), then the
value of structure dynamic is 0 since \( |\Delta_i| \) is 0. It can be implied that the larger the value
of structure dynamic is, more significantly the substructure changed.

**Definition 3.5 Version Dynamic:** Let \( \langle T_1, T_2, \cdots, T_n \rangle \) be the tree representations of
XML documents \( \langle X_1, X_2, \cdots, X_n \rangle \). Suppose \( t \leq T_j \). The version dynamic of \( t \), denoted as
\( V(t) \), is defined as:
\[
V(t) = \frac{\sum_{i=1}^{n-1} v_i}{n-1} \text{ where } v_i = \begin{cases} 1, & \text{if } |\Delta_i| \neq 0; \\ 0, & \text{if } |\Delta_i| = 0; \end{cases}
\]
Consider the 3 different versions of the XML document in Figures 3.2 and 3.3. We calculate the version dynamic value for the substructure rooted at node DM. Here, \( n = 3 \). For the first delta, \(|\Delta_{DM}| \neq 0\), so \( v_1 = 1 \). Similarly, \( v_2 = 0 \). Then, \( \sum_{i=1}^{2} v_i = 1 \). Consequently, the version dynamic of this substructure is 0.50 (1/2). It can be observed that \( V(t) \in [0, 1] \). If \( t \) changed in every version in the history, then the corresponding value of \( \sum_{i=1}^{n} v_i \) is \( n - 1 \), so the version dynamic value is 1. If \( t \) did not change in the history at all, then the value of \( \sum_{i=1}^{n} v_i \) is 0 and version dynamic value is 0. Also, it implies that the larger the value of version dynamic is, more frequently the substructure changed in the history.

A major difference between \( N_i(t) \) and \( V(t) \) is that \( V(t) \) measures the macro-pattern based changes over the history while \( N_i(t) \) measures the micro-pattern based changes between two consecutive versions. Thus, for a substructure there is one value for version dynamic and a sequence of values for structure dynamic. To measure the macro-pattern based change behavior of a substructure in terms of both significance and frequency, we proposed another dynamic metric named degree of dynamic, denoted as \( \text{DoD} \). \( \text{DoD} \) is the extension of structure dynamic by incorporating the version dynamic metric. It represents the overall significance of the structural changes in the history.

**Definition 3.6 Degree of Dynamic:** Let \( (T_1, T_2, \cdots, T_n) \) be the tree representations of XML documents \( (X_1, X_2, \cdots, X_n) \). Suppose \( t \preceq T_j \), \( N_i(t) \) and \( V(t) \) are the values of structure dynamic and version dynamic of \( t \). The degree of dynamic, \( \text{DoD} \), for \( t \) is defined as:

\[
\text{DoD}(t, \alpha) = \frac{\sum_{i=1}^{n} d_i}{(n-1) \times V(t)} \quad \text{where} \quad d_i = \begin{cases} 1, & \text{if } N_i(t) \geq \alpha \\ 0, & \text{if } N_i(t) < \alpha \end{cases}
\]

where \( \alpha \) is the pre-defined threshold for structure dynamic.

The metric \( \text{DoD} \) is defined based on the threshold of structure dynamic. It represents the fraction of versions, where the structure dynamic values for the substructure are no less than the predefined threshold \( \alpha \), against the total number of version the substructure
CHAPTER 3. MINING XML DOCUMENT VERSIONS

has changed over the history. Consider the examples shown in Figures 3.2 and 3.3. We can calculate the DoD value for the substructure rooted at node XML. From the previous examples, we know that the structure dynamic values of this substructure are 0.11 and 0. The version dynamic value is 0.50. Suppose the threshold for structure dynamic is set to 0.10, then the value of DoD is 1 (1/1). If the threshold for structure dynamic is set to 0.30, then the corresponding DoD value will be 0 (0/1). It is obvious that, \( \forall \alpha, \text{DoD}(t, \alpha) \in [0,1] \.

Similar to the structure dynamic, the value of DoD also implies the overall significance of the substructure, the larger the value is, more significant the changes are.

3.3.2 Representation

In this section, we present the representation methods for efficient storage of the historical tree structured data. Specifically, we will present the H-DOM tree structure.

The structure of an XML document can be represented and stored as a tree such as the DOM tree proposed by W3C. However, in our problem of mining evolution of XML document versions, given a sequence of history XML documents, it is not efficient to store them in a sequence of DOM trees as there are overlaps among these DOM trees. We present an H-DOM model to represent the history of changes to XML data. The H-DOM is an extension of the DOM model with some historical properties so that it can compress the history of changes to XML into a single tree. It is a general model to store and represent both the structure of the XML documents and the historical changes to XML data such as insert and delete, which are represented as the dynamic metrics. Formally, we define an H-DOM tree as follows:

**Definition 3.7 H-DOM:** An H-DOM tree is a 4-tuple \( H = (N, A, v, r) \), where \( N \) is a set of object identifiers; \( A \) is a set of labelled, directed arcs \( (p, l, c) \) where \( p, c \in N \) and \( l \) is a string; \( v \) is a function that maps each node \( n \in N \) to a set of values \( (C_n, C_v) \), \( C_n \) is an integer and \( C_v \) is a binary string; \( r \) is a distinguished node in \( N \) called the root.

We now elaborate on the parameters \( C_n \) and \( C_v \). The two parameters are introduced to record the historical changes for each substructure. \( C_n \) is an integer that records the number
of versions that a substructure has changed significantly enough (the structure dynamic is no less the corresponding threshold). $C_v$ is a binary string that represents the historical changes of a substructure. The length of the string is equal to the number of deltas in the XML sequence. The $i$th digit of the string denotes the change status of the structure from $X_i$ to $X_{i+1}$, where the value of 1 means this structure changed, the value of 0 means it did not change. In the H-DOM tree, the $C_v$ value for each structure is lastly updated by using the formula: $C_v(t) = C_v(t_1) \lor C_v(t_2) \lor \cdots \lor C_v(t_j)$, where $t_1, t_2, \cdots, t_j$ are the substructures of $t$.

Figure 3.4(a) is part of the H-DOM for the structure sequence in Figures 3.2 and 3.3. Suppose the threshold for structure dynamic is 0.30, the $C_n$ value of node XML is 2, which means that this structure has changed twice in the history with a structure dynamic value no less than 0.30. The $C_v$ value 100 of node p3 means that this node has changed from $X_1$ to $X_2$. The $C_v$ value of the internal nodes and root node are calculated according to the above formula. With $C_v$ and $C_n$, values of the dynamic metrics can be calculated as follows.

- $N_t(t) = \sum_{j=1}^{n} C_v(d_j)[i]$, where $d_j$ is the list of descendant nodes of $t$, $C_v(d_j)[i]$ is the $i$th digit of $C_v(d_j)$.
- $V(t) = \sum_{i=0}^{n-1} C_v[i]$, where $C_v[i]$ is the $i$th digit of $C_v(t)$; $n$ is the total number of XML documents.
CHAPTER 3. MINING XML DOCUMENT VERSIONS

• \( DoD(t) = \frac{C_v}{\sum_{i=1}^{n} C_v[i]} \), where \( C_v[i] \) is the \( i \)th digit of \( C_v(t) \); \( n \) is the total number of XML documents.

The H-DOM model is inspired by the FP-Tree in association rule mining [79]. It is designed to preserve and compress the historical structural information of XML versions. H-DOM compresses the historical structural data by representing the identical nodes only once in the H-DOM tree, while the related historical information is preserved using a binary string and an integer. Compared to the FP-Tree, the compactness of H-DOM is higher since the same nodes may appear more than once with \( h \)-links in the FP-tree. Moreover, the FCSs can be extracted without any candidate generation process by traversing the H-DOM exactly once, while in FP-Tree there is a conditional FP-Tree generation process. Another feature of the H-DOM tree is that it expresses the temporal features of the XML structures.

3.3.3 Problem Statement

The problem of frequently changing structure mining is to discover those structures that changed significantly and frequently in the history. Based on the above set of metrics, the frequently changing structure is defined as follows.

**Definition 3.8 Frequently Changing Structure:** Let \( \langle T_1, T_2, \cdots, T_n \rangle \) be the tree representations of XML documents \( \langle X_1, X_2, \cdots, X_n \rangle \). The thresholds for structure dynamic, version dynamic, and degree of dynamic are \( \alpha, \beta, \gamma \) respectively. A structure \( t \preceq T_j \) is a frequently changing structure (FCS) in this sequence iff: \( V(t) \geq \beta \) and \( DoD(t, \alpha) \geq \gamma \).

The FCS is defined based on the predefined thresholds of the dynamic metrics. The significance of changes are defined by structure dynamic \( \alpha \) and degree of dynamic \( \gamma \), while the frequency of changes are defined by version dynamic \( \beta \).

Consider the versions of XML shown in Figures 3.2 and 3.3. An example of the frequently changing structure will be the structure rooted at node Publication. This structure may indicate that the corresponding professor is very active in publishing papers as more papers are inserted over time.
Algorithms 3.3.4 Mining Algorithms

In this section, we present the algorithms for discovering the frequently changing structures (FCS). Given a sequence of XML document version, the frequently changing structure mining algorithm consists of two main phases: H-DOM tree construction and FCS extraction.

H-DOM Construction

Algorithm 1 describes the process of H-DOM construction. Given a sequence of historical XML documents, the H-DOM tree is initialized as the structure of the first version. After that, the algorithm iterates over all the other versions by extracting the structural deltas and mapping them into the H-DOM tree. The SX-Diff function is a modification of the X-Diff [169] algorithm, which is the state-of-the-art change detection approach based on the minimal edit script cost model used in [200]. Our SX-Diff function generates only the structural change for two different versions of the document using the X-Diff algorithm. The structural delta is mapped into the H-DOM tree according to mapping rules as described in Algorithm 2. This process iterates until no more XML document is left in the sequence. Finally, the H-DOM tree is returned as the output of this phase. Note that, the change detection problem for unordered XML itself is NP-Complete [200]. In this dissertation, we adopt the state-of-the-art XML change detection algorithm and our focus is mining the structural delta between different versions of XML documents.

74
Algorithm 2 Mapping

Input:
H-DOM tree: $H$
Structural delta: $\Delta$
Threshold of structure dynamic: $\alpha$

Output:
Updated H-DOM tree: $H$

Description:
1: for all $n_i \in \Delta$ do
2: \hspace{1em} if $n_i \neq \emptyset$ then
3: \hspace{2em} for all $n_i \in H$ do
4: \hspace{3em} update $C_n(n_i)$
5: \hspace{2em} \hspace{1em} if $N_i(n_i) \geq \alpha$ then
6: \hspace{3em} \hspace{2em} update $C_v(n_i)$
7: \hspace{3em} \hspace{3em} $n_i = N_i.parent(H)$
8: \hspace{2em} \hspace{2em} end if
9: \hspace{2em} end for
10: \hspace{1em} end if
11: end for
12: Return($H$)

Algorithm 2 describes the mapping function. Given the H-DOM tree and the structural changes, this function is to map the deltas into the H-DOM tree and return the updated H-DOM tree. The idea is to update the corresponding values of the nodes in the H-DOM tree, for all the nodes in the structural delta. Note that each node is identified as the corresponding path from the root node to the target node. The values of the nodes are updated according to following rules:

i) If the node does not exist in the H-DOM tree, then the node is inserted. The value of $C_v$ is set to $000 \cdots 1$ where the $i$th digit of the string is set to 1 and $i$ is the version number of the structural delta. In addition, the value of $N_i$ is calculated. If $N_i \geq \alpha$, then $C_n$ is set to 1 and the $C_n$ values of its parent nodes are incremented by 1 until $N_i$ is less than $\alpha$. Otherwise, $C_n$ is set to 0 and the process terminates.

ii) For nodes that exist in the H-DOM, the value of $C_v$ is updated by inserting a 1 at the $i$th digit of $C_v$ where $i$ is the version number of the structural delta. The value of $C_n$ is also updated based on $N_i$ and $\alpha$. Similarly, if $N_i \geq \alpha$, then $C_n$ is incremented by 1 and the $C_n$ values of its parent nodes are updated based on the same rule until $N_i$ is less than $\alpha$. 

75
Algorithm 3 FCS Mining (top-down)

Input:
H-DOM tree: $H$
Threshold of version dynamic and degree of dynamic: $\beta, \gamma$

Output:
A set of FCS rooted nodes: $F$

Description:
1: for all $n_i \neq \emptyset$ //Top-down BFS do
2: if $V(n_i) \geq \beta$ then
3: if $DoD(n_i) \geq \gamma$ then
4: $F$ is updated by incorporating $n_i$
5: end if
6: else
7: prune all descendants of $n_i$
8: end if
9: end for
10: Return($F$)

$\alpha$. Otherwise, $C_n$ does not change and the process terminates.

FCS Extraction

In this section, we explain the FCS mining/extraction algorithm to extract FCS from the H-DOM tree representation. More specifically, given the H-DOM tree, the values of the required parameters (structure dynamic, version dynamic, and degree of dynamic) for each node are calculated and compared against the predefined thresholds. Since for a FCS, both its version dynamic and degree of dynamic should be no less than the thresholds, we first calculate only one of the parameters and determine whether it is necessary to calculate the other parameter. Because if any of the two parameters does not satisfy the definition, the substructure cannot be a FCS. In our algorithm, the version dynamic for a node is checked against the corresponding threshold first. If it is no less than the threshold, then we check its degree of dynamic. Considering the traversal strategy of the H-DOM tree, two approaches are analyzed: the bottom-up (level by level) approach and the top-down (breadth first) approach. Before we come to the details of the traversal strategies, we present two lemmas that will be used to make the extraction phase more efficient.

Lemma 3.1 Let $n_i, n_j \in N$ be any two nodes. The substructures rooted at $n_i$ and $n_j$ are
Algorithm 4 FCS Mining (bottom-up)

Input:
H-DOM tree: $H$
Threshold of version dynamic and degree of dynamic: $\beta$, $\gamma$

Output:
A set of FCS rooted nodes: $F$

Description:
1: for all $n_i \neq \emptyset$ //bottom-up BFS do
2: if $C_n < \gamma \times V(t_{n_i})$ then
3: $n_i = n_i$ next
4: else
5: if $V(n_i) > \beta$ then
6: if DoD($n_i) \geq \gamma$ then
7: $F$ is updated by incorporating $n_i$
8: end if
9: end if
10: end if
11: end for
12: Return($F$)

denoted as $t_{n_i}$ and $t_{n_j}$ respectively. If $n_i$ is the ancestor of $n_j$, then $V(t_{n_i}) \geq V(t_{n_j})$.

Proof: The proof is intuitive. Based on the previous definition, once a node changes, superstructures that include this node are considered as changed. It indicates that the number of versions a superstructure has changed should be no less than its substructures. Consequently, it can be concluded that the version dynamic of a superstructure should be no less than the version dynamic of its substructures, while the total number of versions is the same. ■

Lemma 3.2 Let $t_1$ and $t_2$ be any two structures and $t_2 \leq t_1$. Given the threshold for DoD as $\gamma$, the necessary condition for structure $t_1$ to be a FCS is that $C_n(t_1) \geq \gamma \times V(t_2) \times (n - 1)$.

Proof: From Lemma 1, we can infer that $V(t_1) \geq V(t_2)$. The necessary condition for structure $t_1$ to be a FCS is that its degree of dynamic is no less than the threshold $\gamma$, which is $\gamma \leq \frac{C_n(t_1)}{V(t_1) \times (n - 1)}$. Then, $C_n(t_1) \geq \gamma \times V(t_1) \times (n - 1)$, while $V(t_1) \geq V(t_2)$, it can be inferred that $C_n(t_1) \geq \gamma \times V(t_2) \times (n - 1)$. ■

Based on the above lemmas, we observed that it is not necessary to traverse the entire H-DOM tree. We can skip checking some structures that cannot be FCSs. Lemma 1 can be
used in the top-down traversal strategy. When we reach a node where its version dynamic is less than the threshold, it is not necessary to further traverse down this substructure since the version dynamic of its substructures will definitely be less than the threshold and they cannot be FCSs. Lemma 2 can be used in the bottom-up traversal strategy. In this case, for any node, rather than calculate its version dynamic value, the $C_n$ value of the node is checked against the value of $\gamma \times V(t_i)$, where $t_i$ is any of its substructures. If $C_n < \gamma \times V(t_i)$, then it is not necessary to calculate the version dynamic and degree of dynamic for this structure since it cannot be a FCS. Based on the lemmas, the top-down FCS mining algorithm and the bottom-up FCS mining algorithm are presented in Algorithm 3 and Algorithm 4.

**Algorithm Analysis**

In this section, we analyze the time complexity and space complexity of the FCS basic algorithms. Since the visualization is trivial, we focus on analysis of the first and second phases.

**Time Complexity:** In phase 1, the H-DOM tree is constructed based on the sequence of historical XML documents. In this phase, each XML document is parsed once and only consecutive versions are compared. Let $\langle |T_1|, |T_2|, \ldots, |T_n| \rangle$ and $\langle |t_1|, |t_2|, \ldots, |t_{n-1}| \rangle$ denote the number of nodes in the sequence of XML documents and the structure deltas respectively. The complexity of SX-Diff is $O(|T_i| \times |T_{i+1}|) \times \max\{\deg(T_i), \deg(T_{i+1})\} \times \log_2(\max\{\deg(T_i), \deg(T_{i+1})\})$ according to [169]. The complexity of the mapping process is $O(|t_i|)$. The SX-Diff and mapping process iterate $k - 2$ times in this phase, while the cost of the initialization is $O(|T_1|)$. Since $|t_i| \leq |T_i|$, the dominant of this iteration is the SX-Diff. The overall complexity of phase 1 is $O((k-2) \times \max\{|T_i| \times |T_{i+1}|\} \times \max\{\deg(T_i), \deg(T_{i+1})\} \times \log_2(\max\{\deg(T_i), \deg(T_{i+1})\}))$, where $i \in [2, k-1]$. In phase 2, the H-DOM is traversed and the parameters for all the potential FCSs are calculated and compared against the predefined thresholds. No matter which traversal strategy we choose, the upper bound of this phase is $O(|T|)$, which is an entire traversal of the H-DOM tree, where $|T|$ is the total number of nodes in the H-DOM tree. In practice, the actual cost of this phase is
substantially cheaper than this, since we use Lemma 1 and Lemma 2 to reduce the traversal space. From the above analysis, it can be inferred that the bottleneck of the FCS mining is the structural change detection process, which is the most expensive process.

**Space Analysis:** In the FCS basic algorithms, the history of XML structural deltas is stored in the H-DOM tree and it is processed in memory. The space cost of this algorithm is the size of the H-DOM tree. Based on the algorithm, we observed that the size of the H-DOM tree depends on the overlaps between the consecutive versions. For the same number of XML documents with the same value of average number of nodes, the more significantly they change, larger the size of the H-DOM is. Since only the structural data is stored and each unique node is store only once, the size of the H-DOM should be no larger than the total size of the sequence of XML documents. However, as the sizes of the XML documents increase or the changes become more significant, or the number of XML documents increases, the size of H-DOM will increase accordingly. However, the upper bound of the space requirement is $O(|t_1 t_2 \cdot \cdot \cdot t_n|)$, where $(t_1, t_2, \cdot \cdot \cdot, t_n)$ are the tree representations of the XML documents sequence $(X_1, X_2, \cdot \cdot \cdot, X_n)$.

### 3.3.5 Optimization Techniques

Based on the analysis of the FCS basic algorithms, in this section we propose three optimization techniques. The compression techniques, the build and merge strategy, and the DTD-based pruning technique. The objective of these techniques is to make the algorithm more scalable by reducing the size of the H-DOM tree.

**Compression Technique:**

In the H-DOM model, suppose there are $n$ versions of XML in the sequence. Then, for each node a length $n$ binary string is used to represent the history of changes. However, we observed that the size of the string can be very large, while only 2 out of $n$ digits are useful since each node itself in the H-DOM could change at most twice, *insertion* and *deletion*. Consequently, rather than using the binary string, we use two integers to represent the changes. Consider the H-DOM tree in Figure 3.4 as an example. For node $p_2$, suppose it is
deleted in the $i+1$th version, then the $C_v$ value of this node will be 1000⋯01 in the basic approach. Now we only store two integers 1 and $i$ to represent the changes. Using the basic algorithm the space requirement is $i$ bites, but using this strategy it only requires 8 bites (for two integers). It is obvious that when $i > 8$, the later strategy is more efficient in terms of space. Usually, to get useful knowledge from the changes, the number of versions is greater than 8.

**Building and Merging Strategy:**

Based on the basic algorithms, we observed that for any structure that has been deleted their $C_n$ and $C_v$ values would not change since no change could happen to them again. Thus, whether this structure is a FCS or not can be determined by then. Different from the FCS basic algorithms, we propose not to keep all substructures in the H-DOM tree. If the structures are not a FCS when they are deleted, only the root nodes are stored in the H-DOM, with the summarized historical information. By using this strategy, the size of the H-DOM tree can be reduced. Consider the H-DOM tree in Figure 3.4. Suppose in the next version the substructures rooted at $DM$ and $DB$ are deleted. Based on the thresholds, substructure $DM$ is a FCS while substructure $DB$ is not. Then, rather than storing the entire substructure of $DM$ and $DB$, only the root node of $DM$ is stored with the summarization of historical information. Similarly, the substructure $DB$ is merged into its parent node as shown in Figure 3.4(b). The building and merging algorithm is shown in Algorithm 5.

**DTD-based Pruning Technique:**

We observed from the history of the structural changes that some of the nodes never change in the history. Thus, it is not necessary to store such information since it cannot be used for FCS mining. If we can prune such nodes during the H-DOM construction phase, then the H-DOM tree will be more compact and the efforts of checking such nodes can be avoided. With the help of DTD and schema, elements and attributes in the XML documents can be categorized into two classes. Elements and attributes that can be inserted or deleted individually are classified to class 1, while elements and attributes that cannot be inserted
Algorithm 5 Building and merging

Input:
- H-DOM tree: \( H \)
- Structural delta: \( \Delta \)
- Threshold of structure dynamic: \( \alpha \)

Output:
- Updated H-DOM tree: \( H \)

Description:
1. for all \( n_i \in \Delta \) do
2. if \( n_i \) is deleted according to the DTD and \( V(n_i) < \alpha \) then
3. update\((n_i, H)\)
4. prune all descendants of \( n_i \)
5. else
6. update\((n_i, H)\)
7. end if
8. end for
9. Return\((H)\)

or deleted individually are in class 2. Using the DTD, we know that elements defined with more than one occurrences can be inserted or deleted while elements that are defined as default and required for exactly one occurrence cannot be deleted or inserted individually. Our DTD-based pruning strategy maps only nodes belong to class 1, while nodes in class 2 are merged into their parent nodes to save space. For example, suppose elements BS, MS and PhD are defined as required subelements with exactly one occurrence for element Edu in Figure 3.1. Then, rather than storing the entire substructure rooted at node Edu, it can be represented as a single node Edu in the H-DOM tree.

3.3.6 Performance Study

Experimental Setup and Dataset

We ran experiments on a PC with Intel Pentium 4, 1.7GHz CPU, 512 RAM, 40G hard disk, and Microsoft Windows 2000. We have implemented two basic algorithms, the bottom-up based algorithm FCS-BASIC-B and the top-down based algorithm FCS-BASIC-T. We also implemented optimization-based algorithms by combining the optimization techniques we proposed: FCS-A is implemented by integrating the three optimization techniques; FCS-C is implemented with the compression technique and the building and merging technique.
### Chapter 3. Mining XML Document Versions

#### Table 3.3: Symbols, Descriptions, and Datasets

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoN</td>
<td>Number of Nodes</td>
<td>α</td>
<td>Threshold of Ni(s)</td>
</tr>
<tr>
<td>NoV</td>
<td>Number of Versions</td>
<td>β</td>
<td>Threshold of V(s)</td>
</tr>
<tr>
<td>PoC</td>
<td>% of Changes</td>
<td>γ</td>
<td>Threshold of DoD(s, α)</td>
</tr>
</tbody>
</table>

(a) Statistics of Datasets

(b) Parameters

<table>
<thead>
<tr>
<th>dataset</th>
<th>NoN</th>
<th>NoV</th>
<th>PoC</th>
<th>α</th>
<th>β</th>
<th>γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10644</td>
<td>20</td>
<td>10%</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>2</td>
<td>21464</td>
<td>20</td>
<td>10%</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>43196</td>
<td>20</td>
<td>10%</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>4</td>
<td>87642</td>
<td>20</td>
<td>10%</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

(c) Description of Datasets

<table>
<thead>
<tr>
<th>Source data</th>
<th>NoN</th>
<th>NoV</th>
<th>PoC</th>
<th>α</th>
<th>β</th>
<th>γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGMOD1</td>
<td>1124</td>
<td>20</td>
<td>10%</td>
<td>-</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>DBLP1</td>
<td>1143</td>
<td>20</td>
<td>10%</td>
<td>0.2</td>
<td>-</td>
<td>0.4</td>
</tr>
<tr>
<td>Synthetic1</td>
<td>1264</td>
<td>20</td>
<td>10%</td>
<td>0.2</td>
<td>0.4</td>
<td>-</td>
</tr>
</tbody>
</table>

(d) Description of Datasets

<table>
<thead>
<tr>
<th>Source data</th>
<th>NoN</th>
<th>NoV</th>
<th>PoC</th>
<th>α</th>
<th>β</th>
<th>γ</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGMOD2</td>
<td>-</td>
<td>30</td>
<td>10%</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>DBLP2</td>
<td>5743</td>
<td>-</td>
<td>10%</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>1264</td>
<td>20</td>
<td>-</td>
<td>0.2</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

(e) Description of Datasets

For XML data sequences without DTDs. Note that the optimization-based algorithms are implemented using the bottom-up traversal strategy since the metrics can be calculated more efficiently in this way.

We use synthetic XML delta sequences generated from three XML documents, which are real and synthetic XML documents. The two real XML documents we use are DBLP and SIGMOD XML downloaded from UW XML repository, while the synthetic XML is generated by IBM XML Generator. From such XML documents, sequences of XML versions are generated by using our synthetic XML delta generator. We do experiments by using

---

datasets of different characteristics and varying the parameters of each algorithm. For each algorithm, different XML datasets are used to show how the datasets affect the performance. Experiments with the same dataset and all possible variations of the parameters have also been done to show how the parameters can affect the performance. The symbols for characteristics of the datasets and parameters of the algorithms are shown in Tables 3.3(a) and 3.3(b) with their descriptions.

Variation of Algorithm Parameters

We evaluate the performance of the four algorithms, FCS-BASIC-T, FCS-BASIC-B, FCS-A, and FCS-C, by varying the thresholds of the three major parameters, $\alpha$, $\beta$, and $\gamma$. Table 3.3(d) shows the characteristics of the datasets and some parameters used in our experiments. Hereafter, we use the symbol "-" to denote the parameters or characteristics of the dataset that will be varied in the experiments. Figure 5.3(a) shows the performance of the algorithms by varying the threshold $\alpha$. We use the SIGMOD1 XML dataset. Figure 5.3(b) shows how the algorithms perform when the threshold $\beta$ changes. We use the DBLP1 XML dataset. Figure 5.3(c) describes how the changes of threshold $\gamma$ may affect their performances. We use the Synthetic1 XML dataset. From the above figures, following observations can be made.

None of the algorithms is sensitive to the changes of $\alpha$, $\beta$ or $\gamma$. However, the overall observation is that as any of the thresholds increases, the execution time decreases. This is due to the fact that when the threshold increases, the pruning techniques are more efficient and the search space of FCS is reduced. The execution time does not change significantly with the changes of thresholds because that the major cost of the algorithms is the cost of SX-Diff, which is independent to the thresholds. As shown in Figure 5.3(d), the SX-Diff cost is more than 50% of the total cost. From Figure 5.3(d), we also observed that as the total number of nodes increases the percentage of SX-Diff cost also increases.

The FCS-BASIC-T algorithm is more stable than others as $\alpha$ changes, but it is more sensitive to the changes of $\beta$. This is due to the fact that FCS-BASIC-T uses the heuristic
CHAPTER 3. MINING XML DOCUMENT VERSIONS

3.5.a: Variation of $\alpha$  
3.5.b: Variation of $\beta$  
3.5.c: Variation of $\gamma$  
3.5.d: Execution Time  
3.5.e: Variation of $\text{NoN}$  
3.5.f: Variation of $\text{NoV}$  
3.5.g: Variation of $\text{PoC}$  
3.5.h: Size of the H-DOM tree

Figure 3.5: FCS Experiment Results
Chapter 3. Mining XML Document Versions

in Lemma 1 to prune the H-DOM tree, and other algorithms use the heuristic in Lemma 2, where Lemma 1 is solely based on $\beta$ and Lemma 2 is based on $\alpha$.

For the same dataset, with the same parameters, we observed that the differences of execution time for the four algorithms are in constant order. It means that although different traversal strategies, optimization techniques are used, the time cost does not change significantly.

Characteristics of Datasets

We evaluate the performance of the four algorithms by varying the characteristics of the datasets. Table 3.3(e) shows the values of the parameters and some of the characteristics of the datasets used in our experiments. Figure 5.3(e) shows the performance of the algorithms using SIGMOD2 by varying $NoN$, the average number of nodes in each XML document, from 8,000 to 40,000 (the corresponding size of each XML document is from 3M to 15M), with 30 versions in the sequence. Figure 5.3(f) presents the performance of the algorithms using DBLP2 by varying $NoV$, the number of version in the sequence from 30 versions to 150 versions. Whereas the average size of each XML document is 2.3M (5743 nodes). Figure 5.3(g) evaluates the performance of the algorithms by varying $PoC$. The Synthetic2 dataset is used. From the above figures, several observations can be made.

As the average number of nodes $NoN$ in the XML document increases, the time cost increases. Changes are more significant compared to the changes in Figures 5.3(f) and 5.3(g). It is because when the average number of nodes increases, the SX-Diff cost increases, the pruning and extraction phase become more expensive too.

As the total number of versions $NoV$ in the XML sequence increases, the execution time of the algorithms increases too. It is obvious that when the total number of versions increases, the number of comparison increases accordingly. Consequently, the cost for detection the structural change increases. Compared with the changes of $NoN$, the changes of $NoV$ do not affect the performance significantly.

As the percentage of changes $PoC$ in the XML sequence increases, the execution time also increases. Since our FCS mining is actually dealing with the deltas rather than the
original sequence, as the PoC increases, the size of the delta increases. Consequently, the SX-Diff, the pruning and extraction phases become more expensive.

**Compression Efficiency Experiments**

We evaluate the space efficiency of the algorithms by comparing the compactness of the H-DOM tree. Table 3.3(c) shows the characteristics of the datasets and parameters used in the experiments. Figure 5.3(h) shows the size of the H-DOM trees for different datasets and different algorithms.

From Figure 5.3(h), we can observe that compared to the original dataset, the H-DOM trees are very compact. The compression rate of the H-DOM tree is almost 50% without any optimization techniques. With the optimization techniques, the FCS-C, and FCS-A are more compact than the FCS-BASIC. Especially, the H-DOM tree built using the FCS-A is the most compact one. The compression rate of the FCS-A is around 30% according to our experiments. This fact also explains why the time cost of the FCS-C and FCS-A are relatively more expensive as shown in the results shown in Figures 5.3(a) to 5.3(g).

From the above observations, we can conclude that our proposed algorithm FCS-BASIC-T and FCS-BASIC-B are efficient and scalable while three optimization techniques, compression technique, building and merging strategy, and DTD-based pruning strategy, have improved the space efficiency substantially. Based on the experiment results, if users want to find out FCS with higher version dynamic, the FCS-BASIC-T is recommended. Otherwise, the FCS-BASIC-B is the best choice, since the three optimization techniques work in a bottom-up manner. The FCS-C algorithm can be applied to any datasets, while the FCS-A can only be used for datasets with DTDs.

### 3.4 Extraction of Interesting FCS

In this section, we elaborate on how FCS mining framework can be used to extract interesting FCS (denoted as FASST). A set of user-specified concepts is used to guide the interesting FCS mining process. These concepts represent the semantic objects in the XML documents.
There are two approaches to obtain such concepts. The first approach is to extract interesting concepts from domain-specific ontology. The second approach is to build the concepts based on DTDs/XML Schemas used in this domain to represent the collection of XML documents. We represent interesting concepts in form of a concept hierarchy that specifies the relation among them. Nodes in the concept hierarchy can be classified as primitive or nonprimitive. The primitive concepts, which represent the basic elements in a domain, reside in the lowest level in the hierarchy; all nonprimitive concepts, which consist of a conglomeration of the primitive concepts, reside in the higher level of the hierarchy. The higher the node's level, the more complex is the concepts it represents. Figure 3.6 shows an example of concept hierarchy. The leaf nodes such as P-ID, and P-NAME are primitive concepts; while internal nodes and root node such as CLIENT and COMPANY are nonprimitive concepts. In our FASST mining, we assume that the specified concept hierarchy is provided by users.

Given the user-specified concept hierarchy, thresholds for evolution metrics, and a sequence of versions of XML documents, the goal of FASST mining is to discover interesting FCS from the document collection. Given a concept C in a specific domain, a structure t in an XML document is an interesting structure with respect to concept C, denoted as \( t \sim C \), if t provides the required information of the concept C. Furthermore, if t is a frequently changing structure then it is called interesting FCS. We now elaborate on the algorithm for discovering FASST.

### 3.4.1 FASST Mining Algorithm

The FASST mining algorithm is similar to the FCS mining algorithm and consists of two main phases: the **H-DOM tree construction** phase and the **FASST extraction** phase.
CHAPTER 3. MINING XML DOCUMENT VERSIONS

Figure 3.7: DTD of the real Yahoo auction data.

Table 3.4: Datasets for evaluation of FASST.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of versions</th>
<th>Average size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_1$</td>
<td>120</td>
<td>2500</td>
</tr>
<tr>
<td>$D_2$</td>
<td>120</td>
<td>varying</td>
</tr>
<tr>
<td>$D_3$</td>
<td>varying</td>
<td>2500</td>
</tr>
<tr>
<td>$D_4$</td>
<td>120</td>
<td>5000</td>
</tr>
<tr>
<td>$D_5$</td>
<td>240</td>
<td>2500</td>
</tr>
</tbody>
</table>

the H-DOM tree construction phase is similar to the one discussed earlier, we focus on the FASST extraction algorithm.

Given the H-DOM tree, the FASST extraction algorithm is to extract all FASST based on the user-defined concept hierarchy and evolution constraints. First the substructures are compared with the user-specified concept hierarchy. If the structures are instances of the concepts in the hierarchy, then the values of the required parameters (version dynamic, and DoD) for each node are calculated and compared against the predefined thresholds. Note that the comparison strategy of the evolution metrics is same as that discussed in FCS mining. Also, we use the bottom-up traversal approach for traversing the H-DOM tree since the set of interesting concepts is represented in a hierarchical manner with primitive concepts in the lower level.

3.4.2 Performance Evaluation

In this section, we evaluate the performance of FASST mining algorithm for extracting interesting FCS. We use real datasets extracted from Yahoo! auction site (http://auctions.yahoo.com/) for our performance study. A portion of the DTD of the XML view of the data is shown.
in Figure 3.7. The maximum depth of the XML document is 7 and the average depth is 5.37. To obtain different versions of the auction data, we issue queries related to all the five categories (music, tv, movies, books, and collectibles) of products periodically and each result set is then transformed into an XML document. For instance, we issue the five queries every hour and convert the results into an XML document. In the following experiments, we assume that users are interested in the following two concepts: product category and product. The dataset consists of 120 versions of auction XML data and is shown in Table 3.4.

Efficiency and Scalability: First, we evaluate the efficiency and scalability of the FASST algorithm and compare it with the FCS algorithm (FCS-BASIC-T). Figure 3.8(a) shows how the running time changes by varying the total number of nodes in the XML documents. There are two ways of increasing the total number of nodes. One way is to increase the number of versions (NoV) in the XML sequence, another way is to increase the average number of nodes (NoN) in each version. To increase the number of versions, we need to crawl the data more frequently, while to increase the size of each version, we need to crawl the data less frequently. Datasets $D_2$ and $D_3$ are used in this set of experiments. We set $\alpha = \beta = \gamma = 0.2$. Both results show good scalability with the total number of nodes, while the running time is more sensitive to the number of versions in the XML sequence than the average number of nodes in each version. The reason is that the change detection process is the most expensive phase of the algorithm as shown in Figure 3.8(b). Moreover, the FASST mining algorithm is around 2 times faster than the FCS algorithm for the benchmark dataset as only a portion of the H-DOM tree is processed.
CHAPTER 3. MINING XML DOCUMENT VERSIONS

Figure 3.9: Effect of FASST on the thresholds of evolution metrics and results of compactness of H-DOM+ tree.

Figure 3.10: Effect of depth of concept substructures and number of interesting FCS.

Figure 3.8(b) shows the cost of each phase in the FASST mining algorithm. The dataset $D_1$ is used and $\alpha = \beta = \gamma = 0.2$. Similar to the FCS mining results, it can be observed that the change detection phase takes a share of up to 40% of the cost. Compared to the cost of SX-Diff in FCS algorithm in Figure 5.3(d), the percentage of cost for detecting changes decreases in the FASST algorithm. This is because only changes to the user-specified interesting structures are detected.

**Effects of Parameters:** Figure 3.9(a) shows how the running time changes by varying the thresholds of evolution metrics. We use the $D_1$ dataset. In the three experiments, we vary one of the thresholds and fixed the thresholds for the other two to 0.2. It can be observed that the running time does not change significantly when the thresholds of evolution metrics are varied. This is due to the fact that the most expensive process, SX-Diff, is independent of the thresholds.

Figure 3.10(a) shows how the depth of the concept structure that users are interested in affect the performance of the FASST algorithm. In this set of experiments, dataset $D_1$ is used. Fixing the thresholds of the evolution metrics to 0.2, the targeted concepts are varied from level 2 (denoted as case 1), level 3 (denoted as case 2), level 2 and 3 (denoted as case 3), 90
CHAPTER 3. MINING XML DOCUMENT VERSIONS

level 4 (denoted as case 4), to level 3 and level 4 (denoted as case 5). It can be observed that the maximum level of the concepts directly affects the performance of the FASST algorithm. The smaller the minimum level, the more efficient is the FASST algorithm. For concepts that have the same maximum level, the performance is quite similar as shown in this figure. This is because the maximum level of the targeted concepts determines the maximum level the FASST algorithm explores for both change detection and computation of values of evolution metrics.

Figure 3.10(b) shows the number of structures in the mining results using the above thresholds for the evolution metrics. It shows that the number of structures in FASST mining result is reduced by almost 40% compared to that of FCS mining result. This is because the search space for discovering FCS is reduced considerably as FASST only process substructures that users are interested in.

Analysis of Mining Results: In the above experiments, we have successfully extracted interesting concepts such as frequently changing products and frequently changing product categories. We now show that these concepts are indeed FCS by analyzing some of the representative interesting FCS in the actual data. Along with this, we also show the evolution patterns of some other concepts that occurred in the auction datasets. These structures were not considered as FCS as they did not satisfy the evolution metrics.

In Figures 3.11(a) and (b), we present the structure dynamic values of three product categories and three specific topics under these categories. Note that the datasets used in Figures 3.11(a) and (b) are crawled every 6 hours and every 2 hours, respectively. Suppose that \( \alpha = \beta = \gamma = 0.2 \). Then, books and comics are two interesting FCS. Note that the dotted lines in Figures 3.11(a) and (b) are the thresholds for structure dynamic. A larger structure dynamic value indicates that more elements are inserted and deleted under the corresponding subtree. For instance, the results in Figure 3.11(a) indicate that the books is one of the very popular categories where people keep bidding frequently and new products are inserted constantly. Figure 3.11(b) shows the historical structure dynamic values for a
set of products. It can be observed that some products became less popular with time (such as TV series), some products became more popular (such as stamps), and other products changed in various ways (such as money). Observed that stamps is a FCS when the threshold values are set to 0.2.

In summary, experimental results show that FASST has good scalability and efficiency. It is more efficient than the FCS algorithm by integrating the user-defined semantic concepts. Moreover, it reduces the number of substructures in the mining results and presents only the targeted substructures that users are interested in.

3.5 Summary

The existing XML data mining approaches consider XML data as snapshot data, while it is dynamic in real life. The dynamic nature of XML data leads to several challenging problems. In this chapter, we propose to mining the frequently changing substructures from XML document versions by taking into account the dynamic nature. Specifically, two approaches are presented to extract the frequently changing substructures and the frequently changing semantic substructures based on our proposed dynamic metrics and tree representations. Experimental results with both synthetic datasets and real datasets show that our mining approaches can discover novel knowledge that cannot be discovered using existing XML mining and querying techniques. Moreover, our mining algorithms are efficient and scalable.
Chapter 4

Mining Historical XML Query Patterns

In the previous chapter, XML document versions are used as the representative of author-centric tree structured data to illustrate the MONETA framework. In this chapter, we shall illustrate the MONETA framework using visitor-centric tree structured data. Specifically, we choose historical XML queries as the example for visitor-centric data and the reasons are as follows. Firstly, XML queries are visitor-centric data and they can be modeled as tree structures [184, 185]. Secondly, XML queries are expected to be used extensively in the future for querying both surface Web and hidden Web as XML is becoming the standard format for data representation and exchange [2]. For example, Niagara [128] and Xyleme [178] are two systems designed for querying XML data on the Web. Thirdly, historical XML queries are dynamic, which is the focus of our MONETA framework. That is, new queries will emerge, old queries may disappear, and the same query may be issued at different frequency over time as shown in Table 1.1. Here frequency refers to the number of times a query is issued during a time period.

Different from author-centric data, the visitor-centric data records interactions between users and Web content. As a result, the evolution of visitor-centric data reflects the changes in Web users' point of views/interests about the corresponding content. For instance, consider a repository containing a set of XML queries posed on a Website over time. We may observe that some queries are issued more frequently over time while others are not. Such
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

change patterns may be attributed to the changes of users' interests. Moreover, such change patterns of users' interests can be used in many applications such as dynamic-conscious caching, hot topic monitoring, and efficient XML indexing. In this chapter, we propose to extract the change patterns of historical XML queries and illustrate the usefulness of such knowledge. Specifically, we focus on XML queries that exhibit a particular type of evolution pattern called conserved query path. We illustrate the usefulness of these paths in the context of XML query caching [205].

The rest of this chapter is organized as follows. We first motivate the research for mining evolution of historical XML queries by reviewing the existing XML query mining approaches and highlighting their limitations. Section 4.2 presents the overview of mining conserved XML query paths and building dynamic-conscious cache strategy using these conserved query paths. Then, in Section 4.3, we present the approach of mining conserved XML query paths in detail. Section 4.4 explains how to build the dynamic-conscious caching strategy using conserved XML query paths. Performances of both the mining algorithm and the cache strategies are studied using synthetic datasets in Section 4.5. Lastly, Section 4.6 concludes this chapter.

4.1 Motivation

Due to the increasing demand of retrieving information from various XML data sources, the efficiency of XML query evaluation has become one of the primary concerns in the XML research community. One of the optimization techniques to improve the query efficiency that has attracted the community is the cache technique. Following the idea of semantic cache [58], which utilizes the historical queries and their results to answer subsequent queries by reasoning their containment relations, different cache strategies have been proposed for XML query evaluation [35, 36, 121, 183, 184, 185]. For instance, suppose at time $t_1$, the semantic cache contains a set of views $V = \{V_1, V_2, \cdots, V_n\}$ and the corresponding queries are $Q = \{Q_1, Q_2, \cdots, Q_n\}$. When a new query $Q_{n+1}$ comes, it inspects each view $V_i$ in $V$ and determines whether it is possible to answer $Q_{n+1}$ from $V_i$. View $V_i$ answers query $Q_{n+1}$
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

if there exists another query $C$ which, when executed on the result of $Q_i$, gives the result of $Q_{n+1}$. It is denoted by $CoQ_i = Q_{n+1}$, where $C$ is called the composing query (CQ). When a view answers the new query, we have a hit, otherwise we have a miss. When we have a hit, query $Q_{n+1}$ is answered by executing query $C$, which is simpler than $Q_{n+1}$, on the result of $Q_i$, which is much smaller than the original XML data source.

One of the major issues in the above problem is to determine which parts of the query results should be cached as the cache size of the system is often limited. Traditional caching strategies consider the contents in a cache as a priority queue. One of the well-established replacement strategies is called Least Recently Used (LRU) [35], which evicts the least recently used objects from the cache when it is full. In the context of XML queries, different cache strategies have been proposed. One of the approaches that utilize the historical XML queries is to cache the results of frequent XML query patterns [183, 184]. XML queries can be represented as rooted unordered trees called query pattern trees (QPTs). The frequent XML query patterns refer to rooted trees that are induced subtree of at least $\text{minsup}$ number of XML queries in the query collection, where $\text{minsup}$ is a user-defined minimum support. For example, Figure 4.1 shows a set of query pattern tree (QPT) representations for four XML queries. If the user-defined $\text{minsup}$ is two, then Figure 4.1(e) is a frequent query pattern since it is a subtree of $QPT_1$ and $QPT_4$. The intuition is that XML query patterns that are frequently issued in the history are expected to be frequently issued in the future as well. Based on this assumption, existing studies focused on the issue of mining frequent XML query patterns from XML query collections.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

As a result, the frequent query pattern mining problem is similar to the frequent subtree mining problem [184]. However, there are several differences between these two problems. For instance, in XML queries, there are special characters such as the wildcard "*" and relative path "/", which make the subtree matching process more expensive [184]. Moreover, existing XML query pattern mining approaches only focused on extracting frequent rooted subtree as each query should start from the root of the XML document schema. In the literature, three different algorithms (XQPMiner [184], FastXMiner [183], 2PXMiner [185]) have been proposed to address this issue.

We observed that existing query pattern-based caching strategies are solely based on statistics obtained by treating historical XML queries as snapshot data [183, 184, 185]. That is, the frequent XML query patterns are based on only the number of occurrences of the query subtrees in the history. Every occurrence of a query subtree contributes equally to the caching strategy regardless of when the query was issued. The occurrence of an XML query issued long time ago is considered as important for caching as the occurrence of an XML query issued recently. However, this may not always be true in real life applications based on the following observations.

- **Dynamic Nature of XML Data:** XML data is often dynamic in many real life applications. New data may be inserted. Outdated data may be deleted, while other data may be updated. As a result, the frequent query subtrees may not remain frequent for the updated XML. For example, a query $Q_1$ is a frequent query in time $t_1$ because it can return many interesting results. However, at time $t_2$ when the data source is updated by deleting most of the results for $Q_1$, then $Q_1$ may not remain frequent as users may not be interested in the remaining results. Moreover, some of the previously frequent query subtrees may not be able to return any results from the updated XML data due to the structural changes to the XML data.

- **Dynamic Nature of XML Queries:** Due to changes to the XML data source, together with changes in users’ interests and their familiarities with the schema of the
XML repository, the workload of XML queries may change over time. Let us reconsider the two queries $QPT_2$ and $QPT_4$ in Figure 4.1. When a new book is released, most people use $QPT_2$ to find the book from the data source as they only know the title of the book. However, after more people read this book, details of the books such as the title of certain sections become available. Also, the price of the book may be different as there may be brand new books and second-hand books. As a result, more and more people use $QPT_4$ to find the book. However, it is possible that $QPT_2$ has been issued so many times in the history that it is still a frequent query subtree, while $QPT_4$ is quite new and is not a frequent query subtree. However, intuitively, more queries similar to $QPT_4$ are expected to be issued in the near future compared to queries similar to $QPT_2$.

From the above observations, it is evident that the existing frequent XML query pattern-based cache strategy may not perform well in a dynamic environment. This is because timestamps of historical XML queries are ignored while such information plays an important role in building an efficient cache strategy. That is, the frequent query patterns cannot accurately monitor the evolution of users' interests due to the dynamic nature of XML source data and XML queries. Reconsidering the two query pattern trees $QPT_2$ and $QPT_4$, suppose both of them have the same number of occurrences from January to December in 2005. However, $QPT_2$ was issued more frequently in January than in other months, while $QPT_4$ was issued more frequently in December than in other months. Assuming other factors, such as query evaluation time and size of the query results, are the same, existing frequent query pattern based cache strategies will assign the same priority to both queries.

However, such cache strategies may not be efficient as the XML query workload is dynamic and may change from time to time where the temporal dimension is ignored. The temporal information associated with the XML queries can be used to build a more efficient cache strategy. For instance, based on the above frequency change patterns for $QPT_2$ and $QPT_4$, one may argue that $QPT_4$ is becoming more popular and should be assigned with
a higher priority than $QPT_2$, while others may argue that the occurrence pattern of the queries may repeat in every year. Thus, the cache strategy should assign a higher priority to $QPT_2$ than $QPT_4$ in the coming month. Both arguments show that a more efficient caching strategy can be built by incorporating the dynamic nature of the XML queries.

How to incorporate the temporal information for efficient caching strategy is a challenging problem for the following reasons: (1) it is difficult to design an efficient algorithm to model the evolution behaviors of historical queries by taking into account both the structural information and temporal information embedded in the query history; (2) given the list of XML queries with various types of evolution patterns, it is difficult to propose an efficient cache strategy that fully utilizes their structural, evolutionary, space requirement, and evaluation cost properties. Moreover, to the best of our knowledge, this is the first approach to model the evolutionary change behavior of tree structured data and apply it for XML query evaluation caching.

4.2 An Overview of Mining Historical XML Queries

As discussed in Chapter 1, there are four types of novel knowledge that can be discovered under the MONETA framework. In this chapter, we are interested in the issue of discovering substructures with specific evolution patterns from historical XML queries. Such substructures can be used to overcome the above-mentioned limitations of existing query pattern-based cache strategies by constructing more efficient dynamic-conscious cache strategies.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

More specifically, we focus on discovering conserved XML query paths. Hereafter XML query paths refer to the rooted query paths (RQPs) in the tree representation of XML queries. For example, /book/section/figure/title is an XML query path of the XML query shown in Figure 4.1(c). Here conserved XML query paths refer to XML query paths that never change or do not change significantly most of the time (if not always) in terms of their support values (the number of occurrences of the query path in the historical query collection during a specific time period). For instance, Figure 4.2 shows the support values for five rooted query paths over eight months. It can be observed that XML query paths RQP_1, RQP_3, and RQP_5 are conserved XML query paths as their support values are relatively stable over time.

Then, based on the conserved XML query paths, we propose a novel and more efficient cache strategies called dynamic-conscious cache strategies. The intuition behind this is as follows. As conserved XML query paths are stable in terms of their support values in the history, it is possible for support values of these query paths to remain relatively stable in the future. As a result, we can have a more accurate prediction on the occurrence of such XML query paths in the future, which is a key factor that affects the efficiency of query pattern-based cache strategies.

With the above objectives in mind, in this chapter, we focus on two major problems: mining conserved XML query paths and building a dynamic-conscious cache strategy.

4.2.1 Mining Conserved XML Query Paths

Given an XML data repository, there will be a collection of XML queries that were issued by different users over a period of time. For each occurrence of a query, there is a timestamp that records the time when the query was issued. The objective of mining conserved XML query paths from historical XML queries is to explore the temporal information associated with the queries along with the number of occurrences of the query paths. Basically, the approach of mining conserved XML query paths consists of three major components, measurement, representation, and mining.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Firstly, for the measurement component, we propose a set of dynamic metrics to measure how the support values of XML queries change over time. Hereafter, in the rest of this chapter, the term changes to XML queries refers to changes to the support values of XML queries. Generally, the changes to XML queries can be monitored using micro-pattern and macro-pattern. To extract such change pattern, the basic operation is to compare the support values of queries over different time period. As different XML data repositories have different volumes of query traffics and the time resolutions for the historical queries can be up to millisecond, we propose to segment the collection of queries into query groups based on the user-defined calendar pattern. For instance, we can segment the query collection on a daily or weekly basis such that each query group contains the queries that are issued during the same day or week. Then, the micro-pattern based dynamic metrics are proposed based on the comparison of the consecutive query groups, while the macro-pattern based dynamic metrics are proposed based on the changes among the entire sequence of query groups. Note that the calendar pattern may be domain dependent and users can specify any calendar pattern with respect to the requirements of the corresponding application.

Next, for the representation component, we propose to model each XML query as a rooted unordered tree, which is called the query pattern tree [184]. As a result, the collection of XML queries can be represented as a collection of query pattern trees with timestamps. Examples of query pattern trees (QPTs) are presented in Figure 4.1. Then, the collection of QPTs is segmented into a sequence of query groups using the user-defined calendar pattern. Figure 4.3 shows the process of segmenting the QPTs on a daily basis. To represent
the query groups, we propose to merge the corresponding query pattern trees into a tree structure called **Query Pattern Group Tree (QPG-Tree)**. The QPG-Tree records the occurrences of each query path within the time window. Further, to represent the whole query collection, the sequence of QPG-Trees is merged into an **Historical Query Pattern Group Tree** (called HQPG-Tree) as shown in Figure 4.3. In the HQPG-Tree representation, the historical occurrences of each query and their evolution patterns can be modeled as either a vector or a set of metrics.

After that, based on the dynamic metrics, we present the formal definition of conserved query paths. Intuitively, conserved query paths are defined as query paths that never change or do not change significantly in terms of their support values during a time period such as $RQP_1$, $RQP_2$, and $RQP_3$ in Figure 4.2. Moreover, we define frequent conserved query paths (FCQPs) and infrequent query paths (ICQPs) based on the average support values in all query pattern groups of conserved query paths. For instance, suppose the thresholds for frequent and infrequent query paths are 0.5 and 0.2, respectively, then $RQP_1$ is an infrequent conserved query path and $RPQ_5$ is a frequent conserved query path.

Lastly, for the mining component, based on the HQPG-Tree representation, we present two algorithms for efficiently extracting FCQPs and ICQPs. Note that the mining results of our algorithms are different from the frequent query pattern mining results in the following two ways. Firstly, conserved query paths are defined based on not only the number of occurrences but also the temporal information of these occurrences, while the temporal information is ignored in the definition of frequent query patterns. Secondly, conserved query paths are linear structures, while frequent query patterns are tree structures. That is, conserved query paths are structurally in a finer granularity than frequent query patterns. Both the frequent query patterns and the conserved query paths have their own advantages and limitations. A smaller granularity is more flexible and can handle a larger number of similar queries but need more caching space than a larger granularity. By limiting the caching space, as we shall see in Section 4.5.2, the conserved query paths-based cache strategies perform better than frequent query pattern-based cache strategies.
4.2.2 Building A Dynamic-conscious Cache Strategy

Typically, a cache system utilizes a replacement manager to decide what to be retained in the cache and what to be discarded when the cache is full [35, 36, 198]. In existing frequent query pattern-based cache systems, the frequent query pattern results are cached based on the query execution costs, sizes of the results, and the occurrences of the queries.

Based on the set of conserved XML query paths, a dynamic-conscious cache strategy can be built by integrating both the temporal information and support values of historical queries. Specifically, we define a rank function to evaluate the priority of the query paths and results to be cached. The rank is calculated based on the historical change pattern, support value, cost of evaluating the query path, and size of the query result. Intuitively, query results of moderate size that are expensive to obtain and expected to be issued frequently are expected to have high ranks and shall have higher priorities to be cached. The general cache strategy is as follows. When the cache is not full, frequent conserved query paths and results with higher ranks are cached. At the same time, in the dynamic environment, the ranks of different conserved query paths may change over time. The ranks of some of the query paths in the cache may increase or decrease over time. For these query paths whose ranks are smaller than the minimum rank in the previous cache, the corresponding results are evicted and other frequent conserved query paths with higher ranks are cached.

Compared with existing cache strategies, the proposed dynamic-conscious cache strategies are different in the following aspects. Firstly, the historical change patterns extracted from temporal information of queries are integrated to build a more robust rank function for the corresponding query result. Secondly, the cache granularity is different. That is, in existing XML query cache strategies, queries or query pattern trees and the corresponding results are considered as the basic elements for caching. However, in our approach, query paths and corresponding results, which have a finer granularity, are considered as the basic elements for caching. This enable us to monitor the users' interests in a more effective way as the query patterns are dynamic, which is represented as the conserved query paths and
cannot be included in the frequent query patterns. In a sense, the conserved query paths can provide more up to date caching strategy than the frequent query patterns in the dynamic environment. As we shall see in Section 4.5.2, the proposed dynamic-conscious cache strategies outperform an existing state-of-the-art frequent query pattern-based cache strategy and traditional LRU-based cache strategy as well.

4.3 Mining Conserved XML Query Paths

In this section, we present the approach of mining conserved XML query paths. The input is a collection of historical XML queries. The objective of this approach is to extract the set of conserved query paths that are frequent or infrequent over time. Basically, this approach consists of three major components: representation, measurement, and mining. Now we elaborate on them in turn.

4.3.1 Representation

In most existing XML query pattern mining approaches, XML queries are modeled as unordered trees called Query Pattern Trees (QPTs) [183, 184, 185]. We adopt the QPT representation method in this chapter. Formally, a query pattern tree is defined as follows.

**Definition 4.9 Query Pattern Tree (QPT):** A query pattern tree is a rooted unordered tree, \( QPT = (V, E) \), where \( V \) is the vertex set and \( E \) is the edge set. The root of the QPT is denoted by root \((QPT)\). For each edge \( e \in E \), \( e = (v_i, v_j) \), where node \( v_i \) is the parent of node \( v_j \). Each vertex \( v \) has a label, denoted as \( v.label \). A \( v.label \) can be any tag name of the elements and attributes, wildcard “*” (denoting any label) and relative path “//” (denoting zero or more labels).

A QPT is a tree structure that represents the hierarchy structure of the predicates, result elements, and attributes in the XML query. For example, Figure 4.1(a) shows the QPT for a query that retrieves the author, price, publisher of books where book/title has value “XML”. Based on the definition of QPT, in existing approaches the rooted subtree of QPT is defined
to capture the common subtrees in a collection of XML queries \[184, 185\]. However, in this chapter, we are interested in rooted query paths, which can provide a finer granularity for caching than rooted subtrees. Rooted query paths are special cases of rooted subtrees that have only one leaf node. Formally, a rooted query path (RQP) is defined as follows.

**Definition 4.10 Rooted Query Path (RQP):** A rooted query path (RQP), \( RQP_1 \), is a sequence of vertexes \( (v_i, v_{i+1}, v_{i+2}, \ldots, v_j) \). Given a QPT, \( QPT_1 \), \( RQP_1 \) is a rooted query path of \( QPT_1 \), denoted as \( RQP_1 \subseteq QPT_1 \), if and only if \( v_i = root(QPT_1) \) and \( \forall i \leq k \leq (j-1), e = (v_k, v_{k+1}) \) exists in \( QPT_1 \).

Reconsider the QPT example in Figure 4.1(a). It consists of several RQPs such as //book, //book/title, //book/author, //book/price, and //book/publisher. In existing approaches, QPTs are modelled as transactions in a transactional database, where the timestamps associated with QPTs are ignored. Here we propose to record the timestamp associated with each QPT. That is, each QPT is represented as a pair \( (QPT_i, t_i) \), where \( t_i \) is the timestamp when \( QPT_i \) was issued. As a result, the collection of queries can be represented as a sequence \( \langle (QPT_1, t_1), (QPT_2, t_2), \ldots, (QPT_n, t_n) \rangle \), where \( t_1 \leq t_2 \leq \ldots \leq t_n \). Furthermore, this sequence is compressed into a smaller sequence of query pattern groups based on their timestamps and a user-defined calendar pattern. Before we define query pattern groups, we formally define the notion of calendar schema, calendar pattern, and temporal containment.

**Definition 4.11 Calendar Schema:** A calendar schema is a relational schema \( R \) with a constraint \( C \), where \( R = (f_n : D_n, f_{n-1} : D_{n-1}, \ldots, f_1 : D_1) \), \( f_i \) is the name for a calendar unit such as year, month, and day, \( D_i \) is a finite subset of positive integers for \( f_i \), \( C \) is a Boolean valid constraint on \( D_n \times D_{n-1} \times \cdots \times D_1 \) that specifies which combinations of the values in \( D_n \times D_{n-1} \times \cdots \times D_1 \) are valid.

For example, suppose we have a calendar schema \( \langle \text{year: \{2000, 2001, 2002\}, month: \{1, 2, 3, \ldots, 12\}, day: \{1, 2, 3, \ldots, 31\} \rangle \) with the constraint that evaluate \( \langle y, m, d \rangle \) to be "true" only if the combination gives a valid date. Then, it is evident that \( \langle 2000, 2, 15 \rangle \) is valid.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

while \(\{2000, 2, 30\}\) is invalid. The reason to use the calendar schema is to exclude invalid time intervals that are generated due to the combinations of calendar units. Specifically, in this chapter, by modifying the constraint, users can further narrow down valid time intervals according to application-based requirements. For instance, if we want to monitor the change patterns of XML queries on a daily basis, then a daily based calendar schema can be defined and used. Hereafter, we use \(\ast\) to represent any integer value that is valid based on the constraint. For instance, if we use \(\ast\) to represent months in the above mentioned calendar schema, then \(\ast\) refers to any integer value from 1 to 12.

**Definition 4.12 Calendar Pattern:** Given a calendar schema \(R\) with the constraints \(C\), a calendar pattern, denoted as \(P\), is a valid tuple on \(R\) of the form \(\langle d_n, d_{n-1}, \ldots, d_1 \rangle\) where \(d_i \in D_i \cup \{\ast\}\).

For example, given a calendar schema \(\langle\text{year}, \text{month}, \text{day}\rangle\), the calendar pattern \(\langle\ast, 1, 1\rangle\) refers to the time intervals “the first day of the first month of every year”. Similarly, the calendar pattern \(\langle2002, \ast, 1\rangle\) represents the time intervals “the first day of every month in year 2002.”

Given a timestamp, the relation between the timestamp and the calendar pattern is defined as follow.

**Definition 4.13 Temporal Containment:** Given a calendar pattern \(\langle d_n, d_{n-1}, \ldots, d_1 \rangle\) denoted as \(P_i\) with the corresponding calendar schema \(R\), a timestamp \(t_j\) is represented as \(\langle d'_n, d'_{n-1}, \ldots, d'_1 \rangle\) according to \(R\). The timestamp \(t_j\) is contained in \(P_i\), denoted as \(t_j < P_i\), if and only if \(\forall 1 \leq l \leq n, d'_l \in d_l\).

For example, given a calendar pattern \(\langle\ast, 2, 12\rangle\) with the calendar schema \(\langle\text{week}, \text{day of the week}, \text{hour}\rangle\), then the timestamp \(2005-09-30\ 12:28\) is not contained in this calendar pattern as it is not the second day of the week, while the timestamp \(2005-09-26\ 12:08\) is.

Based on the above definitions of calendar schema, calendar pattern, and temporal containment, we are now ready to formally define a query pattern group (QPG).
Definition 4.14 Query Pattern Group (QPG): Let \((QPT_1, t_1), (QPT_2, t_2), \ldots, (QPT_n, t_n)\) be a sequence of QPTs. Let \(P_x\) be the user-defined calendar pattern. A query pattern group QPG is a bag of QPTs \([QPT_i, t_i), (QPT_{i+1}, t_{i+1}), \ldots, (QPT_j, t_j)\] such that \(\forall m \ (i \leq m \leq j), t_m \prec P_x, \) where \(0 \leq k \leq n\).

According to the above definition, all the QPTs within the same query pattern group are issued within the same calendar pattern according to the calendar schema. For example, queries that were issued in the same day, week, month, can be partitioned into the same query pattern group. Using this definition, users can define their own time granularity based on their applications and the query traffic of their XML repositories. As a result, the sequence of QPTs is partitioned into a sequence of query pattern groups represented as \(<QPG_1, QPG_2, \ldots, QPG_k>\).

After the QPTs are partitioned into groups, the occurrences of the same QPT within a group are considered as equally important. As a result, for each group of QPTs, we can build a more compact representation, called a query pattern group tree (QPG-tree), to store the aggregated occurrence information. Formally, the query pattern group tree is defined as follows.

Definition 4.15 Query Pattern Group Tree (QPG-Tree): Let \(QPG\) be a query pattern group that consists of a bag of query pattern trees denoted as \(\{QPT_i, QPT_{i+1}, \ldots, QPT_j\}\). A query pattern group tree is a 3-tuple tree, denoted as \(QPG\)-tree = \(<V, E, \mathcal{R}>\), where \(V\) is the vertex set, \(E\) is the edge set, and \(\mathcal{R}\) is a function that maps each vertex to the support value of the corresponding rooted query path (RQP), such that \(\forall RQP \subseteq QPT_k, i \leq k \leq j, \) there exists a rooted query path, \(RQP' \subseteq QPG\)-tree, that is identical to \(RQP\).

Consider the three QPTs in Figures 4.1(a), (b), and (c). The corresponding query pattern group tree is shown in Figure 4.4(a). The query pattern group tree includes all the query patterns and records the support values, which are shown as the values within the nodes of the rooted query paths. Note that for special cases such as "/" and "*", not
only the exact/identical path will be inserted into the QPG-tree, but also the attributes of other paths that are supported by such paths will be updated as well. Moreover, the attributes of paths include "//" and "*" will be updated when other rooted paths support them are encountered as well. Here, we review the definitions of support and extended RQP inclusion [183, 184, 185], which are used in the above definition.

**Definition 4.16 Extended RQP Inclusion:** Given two rooted query paths $RQP_1$ and $RQP_2$. $RQP_1$ is said to be extended included in $RQP_2$, denoted as $RQP_1 < RQP_2$, if and only if: $\forall v \in RQP_1$, there is a match node $v' \in RQP_2$ such that $v$.label $\in \{v'.label, *, //\}$ and any RQP rooted at node $v$ in $RQP_1$ is extended included in the RQP rooted at node $v'$ in $RQP_2$.

**Definition 4.17 Support:** Given a query pattern group $QPG_i$, the support of a RQP is defined as $\Phi_i(RQP) = K / L$, where $K$ denotes the number of times the RQP is extended included in the QPTs in $QPG_i$ and $L$ denotes the number of QPTs in the query pattern group $QPG_i$.

The historical query patterns can now be represented as a sequence of QPG-trees. To record the historical change patterns of the supports to the rooted query paths in the QPG-trees, we propose to merge the sequence of QPG-trees into a single tree called historical QPG-tree (HQPG-tree).
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Definition 4.18 HQPG-tree: Let $\mathcal{G}$ be a sequence of query pattern group trees (QPG-trees), $\mathcal{G} = \{\text{QPG-tree}_1, \text{QPG-tree}_2, \ldots, \text{QPG-tree}_i\}$. An HQPG-tree is a 3-tuple tree, denoted as $\text{HQPG-tree} = (\mathcal{V}, \mathcal{E}, \Psi)$, where $\mathcal{V}$ is the vertex set, $\mathcal{E}$ is the edge set, and $\Psi$ is a function that maps each vertex to a sequence of support values, such that $\forall \text{RQP} \subseteq \text{QPG-tree}_k, 1 \leq k \leq i, \exists \text{RQP'} \subseteq \text{HQPG-tree} \text{ that is identical to } \text{RQP}$. □

The idea of HQPG-tree is similar to the idea of QPG-tree except for the function $\Psi$. In the definition of QPG-tree, the $\mathcal{N}$ function is used to map each vertex to a single real value, which is the support value of the rooted path at that vertex; in the definition of HQPG-tree, the $\Psi$ function is used to map each vertex to a sequence of support values. For example, Figure 4.4(b) shows an example HQPG-tree by partitioning the QPTs in Figures 4.1(a), (b), (c), and (d) into two QPGs. The first three QPTs are in one group, while the last is in another group. The sequences of values associated with each vertex in Figure 4.4 correspond to the support values. It is obvious that the size of the HQPG-tree can be huge when the number of QPGs is large. To compress the representation of sequence of support values, different metrics will be defined in the next section.

4.3.2 Measurement

In order to extract the conserved query paths, it is important to define metric(s) that can quantify the evolutionary characteristics of a specific RQP in history. Intuitively, the lower the degree of evolution of a RQP with respect to its support values is, the more conserved it is in the history.

Recall that to measure the dynamic property of XML queries, two types of dynamic metrics are proposed: macro-pattern metrics and micro-pattern metrics. Macro-pattern metrics are designed to evaluate the overall change trend for the support values of XML queries over time, while micro-pattern metrics are designed to monitor changes for the support values of query paths between consecutive query pattern groups.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Regression-based Metric

Intuitively, the macro-pattern can be modelled using regression models [171]. Hence, we propose a metric called support conservation rate to monitor the changes to supports of query paths in the history using the linear regression model.

\[ \Phi_t(RQP) = \Phi_0(RQP) + \lambda t, \quad \text{where } 1 \leq t \leq n \]

Here the idea is to find a "best-fit" straight line through the data points \{(\Phi_1(RQP), 1), (\Phi_2(RQP), 2), \ldots, (\Phi_n(RQP), n)\}, where \(\Phi_0(RQP)\) and \(\lambda\) are constants called support intercept and support slope respectively. The most common method for fitting a regression line is the method of least-squares [171]. By applying the statistical treatment known as linear regression to the data point

\[ \Phi_0(RQP) = \frac{n \sum_{i=1}^{n} (i \cdot \Phi_i(RQP)) - (\sum_{i=1}^{n} \Phi_i(RQP))(\sum_{i=1}^{n} i)}{n \sum_{i=1}^{n} (\Phi_i(RQP))^2 - (\sum_{i=1}^{n} \Phi_i(RQP))^2} \]

\[ \lambda = \frac{\sum_{i=1}^{n} i - (\Phi_0(RQP) \cdot \sum_{i=1}^{n} \Phi_i(RQP))}{n} \]

Besides the two constants, there is another measure to evaluate how the regression fits the data points actually. It is the correlation coefficient, denoted as \(r\).

\[ r = \frac{n \sum_{i=1}^{n} (\Phi_i(RQP) \cdot i) - (\sum_{i=1}^{n} \Phi_i(RQP))(\sum_{i=1}^{n} i)}{\sqrt{n \sum_{i=1}^{n} (\Phi_i(RQP))^2 - (\sum_{i=1}^{n} \Phi_i(RQP))^2} \sqrt{n \sum_{i=1}^{n} i^2 - (\sum_{i=1}^{n} i)^2}} \]

The correlation coefficient, \(r\), always takes a value between -1 and 1, with 1 or -1 indicating perfect correlation. The square of the correlation coefficient, \(r^2\), represents the fraction of the variation in \(\Phi_t(RQP)\) that may be explained by \(t\). Thus, if a correlation of, say 0.8, is observed between them, then a linear regression model attempting to explain the changes to \(\Phi_t(RQP)\) in terms of \(t\) will account for 64% of the variability in the data [171].

Based on the above linear regression-based model for support history we now propose the metric support conservation rate.
Definition 4.19 Support Conservation Rate: Let \((\Phi_1(RQP), \Phi_2(RQP), \ldots, \Phi_n(RQP))\) be the sequence of historical support values of the rooted query path \(RQP\). The support conservation rate of \(RQP\) is defined as 
\[
R(RQP) = r^2 - |\lambda|.
\]

As we know that in the regression model, the larger the absolute value of the slope, the more significantly the support changes over time. At the same time, the larger the value of \(r^2\), the more accurate is the regression model. Hence, the larger the support conservation rate \(R(RQP)\), the support values of the \(RQP\) change less significantly. In other words, the support values of a \(RQP\) are more conserved when the value of support conservation rate increases.

Also from the regression model, it can be inferred that \(|\lambda| < \frac{1}{n}\) as \(0 \leq \Phi_1(RQP) \leq 1\). In real life the value of \(n\) can be huge, thus \(|\lambda| \ll r \leq 1\). Consequently, we can guarantee that \(0 \leq R(RQP) \leq 1\). When \(R(RQP) = 1\), the support of this \(RQP\) is a constant where \(r^2 = 1\) and \(\lambda = 0\).

Group-wise Metrics

The regression-based metric can efficiently monitor the macro-pattern. In this section, we propose another set of metrics that monitor both the micro-pattern and the macro-pattern. This set of metrics is called group-wise metrics because they are defined based on the changes to the consecutive query pattern group pairs.

Definition 4.20 Group Dynamic: Let \(QPG_i\) and \(QPG_{i+1}\) be any two consecutive XML query pattern groups. For any rooted query path, \(RQP\), the group dynamic from group \(i\) to group \(i+1\), denoted as \(G_i(RQP)\), is defined as 
\[
G_i(RQP) = |\Phi_{i+1}(RQP) - \Phi_i(RQP)|.
\]

The group dynamic measures the changes of support for a rooted query path, \(RQP\), between any two consecutive query pattern groups. As a result, for a sequence of \(n\) query pattern groups, there will be a sequence of \(n-1\) group dynamic values. As group dynamic only measures changes between query pattern groups, we define sequence dynamic to measure the overall change patterns in the entire query pattern group sequence.
Definition 4.21 Sequence Dynamic: Let \((QPG_1, QPG_2, \ldots, QPG_n)\) be a sequence of query pattern groups in the history. For any rooted query path, \(RQP\), the sequence dynamic in this sequence, denoted as \(S(\alpha, RQP)\), where \(\alpha\) is the pre-defined threshold for group dynamic, is defined as

\[
S(\alpha, RQP) = \frac{\sum_{i=1}^{n-1} d_i}{n - 1}
\]

where \(d_i = \begin{cases} 
1, & \text{if } G(RQP) \geq \alpha \\
0, & \text{if } G(RQP) < \alpha 
\end{cases}
\)

The sequence dynamic measures the percentage of query pattern groups where the support of a rooted query path changed significantly from the previous query pattern group. However, small changes in the history can also cause significant changes over time. For instance, if the support value of a query path keeps increasing/decreasing constantly with very small group dynamic values such as 0.005. At the end, we may find out that the support values in the first few query pattern groups are substantially different from the support values in the last few query pattern groups when the number of query pattern groups is very large. To measure the aggregated effect of individual changes, we propose a metric named aggregated dynamic.

Definition 4.22 Aggregated Dynamic: Let \((QPG_1, QPG_2, \ldots, QPG_n)\) be a sequence of query pattern groups in the history. Let \(RQP\) and \(i\) be a rooted query path and its support value in \(QPG_i\). The aggregated dynamic for \(QPT\), denoted as \(A(QPT)\), is defined as:

\[
A(QPT) = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n-1} (\Phi_i - \Phi_{i+1})^2}
\]

Different from standard deviation, the definition for aggregated dynamic incorporates the temporal relationships in the historical sequence. That is, for sequences of support values that have the same average support and support deviation values, the values of aggregated dynamic can be substantially different. For example, suppose we have two sequences of
support values \( (0.1, 0.2, 0.3, 0.4, 0.5, 0.6) \) and \( (0.1, 0.6, 0.2, 0.5, 0.3, 0.4) \). The average support and standard deviation values for them are the same while the A values are different \((0.1 \text{ and } 0.33 \text{ for the first and second sequences respectively})\). Similar to the definition of standard deviation, here we use the aggregated differences to show the significance of changes to the sequence of support values. It can be observed that the value of A is within the range of \([0, 1]\). A large value of A indicates more frequent and significant changes of the support values in the history.

### 4.3.3 Problem Statement

Based on the dynamic metrics proposed in the previous section, we formally define the problem of conserved query paths mining in this section. There are two groups of rooted query paths, frequently conserved query paths (FCQPs) and infrequently conserved query paths (ICQPs), which are important for efficient query caching. We can summarize their characteristics as follows: (1) The support values of the rooted query paths are either large enough or small enough; and (2) Their support values do not change significantly in the history. We now formally define conserved query paths. Note that we present two definitions for conserved query paths by using the regression-based metric and group-wise metrics, respectively.

**Definition 4.23 Conserved Query Path:** Let \( Q \) be a collection of historical XML queries, which is represented as a sequence of query pattern groups \((QPG_1, QPG_2, \ldots, QPG_n)\). Let \( \alpha, \beta, \gamma \) and \( \zeta \) be the thresholds for group dynamic, sequence dynamic, aggregated dynamic, and support conservation rate, respectively. Then,

- a rooted query path, \( RQP \), is a conserved query path according to the regression-based metric if \( R(RQP) \leq \zeta \)

- a rooted query path, \( RQP \), is a conserved query path according to the according to the group-wise based metrics if \( S(\alpha, RQP) \leq \beta \) and \( A(RQP) \leq \gamma \).
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Note that the regression-based definition of conserved query paths focuses on macro-pattern, while the group-wise based definition of conserved query paths focuses on both macro-pattern and micro-pattern. The effectiveness of the two definitions will be compared in terms of the mining results in Section 4.5. Next, we define the group average support metric, based on which we will define frequent conserved query paths and infrequent conserved query paths.

**Definition 4.24 Group Average Support:** Let \( \langle QPG_1, QPG_2, \ldots, QPG_n \rangle \) be a sequence of query pattern groups in the history. The group average support of a rooted query path, \( RQP \subseteq QPG_i \) \((0 \leq i \leq n)\), denoted as \( \Phi(RQP) \), is defined as \( \Phi(RQP) = \frac{1}{n} \sum_{i=1}^{n} \Phi_i \).

The group average support value is defined to measure the overall support for the corresponding query path. Based on the average support value, we are interested in the following two groups of conserved query paths.

**Definition 4.25 FCQP and ICQP:** Let \( RQP \) be a conserved query path. Let \( \xi \) and \( \xi' \) be the minimum group average support and maximum group average support thresholds respectively. Then,

- \( RQP \) is a Frequent Conserved Query Path (FCQP) if and only if \( \Phi(RQP) \geq \xi \);
- \( RQP \) is an Infrequent Conserved Query Path (ICQP) if and only if \( \Phi(RQP) \leq \xi' \).

Note that there are two definitions for conserved query path, frequent conserved query path, and infrequent conserved query pattern. In the subsequent sections, we shall design two algorithms for each of the definitions. Moreover, the mining results of the two algorithms will be evaluated by comparing the efficiencies of the corresponding dynamic-conscious cache strategies.
Algorithm 6 QPG-tree Construction

Input:
A bag of QPTs: \([QPT_1, QPT_2, \ldots, QPT_n]\)

Output:
The query pattern group tree: QPG-tree

Description:
1: Initialize QPG-tree as the first QPT \(QPT_1\)
2: for all \(2 \leq i \leq n\) do
3: for all \(RQP \subseteq QPT_i\) do
4: for all \(RQP' \subseteq QPG-\text{Tree}\) do
5: if \(RQP' \prec RQP\) then
6: update the support of \(RQP'\)
7: end if
8: if \(RQP \not\subseteq QPG-\text{Tree}\) then
9: Insert \(RQP\) to \(QPG-\text{Tree}\)
10: end if
11: end for
12: end for
13: end for
14: Return(QPG-tree)

4.3.4 Mining FCQPs and ICQPs

Given the collection of historical XML queries, in this section, we present two algorithms to extract the sets of FCQPs and ICQPs with respect to the user-defined parameters. The mining algorithms consist of two major phases: the construction of HQPG-Tree phase and the query paths extraction phase. We elaborate on them in turn.

Construction of HQPG-Tree

Given a collection of XML queries, an HQPG-tree is constructed in the following way. Firstly, the queries are transformed into QPTs. Then, the QPTs are partitioned into groups based on the timestamps and user-defined calendar pattern, where each QPG is represented as a QPG-tree. After that, there is a sequence of QPG-trees, which are then merged together into an HQPG-tree. As the process of transforming queries into QPTs and partitioning QPTs into groups are straightforward, we present the details of constructing the QPG-tree and merging QPG-trees in Algorithm 6 and Algorithm 7, respectively.

The algorithm of constructing the QPG-tree is shown in Algorithm 6. Firstly, the QPG-tree is initialized as the first QPT in that group. Next, other QPTs in the group are compared with the QPG-tree to construct the structure of the QPG-tree as shown in Lines
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

2-13. Each RQP in the corresponding QPT is compared with the existing QPG-tree, the support values for these RQP's in the QPG-trees that are extended included by this RQP are updated. Moreover, if this RQP is not included in the existing QPG-tree, then the QPG-tree is updated by inserting this RQP into the tree. This process iterates for all the RQPs in all the QPTs in the corresponding query pattern group. An example QPG-tree is shown in Figure 4.4(a) for the set of QPTs in Figures 4.1(a), (b), and (c).

The algorithm of merging the sequence of QPG-trees into the HQPG-tree is shown in Algorithm 7. The basic idea of the algorithm is similar to the algorithm for construction of the QPG-tree. Firstly, the HQPG-tree is initialized with the first QPG-tree in the sequence. Then, for each RQP in the rest of the QPG-tree sequence, the corresponding RQP (if exists) and \( RQP' \) that are extended included by this RQP in the current HQPG-tree is updated. If the RQP does not exist in the current HQPG-tree, then it is inserted into the HQPG-tree. Note that rather than increasing the support values of the corresponding RQPs, a vector that represents the historical support values is created for each RQP. For example, for these RQP's, the support value in the current query pattern group is appended into the existing vector of supports. If the RQP does not exist in the HQPG-tree, then the vector of supports for this RQP should be a vector started with \( i-1 \) number of 0's, where \( i \) is the ID of the current query pattern group. Figure 4.4(b) shows an example of HQPG-tree, where the vectors of supports are associated with each rooted query path.

**FCQPs and ICQPs Extraction**

Given the HQPG-tree, the FCQPs and ICQPs are extracted based on the user-defined thresholds for the corresponding dynamic metrics. Corresponding to the two definitions for the conserved query paths, two algorithms are presented. The first algorithm is based on the group-wise metrics, called them as GCQP-Miner (group-wise conserved query path miner), and the second algorithm is based on the regression metric, called RCQP-Miner (regression-based conserved query path miner). In both algorithms, the top-down traversal strategy is used to enumerate all candidates of both frequent and infrequent conserved query paths.
Algorithm 7 HQPG-tree Construction

Input:
- A sequence of QPG-trees: \( (QPG_1, QPG_2, \ldots, QPG_n) \)

Output:
- The historical query pattern group tree: HQPG-tree

Description:
1. Initialize HQPG-tree as the first QPG-Tree \( QPG_1 \)
2. for all \( 2 \leq i \leq n \) do
   3. for all \( RQP \subseteq QPG_i \) do
      4. for all \( RQP' \subseteq HQPG\text{-}Tree \) do
         5. if \( RQP' \prec RQP \) then
            6. update the vector of supports for \( RQP' \)
         7. end if
      8. if \( RQP \not\subseteq HQPG\text{-}Tree \) then
         9. Insert \( RQP \) to HQPG-Tree
     10. end if
   11. end for
3. end for
4. end for
5. Return(HQPG-tree)

Algorithm 8 GCQP-Miner

Input:
- An historical query pattern group tree: HQPG-tree
- The user-defined thresholds \( \alpha, \beta, \gamma, \xi, \xi' \)

Output:
- Sets of FCQPs and ICQPs: \( F \) and \( I \)

Description:
1. for all \( RQP \subseteq HQPG\text{-}tree \) (top-down)
2. if \( \xi > \Phi > \xi \) end if
3. if \( \Phi \geq \xi, S(\alpha, RQP) \leq \beta \) and \( A(RQP) \leq \gamma \)
4. \( F = F \cup RQP \)
5. end if
6. if \( \Phi \leq \xi', S(\alpha, RQP) \leq \beta \) and \( A(RQP) \leq \gamma \)
7. \( I = I \cup RQP \)
8. end if
9. end for
10. Return \((F, I)\)

We use the top-down traversal strategy based on the downward closure property of the group average support values for rooted query paths.

Lemma 4.3 Let \( RQP_1 \) and \( RQP_2 \) be two rooted query paths in an HQPG-tree. If \( RQP_1 \) is included in \( RQP_2 \), then \( \Phi(RQP_1) \geq \Phi(RQP_2) \).

Proof: Based on the definition of query pattern group tree, it can be inferred that \( \Phi_i(RQP_1) \geq \Phi_i(RQP_2) \) for all \( i \) \((1 \leq i \leq n)\) because whenever \( RQP_2 \) is extended included
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Algorithm 9 RCQP-Miner

Input:
- An historical query pattern group tree: HQPG-tree
- The user-defined thresholds $\zeta, \xi, \xi'$

Output:
- Sets of FCQPs and ICQPs: $F$ and $I$

Description:
1: for all $RQP \subseteq$ HQPG-tree (top-down)
2: if $\xi > \Phi > \xi'$
3: prune all the children of this RQP
4: end if
5: if $\Phi \geq \xi$ and $R(RQP) \leq \zeta$
6: $F = F \cup RQP$
7: end if
8: if $\Phi \leq \xi'$ and $R(RQP) \leq \zeta$
9: $I = I \cup RQP$
10: end if
11: end for
12: Return $(F, I)$

in a QPT, $RQP_1$ is also extended included in that QPT provided that $RQP_1$ is included in $RQP_2$. As a result, it is evident that $\Phi(RQP_1) \geq \Phi(RQP_2)$. 

Based on the above lemma, we can prune the HQPG-tree during the top-down traversal. That is, for those RQPs whose $\Phi$ are smaller than $\xi$, then no extensions of the RQPs can be FCQPs. Similarly, for these RQPs whose $\Phi$ are smaller than $\xi'$, then all of their extensions also satisfy this condition to be ICQPs. More importantly, as only rooted query paths are considered, all the rooted sub-paths of an ICQP should have a $\Phi$ that is no larger than the $\xi'$, according to the definition of ICQPs. As a result, these rooted query paths whose $\Phi$ are between the $\xi$ and $\xi'$ are pruned as they can be neither FCQPs nor ICQPs.

The GCQP-Miner algorithm is shown in Algorithm 8. The idea is to first compare the values of $\Phi$ with the threshold of group average support. In this case, some candidates can be pruned. After that, the value of $S(\alpha, RQP)$ is calculated and compared with $\beta$. If $S(\alpha, RQP) \leq \beta$, then the value of $A(RQP)$ is calculated and compared with $\gamma$. Otherwise, if $S(\alpha, RQP) > \beta$, then we do not need to calculate the value of $A(RQP)$. If $S(RQP) \leq \gamma$, then the RQP is inserted into the corresponding group of FCQPs or ICQPs. The reason for comparing the thresholds in this order is based on the cost of calculating these values. As the value of $S(RQP)$ is the most expensive one, it is only calculated for these candidates.
that satisfy all other constraints.

The RCQP-Miner algorithm is shown in Algorithm 9. The basic idea is the same as that of the GCQP-Miner algorithm: traverse the HQPG-tree in a top-down manner and compare the values of the dynamic metrics with the user-defined thresholds to enumerate all the frequent and infrequent conserved query paths. The only difference between GCQP-Miner and RCQP-Miner is that they use different dynamic metrics.

4.4 Dynamic-Conscious Cache Strategy

Based on the FCQPs and ICQPs as discussed in the previous sections, we now present how to utilize such conserved query paths to build the dynamic-conscious cache strategy. There are two major phases, ranking the conserved query paths for caching and building the dynamic-conscious caching strategy. Corresponding to the two types of results, which are extracted using the GCQP-Miner and RCQP-Miner algorithms, respectively, a rank function is presented. Then, the dynamic-conscious cache strategies are built based on the ranks of the query paths. Performances of the rank function using the two mining algorithms are compared and the usefulness of the two algorithms will be evaluated by comparing the cache strategies with and without rank functions.

4.4.1 Rank Functions

The objective of the rank functions is to rank the query paths for efficient cache replacement strategy because the number of conserved query paths can be huge while the available space for caching is limited. The intuitive idea is to assign high rank scores to query paths that are expected to be issued frequently. However, there are other factors such as the query evaluation cost and the size of the query result, which are independent from the GCQP-Miner and RCQP-Miner algorithms, are also important for efficient caching. Hence, we propose a rank function that takes all possible factors into account to assign different priority scores to these conserved query paths. We use the two sets of dynamic metrics used in the GCQP-Miner and RCQP-Miner to estimate the expected number of occurrences of the query paths.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Here, based on the two sets of dynamic metrics, together with other factors, we present a rank function. Note that the dynamic metrics are only used to predict the number of occurrences of the query paths, while the other factors are used in the similar way as they are used in other cache strategies [35, 36, 121, 183]. For instance, the cost of evaluating a query path is the time to execute this query against the XML data source without any cache strategy, while the size of the result is the actual size of the view that stores the query result.

Definition 4.26 Rank Functions: Let RQP be a rooted query path. Let the cost of evaluating a query path (denoted as \( Cost_{\text{eval}}(RQP) \)) is the time to execute this query against the XML data source without any caching strategy, while the size of the result (denoted as \( \text{result}(RQP) \)) is the actual size of the view that stores the query result, \( A(RQP) \) is the aggregated dynamic. Then the rank function, \( R \), is defined as:

- If GCQP-Miner is used to extract ICQPs and FCQPs, then
  \[
  R(RQP) = \frac{\text{Cost}_{\text{eval}}(RQP) \times \overline{F}(RQP)}{S(a, RQP) \times A(RQP) \times |\text{result}(RQP)|}
  \]

- If RCQP-Miner is used to extract ICQPs and FCQPs, then
  \[
  R(RQP) = \frac{\text{Cost}_{\text{eval}}(RQP) \times \overline{F}(RQP)}{K(RQP) \times |\text{result}(RQP)|}
  \]

The problem of dynamic-conscious caching for XML query processing is how to utilize the FCQPs and ICQPs with their rank scores in such a way that the query processing cost for future incoming queries can be minimized. The basic caching strategy is to cache the results for the FCQPs with the largest rank scores by replacing the cached results of the RQPs with smaller rank scores. Different from existing query pattern based approaches, in our approach, not only the support values of the RQPs are considered, but also the historical change patterns is taken into account in the rank functions that assign priorities of caching to each rooted query path.

4.4.2 Dynamic-Conscious Caching Strategy

In this section, we present the dynamic-conscious caching approach. As shown in Figure 4.5, firstly, according to the rank functions defined in the previous section, the sets of FCQPs
and ICQPs are ranked in descending order of the ranks. Then, based on the set of sorted conserved query paths, some of the conserved query paths and their results are cached in the cache manager. When a new query comes, the query is parsed and decomposed into a set of query paths and composing query. Then, the cached results of these query paths are mapped back using the RQP register. The query rewriter then rewrites the query by combining the cached query results with the composing query. In this query evaluation process, the ranks of these query paths in the cache are updated and query paths with smaller ranks are evicted such that query paths with large ranks can be cached. Also, the conserved query paths mining results are updated when certain number of queries have been evaluated in the system and all the cached query paths and results will be updated consequently.

The cache algorithm is shown in Algorithm 10. When a new query $q_x$ comes, different from the query pattern tree based caching, $q_x$ may match to more than one of the RQPs in the set of FCQPs, which are denoted as $M$. Hence, $q_x$ can be considered to be the join of the RQP with the highest ranks and the composing query iteratively. Finally, it is decomposed into a set of cached RQPs and the composing query $q'_x$. The answers are obtained by evaluating the composing query and joining the corresponding results. Then, for all RQPs that are contained in $q_x$, the corresponding ranks are updated with respect to the changes of $\Phi$. Note that, we do not update the FCQPs and ICQPs with the corresponding values.
Algorithm 10 Dynamic-Conscious Caching Strategy

Input:
A new XML query: $q_x$
FCQPs and ICQPs in rank score descending order: $F_p$ and $I_p$

Description:
1. $M = \{RQP_i | RQP_i < q_x \} \cap F_p$
2. if $M \neq \emptyset$
3. choose a sequence of ordered $RQP_i \in M$
4. decompose $q_x = RQP_i \circ \cdots \circ RQP_j \circ q'_x$
5. end if
6. evaluate the query by combining the results
7. for all $RQP_i \cdots RQP_j \in M$
8. update $R(RQP_i)$
9. if $R(RQP_i) < \text{Min}(R(RQP))$
10. evict the cached result of $RQP_i$ from caching
11. end if
12. end for
13. $C$ is the set of queries that have been cached
14. if $M' = C \cap I_p \neq \emptyset$
15. evict $RQP \in M'$
16. end if
17. while there is space left in the cache
18. caching the RQP with maximum rank but not in the cache
19. end while

of the dynamic metrics in the caching process, rather they are only updated after certain number of queries have been evaluated (will be discussed later) as the cost of mining these conserved query paths are expensive. If the rank for any of these RQPs falls beyond the minimum value of these RQPs in the cache, the corresponding query results will be evicted. Also, based on the list of infrequent conserved query paths, if these RQPs that have been cached are within the list of infrequent conserved query paths, the corresponding results in the cache have to be evicted. As a result, there may be some space in the cache available, if the space is enough, among these RQPs in $F_p$ but have not been cached, the RQP with maximum rank is cached till there are no cache space left.

For example, given a query $q_x$, it can be decomposed into the intersection of a composing query $q'_x$ and a list of RQPs $RQP_i$ that have been cached. For the specific query $q_x$, the corresponding results can be obtained by querying the intersection of the results for these RQPs in $RQP_i$ using $q'_x$. Also, the ranks of all the RQPs in the frequent conserved query paths are updated with respect to the updated average support value. If any of the updated rank of a RQP $RQP_i$ is less than the minimum rank in the cached RQPs, the corresponding
results will be evicted. At the same time, the RQPs whose results have been cached are checked against the list of infrequent conserved query paths, if any of them $RQP_j$ is in the list, the corresponding content is evicted from the caching. As a result, there will be some free caching space. Then, these top RQPs in the list of frequent conserved query paths with high ranks but have not been cached are cached accordingly until there is no enough space.

In summary, our dynamic conscious caching strategy is superior to the LRU caching strategy as it incorporates the ranks of the rooted query paths, which takes into account both the temporal evolution patterns, query evaluation cost, and the required caching space.

4.5 Performance Study

In this section, we evaluate the performance of the proposed mining algorithms and caching strategies with extensive experiments. The mining algorithms and the caching strategy are implemented in Java. All the experiments were conducted on a Pentium IV PC with a 1.7Ghz CPU and 512 MB RAM, running Microsoft Windows 2000 Professional. Two sets of experiments are conducted. The first set is to evaluate the mining algorithms for their efficiency and scalability. The second set of experiments is to compare our proposed caching strategy with the state-of-the-art frequent query pattern-based caching strategy and LRU-based cache strategy.

In the following experiments, two set of synthetic datasets are used. They are generated based on the DBLP.DTD \(^1\) and SSPlay.DTD \(^2\). Firstly, a DTD graph is converted into a DTD tree by introducing some "//" and "**" nodes. Then, all possible rooted query paths are enumerated. Similar to [183, 185], the collection of QPTs is generated based on the set of rooted query paths using the Zipfian distribution and these QPTs are randomly distributed in the temporal dimension. That is, most of the numbers of occurrences of QPTs are near the average number, while the number of very frequent and very rare QPTs is small. Table 4.1 shows two sets of example QPTs in the DBLP and SSPlay datasets. Each basic dataset

\(^1\)http://dblp.uni-trier.de/xml/dblp.dtd
\(^2\)http://www.kelschindexing.com/shakesDTD.html
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

<table>
<thead>
<tr>
<th>QPT ID</th>
<th>Timestamp</th>
<th>DBLP</th>
<th>SSPlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>day 1</td>
<td>dblp//key</td>
<td>play//act//scene</td>
</tr>
<tr>
<td>2</td>
<td>day 2</td>
<td>dblp/*/title[author=&quot;M. Lee&quot;]</td>
<td>play//stage</td>
</tr>
<tr>
<td>3</td>
<td>day 2</td>
<td>dblp/article/author</td>
<td>play/*/stage</td>
</tr>
<tr>
<td>4</td>
<td>day 3</td>
<td>dblp//author</td>
<td>play/prologe//onstage</td>
</tr>
<tr>
<td>5</td>
<td>day 3</td>
<td>dblp/book//editor</td>
<td>play//act//scene//stage</td>
</tr>
<tr>
<td>6</td>
<td>day 5</td>
<td>dblp/*/year</td>
<td>play//speechblock</td>
</tr>
<tr>
<td>7</td>
<td>day 5</td>
<td>dblp//page</td>
<td>play//epilogue/*</td>
</tr>
<tr>
<td>8</td>
<td>day 5</td>
<td>dblp/article/author</td>
<td>play//prologe//onstage</td>
</tr>
</tbody>
</table>

Table 4.1: Example QPTs in the DBLP and SSPlay Datasets

consists of up to 3,000,000 QPTs, which are divided into 1000 query pattern groups. The characteristics of the datasets are shown in Figure 4.6(a).

4.5.1 Mining Algorithms

Firstly, the efficiency of the mining algorithms is evaluated by varying the user-defined thresholds and using different datasets. The two algorithms WCQP-Miner and RCQP-Miner are first evaluated separately. Then their mining results are compared with each other. However, a comparison of their efficiency is not possible because they use different metrics and the mining results are different.

For the WCQP-Miner algorithm, there are four parameters $\alpha$ (group dynamic threshold), $\beta$ (sequence dynamic threshold), $\gamma$ (aggregated dynamic threshold), $\xi$ (average group support threshold) that can be varied. Similarly, for the RCQP-Miner algorithm, there are two parameters $\zeta$ (support conservation rate threshold) and $\xi$ (average group support threshold) that can be varied. Note that hereafter we use $\xi/10$ to denote $\xi'$.

Algorithm Efficiency

We evaluate the efficiency of the algorithms using different datasets. Basically, the datasets can be characterized by two features: the average size of query pattern groups and the number of query pattern groups (the size of the time window). Experiments have been done by varying one of the parameters and fixing the others and the results are presented in Figure 4.6. Figures 4.6(b) and (c) show the running time of the WCQP-Miner algorithm when the size of the dataset increases. In the first case, the number of query pattern groups
increased while the average size of each query pattern group is fixed. In the second case, the average size of each query pattern group increases while the number of query pattern groups is fixed. Note that, in the above experiments, the DBLP dataset is used and the parameters for the WCQP-Miner algorithm are fixed as follows: $\alpha = 0.02$, $\beta = 0.05$, $\gamma = 0.02$, and $\xi = 0.25$. It can be observed that when the size of the dataset increases, the running time increases as well. The reason is intuitive as the size of the HQPG-tree will be larger, which may require more time for the tree to be constructed and there will be more candidate conserved query paths to be enumerated.

Figure 4.6(d) and Figure 4.7(a) show the running time of the RCQP-Miner algorithm when the size of the dataset increases. The running time changes in the same pattern as the WCQP-Miner algorithm. In this set of experiments, the SSPlay dataset is used and the parameters for the RCQP-Miner algorithm are fixed as follows: $\xi = 0.25$ and $\zeta = 0.05$. 

124
Moreover, we conduct experiments to show the compactness of the two data structures we proposed. The sizes of the QPG-tree and HQPG-tree are compared with the size of the original QPTs. Figure 4.7(b) shows the space requirements for the DBLP and SSPlay datasets. It can be observed that the QPG-tree and the HQPG-tree are very compact and their space requirements are approximately 50% and 70% less than that of the original dataset, respectively. Relatively, HQPG-tree is much smaller than the QPG-trees. That is because there are many QPG-trees while there is only one HQPG-tree for the entire collection of XML queries.

Effects of Parameters

As there are four thresholds: \( \alpha, \beta, \gamma, \) and \( \zeta \) for the RCQP-Miner algorithm, experiments are conducted by varying one of the thresholds and fixing the others. For instance, in Figure 4.7(c), \( \alpha = 0.01 * k, \beta = 0.05, \gamma = 0.02, \zeta = 0.2^3 \) means that we fix the values of \( \beta, \gamma \)
and $\xi$ vary, and $\alpha$ from 0.01 to 0.05 as $k=1, 2, 3, 4,$ and 5. In the following experiments, the DBLP dataset with 300,000 queries is used. The results in Figure 4.7(c) show that the running time of RCQP-Miner increases when $\alpha$ increases, and/or $\beta$ increases, and/or $\gamma$ increases, and/or $\xi$ increases. Moreover, we observed that the changes to $\xi$ and $\alpha$ have more significant effects on the running time than the changes to $\beta$ and $\gamma$. It is because the values of $\xi$ affect the total number of FCQPs and ICQPs and the values of $\alpha$ affect not only group dynamic but also sequence dynamic.

Similarly, the two thresholds, $\xi$ (support conservation rate threshold) and $\zeta$ (average group support threshold), are varied to evaluate their effects on the running time of the RCQP-Miner algorithm. The results are shown in Figure 4.7(d). Note that the SSPlay dataset with 900,000 queries is used. The representation method used of the WCQP-Miner algorithm is employed here. It can be observed that the running time increases as the values of $\xi$ increase and/or the values of $\zeta$ increase. The reason is that when the values for any of the two parameters increase, the number of conserved query paths increases. As a result, it requires more time to enumerate all candidate query paths.

Comparison of Mining Results

As the two algorithms use different dynamic metrics, to compare the mining results, we define the overlap measure as follows.

**Definition 4.27 Overlap:** Let $FCQP_G$ and $ICQP_G$ be the frequent conserved query paths and infrequent conserved query paths in the WCQP-Miner mining results. Let $FCQP_R$ and $ICQP_R$ be the frequent conserved query paths and infrequent conserved query paths in the RCQP-Miner mining results. The overlap between the two sets of mining results is defined as:

\[
\text{Overlap} = \frac{1}{2} \times \left( \frac{|FCQP_G \cap FCQP_R|}{|FCQP_G \cup FCQP_R|} + \frac{|ICQP_G \cap ICQP_R|}{|ICQP_G \cup ICQP_R|} \right)
\]

Note that $|FCQP_R|$ refers to the number of unique query paths in the set of frequent conserved query paths. Basically, the overlap value is defined as the number of shared
conserved query paths divided by the total number of unique conserved query paths in both mining results. Based on this definition, it is evident that the larger the overlap value, the more similar the mining results are. If overlap is 1, then the two mining results are identical. In this definition the whole set of mining results are taken into consideration. However, in some applications such as caching, only the top-\(k\) frequent conserved query paths in the results are important. As a result, we further define overlap@\(k\) as follows.

**Definition 4.28 Overlap@\(k\):** Let FCQPG\((k)\) be the top-\(k\) frequent conserved query paths in the WCQP-Miner mining results. Let FCQPR\((k)\) be the top-\(k\) frequent conserved query paths in the RCQP-Miner mining results. The overlap@\(k\) between the two sets of mining results is defined as:

\[
\text{Overlap@}\(k\) = \frac{|FCQPG\((k) \cap FCQPR\((k)\)|}{|FCQPG\((k) \cup FCQPR\((k)\)|}
\]

We vary the thresholds for all dynamic metrics, and the experimental results with the SSPlay dataset is shown in Table 4.2. It can be observed that the overlap value between the mining results depends on setting of the parameters. However, through our experiments, as shown in Table 4.2, the overlap value can be very close to 1 when the parameters are appropriately set. That is, the two algorithms can produce the same result by tuning the parameters, which indicates that the regression-based metric can efficiently monitor both the micro-pattern and macro-pattern. Moreover, it can be observed that the top-10 frequent conserved query paths after rank are exactly the same. Even for the top-60 frequent conserved query paths, their results can be exactly the same with appropriate parameter settings. The quality of the mining results shall be further compared in terms of the corresponding cache strategy in the next section.

### 4.5.2 Caching Strategy Comparison

In this section, we evaluate the effectiveness of the dynamic-conscious caching strategies by comparing them with the existing approaches. We implemented the caching strategy by incorporating the knowledge of FCQPs and ICQPs as stated in the previous section. From
the original collections of QPTs, some QPTs are chosen as basic query paths and are extended to form the future queries. To select the basic query paths, queries that issued more recently have a higher possibility of being chosen. That is, given a sequence of \( n \) QPGs, \( \frac{n-1}{2^{i+1-n}} \) QPTs are selected from the \( i \)th group. Then, the set of selected queries are extended according to the corresponding DTD. Future queries are generated by extending previous query paths with randomly selected query paths. Note that for each of the following experiments, 10 sets of queries are generated for evaluation and the figures show the average performance.

Examples of the 10 sets of queries are generated based on the QPTs in Table 4.1.

Basically, 6 caching strategies are implemented: WCQP-Miner based dynamic conscious cache (denoted as WCQP), regression-based dynamic conscious cache (denoted as RCQP), WCQP-Miner based dynamic conscious cache without rank (denoted as WCQP-R), regression-based dynamic conscious cache without rank (denoted as RCQP-R), the original LRU caching strategy (denoted as LRU), and the frequent query pattern based caching strategy (denoted as QP) [185]. To evaluate the effectiveness of the proposed caching strategies,
two performance metrics are defined. The cost ratio is proposed to represent the response time with dynamic-conscious cache strategies against the response time without caching strategy for all the query examples. Another metric is average response time, which is the overall response time divided by the total number of query examples. Note that the FCQPs and ICQPs used in the following experiments are discovered using the WCQP-Miner and RCQP-Miner, by setting $\alpha = 0.02$, $\beta = 0.02$, $\gamma = 0.01$, $\zeta = 0.01$, and $\xi = 0.2$.

**Response Time and Cost Ratio**

Figure 4.8(a) shows the average response time of the six approaches while varying the number of queries from 10000 to 50000. Note that the cache size is fixed at 40MB. It can be observed that generally WCQP, WCQP-R, RCQP, and RCQP-R perform better than QP and LRU. It can be observed that when the number of queries increases, the gaps between our approaches and existing approaches increases as well. For instance, our cache strategies can be up to 5 times faster than QP approach and 10 times faster than the LRU approach when the number of queries is up to 50000. Also, the rank-based dynamic-conscious cache strategies outperform the dynamic-conscious cache strategies without rank scores. Generally, it can be observed that as the number of queries increases, the average response time decreases. That is because when the number of queries increases, more historical behaviors can be incorporated and the frequent query patterns and conserved query paths can be more accurate. Hence, the average response time decreases in the corresponding cache strategies. Also it can be observed that by ranking the mining results, the corresponding dynamic-conscious cache strategies become more efficient.

Figures 4.8(b) shows the performance of the six cache strategies in terms of the cost ratio measure. Note that the number of queries is fixed at 2000, while the cache size varies from 20Mb to 100Mb (for the SSPlay dataset). It can be observed that WCQP, WCQP-R, RCQP, and RCQP-R perform better than QP and LRU. Moreover, as the cache size increases, the cache strategies perform better but the gaps between our approaches and existing approaches decrease. That is, when the cache space is large enough, most cache strategies have similar performances.
CHAPTER 4. MINING HISTORICAL XML QUERY PATTERNS

Figure 4.9: Experimental results (4).

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>0.52</td>
<td>0.52</td>
<td>0.52</td>
<td>0.52</td>
<td>0.52</td>
<td>0.52</td>
</tr>
<tr>
<td>RCQP-R</td>
<td>0.29</td>
<td>0.32</td>
<td>0.37</td>
<td>0.43</td>
<td>0.51</td>
<td>0.68</td>
</tr>
<tr>
<td>WCQP-R</td>
<td>0.36</td>
<td>0.41</td>
<td>0.48</td>
<td>0.56</td>
<td>0.63</td>
<td>0.71</td>
</tr>
<tr>
<td>QP</td>
<td>0.39</td>
<td>0.45</td>
<td>0.53</td>
<td>0.61</td>
<td>0.74</td>
<td>0.86</td>
</tr>
</tbody>
</table>

Table 4.3: Effects of $q$ in cost ratio.

Also, experiments have been done to show how the number of query pattern groups in the historical collection affects the performance of our cache strategies. Figures 4.9(a) and (b) show how the average response time and cost ratio change when the number of query pattern groups increases. The SSPlay dataset is used and the average size of each query pattern group is 300. We vary the number of query pattern groups from 3000 to 15000. It can be observed that the dynamic-conscious cache strategies perform better when there are more query pattern groups used in the mining algorithm for a fixed group size. The reason is that when the number of query pattern groups is large, our conserved query paths are accurate, thus, the cache strategies become more efficient.

Effects of Mining Results in Caching

The above results show that our caching strategy outperformed existing XML query pattern based caching strategies. However, while these FCQPs and ICQPs need to be updated whenever new queries are issued and the process of mining and ranking these FCQPs and ICQPs is expensive.

To avoid such expensive overhead, we propose to incremental FCQPs and ICQPs mining
and ranking only when the number of new queries have been issued is larger than a pre-defined threshold. That is, we propose to incrementally mine the FCQPs and ICQPs only after a certain number of new queries $\delta_q$ has been issued by the users. Moreover, since the mining phase is expensive, it can be conducted offline. Here, $\delta_q$ is defined as:

$$\delta_q = q\% \times \sum_{i=1}^{n} L_i$$

where $q$ is between 1 to 100, which represent the percentage value, $L_i$ is the total number of queries in the $i$th query pattern group in the $n$ groups of queries where the FCQPs and ICQPs are extracted from.

Experiments have been done with various $q$ values to show how this parameter will affect the quality of our caching strategy. Note that, from the running cost point of view, the larger the value of $q$ is, the less the overhead is. Table 4.3 shows the performance of our proposed approaches compared with the LRU and QP approaches where 45000000 queries in the history are used for mining FCQPs and ICQPs, and the cache size is fixed to 50MB. Specifically, it can be observed that when $q$ increases, the cost ratio increases for three of the caching approaches: RCQP-R, WCQP-R, and QP, whereas the cost ratio for the LRU approach does not change. Note that, for the QP approach, rather than repeatedly updating the frequent query patterns whenever new queries are issued, the same strategy of periodically updating the mining results is used. It can be observed that the performance of our proposed RCQP-R and WCQP-R approaches are better than the QP approach for any $q$ value, and RCQP-R and WCQP-R are better than the LRU approach when $q$ is less than 0.5. That is, our proposed approaches can improve the query evaluation without updating the FCQPs and ICQPs for half the number of queries used in mining process.

Figure 4.10 shows how different $q$ values affect the cost ratio of the proposed caching approaches under different settings: the number of historical queries used for the mining of FCQPs and ICQPs, and cache size. It can be observed that our approaches produce good performance (cost ratio $< 0.5$) in most cases when $q < 0.5$. That is, independent from
the settings of other parameters, the proposed cache strategy can process half number of the queries used in the mining algorithm efficiently without updating the mining results. Further it can be observed that the cost ratio is smaller when the cache size is larger. Also it can be observed that when \( q \) is smaller, the cost ratio is smaller as well. That is, the dynamic behavior of the query patterns may change after certain number of queries have been processed. Hence, we need to optimize the frequency of FCQPs and ICQPs mining and ranking by balancing the cost of mining with the cost ratio of query evaluation. In real applications, the frequency of updating FCQPs, ICQPs, and their ranks is dependent from the properties of queries and the database.

### 4.6 Summary

Existing XML query mining approaches and XML cache strategies ignored the temporal information associated with XML queries under the MONETA framework. In this chapter, we proposed to mining the evolution of historical XML queries under the MONETA framework. Firstly, by integrating the temporal information and the support values, we define the frequent conserved query paths and infrequent conserved query paths. To extract
these conserved query paths, a group-wise query path mining algorithm and a regression-based query path mining algorithm are proposed. After that, using these conserved query paths and our proposed rank functions, four dynamic-conscious cache strategies were proposed. Experimental results show that our conserved query path mining algorithms are efficient and scalable. Moreover, the dynamic-conscious cache strategies outperform an existing state-of-the-art frequent query pattern-based cache strategy and a classic LRU cache strategy.
Chapter 5

Mining Evolution of Website Log Data

In the previous chapters, the issue of extracting pattern-based substructures from tree-structured Web data has been exploited in the context of both author-centric and visitor-centric data. In this chapter, we further illustrate the MONETA framework in the context of Website log data (also called Web usage data), which is another example of visitor-centric data that records the navigation behaviors of Web users to specific Websites. This chapter is a transitive chapter that connects the previous chapters and later chapters in the following two aspects. Firstly, different from the previous chapters, the Website log data shall be considered as an illustrative example to show how we move from tree-structured data to graph-structured data. That is, later on, we shall focus on graph structured Web data. Secondly, rather than only extracting pattern-based substructures, different analysis techniques such as clustering, time-dependent models, and semantic extraction shall be exploited in the coming chapters. Specifically, in this chapter, we focus on not only pattern-based substructure extraction but also clustering and semantic extraction based on the evolution of substructures.

The rest of this chapter is organized as follows. Firstly, in Section 5.1, we present the background of Web usage mining and an overview of existing Web usage mining techniques. In Section 5.2, the motivation of mining evolution of Website log data is discussed. Then, the three major issues of mining evolution of Website log data: Web access motif extraction,
5.1 Web Usage Mining

In this section, we review existing Web usage mining approaches. Firstly, we present the backgrounds of Web usage data mining such as the data source, major phases for Web usage mining, and usage data representation. Then, some representative of the state-of-the-art Web usage mining techniques are discussed.

5.1.1 Background

The data source of Web usage mining is the Web log data, which can be collected at different levels. Basically, there are four types of Web log data: client-level log, proxy-level log, server-level log, and content-level log as shown in Figure 5.1. Client-level log generally refers to behaviors of a single-user for multi-sites. Proxy-level log and content-level log depict behaviors of multi-users for multi-sites. Server-level log records multi-user behaviors for single-site. In the literature, most of existing Web usage mining approaches focus on analyzing the server-level log data. The reason is that the server-level log data explicitly records the navigation behaviors of all users for a specific Website. Moreover, there are many Website-level applications such as Website designing, personalization, and site-level searching that can benefit from server-level log data mining. In this chapter, we focus on
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

Table 5.1: Example of WASs

<table>
<thead>
<tr>
<th>S.ID</th>
<th>WASs</th>
<th>S.ID</th>
<th>WASs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(a, b, c, a, f, g)</td>
<td>1</td>
<td>(a, b, c, a, f, g)</td>
</tr>
<tr>
<td>2</td>
<td>(a, b, e, h, a, f, g)</td>
<td>2</td>
<td>(b, d, c, x)</td>
</tr>
<tr>
<td>3</td>
<td>(e, f, g, i, n)</td>
<td>3</td>
<td>(e, f, g, i, n)</td>
</tr>
<tr>
<td>4</td>
<td>(b, d, c, a, e)</td>
<td>4</td>
<td>(b, e, h, b, d, c, n, f, g)</td>
</tr>
</tbody>
</table>

(a) The first month  (b) The second month

(c) The third month  (d) The fourth month

The server-level log as well. Hereafter, in the rest of this chapter, the term Website log/usage data refers to the server-level log data.

The Website log data records the user IP address, user ID, time/date, request, status, bytes, referrer, and agents. The user IP address and user ID are used to identify different users. The request information contains the URL of the Web pages being accessed and the corresponding methods for this request such as GET and POST. The Web usage data can be stored in the Common Log Format (CLF) or Extended Common Log Format (ECLF) [119].

5.1.2 Overview of Web Usage Mining (WUM)

Web Usage Mining (WUM), the application of data mining techniques to discover usage patterns from Web data, has been an active area of research and commercialization [151]. Generally, the Web usage mining process can be considered as a three-phase process, which consists of data preparation, pattern discovery, and pattern analysis [151]. Since the last phase is application-dependent, let us briefly describe the first two phases. In the first phase, the Web log data are transformed into sequences of events (called Web Access Sequences (WASs)) based on the identification of user sessions and the corresponding timestamps. For example, given a Web log archive that records the navigation history of a Web site, by using some existing preprocessing techniques [37, 173], the raw log data can be transformed
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

into a set of WASs. Table 5.1 shows some examples of such WASs. Here S.ID represents a
sequence id. A WAS such as \((a, b, d, c, a, f, g)\) denotes a visiting sequence from Web page \(a\)
to pages \(b, d, c, a, f\) and finally to page \(g\). Each sub-table in Table 5.1 records the collection
of WASs for a particular month. In the second phase, statistical methods and/or data
mining techniques are applied to extract interesting patterns such as Web Access Patterns
(WAPs)\cite{135}. A WAP is a sequential pattern in a large set of WASs, which is visited
frequently by users \cite{135}. That is, given a support threshold \(\xi\) and a set of WASs (denoted
as \(A\)), a sequence \(W\) is a WAP if \(W\) appears as a subsequence\(^1\) in at least \(\xi \times |A|\) Web access
sequences of \(A\). For clarity, we call such a WAP a frequent WAP. Consequently, a sequence
that appears in fewer than \(\xi \times |A|\) Web access sequences in \(A\) is called an infrequent WAP.
These patterns are stored for further analysis in the third phase.

Given a WAS denoted as \(A = (p_1, p_2, p_3, \ldots, p_n)\), in the literature, there are various
ways to represent the relationship among Web pages in the sequence \cite{135, 173}. In \cite{135}, a
WAS is represented as a flat sequence, while in \cite{173} a WAS is represented as an unordered
tree, which was claimed more informative with the hierarchical structure.

5.1.3 Existing Web Usage Mining Approaches

Existing Web usage data mining techniques include statistical analysis \cite{151}, association rules
\cite{83}, sequential patterns \cite{135}, and dependency modeling \cite{76}. Among them, the issue of Web
access pattern mining is the foundation of Web usage mining. In this section, we review the
state-of-the-art Web access pattern mining techniques and their extensions considering the
dynamic nature of Web usage data.

Web access pattern mining is defined to extract hidden patterns from the navigation
behavior of Web users \cite{37}. Different algorithms have been proposed to mine different types
of access patterns from the Web logs such as the maximal frequent forward sequence mining
\cite{37}, the maximal frequent sequence mining with backward traversal \cite{173}. There are also
algorithms for general Web access pattern mining such as the WAP-Mine \cite{135}, the maximal
and closed access pattern mining \cite{44, 174}, etc.

\(^1\)If there are two WASs \(A_1 = (B, E, A)\) and \(A_2 = (A, B, C, E, A)\), then \(A_1\) is a subsequence of \(A_2\).
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

The maximal frequent forward sequence mining approach in [37] transforms the browsing patterns of users to maximal forward sequences. This work is based on statistically dominant paths. A maximal forward reference is defined as the sequence of pages requested by a user up to the last page before backtracking. Suppose the traversal log contains the following traversal path for a user: \( (A, B, C, B, D) \), after the transformation, the maximal forward sequences are \( (A, B, C) \) and \( (A, B, D) \). However, this approach may lose some useful information about the structure [173]. Considering the above example, the structural information about the need to have a direct link from page \( C \) to page \( D \) is lost.

The maximal frequent sequence proposed in [173] keeps backward traversals in the original sequences and thus preserve some structural information, i.e., the sibling relation within a tree structure. However, the maximal frequent sequence approach requires contiguous page access in the pattern, which limits the patterns to be found. Suppose there are two sessions \( (A, B, C, A, E, A, D) \) and \( (A, B, C, A, D) \). The access pattern of \( (A, B, C, A, D) \) may be missed since the access patterns \( (A, B, C) \) and \( (A, D) \) are not contiguous in the first session.

Two other approaches have been proposed in [44, 174] to overcome this limitation by considering the sessions as unordered trees. In [174], the authors proposed a compact data structure, FST-Forest to compress the trees and still keeps the original structure. A PathJoin algorithm was proposed to generate all maximal frequent subtrees from the forest. However, this approach is neither scalable nor efficient. In [44], Chi et al. use the canonical form to represent the unordered trees. The authors proposed the CMTreeMiner algorithm to mine the maximal and closed frequent subtrees. It is claimed to be much faster than the PathJoin algorithm.

Another general algorithm is the WAP-Mine approach proposed in [135]. The authors proposed a compressed data structure, WAP-tree, to store data from the Web logs. The WAP-tree structure facilitates the mining of frequent Web access patterns. Different from the Apriori-based algorithms, the WAP-Mine algorithm avoids the problem of generating explosive numbers of candidates. Experimental results show that the WAP-Mine is much
faster than the traditional sequential pattern mining techniques. Although the WAP-tree
technique improved the mining efficiency, it recursively reconstructs large numbers of inter-
mediate WAP-trees that require expensive operations of storing intermediate patterns.

Considering the dynamic property of the datasets, there are several techniques proposed
recently for maintaining and update previously discovered knowledge. They focus on two
major issues. One is to actualize the knowledge discovered by detecting changes in the
data such as the DEMON framework proposed by Ganti et al. [69]. Another is to detect
interesting changes in the KDD mining results such as the FOCUS framework proposed
by Ganti et al [68], PAM proposed by Baron et al. [20], and the fundamental rule change
detection tools proposed by Liu et al. [115].

5.2 Motivation

Existing Web usage mining approaches, as discussed in the previous section, focus on im-
proving the efficiencies of Web access pattern mining algorithms and detecting/updating
the corresponding mining results. We observed that these approaches have the following
limitations.

- Existing Web access pattern mining approaches did not fully utilize the temporal
  information associated with the Web usage data. That is, these approaches either
  ignore the temporal information with Web usage data, or focus on updating/detecting
  changes to the mining results, while the dynamic and temporal properties of the Web
  usage source data are ignored.

- The dynamic nature of Web usage has attracted some research attentions in the Web
  usage mining community. However, existing works focus on the issue of updating and
detecting changes to the Web usage mining results. That is, none of the existing
approaches address the issue of extracting semantics (e.g., patterns and underlying
reasons) behind the Web usage data dynamic.
• Most of the existing Web usage mining approaches focus on only the Web usage data itself. However, it is possible to get better Web mining results by integrating Web usage mining with Web content mining and Web structure mining, especially for the issue of extracting semantics from evolution of Website data.

Considering the above limitations of existing Web usage mining approaches, in this chapter, we focus on mining the changes behind dynamic Web data. The Web usage data is used as an example. Specifically, the following three major issues: evolution-pattern based Web access sequence extraction, clustering of Web access sequences based on evolution-patterns, and semantic extraction from evolution of Website data, will be discussed. Note that these issues have been discussed in detail in Chapter 1, here we review them briefly. Firstly, in the issue of evolution-pattern based Web access sequence extraction, Web access sequences with specific evolution-patterns are extracted in the context of tree representation. The evolution-patterns are measured by a set of metrics that are defined in the similar way as they are defined in the previous chapters. For the issue of clustering Web access sequences based on evolution-patterns, a more robust semantic similarity is defined for the tree structured Web access sequences by incorporating their evolution-pattern in the history. From the results of clustering Web access sequences, we observe that the dynamic nature of Web usage data is event driven. As a result, we address the issue of semantic extraction using the approach of event detection from evolution of Website data in the context of graph representation. In the following sections, we shall elaborate on the three issues in turn.

5.3 WAM Mining

In this section, we focus on extracting Web Access Motifs (WAMs). To define WAMs, not only the frequencies of the WAsSs are considered as important, the consistences of their change patterns in the history are also taken into account. The basic idea is that WAMs refer to a subset of WAsSs, whose frequencies are consistent in the history and are within the user defined range. More specifically, we are interested in the popular/frequent
WAMs and unpopular/infrequent WAMs. Such WAMs are useful for many applications, such as intelligent Web advertisement, Web site restructuring, business intelligence, and intelligent Web caching [206]. In the rest of this section, the issues of representation, metrics, and mining algorithms will be discussed in turn. Lastly, the experimental results shall be presented to evaluate our proposed WAM mining approaches.

### 5.3.1 Representation of Historical Web Usage data

In this approach, we adopt the unordered tree representation of WAS. A WAS tree is defined as $T_A = (r, N, E)$, where $r$ is the root of the tree that represents Web page $p_1$; $N$ is the set of nodes where $N = \{p_1, p_2, \ldots, p_n\}$; and $E$ is the set of edges in the maximal forward sequences of $A$. An example of WAS tree is shown in Figure 5.2(a), which corresponds to the first WAS shown in Table 5.1(a).

As a result, a WAS group consists of a bag of WAS trees. Here, all occurrences of the same WAS within a WAS group are considered as identical. Then the WAS group can also be represented as an unordered tree by merging the WAS trees. We propose an extended WAS tree to record the aggregated support information about the bag of WASs within a WAS group. The extended WAS tree is defined as follows.

**Definition 5.29 Extended WAS Tree:** Let $G = [A_1, A_2, \ldots, A_k]$ be a bag of WASs, where each WAS $A_i$, $1 \leq i \leq k$, is represented as a tree $T_{A_i} = (r_i, N_i, E_i)$. Then, the extended WAS is defined as $T_G = (r, N, E, \Theta)$, where $N = N_1 \cup N_2 \cdots \cup N_k$; $E = E_1 \cup$
Consider the first WAS group in Table 5.1. The corresponding extended WAS tree is shown Figure 5.2(b), where the value associated with each node is the $\Theta$ value. It can be observed that the common prefix for different WAS trees is presented only once in the extended WAS tree. For example, the common prefix of $(a, b, d, c, a, f)$ and $(a, b, e, h, a, f, g)$ is $(a, b, a, f, g)$, which is presented once in the extended WAS tree. Details of how to construct the extended WAS tree is similar to the H-DOM tree and H-QPG-tree construction discussed in Chapter 4.

The simplistic method of representing the historical WAS groups is to merge the sequence of extended WAS trees together to form an historical WAS tree (called H-WAS tree) in a similar way as we have merged the WAS trees to form the extended WAS tree. However, the H-WAS tree and extended WAS tree are different in several aspects. Firstly, all occurrences of the same WAS tree in one WAS group are considered to be equal, while occurrences of the same extended WAS tree in a sequence of WAS groups may have different support values. Secondly, the order of extended WAS trees is important in the construction of the H-WAS tree, while the order of WAS trees is not important in the construction of the extended WAS tree. Moreover, the purpose of the extended WAS tree is to record the support values in a specific WAS group, while the purpose of the H-WAS tree is to record the history of support values of the WASs. Intuitively, the historical support values in the H-WAS tree may be represented as a time series, where the $i^{th}$ element represents the support values of the WAS in the $i^{th}$ WAS group.

**Definition 5.30 H-WAS Tree:** Let $H_G = (G_1, G_2, G_3, \ldots, G_k)$ be a sequence of $k$ WAS groups, where each WAS group $G_i$, $1 \leq i \leq k$, is represented as an extended WAS tree, $T_{G_i} = (r_i, N_i, E_i, \Theta)$. Then, the H-WAS tree is defined as $H_G = (r, N, E, \Lambda)$, where $r$ is a virtual root; $N = N_1 \cup N_2 \cup \cdots \cup N_k$; $E = E_1 \cup E_2 \cdots \cup E_k$; and $\Lambda$ is a function that maps each node in $N$ to the sequence of historical support values of the corresponding WAS.
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

Note that, in the H-WAS tree, there is a sequence of support values for each node; while there is only one support value for each node in the extended WAS. Rather than using the entire sequence of support values, we propose different dynamic metrics to summarize the history of support values and make the H-WAS tree more compact.

5.3.2 Dynamic Metrics

Given a collection of historical Web usage data, according to the representation method introduced in the previous Section, it can be represented as an H-WAS tree. Then, for a WAS A, there is a sequence of support values \( H_A = \{ \Phi_1(A), \Phi_2(A), \Phi_3(A), \ldots, \Phi_k(A) \} \). We propose two metrics support range and conservation rate to summarize the sequence of historical support values.

Firstly, maximum popularity support of A (denoted as \( M_A \)) is defined as \( M_A = \Phi_i \) where \( \Phi_i \geq \Phi_j \forall 0 \leq j \leq k \) and \( i \neq j \). Similarly, minimum unpopularity support of A (denoted as \( U_A \)) is \( U_A = \Phi_r \) where \( \Phi_r \leq \Phi_j \forall 0 \leq j \leq k \) and \( r \neq j \). Then, the pair \( (M_A, U_A) \) is called the support range of A (denoted as \( R = (M_A, U_A) \)).

To model the change patterns in the history, we propose to model the sequence of support values using linear regression model as we discussed in Chapter 4.

\[
\Phi_t(A) = \Phi_0(A) + \lambda t, \text{ where } 1 \leq t \leq k
\]

Based on the linear regression-based model for the Web usage data and the corresponding correlation coefficient, \( r \), we now propose the metric conservation rate.

**Definition 5.31 Conservation Rate:** Let \( \{ \Phi_1(A), \Phi_2(A), \ldots, \Phi_k(A) \} \) be the sequence of historical support values of a WAS A, where \( \Phi_i(A) \) represents the \( i^{th} \) support value for A and \( 1 \leq i \leq k \). The conservation rate of WAS A is defined as \( C_A = r^2 - |\lambda| \).

Note that the larger the absolute value of the slope, the more significantly the support changes over time. At the same time, the larger the value of \( r^2 \), the more accurate is the regression model. Hence, the larger the conservation rate \( C_A \), the support values of the...
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

WAS change less significantly. In other words, the support values of a WAS are more conserved with the increase in the conservation rate. Also from the regression model, it can be inferred that \(|\lambda| < \frac{1}{k}\) as \(0 \leq \Phi_t(A) \leq 1\). In real life the value of \(k\) can be huge, thus \(|\lambda| \ll r \leq 1\). Consequently, we can guarantee that \(0 \leq C_A \leq 1\). When \(C_A = 1\), the support of WAS \(A\) is a constant where \(r^2 = 1\) and \(\lambda = 0\). Then, we formally define the popular WAM and unpopular WAM as follows.

**Definition 5.32 Popular/Unpopular WAM:** Given the popularity threshold \(\alpha\) and a conservation threshold \(\mu\), a WAP \(W\) is a popular WAM if and only if \(\forall W' \subseteq W, C_{W'} \geq \mu\) and \(M_{W'} \geq \alpha\). Similarly, given the unpopularity threshold \(\beta\), a WAP \(W\) is an unpopular WAM if and only if \(\forall W' \subseteq W, C_{W'} \geq \mu\) and \(U_{W'} \leq \beta\).

### 5.3.3 WAM Mining Algorithms

Based on the definitions of popular WAMs and unpopular WAMs, our objective of WAM mining is to find all popular and unpopular WAMs in the historical Web log data given some popularity and unpopularity thresholds, and conservation threshold.

Given the H-WAS tree, with the user-defined threshold for conservation rate (\(\mu\)), popularity threshold (\(\alpha\)), and unpopularity threshold (\(\beta\)), the WAM extraction phase is actually a traversal over the H-WAS tree. The process of H-WAS tree construction is similar to the algorithms for H-DOM tree construction and H-QPG-tree construction. The WAM extraction is to enumerate all possible WASs and check the metric values against the corresponding user-defined thresholds. As the number of WAS groups can be huge, the size of the H-WAS tree can be huge if we store all the support value sequences. As a result, we present two algorithms. One is called AGGREGATED WAM-MINER (AGG-W-M), which summarizes sequences of support values using the dynamic metric values, the other is called INCREMENTAL WAM-MINER (INC-W-M), which supports incremental mining by storing sequences of support values.

The AGG-W-M algorithm is shown in Algorithm 11. Here, the support range is first compared with \(\alpha\) and \(\beta\) to determine the potential groups of popular WAMs and unpopular
Algorithm 11 AGG-W-M

Input: The H-WAS tree with values for the dynamic metrics: $H_G$
Thresholds: $\mu$, $\alpha$, and $\beta$

Output: The popular and unpopular WAMs: $W_P$ and $W_U$

1: for all node $n_i \in H_G$ do
2:   if $M_{n_i} \geq \alpha$ then
3:     if $C_{n_i} \geq \mu$ then
4:       $W_P = n_i \cup W_P$
5:     end if
6:   else
7:     if $U_{n_i} \leq \beta$ then
8:       if $C_{n_i} \geq \mu$ then
9:         $W_U = n_i \cup W_U$
10:      end if
11:   end if
12: else
13:   prune $n_i$
14: end if
15: end for
16: Return($W_P, W_U$)

WAMs to which the corresponding WAP belongs to. If $M_A \geq \alpha$ and $U_A \leq \beta$ then, the conservation rate is further compared with the threshold $\mu$. These WAPs whose conservation rate is no greater than $\mu$ are assigned to the popular WAMs and unpopular WAMs accordingly. Lastly, the sets of popular and unpopular WAMs are returned.

Example Let us take the H-WAS tree in Figure 5.2(c) as an example. Let $\alpha = 0.3$, $\beta = 0.05$, and $\mu = 0.7$. First, we check the root of the H-WAS tree, its $M_r > 0.3$ and $C_r > 0.7$, then node $a$ is included in the popular WAMs. Then, nodes $b$, $d$, $c$ are checked in a similar way. In this example, node $e$ is pruned out but its child node $h$ is included, then node $e$ is directly linked to node $b$ in the final result.

The INC-W-M algorithm is shown in Algorithm 12. Here, firstly, the support range is calculated on the fly and compared with the $\alpha$ and $\beta$ value. Then, if the support range is out of the range for popular and unpopular WAMs, we do not need to calculate the conservation rate values at all. Otherwise, we calculate the conservation rate on the fly and compare the value with $\mu$. These WAPs whose conservation rate is no greater than $\mu$ are assigned to the popular WAMs and unpopular WAMs accordingly. Lastly, the sets of popular and unpopular WAMs are returned.

The difference between this two WAM mining algorithms is that AGG-W-M is space efficient but it does not support incremental mining and changes of parameter values. That
Algorithm 12 INC-W-M

Input: The H-WAS tree with sequences of support values: \( H_G \)
Thresholds: \( \mu, \alpha, \) and \( \beta \)

Output: The popular and unpopular WAMs: \( W_p \) and \( W_U \)

1: for all node \( n_i \in H_G \) do
2: Calculate the support range
3: if \( M_{n_i} \geq \alpha \) then
4: Calculate the conservation rate
5: if \( C_{n_i} \geq \mu \) then
6: \( W_p = n_i \cup W_p \)
7: end if
8: else
9: if \( U_{n_i} \leq \beta \) then
10: Calculate the conservation rate
11: if \( C_{n_i} \geq \mu \) then
12: \( W_U = n_i \cup W_U \)
13: end if
14: end if
15: else
16: prune \( n_i \)
17: end if
18: end for
19: Return\( (W_p, W_U) \)

is, when new access sequences are inserted, the whole H-WAS-tree has to be re-constructed again. Whereas, the INC-W-M requires more space but is more flexible as it supports incremental mining. That is, rather than re-construct the whole H-WAS-tree, only the new data are incorporated. In summary, the INC-W-M is designed for very dynamic datasets, while AGG-W-M is designed for very large but relatively stable datasets. Performance of the two algorithms shall be compared in the next section.

5.3.4 Experimental Results

In this section, we present experimental results to evaluate the performance of our proposed AGG-W-M and INC-W-M algorithms. All experiments were conducted on a P4 1.80 GHz PC with 512Mb main memory running Windows 2000 professional. The algorithm is implemented in Java.

Both real and synthetic Web log datasets are used in the experiments. The real data is the Web log \( \text{UoS} \) obtained from the Internet Traffic Archive [101]. It records the historical visiting patterns for University of Saskatchewan from June 1, 1995 to December 31, 1995. There were 2,408,625 requests with 1 second resolution and 2,981 unique URLs. The synthetic data set is generated using the synthetic tree generation program used in [193].
The characteristics of the synthetic data we used are shown in Table 5.2. The program first constructs a tree representation of the web site structure based on two parameters, the maximum fan out of a node (denoted as $F$) and the maximum depth of the tree (denoted as $D$). Based on the web site structure, a collection of WASs with the corresponding timestamps are generated by mimicking the user behaviors. In Table 5.2, $\overline{S}$ is the average size of the WASs and $N$ is the number of WASs in the corresponding datasets.

### Scalability and Efficiency

As the size of the web usage data collection can be affected by two factors: the number of WASs (indicates the size of the time window for a given dataset) and the average size of each WAS, two sets of experiments have been conducted to evaluate the scalability of our proposed algorithm. In the first set of experiments, denoted as $E_1$ in Figure 5.3(a), synthetic datasets $D_1$, $D_2$, $D_3$, $D_4$, and $D_5$ are used, where the average size of each WAS is fixed while the number of WASs is varied. In the second set of experiments, denoted as $E_2$ in Figure 5.3(a), synthetic datasets $D_2$, $D_6$, $D_7$, $D_8$, and $D_9$ are used, where the number of WASs is fixed while the average size of each WAS varies.

Figure 5.3(a) shows the running time of the algorithm as the total number of nodes in the dataset increases. The user defined time interval, $\alpha$, $\beta$, $\mu$ are set to 12 hours, 0.01, 0.025, and 0.8 accordingly. The running time increases as the total number of nodes increases from 100k to 600k. The reason is that with more nodes, both the cost of constructing the trees and the traversal over the H-WAS tree becomes more expensive. However, we observed
that even for the same total number of nodes, the running time is much expensive when the number of WASs is large and the average size of each WAS is small. This is because the cost of calculation of $\Phi_0$ and the conservation rate is quite expensive when the number of extended WAS trees is large. Note that for the same user-defined time interval, a larger number of WASs indicates that there are more extended WAS trees. It can be observed that the Inc-W-M outperformed the Agg-W-M in terms of running time as we explained in the algorithm section. It can be observed that the cost of calculating the conservation rate values is very expensive, which is the gap between the two algorithms shown in Figure 5.3(a). As a result, the cost of calculating the conservation rate for WASs other than the candidates makes the running time of Agg-W-M almost doubles the running time of Inc-W-M.

Besides the size of the datasets, experiments are also conducted to show how various
parameters such as user-defined time intervals, conservation rate, popularity threshold, and unpopularity threshold, affect the efficiency of the mining algorithm. Figure 5.3(b) shows how the user-defined time interval affects the running time using $D_1$, $D_4$ and $D_9$. We set $\alpha = 0.1$, $\beta = 0.005$, and $\mu = 0.8$. Here, we use the average number of WASs in the WAS groups to represent the size of the time interval. It can be observed that the running time decreases as the size of the user-defined time interval increases. The reason is that the number of extended WAS trees is small as the average size of the WAS group increases. As a result, the computation cost of calculating the support range and conservation rate decreases. Similarly, the experiments verified that the INC-W-M outperformed the AGG-W-M in terms of running time.

Figure 5.3(c) shows the relationship between the running time and the thresholds for INC-W-M algorithm using $D_9$. There are three variables in this figure, the x-axis $k$ changed from 1 to 5, and the values of $\alpha$, $\beta$, and $\mu$ are dependent on $k$. For example, in the first set of experiment, $\beta = 0.005$ and $\mu = 0.8$; while $\alpha = 0.05 \times k$. Similarly, the values of $\beta$ and $\mu$ are changed in a similar way in the remaining two experiments. It can observed that when $\alpha$ increases, the running time decreases because the number of popular WAMs decreases accordingly. When $\beta$ increases, the running time increases because there are more unpopular WAMs. When $\mu$ increases, the running time is almost stable, which is because of the computation cost is independent of the threshold of conservation rate. Note that the
running time of AGG-W-M algorithm does not change when the thresholds changed because the cost of calculating values of the dynamic metrics are always required to summarize the sequences of support values.

Figure 5.4 shows the size of the two \(H-WAS\) tree in the AGG-W-M and Inc-W-M algorithms. The first 5 datasets in Table 5.2 were used. From the results, it is obvious that the Inc-W-M outperformed the AGG-W-M algorithm in terms of memory space requirement. As we mentioned, the reason is that in the Inc-W-M approach the entire support sequences are stored while in the AGG-W-M approach only the values for the dynamic metrics were stored.

Quality of Popular and Unpopular WAMs

As there are four parameters, the user-defined time interval, \(\alpha\), \(\beta\), and \(\mu\), in our algorithm, in this section, we investigate how the four parameters affect the quality of the mining results. By varying one parameter and fixing the values for the other three parameters, the effects of each parameter are evaluated in the following experiments. Note that the size of the time interval is measured by the average number of \(WAS\) in each \(WAS\) group. In the following experiments, the \(UoS\) real dataset is used.

In the first set of experiments, \(\alpha, \beta\) and \(\mu\) are fixed to 0.1, 0.005, and 0.8 respectively, the user-defined time interval varies from 40 to 200. Table 5.3(a) shows the number of popular WAMs and unpopular WAMs with different user-defined time interval. We observed that

<table>
<thead>
<tr>
<th>(P_1)</th>
<th>(P_2)</th>
<th>Accuracy</th>
<th>(\alpha)</th>
<th>(\beta)</th>
<th>(\mu)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>0.94</td>
<td>0.4</td>
<td>0.05</td>
<td>0.8</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>0.93</td>
<td>0.3</td>
<td>0.05</td>
<td>0.8</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>0.95</td>
<td>0.4</td>
<td>0.01</td>
<td>0.7</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>0.93</td>
<td>0.4</td>
<td>0.05</td>
<td>0.8</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>0.94</td>
<td>0.4</td>
<td>0.05</td>
<td>0.6</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>0.93</td>
<td>0.4</td>
<td>0.05</td>
<td>0.8</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0.93</td>
<td>0.4</td>
<td>0.05</td>
<td>0.8</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0.94</td>
<td>0.3</td>
<td>0.05</td>
<td>0.8</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>0.93</td>
<td>0.3</td>
<td>0.05</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 5.3: WAM experimental results.
as the time interval increases, the number of popular and unpopular WAMs increases. By looking into the results, we observed that popular and unpopular WAMs with smaller user-defined time intervals are also popular and unpopular WAMs with larger user-defined time intervals. We also compare the number of popular WAMs extracted by our AGG-W-M with the number of popular WAMs extracted by repeatedly using WAP-Mine\(^2\)[135]. We observed that the WAP-Mine based approach cannot extract all the popular WAMs. Note that, in the WAP-Mine-based popular WAM extraction approach, the conservation rate is calculated using the number of times a WAP is frequent in the sequence of WAS groups divided by the total number of WAS groups.

By fixing the user-defined time interval to 40, the effects of the other three parameters are evaluated in similar ways. Figure 5.3(d) shows how the total number of popular and unpopular WAMs changes with different \(\alpha\), \(\beta\), and \(\mu\). Here, we introduce a variable, \(k\), as the x-axis. Then, the values of \(\alpha\), \(\beta\), and \(\mu\) are represented using \(k\). For example, in the first set of experiments, \(\beta = 0.005\) and \(\mu = 0.8\); while \(\alpha= 0.05 \ast k\). It can be observed that the total number of WAMs increases as \(\beta\) increases, \(\alpha\) decreases, or \(\mu\) decreases.

Table 5.3(b) shows the quality of the regression-based model for extracting WAMs. In this experiment, the UoS dataset is partitioned into 30 WAS groups and is divided into two parts, denoted as \(P_1\) and \(P_2\). \(P_1\) is used to construct the regression model and \(P_2\) is used to evaluate the accuracy of the model. That is, we extract the popular and unpopular WAMs in \(P_1\) using the regression model and check whether these are still popular/unpopular WAMs in \(P_2\). The accuracy is defined as the percentage of popular/unpopular WAMs obtained from \(P_1\) that are still popular/unpopular WAMs in \(P_2\). Formally, let \(R_1\) and \(R_2\) be the sets of popular and unpopular WAMs returned by the AGG-W-M using \(P_1\). Let \(Z_1\) and \(Z_2\) be the sets of popular and unpopular WAMs based on the entire dataset. Then accuracy is denoted as \(\frac{1}{2}(\frac{|R_1 \cap Z_1|}{|Z_1|} + \frac{|R_2 \cap Z_2|}{|Z_2|})\). The results show that the accuracy of our model is quite high for different size of \(P_1\). Furthermore, the quality of the model is not affected by the user-defined

\(^2\)Downloaded from http://www.cs.ualberta.ca/~tszhu
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

Figure 5.5: Support of WASs over a time period

thresholds as here we only identify whether a WAM is still popular/unpopular in $P_2$. The reason is that the more training data is used, the more accurate the results are.

5.3.5 Summary

In this section, we focus on discovering novel knowledge by analyzing the change patterns of historical web access sequence data. Specifically, we propose two algorithms called AGG-W-M and INC-W-M that extract popular and unpopular Web Access Motifs (WAMs) from historical web usage data. WAMs are WAPs that never change or do not change significantly most of the time (if not always) in terms of their support values during a specific time period. WAMs are useful for many applications, such as intelligent web advertisement, website restructuring, business intelligence, and intelligent web caching. Experimental results on both synthetic data and real datasets show that AGG-W-M and INC-W-M are efficient and scalable. More importantly, they can extract novel knowledge that cannot be discovered by existing web usage mining approaches.

5.4 CLEOPATRA: Evolutionary Pattern-based Clustering of WAS

In this section, we focus on clustering of WASs based on the characteristics of their evolution over time. The intuition behind this is that WASs are event/task driven, in turn, WASs
related to the same event/tasks are expected to be accessed in a similar way over time. For example, consider Figure 1(b), which depicts the support values \( y \)-axis of five WASs (denoted as \( A_1, A_2, A_3, A_4, \) and \( A_5 \)) from time period 1 to 6 \( x \)-axis. Note that \( i \) in the \( x \)-axis represents a time period (e.g., day, week, month etc.) and not a particular time point.

It can be observed that evolutionary pattern of the supports for \( A_1, A_3, \) and \( A_5 \) are very similar over time (like the letter “W”). Similarly, the evolutionary patterns of supports for \( A_2 \) and \( A_4 \) are similar (like the letter “M”). However, the “W” and “M” clusters cannot be discovered by existing Web usage mining techniques due to the fact that they focus either on the structural patterns within individual time intervals or sequential pattern of web page items, whereas the above cluster examples are the combination of both structures and sequential information. To extract those clusters, we propose the dynamic metrics based structural similarity measure and the CLEOPATRA (CLustering of EvOlutionary PATteRn-based Web Access sequences) algorithm. Note that our contribution in this section is the temporal and structural based similarity measure rather than the clustering algorithm.

### 5.4.1 Dynamic Metrics

In this approach, the \( H\text{-WAS} \) tree representation of the historical Web usage data discussed in the previous section is used. Rather than using the entire sequence of support values, we propose another two metrics called local dynamic and global dynamic to summarize the history of support values. The reason for proposing the new dynamic metrics is that the previous conservation rate and average support value can measure the macro-pattern based changes and produce high quality results for extracting pattern-based WASs (verified in Chapter 4). However, for clustering and semantic extraction, we need to use both the micro-pattern and the macro-pattern based metrics. The following metrics are extended from the version dynamic and degree of dynamic defined in the previous chapters.

**Definition 5.33 Global Dynamic:** Given a WAS, \( A \), with the corresponding support count sequence \( H_A = (\Phi_1(A), \Phi_2(A), \ldots, \Phi_n(A)) \), the global dynamic, denoted as \( \omega(A) \), is
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

defined as:

\[ \omega(A) = \frac{1}{n-1} \sum_{i=1}^{n-1} d_i \text{ where } d_i = \begin{cases} 1, & \text{if } \Phi_i(A) \neq \Phi_{i+1}(A); \\ 0, & \text{otherwise} \end{cases} \]

\[
\text{Definition 5.34 Local Dynamic: Given a WAS, } A, \text{ with the corresponding support count sequence } H_A = (\Phi_1(A), \Phi_2(A), \ldots, \Phi_n(A)), \text{ the local dynamic, denoted as } \chi(A), \text{ is defined as a sequence } \\
\chi(A) = (\chi_1(A), \chi_2(A), \ldots, \chi_{n-1}(A)), \text{ where } \\
\chi_i(A) = \frac{\Phi_i(A) - \Phi_{i+1}(A)}{\max\{\Phi_i(A), \Phi_{i+1}(A)\}}, \\
\text{for } 1 \leq i < n-1. \]

Figure 5.2(c) shows a part of an H-WAS tree, where the associated values are the corresponding global dynamic value, and the sequence of local dynamic values. The global dynamic measures how frequently the WAS changed and the local dynamic measures how significant are the changes in the history. Furthermore, based on the local dynamic metric, we propose an evolutionary pattern-based distance to measure the relationships between WASs.

\[
\text{Definition 5.35 Evolutionary Pattern-based Distance: Given two WASs } (A_1 \text{ and } A_2), \text{ the evolutionary pattern-based distance between } A_1 \text{ and } A_2, \text{ denoted as } D(A_1, A_2), \text{ is defined as:} \\
D(A_1, A_2) = \sqrt{(\bar{x}_1(A_1) - \bar{x}_1(A_2))^2 + \cdots + (\bar{x}_{n-k+1}(A_1) - \bar{x}_{n-k+1}(A_2))^2} \\
\text{where } \bar{x}_i(A_j) = \frac{1}{k} \sum_{i}^{i+k-1} \left( \frac{\chi_i(A_j) - \overline{\chi}(A_j)}{\sigma(A_j)} \right), \text{ } k \text{ is the user defined window size, } \overline{\chi}(A_j) \text{ and } \sigma(A_j) \text{ are the average support count value and standard deviation of } \chi(A). \)

Note that, the above evolutionary pattern-based distance measure is actually the Euclidean distance between the smoothed \(\chi(A)\) sequence using the moving average. This distance measure can handle WASs with different baseline, scale, and time offset. Such properties are highly desired in this specific problem for the follow reasons. Firstly, the average \(\chi(A)\), which can be viewed as the baseline for the \(\chi(A)\) sequence, for WASs that are related to the same event/task may vary a lot while their evolutionary patterns are similar.
Secondly, the effects of event/task on different WASs can be different, which makes the scales of changes ($\chi(A)$) to those WASs different. Thirdly, there may be a different time delays for different WASs related to the same event/task, which may cause the time offset among $\chi(A)$ sequences.

5.4.2 CLEOPATRA Algorithm

Given a collection of WASs, with an evolutionary pattern-based distance $D$ and the global dynamic, the objective of the CLEOPATRA algorithm is to partition WASs into clusters such that WASs within the same cluster are more similar/closer to each other than to WASs in other clusters.

The CLEOPATRA algorithm consists of two major phases: node-based clustering and subtree-based clustering.

Phase 1: Node-based Clustering

The objective of this phase is to categorize individual nodes, which represent WASs from the root to the current nodes, with similar evolutionary patterns in the H-WAS tree into clusters. Hereafter, clustering individual nodes refer to clustering WASs that starts from the root and ends at the corresponding leaf nodes. Algorithm 13 consists of two phases, a two-level clustering phase and an iterative refinement phase. In the first phase, given an H-WAS tree, firstly, it is clustered based on the global dynamic associated with the individual nodes. That is, individual nodes the support counts of which changed with similar frequency are grouped into the same cluster. Then, using the evolutionary pattern-based distance, the global dynamic based clustering results are further partitioned into smaller clusters. In the second phase, the iterative refinement phase, the merging and splitting algorithms are used to refine the quality of the clustering results. The reason is that in the first phase, the two metrics global dynamic and evolutionary pattern-based distance are used separately, when the merging and splitting operations converge, the results will be more accurate.

Note that we use the DBSCAN algorithm [63] to cluster the individual nodes in the H-WAS tree in this phase for the following reasons. First, the DBSCAN algorithm needs no
Algorithm 13 Node-based Clustering Algorithm

Input: H-WAS tree: \( H \)
Output: a set of clusters \( C \)
1: \( C' = \text{DBSCAN}(H, \omega(A)) \)
2: for all Node pairs \((N_i, N_j)\) in cluster \( c'_i \in C' \) do
3: calculate \( D(N_i, N_j) \)
4: end for
5: \( C = \text{DBSCAN}(c'_i, D), \forall c'_i \in C' \)
6: for Stop = False do
7: \( C' = \text{Split}(C) \)
8: \( C' = \text{Merge}(C') \)
9: end for
10: Return \( (C) \)

Algorithm 14 Merging operation

Input: A set of clusters \( C \), distance threshold \( \epsilon \) for DBSCAN
Output: Refined clusters \( C' \)
1: for cluster \( C_j \in C \) do
2: calculate the centroid point \( C(C_j) \)
3: end for
4: for all \( C_j, C_k \in C \ & \ C(C_j) \neq C(C_k) \) do
5: if \( D(C(C_j), C(C_k)) < 2 \ast \epsilon \) then
6: merge them into a new cluster
7: calculate the new centroid point
8: end if
9: end for
10: Return clusters \( C' \)

prior knowledge about the number of clusters in the data collection. This is an advantage of the density-based clustering algorithms. Secondly, the naive DBSCAN approach has the time complexity of \( O(N \log N) \), where \( N \) is the total number of points in the database, using spatial indexing techniques. Moreover, the DBSCAN algorithm is able to discover clusters with arbitrary shapes and is efficient for very large database. Notice that here the distances between nodes in the H-WAS tree are the Euclidean distances calculated based on the smoothed \( x(A) \) sequence generated using the moving average.

In the first phase, the reason for designing a two-level clustering algorithm is to avoid computational cost. In the first level, the global dynamic values are used for producing a preliminary results as the global dynamic values are easier to obtain while the cost for calculating the evolutionary pattern-based distances are relatively more expensive. By doing
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

this, the computational cost for calculating the evolutionary pattern-based distances for nodes that are not expected to be in the same cluster can be reduced.

In the second phase, the merging and splitting operations are proposed to refine the clustering results in the first phase. The intuition behind is that it is possible that the first level of global dynamic based clustering results may not fully reflect the evolution pattern-based distances between the nodes. Using this iterative merging and splitting operations, which will converge to certain results, we can guarantee that node-based clustering results are accurate, which is the foundation for the sub-tree based clustering in the next phase.

Specifically, merging operation is shown in Algorithm 14. Firstly, for each cluster a virtual centroid is obtained. Then, the distances between those centroids are calculated using the proposed evolutionary pattern-based distance measure. For clusters whose centroids are within a distance of $2\epsilon$ will be merged together to form a new cluster, where $\epsilon$ is the radius parameter for the DBSCAN algorithm [63]. After that, the splitting operation is then performed on the new clustering results to split them into new clusters if possible. This splitting process is based on the DBSCAN algorithm as well. The distance used in the splitting operation is the evolutionary pattern-based distance. Note that, the splitting and merging operations iterate till the clustering results converge or reach certain user-defined stop criteria.

Phase 2: Subtree-based Clustering

The output of the node-based clustering phase is a set of clusters that consist of sets of individual nodes with similar change patterns. However, given a cluster of individual nodes that have similar change patterns cannot explain the reasons underlying those patterns. In this section, the individual nodes within clusters are merged together to form subtrees, which can represent higher level concepts or objects and make it easier to reason the factors behind the evolution. Note that, the subtree construction process is guided by not only the implicit links hidden behind the Web usage data, which is the structure of the H-WAS tree, but also the evolutionary pattern-based links within clusters.
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

Definition 5.36 Evolutionary Degree: Let $C = \text{NodeClust}(H)$ be a function that implements the node-based clustering phase where $H$ is the $H$-WAS tree and $C$ is the set of clusters returned by the function. Let $B(i,j) = \text{Edge}(n_i, n_j)$ be a function that takes in two nodes $n_i$ and $n_j$ and returns 1 if there exists or 0 if there does not exist an edge $(n_i,n_j)$ in $H$. Let $C_x = \{n_1, n_2, \ldots, n_{|C_x|}\}$ and $C_x \subseteq C$. Then, the evolutionary degree of $n_i \in C_x$ (denoted as $\mathcal{E}(n_i)$) is defined as follows:

$$\mathcal{E}(n_i) = \sum_{j=1}^{|C_x|} B(i,j), \text{ where } i \neq j \text{ and } 0 < j < |C_x|$$

From the above definition, it can be observed that nodes that have large evolutionary degree are expected to form large subtrees. In this section, we propose to extract the list of subtrees for each cluster. Firstly, nodes in each cluster are ranked based on the evolutionary degree in descending order. Then, to make sure that WASs in the same subtree have similar evolutionary patterns with each other, we propose a metric, called intra similarity, based on which the cluster subtree is defined.

Definition 5.37 Intra Similarity: Let $C = \text{NodeClust}(H)$ and $C = \{C_1, C_2, \ldots, C_n\}$. Let $t_j$ be a subtree of $H$ and $N_t$ be the set of nodes in $t_j$. Let $K = \{K_1, K_2, \ldots, K_i\}$, where $K_r = |N_t \cap C_r| \forall \ 0 \leq r \leq i$ and $r \leq n$. Then, the intra similarity of $t_j$, denoted as $\text{IS}(t_j)$, is defined as: $\text{Max}(K) / |N_t|$, where $\text{Max}(K)$ is the maximum value in $K$.

Definition 5.38 Cluster Subtree: Let $t_j = (N_j, A_j)$ be a subtree of $H$ such that $N_j \subseteq C_x$ and $C_x \in C$ where $C = \text{NodeClust}(H)$. Then $t_j$ is a cluster subtree if $\text{IS}(t_j) \geq \beta$ where $\beta$ is a user-defined threshold.

The algorithm for extracting subtree clusters is presented in Algorithm 15. The input of the subtree-based clustering algorithm is a set of clusters with sorted nodes. Firstly, the node with maximum evolutionary degree is selected and the corresponding subtree that
includes all the nodes that are connected to that node is constructed and tested against the threshold value of $T_S$. If this subtree is a cluster subtree, then all the nodes in this subtree are eliminated from the list of subtrees in that cluster. Otherwise, if this subtree is not a cluster subtree, then the evolutionary degree of this node is set to -1. This process iterates till all the nodes in the subtree are tested.

**Algorithm 15 Subtree-based Clustering**

**Input:** Clusters with sorted nodes $C$, $T_S$ threshold $\beta$

**Output:** Clusters of subtrees $CoS$

1: for all cluster $C_j \in C$ do
2: for all node $n_x$ with the largest $\bar{E}(n_x)$ where $\bar{E}(n_x) > 0$ do
3: prune all the leaf nodes that are in different cluster with $n_x$ iteratively
4: calculate the $T_S$ of the subtree rooted at $n_x$
5: if $T_S (Tree(n_x)) \geq \beta$ then
6: insert this subtree into the CoS list
7: prune all the leaf nodes in this subtree from this cluster
8: else
9: $\bar{E}(n_x) = -1$
10: end if
11: end for
12: end for
13: Return($CoS$)

### 5.4.3 Experimental Results

Our experiments focus on two aspects: the quality and novelty of the clustering results. To evaluate the quality of the our clustering results, two quality metrics, *Homogeneity* and *Separation* [144, 143], are used. Note that the class information in the datasets is not available, thus we cannot use quality metrics that require prior knowledge about class labels. *Homogeneity* and *separation* are metrics used to measure the compactness of the cluster. The average homogeneity ($H_{avg}$) and minimum homogeneity ($H_{min}$) are the average similarity and minimum similarity between all the elements in the cluster, respectively. The average separation ($S_{avg}$) and maximum separation ($S_{max}$) are defined as the average overall similarity and the maximum overall similarity between clusters, respectively. Formally, they are defined as follows.
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

<table>
<thead>
<tr>
<th>Dataset</th>
<th>(\epsilon)</th>
<th>(k)</th>
<th>(\beta)</th>
<th>(H_{avg})</th>
<th>(H_{min})</th>
<th>(S_{avg})</th>
<th>(S_{max})</th>
<th># clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>UoS</td>
<td>0.05</td>
<td>30</td>
<td>0.8</td>
<td>0.81</td>
<td>0.16</td>
<td>0.21</td>
<td>0.46</td>
<td>46</td>
</tr>
<tr>
<td>UoS</td>
<td>0.10</td>
<td>60</td>
<td>0.8</td>
<td>0.79</td>
<td>0.13</td>
<td>0.23</td>
<td>0.51</td>
<td>38</td>
</tr>
<tr>
<td>UoS</td>
<td>0.15</td>
<td>60</td>
<td>0.75</td>
<td>0.78</td>
<td>0.17</td>
<td>0.19</td>
<td>0.48</td>
<td>34</td>
</tr>
<tr>
<td>UoS</td>
<td>0.20</td>
<td>90</td>
<td>0.75</td>
<td>0.78</td>
<td>0.14</td>
<td>0.20</td>
<td>0.46</td>
<td>36</td>
</tr>
<tr>
<td>Calgary</td>
<td>0.05</td>
<td>30</td>
<td>0.8</td>
<td>0.80</td>
<td>0.14</td>
<td>0.23</td>
<td>0.45</td>
<td>71</td>
</tr>
<tr>
<td>Calgary</td>
<td>0.10</td>
<td>60</td>
<td>0.8</td>
<td>0.79</td>
<td>0.15</td>
<td>0.16</td>
<td>0.38</td>
<td>68</td>
</tr>
<tr>
<td>Calgary</td>
<td>0.15</td>
<td>60</td>
<td>0.75</td>
<td>0.71</td>
<td>0.13</td>
<td>0.17</td>
<td>0.38</td>
<td>63</td>
</tr>
<tr>
<td>Calgary</td>
<td>0.20</td>
<td>90</td>
<td>0.75</td>
<td>0.75</td>
<td>0.06</td>
<td>0.13</td>
<td>0.32</td>
<td>62</td>
</tr>
</tbody>
</table>

Table 5.4: CLEOPATRA experimental results using DBSCAN.

\[
H_{avg} = \frac{1}{M} \sum_{1<j, C(A_i)=C(A_j)} S(A_i, A_j) \quad (Eq. 5.1)
\]
\[
H_{min} = \min_{C \in C} \frac{2 \cdot \sum_{i<j \in C'} S(A_i, A_j)}{|C'| \cdot (|C'| - 1)} \quad (Eq. 5.2)
\]
\[
S_{avg} = \frac{2}{n(n-1) - 2M} \sum_{1<j, C(A_i) \neq C(A_j)} S(A_i, A_j) \quad (Eq. 5.3)
\]
\[
S_{max} = \max_{C, C' \in C} \sum_{A_i \in C, A_j \in C'} S(A_i, A_j) |C| \cdot |C'| \quad (Eq. 5.4)
\]

where \(n\) is the total number of WAS subtrees; \(A_i\) is the \(i\)th WAS subtree; \(M\) is the total number of node pairs that are within the same cluster; \(C\) is the set of clusters in the result and \(|C|\) is the size of the set; \(C(A_i)\) is the cluster to which \(A_i\) belongs. Note that, here we transform the evolutionary pattern-based distance to the similarity measure \(S\) such that we can use the above cluster quality metrics. That is, \(S(A_i, A_j) = e^{-D(A_i, A_j)}\). The larger homogeneity implies a better result, while a larger separation shows a worse result.

Table 5.4 shows the quality of the clustering results with different parameters for the DBSCAN algorithm, size of moving window in the moving average, and the intra similarity threshold. The reason of using the above cluster quality metrics is that due to privacy the original URLs of Web pages in the Web usage dataset are not available. Therefore, the ground truth of the clusters is not available. However, from the numbers in Table 5.4, compared with the corresponding values in other applications that using above quality metrics, the quality of our results is comparable to the state-of-the-art clustering results in [144, 143].

To demonstrate the robustness of the proposed similarity/distance measure, the graph cut algorithm [147] is used as an alternative for the DBSCAN algorithm. The basic idea of the
5.4.3 Experimental Results

Table 5.5: CLEOPATRA experimental results using Graph Cut.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$H_{avg}$</th>
<th>$H_{min}$</th>
<th>$S_{avg}$</th>
<th>$S_{max}$</th>
<th>$#$ clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>UoS</td>
<td>0.83</td>
<td>0.14</td>
<td>0.20</td>
<td>0.42</td>
<td>46</td>
</tr>
<tr>
<td>UoS</td>
<td>0.82</td>
<td>0.15</td>
<td>0.21</td>
<td>0.44</td>
<td>38</td>
</tr>
<tr>
<td>UoS</td>
<td>0.81</td>
<td>0.16</td>
<td>0.13</td>
<td>0.46</td>
<td>34</td>
</tr>
<tr>
<td>UoS</td>
<td>0.82</td>
<td>0.12</td>
<td>0.17</td>
<td>0.43</td>
<td>36</td>
</tr>
<tr>
<td>Calgary</td>
<td>0.81</td>
<td>0.13</td>
<td>0.15</td>
<td>0.34</td>
<td>68</td>
</tr>
<tr>
<td>Calgary</td>
<td>0.78</td>
<td>0.04</td>
<td>0.12</td>
<td>0.30</td>
<td>62</td>
</tr>
</tbody>
</table>

The graph cut algorithm is to partition the graph representation of data points into subgraphs by cutting these edges with the smallest similarity that connects these subgraphs. The corresponding clustering results are shown in Table 5.5. It can be observed that the quality of the clustering results are better than the DBSCAN approach. That is, our similarity measure is robust to both density based and hierarchical clustering algorithms.

Considering the novelty of our clustering results, although there are no quantified measures, we have the following observations from the DBSCAN and graph cut based clustering results.

- In the CLEOPATRA clustering results, we found many WAS pairs that are in the same cluster are very far away in the $H$-WAS-tree while the evolutionary patterns are quite similar. Such clustering results can be useful for exploring the hidden factors that lead to the evolution of the corresponding WASs.

- The overall structures of the clusters are quite similar in the CLEOPATRA clustering results. Suppose there are two clusters $C_1$ and $C_2$, where $C_1 = \{A_1, A_2, A_3\}$ and $C_2 = \{A_4, A_5, A_6\}$. It can be observed that Web pages in the same cluster such as $A_1$, $A_2$, and $A_3$ may not be siblings or connected, but Web page pairs in two clusters such as $\{A_1, A_4\}$, $\{A_2, A_5\}$, and $\{A_3, A_6\}$ are siblings or connected. That is the clusters are similar to instances of the same schema.

5.4.4 Summary

In this section, we proposed the first approach of clustering historical WASs based on the evolutionary patterns. That is, Web access sequences that have similar evolution patterns in
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

terms of their support values in the history are clustered into the same group. The intuition behind is that Web access sequences are task/event-driven, similar changes patterns in their support values indicate their semantic similarity. Experiments with real life datasets show CLEOPATRA can efficiently produce high quality clusters that cannot be discovered using existing web usage mining techniques.

5.5 Integrated Website Event Detection: iWED

In this section, we present the Website-based event detection application to illustrate the usefulness of the evolution pattern-based clustering and the feasibility of extracting semantics from the evolution pattern as well. In this event detection approach, we integrate both visitor-centric data and author-centric data to distinguish semantically very similar events that are difficult to be separated using existing approaches. The key features of this approach is that the visitor-centric data evolves with the occurrence of real world events. Based on this assumption, we believe that similar events can be distinguished by integrating and analyzing the evolution of visitor-centric data.

For example, suppose Figure 5.6(a) shows a subset of hyperlinked Web pages; Figure 5.6(b) shows the implicit links extracted from the corresponding usage data; and Figure 5.6(c) shows the evolution pattern of Web usage data (the y-axis shows the frequency of a Web page being accessed over the time intervals shown in the x-axis). Here, there is an implicit link between two Web pages if and only if they were accessed consecutively in the
Web access sequences [176]. The evolution pattern of Web usage data refers to how the Web pages changed in the history in terms of their supports [206].

It can be observed that from only Figure 5.6(a), it is difficult to distinguish sibling pages such as e and f even if they correspond to different events. However, with the evolution of Web usage data as shown in Figure 5.6(c), connected Web pages with similar content but corresponding to different events can be distinguished. For example, in Figure 5.6(c), pages e and g have similar evolution pattern while pages e and f have different evolution pattern. At the same time, Web pages that are not connected by hyperlinks but corresponding to the same event can be identified using implicit links in Figure 5.6(b), since they are expected to be accessed together. As shown in Figure 5.6(b), the implicit link between Web pages b and g, which are not connected by hyperlink in Figure 5.6(a), implies that b and g have a possibility to represent the same event. Note that we focus on detecting events in a specific Website as it is extremely difficult to gather Web usage data of the entire Web.

5.5.1 Data Representation and Metrics

Note that in this approach, rather than model the historical Website data as tree structures, they are modelled as graphs. In this section, we first discuss how to represent Web structure, Web content, and Web usage data of a Web site using structure graph, content graph, and usage graph, respectively. Then, we present how these three types of graphs are integrated using a multigraph.

Structure Graph

The Web structure data here refers to the set of Web pages and hyperlinks between them. It can be modelled as a structure graph, $G_s = (V_s, E_s)$, where each vertex in $V_s$ is a Web page and each edge in $E_s$ represents the structure similarity (will be defined later) between the two pages that are connected by this edge. Note that the structure similarity reflects the similarity between Web pages in terms of structure. The intuition is "two Web pages are structurally similar if they are linked with similar Web pages" [87]. As the base case, we consider a Web page maximally similar to itself, to which we can assign a structure similarity.
score of I. Referring back to Figure 5.6(a), Web page e and f are similar because they are both linked to Web page b. Hence, the structure similarity is defined as follows.

Given two Web pages \( i \) and \( j \) in \( V_s \), the structure similarity between them is denoted as \( S_s(i, j) \), where

\[
S_s(i, j) = \frac{C}{|D(i)| \cdot |D(j)|} \sum_{m=1}^{|D(i)|} \sum_{n=1}^{|D(j)|} S_s(D_m(i), D_n(j))
\]

Here \( C \) is a constant between 0 and 1, \( |D(i)| \) is the degree of vertex \( i \) in the graph and \( D_m(i) \) is the \( m^{th} \) neighbor of vertex \( i \). It is obvious that this similarity is an iterative function where similarities between Web pages are propagated through recursions. That is, the value of \( S_s(i, j) \) in the \( t^{th} \) iteration, denoted as \( S_{st} \), is based on the values of the \( t-1^{th} \) iteration. Moreover, it has been proven that this recursive function is nondecreasing and it will converge eventually [87]. We initialize the recursions with \( S_{s0} \): if \( i = j \), then \( S_{s0}(i, j) = 1 \); otherwise \( S_{s0}(i, j) = 0 \).

Content Graph

The Web content data refers to the content of each Web page. The Web content data is modelled as a content graph, \( G_c = (V_c, E_c) \), where each vertex in \( V_c \) is a Web page and each edge in \( E_c \) represents the semantic similarity between two pages. It has been experimentally proven that cosine measure is one of the best measures for Web content clustering [152]. Hence, we use the cosine measure to quantify semantic similarity between two pages. Here the classic vector space model in the information retrieval area is used to represent each Web page. Given a Web page \( i \), using some stemming algorithm, it will be represented as a vector, \( \overline{X_i} \), which correspond to the TF.IDF of the keywords after stemming [152]. Then, the semantic similarity between two Web pages \( i \) and \( j \), denoted as \( S_c(i, j) \), is defined as follows.

\[
S_c(i, j) = \frac{(\overline{X_i} \cdot \overline{X_j})}{||\overline{X_i}|| \cdot ||\overline{X_j}||}
\]

where \((\overline{X_i} \cdot \overline{X_j})\) is the dot product of the two vectors and \( ||\overline{X_j}|| \) denote the length of vector \( \overline{X_j} \).
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

Usage Graph

The usage data refers to the access log of the Web pages. It also can be modelled as a graph, called usage graph, \( G_u = (V_u, E_u) \), where each vertex in \( V_u \) is a Web page and each edge in \( E_u \) represents the usage pattern-based similarity between two pages.

Given two Web pages, \( i \) and \( j \), with the user-defined calendar pattern, the corresponding support values are represented as ( \( \Phi_1(i), \Phi_2(i), \Phi_3(i), \ldots, \Phi_k(i) \) ) and ( \( \Phi_1(j), \Phi_2(j), \Phi_3(j), \ldots, \Phi_k(j) \) ). Then, the usage pattern-based similarity, denoted as \( S_u(i,j) \), is defined as follows.

\[
S_u(i,j) = \lambda \times e^{-D} + (1 - \lambda) \times \frac{\sum_{t=1}^{k}(\Phi_t((i,j)) + \Phi_t((j,i)))}{\sum_{t=1}^{k}(\Phi_t(i) \cup \Phi_t(j))}
\]

where \( D = \sqrt{\sum_{t=1}^{k}|\Phi_t(i) - \Phi_t(j)|^2} \)

Note that, the usage pattern-based similarity is a linear combination of the evolution pattern-based similarity and the implicit link-based similarity. The evolution pattern-based similarity is denoted as \( e^{-D} \), where \( D \) is the Euclidian distance between the support sequences \( H(i) \) and \( H(j) \). The implicit link-based similarity is represented as the percentage of \( \mathcal{WAS} \)s that contain \( i \) and \( j \) consecutively against the total number of \( \mathcal{WAS} \)s that contain at least one of \( i \) and \( j \). Here, \( \lambda \) and \( 1 - \lambda \) are the weights of evolution pattern-based similarity and the implicit link-based similarity. It is obvious that both the evolution pattern-based similarity and implicit link-based similarity are within the range between 0 and 1. Similarly, the usage pattern-based similarity is between 0 and 1.

Multigraph

We merge the above three graphs using a multigraph, which includes Web structure, Web content, and Web usage data in a Website. A multigraph is a graph whose edges are unordered pairs of vertices, and the same pair of vertexes can be connected by multiple edges.
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

In this case, there are three edges for each pair of vertexes. These three edges represent the edges of structure graph, content graph, and usage graph. Formally,

**Definition 5.39 Multigraph** A multigraph is represented as a 3-tuple $M = (V, E, f)$, where $V$ is a set vertexes, $E$ a set of edges, and $f$ is a function $f(e_i) = \{(u,v) \mid u, v \in V; u \neq v\}$ that takes an edge $e_i \in E$ and returns the set of Web pages $u$ and $v$ that are connected by $e_i$. Two edges $e_i$ and $e_j$ are called parallel or multiple edges if $f(e_i) = f(e_j)$.

An example of the multigraph representation of Website data is shown in Figure 5.7 with the corresponding structure graph, content graph, and usage graph. Note that, the similarities between disconnected Web pages are 0 and the weights of the edges represent the corresponding similarity values.

5.5.2 iWED

In this section, we first define the Website-based event detection problem. Then, the iWED algorithm is presented.

**Website-based Event Detection Problem**

Based on the multigraph representation of the Website related data, each real world event corresponds to a strongly connected subgraph in the multigraph. That is, a real world event can be represented as a set of structurally and semantically strongly connected Web pages with similar usage patterns in the multigraph. The Website based event detection problem is to extract such subgraphs from the multigraph representation.

**Definition 5.40 Website-based Event Detection** Let $M$ be the multigraph representation of the Website related data. The problem of Website based event detection is to extract
CHAPTER 5. MINING EVOLUTION OF WEBSITE LOG DATA

A set of subgraphs \{ M_1, M_2, \ldots, M_k \} from multigraph M, where Web pages within any subgraph \( M_i \) \( (1 \leq i \leq k) \) are more strongly connected to each other than to Web pages in other subgraphs, in terms of structure similarity, content similarity, and usage pattern-based similarity. Each subgraph is then expected to represent a real-world event.

iWED Algorithm

In this section, we present the iWED event detection algorithms based on the multigraph representation of the Website data. To extract the strongly connected subgraphs from a graph, different graph cut algorithms have been proposed. We adopt the normalized graph cut algorithm, which is widely used in object extraction from image data and frame segmentation of video data [147]. We begin by briefly discussing the graph cut algorithm, which is the foundation of our proposed algorithm.

Graph Cut Algorithm

Given a graph \( G = (V, E) \), where the nodes of the graph are points in the feature space, and an edge is formed between every pair of vertices. The weight of each edge, \( w(i, j) \) represents the similarity between vertex \( i \) and vertex \( j \). In graph theory, graph cut is to partition the set of vertices into disjoint sets \( V_1, V_2, \ldots, V_m \), where by certain measure the similarity among vertices within \( V_i \) is high and across different sets \( V_i, V_j \) is low. As presented in [148], the problem of graph cut is decomposed into a sequence of recursive bi-partition processes and there are different measures for the bi-partition process. We adopted the normalized graph cut [147], the complexity of the algorithm is \( O(VE^2) \), where \( V \) and \( E \) are the number of vertices and edges.

A graph \( G = (V, E) \) can be partitioned into two disjoint sets \( A \) and \( B \), where \( A \cup B = V \) and \( A \cap B = \emptyset \) by removing edges connect the two sets. The similarity between the two sets can be computed as the total weights of the edges have been removed. In graph theoretic language, it is called cut and is defined as follows.

\[
\text{cut}(A, B) = \sum_{u \in A, t \in B} w(u, t).
\]
To avoid the unnatural bias for partitioning out small sets of points, the normalized graph cut was proposed [147]. Instead of looking at the value of total edge weight connecting the two partitions, normalized graph cut computes the cut cost as a fraction of the total edge connections to all the nodes in the graph. The disassociation measure of normalized cut, denoted as $Ncut$, is defined as follows.

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

where $assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$ is the total connections from vertex $A$ to all vertexes in the graph and $assoc(B, V)$ is similarly defined. Thus, the problem is to minimize the $Ncut$ value, which is NP-complete. Shi et al. proposed to solve this problem by using the eigenvectors [147]. As there are more than two partitions in our problem, we adopt the recursive 2-way cut algorithm, which works as follows. Note that the number of groups segmented by this method is controlled directly by the maximum allowed $Ncut$.

1. Given a set of features, set up a weighted graph $G = (V, E)$, compute the weight on each edge, and summarize the information into $W$ and $D$.
2. Solve $(D - W)x = \lambda Dx$ for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph by finding the splitting point such that $Ncut$ is minimized.
4. Decide if the current partition should be subdivided by checking the stability of the cut, and ensuring $Ncut$ is below the pre-specified value.
5. Recursively repartition the segmented parts if necessary.

**iWED Multigraph Cut Algorithm**

The three similarity measures, $S_s$, $S_c$, and $S_u$, can be classified into two categories: topic similarity and evolution similarity. Topic similarity is the combination of the structure similarity ($S_s$) and the semantic similarity ($S_c$), while evolution similarity is the usage pattern-based similarity ($S_u$). Based on these two categories, we propose two variants of iWED algorithms.
for cutting the multigraph. The first approach, called the fusion approach, fuses the two types of similarity measures together and cut the graph by treating the multiedges between two vertexes as a single edge. The second approach, called the level-wise approach, cuts the graph with the topic and evolution similarity measure separately. We now elaborate on these two approaches.

**Fusion Approach:** The fusion approach, denoted as \( FUS \), integrates the three similarity measures together using linear combination with different weights. Such kind of fusion has been extensively used in combining different types of similarity measures in Web content analysis [87]. In the fusion approach, a new similarity \( S \) is proposed as follows.

\[
S = \alpha S_s + \beta S_c + \gamma S_u
\]

where \( \alpha, \beta, \gamma \) are the weights for the corresponding similarity measure, and \( \alpha + \beta + \gamma = 1 \). Then, the multigraph is transformed to a normal graph, where the weight of each edge is represented by \( S \). The graph is then cut using the normalized graph cut algorithm.

**Level-wise Approach:** In the level-wise approach, the topic similarity and the evolution similarity are used to cut the multigraph separately. Note that, the topic similarity, denoted as \( S_T \), defined as the fusion of structure similarity and semantic similarity. There are two alternative level-wise approaches. In the first approach, denoted as \( LTF \) (Level-wise Topic First), the multigraph is cut based on the topic similarity, which corresponds to only two types of edges in the multigraph, and the result, \( C_T \), is returned. Then, each subgraph in \( C_T \) is cut again based on the evolution similarity and the final result, \( C_F \), is returned. In the second approach, denoted as \( LEF \) (Level-wise Evolution First), the multigraph is first cut based on the evolution similarity and the result, \( C_E \), is returned. Then, each subgraph in \( C_E \) is cut again using the topic similarity and the result \( C_F \) is returned. The underlying intuition is that, in the first approach, Web pages are clustered into semantic topics before they are clustered into events as each event is expected to be a set of semantically similar Web pages that have similar usage patterns. In the second approach, firstly Web pages that
correspond to similar types of events are gathered together and then clustered based on their semantic relationships.

For both the fusion approach and the level-wise approach, we present the clustering results with a hierarchical structure. That is, at the first recursion of the 2-way graph cut algorithm, there are two partitions. After that each partition is further cut into two child partitions and so on. However, not all the subgraphs correspond to real world events. To identify real world events and exclude outliers, we propose an intra-cluster similarity measure, $S_{\text{intra}}(G')$, for any subgraph $G'$.

$$S_{\text{intra}}(G') = \frac{2 \sum_{i,j \in G'} S(i,j)}{|G'| \times (|G'| - 1)} \text{ where } i \neq j \text{ and } i, j \in G'$$

Based on this similarity measure, a threshold $\tau$ in the range of $[0, 1]$, is proposed to distinguish the event-based subgraph and the non-event-based subgraph. Given a subgraph, $G'$, in the graph cut results, it corresponds to a real world event if and only if $S_{\text{intra}}(G') \geq \tau$.

### 5.5.3 Performance Evaluation

In this section, the experimental results are presented to show the performance of our proposed event detection approaches. The three approaches, $FUS$, $LTF$, and $LEF$, are implemented and compared to the baseline approach, $Bl$, which only takes the structure and content of Web pages using the corresponding similarity measures.

#### Datasets

In our experiments, a synthetic e-commerce Website dataset is used. Even though there are some real Web usage datasets available, but due to privacy issue the original URLs and Web pages are not available and cannot be used in our experiments. The synthetic dataset we generated consists of 3000 products and 20000 unique Web pages. The 3000 products belong to 50 categories, where the content of the Web pages are generated according the attributes of products in different categories (we use the schema extracted from http://www.bargaincity.com.sg, which is the one of the biggest e-commerce Websites in Singapore). The usage data are generated in three steps. Firstly, the Web access sequences...
are generated using uniform random generation. Then, we synthesized a list of 200 events (40 burst events such as one day only promotion and release of new products, 80 periodic events such as weekend promotion and new semester promotion, 40 increasing events such as price of a popular product keeps decreasing, 40 decreasing events such as some products are fading out of the market). For example, access patterns of Web pages that are related to promotions, releasing of new products, and periodic products, are generated in such a way that they are visited more frequently for specific time periods. Note that, some of the events may have overlaps in time, the same periodic pattern, or similar burst patterns. Lastly, certain percentage of noise access sequences are randomly inserted into the Web usage data to simulate the real life usage data. In our following experiment, if not specified, 5% of noise are inserted as it is the average percentage of noise in the Web usage data collection according to the study in [156]. In total, there are 20,000,000 unique page requests in the synthetic Web usage data, which are partitioned into 200 access groups.

Evaluation Measure

As the event detection results are set of events, which consist of sets of Web pages, it is different from existing classification algorithms. Although, we have the set of labelled events with corresponding Web pages, the precision and recall measures in our event detection approach are different for the following reasons. Since an event consists of many Web pages, the event may be detected but the corresponding Web pages may not be accurate. That is, some pages may be missed and some non-related pages may be included. For example, given a real world event $E = \{P_1, P_2, P_3, P_4, P_5\}$, there may be one corresponding event $E' = \{P_1, P_3, P_4, P_7, P_8\}$ in the detection results. However, for one real world event, it is possible that there may be more than two corresponding events in the results. For example, given a real world event $E = \{P_1, P_2, P_3, P_4, P_5\}$, there may be two corresponding events $E' = \{P_1, P_3, P_4, P_7, P_8\}$ and $E'' = \{P_2, P_5, P_9\}$ in the detection results. For example, given a promotion, in the first few days some popular products may attract a lot of attention, while in the last few days other products that are not so popular also attract a lot of attention.
As a result, it is possible that the two sets of products are split into two events. We propose precision/recall measure for event detection based on the commonly-used precision/recall from IR.

Let \( \mathcal{E} = \{E_1, E_2, \cdots, E_n\} \) be the set of detected events based on our proposed approach and \( \mathcal{E}' = \{E'_1, E'_2, \cdots, E'_m\} \) be the set of labelled events in the dataset, where each event \( E_i \) consists of a set of Web pages \( \{P_{i1}, P_{i2}, \cdots, P_{ik}\} \). For each \( E_i \), the corresponding real event \( E'_j \) with the largest value of \( |E_i \cap E'_j| \) is selected, where \( |E_i| \) is the number of pages included in that event while \( |E_i \cap E'_j| \) is the number of common pages included in both \( E_i \) and \( E'_j \). Also, for each real world event \( E'_j \), the corresponding event \( E_i \) with the largest value of \( |E_i \cap E'_j| \) is selected from the results. Moreover, for different events in the real world, their corresponding events in the results should be different and vice versa. Then, the precision and recall are defined as follows.

\[
Pr = \frac{\sum_i |E_i \cap E'_i|}{|\mathcal{E}|}, \quad Re = \frac{\sum_j |E'_j \cap E_j|}{|\mathcal{E}'|}
\]

**Experimental Results**

Two sets of experiments have been conducted to evaluate our proposed event detection approaches. Firstly, comparison of our proposed event detection approaches with the baseline approach is presented. Secondly, we show the effects of intra-similarity threshold \( \tau \) on the quality of the detected events. Within each set of results, both the overall performance and the performance for each type of events are presented. Lastly, we discuss about how to set the fusion parameters in the \( FUS \) approach. Note that, the \( \lambda \) value in the usage pattern-based similarity is set to 0.5 for the following experiments.

**Quality of Results** Table 5.6(a) shows the performance of the four approaches with the precision, recall, and \( F_1 \) measure\(^3\). It can be observed that the \( LEF, FUS, \) and \( LTF \) approaches outperform the baseline approach, \( Bl \), which shows the improvement of integrating the usage data and their evolution patterns. Among our proposed approaches, the \( LEF \)

---

\(^3\)The \( F_1 \) measure is computed as \( F_1 = \frac{2 \cdot Pr \cdot Re}{Pr + Re} \)
and \( FUS \) archive better performances than the \( LTF \) approach. This is because some of the synthetic events in our dataset usually cover more than one semantic topic. For instance, for a Christmas Promotion, most products from all categories are on the list and their access patterns change in a similar way (a burst of visits during the same time interval). As a result, if we first use the topic similarity to cut the graph, in the \( LTF \) approach, some events may be cut into sets of smaller events. However, the \( FUS \) and \( LEF \) approaches can discover them as a single event. Tables 5.6(b), (c), and (d) show the performance of our approaches with respect to different types of events.

**Effects of Parameters** In the above experiments, weights of the three similarity measures are set to 0.30, 0.20, and 0.50, which are experimentally proved to be the optimal values for our dataset. The threshold for intra-cluster similarity is set to 0.6. Next, experiments are conducted to show the effects of intra-cluster similarity threshold, \( \tau \). Tables 5.6(e) and (f) show the quality of the event detection results of the \( FUS \) and \( LEF \) approaches by varying
the corresponding \( \tau \) values. The results are for all types of events. Observe that the effects of threshold \( \tau \) are similar for the three types of events. When the value of \( \tau \) increases from 0.3 to 0.7, the quality of the event detection results becomes better; when the value of \( \tau \) increases from 0.7 to 0.9, the quality of the event detection results becomes worse. This is because when the threshold for intra-cluster similarity is too small/large, the number of events detected may be too many/few. While the number of real world event is fixed, the performance of the approaches decreases when the threshold is close to the two extremes.

Table 5.6(g) shows how the FUS algorithm performances when \( \alpha, \beta, \) and \( \gamma \) are set to the best values, while the percentage of noise changes from 3\% to 7\%. It can be observed that when the percentage of noise increases, both the precision and recall decrease, and so does the \( F_1 \) value. The reason is the clustering problem is becoming more difficult. Tables 5.6(h) and (i) show how the FUS algorithm performances when different weight schemes are used. It can be observed that the usage pattern-based similarity is more important than the other two similarities.

Generally, from the results shown in Table 5.6, it is evident that the FUS approach performs relatively better than other approaches in most cases. This is because, in the FUS approach, the weights of different types of similarities can be tuned. In our experiments, we show the average results of the FUS approach. During tuning the parameters, we have following observations.

- The usage pattern-based similarity significantly improves the clustering results. For instance, even in the LTF and LEF approach, the clustering results are much better than the baseline approach in all cases.

- As for the dataset used in this experiment, we can conclude that the structure similarity is less important than the usage pattern-based similarity but more important than the content similarity. As shown in the above experiments, the best values for the parameters in the fusion approach are: \( 0.3 \leq \alpha \leq 0.4, \ 0.1 \leq \beta \leq 0.3, \) and \( 0.4 \leq \gamma \leq 0.5. \)
In summary, this section is motivated by the fact that existing event and object detection approaches only analyze the content and structure data of a website. Specifically, we integrate the author-centric and visitor-centric data to detect real-world events. Experimental results show that our proposed approaches can produce promising results.

5.6 Summary

Motivated by the fact that existing Web usage mining techniques ignored the dynamic nature of Web usage data, in this chapter, we propose to mining novel knowledge from the evolution of Web usage data. By taking into account the temporal information, firstly, two types of novel knowledge are discovered in this chapter. They are Web access motif (Web access sequences whose support values are relatively stale over time) and Web access clusters (sets of Web access sequences that have similar evolution patterns in terms of their historical support values). Moreover, by integrating both the author-centric and visitor-centric data, we propose to detect events from Website related data such as Website structures, Website content, and the evolution of Website usage data. Experimental results with real and synthetic datasets showed that our proposed approaches can produce different types of novel knowledge such as WAMs, WASs clusters, and events that cannot be discovered using existing techniques or their variants. More importantly, our proposed approaches are efficient and scalable as well.
Chapter 6

Building Time-Dependent Semantic Query Similarity Model

Different from the previous chapters, in this chapter, we focus on monitoring the evolution of graph structured data. Specifically, we propose to build a time-dependent semantic query similarity model from the evolution of Web search engine click-through data, which is also called search engine query log data. Here semantic query similarity refers to the similarities between query terms according to the similarities between corresponding relevant documents [56]. The intuition behind is that a more accurate semantic query similarity model can be constructed by incorporating the temporal information associated with historical queries being issued and corresponding Web pages being clicked. Note that such temporal information are ignored by existing query similarity models [56, 154, 172, 177].

The rest of this chapter is organized as follows. Firstly, we present the background of Web search engine log data such as the data source and existing click-through data analysis approaches in Section 6.1. Then, the motivation of building time-dependent semantic query similarity is presented by highlighting limitations of existing works in Section 6.2. Section 6.3 presents the time-dependent semantic query similarity model using marginalized kernel. Section 6.4 shows the empirical studies of the time-dependent query similarity model with real data collection obtained from MSN search engine. Section 6.5 concludes this chapter. A preliminary version of this chapter is to appear in [212].
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

Table 6.1: Example of the click-through data.

<table>
<thead>
<tr>
<th>IP address</th>
<th>Query</th>
<th>Page</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX.XXX.XXX</td>
<td>BMW</td>
<td><a href="http://www.bmw.com">http://www.bmw.com</a></td>
<td>14:02 02112005</td>
</tr>
<tr>
<td>XXX.XXX.XXX</td>
<td>SVM</td>
<td><a href="http://svm.first.gmd.de">http://svm.first.gmd.de</a></td>
<td>15:31 03252005</td>
</tr>
<tr>
<td>XXX.XXX.XXX</td>
<td>MSN</td>
<td><a href="http://www.MSN.com">http://www.MSN.com</a></td>
<td>21:14 02142005</td>
</tr>
</tbody>
</table>

Figure 6.1: Bipartite graph representation of click-through data.

6.1 Background

In this section, we first introduce the Web search engine click-through data with examples and summarize different representation methods for the click-through data in the literature. Then, existing works on analyzing click-through data will be reviewed. Specifically, we focus on research in the area of query expansion using click-through data.

6.1.1 Click-Through Data

Click-through data, representing query log of Web search engines, keep the records of interactions between Web users and the searching engines. Similar to the transaction data in the supermarket, the click-through data consist of a sequence of users sessions in some format like <IP address, query, clicked page, time>. Table 6.1 shows two example user sessions. Note that due to privacy issue, we do not provide the exact IP address. Recently, more and more attributes in the user sessions such as user profile and rank of the clicked web page are stored in the click-through data for further analysis. As a result, the Web search engine click-through data has become a rich source for analyzing users’ searching behaviors for personalizing and improving current search engines.

In the literature, there are two different ways to represent the click-through data. One
way is to represent the click-through data as session databases where each session represents a pair of a query and a page that the user issued and clicked (hereafter, we call such pairs as *query-page pairs*) as shown in Table 6.1 [56, 172]. Recently, some variants of this representation have been proposed by taking into account the rank of the page and the ID of the corresponding users [90, 154]. The other representation method is to use a bipartite graph, like the example in Figure 6.1, where the queries and pages are represented as two sets of nodes and the query-page pair co-occurrence relationships are represented as the edges between the corresponding nodes [22, 177].

### 6.1.2 Existing Click-through Data Analyzing Approaches

With the popularity of Web search engines, a huge amount of click-through data has been accumulated and available for analysis. Recently, a large body of literature has focused on mining the click-through data for query expansion [56], query and page clustering [172], ranking optimization [90], metadata generation [177], etc. In [172], the click-through data is used to improve the content-based query clustering for the application of FAQ identification. In [90], the author proposed to optimize the ranking function of search engines by using the click-through data. The basic idea is to training a ranking function using the SVM with the click-through data. More recently, in [177], Guo et al. proposed an iterative reinforced algorithm to utilize the user click-through data to improve search performance. The algorithm can successfully generate extra metadata for Web page by exploring the interrelations between queries and Web pages.

It has been observed that the real dilemma for most of the existing Web search engines is that users request for accurate search results while they only provide queries of limited length, which is usually less than two on average according to [172]. Recently, a lot of work has been done in the Web search community to expand the query terms with similar keywords for refining the search results [22, 56, 172, 175, 177]. The basic idea is to use the click-through data, which record the interactions between users and a search engine, as the feedback to learn the similarity between the query keywords. The existing work
can be generally classified into two categories: document term based query expansion and query term based query expansion. The document term based query expansion, which was firstly introduced in [56] by Cui et al., is to measure similarity between search queries and document terms based on the query log data of search engines. The basic idea is that if a set of documents is often selected for the same queries, then the terms in these documents are strongly related to the queries. Thus the probabilistic similarity between query terms and document terms can be calculated based on the query log data. The query term based query expansion [22, 177] is to measure similarity between query terms using the similarity propagation of Web pages being clicked. The intuition behind is that Web pages are similar if they are visited by users issuing similar queries, and queries are similar if the corresponding users visit similar Web pages.

More recently, several efforts began to analyze the temporal and dynamic nature of the click-through data [23, 45, 146, 161]. In [23], Beitzel et al. proposed the first approach to show the changes of popularities on an hourly basis. With the categorization information of the Web queries, the results show that query traffic from particular topical categories differs from both the query stream as a whole and queries in other categories. Moreover, Shen et al. [146] proposed to investigate the transitions among the topics of pages visited by a sample of Web search users. They constructed a model to predict the transitions in the topics for individual users and groups of users. Vlachos et al. [161] suggested to identify similar queries based on the historical demand patterns, which are represented as time series using the best Fourier coefficients and the energy of the omitted components. Similarly, Chien and Immorlica [45] proposed to find semantically similar queries using the temporal correlation.

6.2 Motivations

Although click-through data has been receiving considerable attention on measuring similarity between query terms in the Web research community, most of existing work ignored an important fact, i.e., the similarity between query terms often evolves over time. Here the
similarities between query terms are usually obtained by the similarity propagation between queries, pages, and their co-occurrences [177]. The dynamic nature of query terms similarity over time, which is usually embedded in the click-through data implicitly, can be attributed to many factors, such as seasons, holidays, and special events, etc. Traditional methods without temporal consideration have limitations to measure such underlying semantic similarity of query terms accurately.

Let us illustrate with an example to compare two different approaches of measuring similarity between two queries over time. Consider Figure 6.2, the dotted line represents the first method that measures the similarity value at each time point based on all the click-through data available at that time. We refer to this method as the incremental approach. Different from the incremental approach, the other approach, as shown in the solid line, measures the similarity using click-through data that are within the corresponding time interval. We refer to this method as the interval-based approach.

From the comparison, we can see that the interval-based approach can better reflect the temporal factor than the incremental approach. For instance, in the interval-based approach, the similarity values of the second and third month are as high as 0.8, while the similarity values in the fourth and fifth month are as low as 0.2. But in the incremental approach, the similarity values from the second month to the fifth month are 0.6, 0.68, 0.55, and 0.48, respectively. This shows that the similarity values in the incremental approach are relatively
fixed, which is usually not able to evidently reflect the dynamic nature of similarities between
the query terms. The intuition is that the smaller the time interval, the more accurate the
interval-based query similarity.

Therefore, it becomes a challenging and important task to develop an effective model
for similarity measure from the click-through data that can take advantage of the dynamic
nature of queries and pages over time. In particular, the following two challenging issues
need to be addressed:

- How to exploit the click-through data for semantic similarity measure of queries in
terms of temporal consideration?
- How to develop an effective model that can reflect both explicit content similarity and
  implicit similarity between queries?

In this chapter, we propose a time-dependent framework to measure the semantic similari-
ity between Web search queries from the click-through data. Specifically, we propose a novel
time-dependent query term semantic similarity model, which can exploit the click-through
data more effectively than traditional approaches such as the incremental approach shown
in Figure 6.2. Our time-dependent model monitors the terms' similarity over the temporal
dimension and attempts to discover the evolution pattern from the click-through data. Note
that in rest of this chapter, the term evolution pattern of the query terms' similarity refers
how the similarity values vary over time.

Overview and Novelty

The basic idea of our solution is to construct a model from the click-through data by parti-
tioning the click-through data into sequences of sub-groups with respect to certain predefined
calendar schema and calendar patterns. For example, to monitor query similarity that may
change on a daily basis, the click-through data is partitioned into sequences of subgroups,
where each sequence consists of click-through data of an individual day. Then, using pro-
posed semantic similarity measure methodology based on the marginalized kernel function,
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

A daily based temporal similarity model can be constructed. As a result, the time-dependent model can accurately reflect the daily-based patterns such as large or small values of query similarities during the specific days.

Compared with the existing approaches, our approach differs from them in the following important aspects. Instead of only discovering Web queries that have specific characteristics, such as burst and periodic queries in [161], our scheme models the similarity of queries based on the predefined calendar schema and calendar patterns. This makes our scheme more general in modelling the temporal feature of queries, which is difficult to achieve by existing techniques. Further, our scheme also critically differs from the approach in [45], which attempts to figure out a unified similarity or correlation between two terms that may not exist for every case and may change rapidly in different situations of real-world environment. Moreover, different from other previous works on categorized Web queries [23, 146], which monitor the differences and transitions between queries from different categories in terms of the frequency of being issued, we focus on exploring queries in terms of their semantic similarity over time. Finally and importantly, in both [161, 45] and [23], only the frequency of the queries are considered, while in our approach both the queries and the corresponding pages being clicked are engaged, together with the relationships between queries and pages. Different from previous approaches, we propose a new time-dependent semantic similarity model formulated by the marginalized kernels in a probabilistic framework to explore explicit content similarity and implicit similarity from the click-through data very effectively.

6.3 Time-Dependent Query Similarity Model

Based on the calendar schema and calendar patterns defined in Chapter 4.3.1, our time-dependent query similarity model is constructed using the marginalized kernel technique, which can exploit both explicit similarity and implicit similarity from the click-through data effectively.
6.3.1 Probabilistic Similarity Measure

Different from the previous query expansion approaches that use the query log and the actual Web pages to extract similarity between terms in the query space and terms in the document space [56, 146], we only employ the query log. We propose a dual approach of the existing information retrieval model by representing each query term as a vector of documents, namely $\vec{q} = < w_1, w_2, \ldots, w_n >$ in which $w_i$ represents the projection weight on the $i^{th}$ page. In our approach, this weight is calculated from the Page Frequency (PF) and Inverted Query Frequency (IQF), which are formally defined as follows:

**Definition 6.41 PF.IQF:** Given a query $\vec{q}$ and a Web page $p_i$ that has been clicked by users who issued $\vec{q}$ via the search engine, the Page Frequency (PF) and the Inverted Query Frequency (IQF) are defined as:

$$
PF(\vec{q}, p_i) = \frac{f(\vec{q}, p_i)}{\sum_j f(\vec{q}, p_j)}, \quad IQF(p_i) = \log \frac{|q|}{|< \vec{q}, p_i >|}
$$

Here $f(\vec{q}, p_i)$ is the number of times that page $p_i$ has been clicked by users who issued the query $\vec{q}$. $\sum_j f(\vec{q}, p_j)$ refers to the total number of times that the pages have been clicked by users who issued the query $\vec{q}$. $|q|$ refers to the total number of times that the query $\vec{q}$ has been issued and $|< \vec{q}, p_i >|$ refers to the times that the page $p_i$ has been clicked by users who issued $\vec{q}$. As a result, the weight of page $p_i$ is calculated as $w_i = PF(\vec{q}, p_i) \times IQF(p_i)$.

Based on the document weight vector representation, the similarity between two queries in content can be defined by a cosine kernel function as follows.

**Definition 6.42 Content Similarity Measure:** Given two queries $\vec{q}_1$ and $\vec{q}_2$, their probabilistic similarity in content, denoted as $K_{cos}(\vec{q}_1, \vec{q}_2)$, is defined as:

$$
K_{cos}(\vec{q}_1, \vec{q}_2) = \frac{\vec{q}_1 \cdot \vec{q}_2}{||\vec{q}_1|| \cdot ||\vec{q}_2||}
$$

Note that, in the above similarity measure, all occurrences of queries and Web pages are considered to be equally important and the timestamps are not used.
6.3.2 Time-Dependent Semantic Similarity Model

Based on the content similarity measure and the calendar schema/patterns proposed in Chapter 4.3.1, we now present the framework of the time-dependent query semantic similarity model. Before going into the details of our framework, let us first review the relationship between the timestamp and the calendar pattern to facilitate our following discussions.

For example, given a calendar pattern < *, 2, 12 > with the calendar schema < week, day of the week, hour >, the timestamp 2005-09-30 12:28 is not contained in this calendar pattern as it is not the second day of the week, while the timestamp 2005-09-26 12:08 is.

Definition 6.43 Click-Through Subgroup (CTS): Given a calendar schema $S$ and a set of calendar patterns \{CAP_1, CAP_2, \ldots, CAP_m\}, the click-through data can be segmented into a sequence of click-through subgroups (CTSs) $< CTS_1, CTS_2, \ldots, CTS_m >$, where all query-page pairs $< q, p, t_i > \in CTS_l$, $t_i < CAP_l$, $1 \leq l \leq m$.

Based on the above definitions, in general, given a collection of click-through data, we can first partition the data into sequences of CTSs based on the user-defined calendar pattern and the corresponding timestamps. For example, given a weekly based calendar schema < week, day > and a list of calendar patterns < *, 1 >, < *, 2 >, \ldots, < *, 7 >, the click-through data will be partitioned into sequences of 7 CTSs $< CTS_1, CTS_2, \ldots, CTS_7 >$, where CTS_i represents the group of click-through data whose timestamps are contained in the i\textsuperscript{th} day of the week.

After that, the query similarities are computed within each subgroup and are aligned into a sequence to show the patterns of historical change. At the same time, a model is generated, with which we can obtain the query similarities by inputting queries and timestamps. Given the above example, we can obtain the query similarity on each day of the week. Moreover, we can monitor how the query similarity changes over time within each week in a daily basis. Also, given two queries and the day of a week, the query similarity can be returned. At the same, the actual data in this time interval is collected to further improve the accuracy of
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

the time dependent query similarity model. Hereafter, we focus on how to construct the time-dependent query similarity model based on sequences of click-through subgroups.

In order to learn the implicit similarity embedded in the click-through data, we first apply clustering techniques on the data to find the cluster information in each click-through subgroup. When the cluster results are obtained, we then formulate our semantic similarity model by the marginalized kernel technique that can unify both the explicit content similarity and the implicit cluster semantics very effectively. Before the discussion of our semantic similarity model, we first discuss how to cluster the click-through data efficiently.

Clusters of Click-through Pages

Given a click-through subgroup, we can obtain clusters of click-through pages $\Omega = \{c_1, c_2, \ldots, c_k\}$ by grouping the pages that are similar in semantics, where $k$ is determined by some clustering algorithm. In the literature, several clustering methods have been proposed to cluster Web pages in the click-through data using the query-page relation and propagation of similarities between queries and pages [22, 177]. In [22], an agglomerative clustering method is proposed. The basic idea is to merge the most similar Web pages and queries iteratively. Originally, the similarity is defined based on the overlaps of neighbors in the bipartite graph representation of the click-through data as shown in Figure 6.1.

For the efficiency reason, we adopt the agglomerative clustering method in [22]. In our clustering approach, neighbors in the bipartite graph are assigned with different weights instead of being taken as equal. The intuition is that the strength of the correlation between two query-page pairs may be quite different. Here, the strength refers to number of co-occurrences of the query-page pair. For example, the strength of a query-page pair that co-occurs once should not be equal as the strength of a query-page pair that co-occurs thousands of times. Hence, we represent the weights of the neighbors based on the number of times the corresponding query-page pairs co-occur. That is, the weight of a page for a given query is the number of times that page was accessed against the total number of times the corresponding query was issued. Similarly, the weight of a query for a given page is the
number of times the query was issued against the total number of times the corresponding page was visited. Then each query is represented as a vector of weighted pages, and each page is represented as a vector of weighted queries. Then similarities between pages or queries are calculated based on the cosine similarity measure.

More details about the clustering algorithm can be found in [22]. Note that the clustering algorithm is applied on each of the click-through subgroups. Based on the clustering results, we now introduce the marginalized kernel technique, which can effectively explore the hidden information for similarity measure in a probabilistic framework [93, 160].

**Definition 6.44 Marginalized Kernel:** Assume that a visible variable \(x\) is described as \(x \in X\), where the domain \(X\) is a finite set. Suppose a hidden variable \(h\) is described as \(h \in H\), where \(H\) is a finite set. A joint kernel \(K_Z(z, z')\) is defined between the two combined variables \(z = (x, h)\) and \(z' = (x', h')\). The marginalized kernel in \(X\) is defined by taking the expectation with respect to the hidden variables as follows:

\[
K(x, x') = \sum_{h \in H} \sum_{h' \in H} p(h|x)p(h'|x')K_z(z, z')
\]

In the above definition, the terms \(p(h|x)\) and \(p(h'|x')\) are employed to describe the uncertainty of the hidden variables \(h\) and \(h'\) related to the visible variables \(x\) and \(x'\), respectively. The marginalized kernel models the probability of similarity between two objects by exploiting the information with the hidden representations. Given the above definition of the marginalized kernel function, we employ it to formulate our time-dependent kernel function for semantic similarity measure of queries as follows. Note that, the hidden variable here is to model the real world events or other real world factors that affects the similarity between queries.

**Definition 6.45 Time-Dependent Query Semantic Similarity Measure:** Given two queries \(\bar{q}\) and \(\bar{q}'\), together with a specific timestamp \(t\), the time-dependent semantic similarity between the two queries is measured by a time-dependent marginalized kernel function

186
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

$K_T(q, q'|t)$ as follows:

$$K_T(q, q'|t) = \sum_{c} \sum_{c'} K_Q(Q_{c|t}, Q'_{c'|t}) p(c|q, t) p(c'|q', t)$$

$$= K_{\cos}(q, q'|t) \left( \sum_{c} \sum_{c'} \varphi(c, c'|t) p(c|q, t) p(c'|q', t) \right)$$

$$= K_{\cos}(q, q'|t) \left( \sum_{c \in \Omega(t)} p(c|q, t) p(c|q', t) \right)$$

where $c$ and $c'$ are the guessed clusters given the queries, $Q_{c|t} = (q, c|t)$ and $Q'_{c'|t} = (q', c'|t)$.

$K_Q$ is a joint kernel, $\varphi(c, c'|t)$ is a function whose value is equivalent to 1 if $c$ and $c'$ are the same and 0 otherwise, and $q_t$ and $q'_t$ are time-dependent query vectors.

In the above formulation, the joint kernel $K_Q(Q_{c|t}, Q'_{c'|t})$ is defined on the two combined query variables as follows:

$$K_Q(Q_{c|t}, Q'_{c'|t}) = \varphi(c, c'|t) K_{\cos}(q, q'|t),$$

where $\varphi(c, c'|t)$ is a function to indicate whether $c$ and $c'$ are the same cluster of click-through data, and $K_{\cos}(q, q'|t)$ is a time-dependent joint cosine kernel on the two time-dependent query vectors $K_{\cos}(q, q'|t) = \frac{q_t \cdot q'_t}{||q_t|| \cdot ||q'_t||}$. Note that the query vectors are only computed on the subgroup $CTS_t$, to which the given timestamp $t$ belongs.

From the definition of time-dependent marginalized kernel, we can observe that the semantic similarity between two queries given the timestamp $t$ is determined by two factors. One is the time-dependent content similarity measure between queries using the cosine kernel function; another is the likelihood for two queries to be grouped in a same cluster from the click-through data given the timestamp.

6.4 Empirical Evaluation

In this section we conduct a set of empirical studies to extensively evaluate the performance of our time-dependent query semantic similarity model. In the rest of this section, we first describe the dataset used in our evaluation and the experimental setup in our experiments.
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

Then, we show several empirical examples to illustrate the real-world results using our time-dependent framework. After that, we discuss the quality measure metric used in our performance evaluation. Finally, the quality of the time-dependent query similarity model is evaluated under different scenarios.

6.4.1 Dataset

A real click-through dataset collected from Microsoft MSN search engine is used in our experiments. The click-through data contains 15 million records of query-page pairs over 32 days from June 16, 2005 to July 17, 2005. The size of the raw data is more than 22 GB. In the following experiments, the entire click-through data is partitioned into subgroups based on the user-defined calendar schema and calendar patterns. For instance, given the calendar schema $<\text{hour}, \text{day}, \text{month}>$ with the calendar pattern $<1,*,*>, <2,*,*>, \ldots, <24,*,*>$, the click-through data is partitioned into a sequence of 24 subgroups, where each group consists of the query-page pairs occurred during a specific hour of everyday. Then, the average number of query-page pairs in each group is around 59,400,000.

6.4.2 Empirical Examples

In this subsection, we present a set of examples of query term similarity evolution over time extracted from the real click-through data collected from MSN search engine. As there are many different types of evolution patterns, here we present some of the representatives.

Figure 6.3(a) shows the similarities for two query pairs ("kid", "toy") and ("map", "route") on a daily basis in the 32 days. It can be observed that the similarities changed periodically in a weekly basis. That is, the similarities changed repeatedly: starting low in the first few days of the week and ending high in the weekend. To reflect such time-dependent pattern, we apply our time-dependent query similarity model to the two query pairs. Here the calendar schema and calendar patterns used are $<\text{day}, \text{week}>$ and $<1,*,>, <2,*,>, \ldots, <7,*>$. Figure 6.3(b) shows the time-dependent query similarity measurement for the two query pairs in Figure 6.3(a). It can be observed that the time-dependent query similarity model can efficiently summarize the dynamics of the similarity over time on a weekly basis.
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

Note that Figure 6.3(b) corresponds to the boxed area in Figure 6.3(a). However, as shown in Figure 6.3(c), the incremental approach cannot accurately reflect the highs and lows of the similarity values. Note that the calendar schema and calendar patterns used in the model are user-defined with related domain knowledge. With inappropriate calendar schema and calendar patterns, we may not be able to construct accurate time-dependent query similarity models. For instance, for the same query pairs, if we use <hour, day> and <1,*>, <2,*>, ..., <24,*> as calendar schema and calendar patterns (shown in Figure 6.3(d)). It can be observed that there are no predictable change patterns, hence there is no hour of the day based time-dependent model that can accurately model the similarity.

Figure 6.3(e) shows the similarity measurement for two query pairs ("weather", "forecast") and ("fox", "news") over one and a half day on hourly basis. It can be observed that "fox" query pairs have two peak values in every day and this pattern repeatedly occur in the dataset. Based on this observation, we propose to model their similarity using the time-dependent query similarity model with a hourly based calendar patterns. That is, the calendar schema and calendar patterns used are <hour, day> with <1,*>, <2,*>, ..., <24,*>. Figure 6.3(f) shows the time-dependent query similarity model. Figure 6.4(a) shows that we cannot generate the time-dependent similarity model if we only have 30 hours of click-through data as there is not any observable pattern. Similarly, Figure 6.4(b) shows the similarity values calculated using the incremental approach, which is clearly not accurate compare to the time-dependent similarity model.

Figure 6.4(c) shows the similarity measurement of two query pairs ("father", "gift") and ("firework", "show") on a daily basis. The corresponding similarity values extracted using the incremental approach are shown in Figure 6.4(d). It can be observed that the time-dependent model cannot be constructed for the two sets of query pairs from the data available in our collection. The reason is that to track event-based query pairs' similarity, e.g., the "father's day" based query pairs' similarity, we need at least years' of click-through data since such events happen only once every year. Note that the previous examples, which
can be modeled using the time-dependent model are within a time interval of 32 days such as weekly based and hour of the day based (our click-through dataset only contains data for 32 days).

6.4.3 Quality Measure

To evaluate the quality of the time-dependent query similarity model, the dataset is partitioned into two parts. The first part consists of a collection of click-through data in the first few days, while the second part consists of the click-through data in the rest. Note that the timestamps of click-through data in the first part must be earlier than the timestamps of the click-through data in the second part. The reason is that we will use the first part
as training data to construct the time-dependent query similarity model, while the second part is used to evaluate the model. Moreover, partitioning of the click-through dataset also depends on the user-defined calendar schema and calendar patterns. For example, to build a weekly based model, the training data should at least cover a time duration of one week; a yearly based time-dependent model cannot be constructed using click-through data of a few days.

Once the time-dependent query similarity model is constructed, given a query pair, the similarity can be obtained by matching the corresponding calendar patterns in the model. For example, with the weekly based query similarity model as shown in Figure 6.3(b), the query similarity between “kid” and “toy” can be derived based on the day of the week. We call the similarity derived from the model as predicted similarity.

Then, the predicted similarity value is compared with the exact similarity value that calculated using the actual dataset. For example, with a weekly based similarity model constructed using the dataset in the first two weeks, the query similarity on the third Monday can be predicted, denoted as \( S' \). Then, the exact similarity is calculated with the dataset.
in the third Monday. Given the predicted similarity value $S'$ and the exact similarity value $S$, the accuracy of the model is defined as $\frac{|S-S'|}{S}$, where $|S-S'|$ is the absolute difference between the two values. Similarly, for the incremental similarity calculation approach, the same definition of accuracy is used so that we can compare the two approaches.

In the following experiments, a set of 1000 representative query pairs is selected from the query page pairs that have similarities larger than 0.3 in the entire click-through data. Some of them are the top queries in the week or month, some are randomly selected, while others are selected manually based on the related real world events such as "father's day" and "hurricane". Note that the accuracy values shown in Table 6.2 are the average accuracy values of all the testing query pairs.

6.4.4 Performance Evaluation

To evaluate the accuracy of the time-dependant query similarity model, three sets of experiments have been done. Firstly, the sizes of the data collections that is used to constructing and testing the time-dependant query term similarity model are varied. For example, we use the first twenty days as training data and use the 11 days left as testing data or we use the first thirty days as training data and use the last day left as testing data, etc. Note that as the size of the testing data increases, the distance between the training data and test data increases as well. Secondly, only the size of the data collection that is used to constructing the time-dependent model is varied. Whereas the testing data is always collected from the other day. For example, we use the first twenty days as training data and use data in the 21st day as testing data. Thirdly, the distance between the training data and testing data is varied while the sizes of the training data and testing data are fixed. Note that the distance between the two data collections is the distance between the latest query-page pairs in the two collections. For instance, we can use the first twenty days as training data and use data in the 21st day as testing data for the case where distance is 1. If the distance is 2, then data in the 22nd day is used for testing. Note that all possible combinations of training and testing data that satisfy the distance constraint are used and the average accuracy values
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

<table>
<thead>
<tr>
<th>Trainingdata</th>
<th>Testingdata</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>22</td>
<td>0.784</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>0.873</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>0.892</td>
</tr>
<tr>
<td>25</td>
<td>7</td>
<td>0.921</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>0.968</td>
</tr>
</tbody>
</table>

Table 6.2: Quality of the Time-dependent model (1)

are presented. In the following experiments, if not specified, the calendar schema \(< hour, day, month >\) is used with the calendar pattern \(< 1, *, * >, < 1, *, * >, \cdots, < 24, *, * >\).

Table 6.2 shows the quality of the time-dependent query similarity model by varying the sizes of data that are used for constructing the model and testing the model. It can be observed that when the size of the training data increases and size of the testing data decreases, the accuracy of the time-dependent model increases as well. When the size of the training and testing data are similar, the accuracy can be as high as 87.3%. Note that here all the click-through data in the 32 days are used. We use the first part of the data as training data and the rest as testing data. The reason behind may be that when the training is not large enough to cover all the possible patterns, then the time-dependent model may not be able to produce accurate results.

Figure 6.5(a) shows the quality of the time-dependent query similarity model by varying the size of data that is used for construction the model and fixing the size of data that is used for testing to 1. Three different calendar schemas and calendar patterns are used as well. It can be observed that when the size of the training data increases, the accuracy of the time-dependent model increases as well. This fact is just as we expected: as the size of the training data increases, performance of the model is expected to increase.

Figure 6.5(b) shows how the quality of the time-dependent query similarity model changes by varying the time distance between the data collection that is used for testing and the data collection used for constructing the model. For example, when the distance is 1 and the training data size is 10, we summarize all the accuracy values that use the \(i\) to \(10+i\) days as training and use the \(10+1+i\) as testing. It can be observed that when the distance increases, the accuracy of the time-dependent model decreases. At the same time, when the
size of the training data increases, with the same distance, the accuracy value may increase. The reason behind this set of data shows that the time-dependent model is more accurate if the most recent data is incorporated as the time-dependent model may be modified.

Moreover, we implemented an incremental query similarity model and compare the prediction accuracy with the time-dependant approach. Note that for the two approaches both the data that are used for building the model and the data that are used for testing are the same (The first part of the data is used for training and the rest is used for testing). In the following experiments, three calendar schema and calendar pattern pairs are used. The calendar schema and calendar patterns are \(<\text{hour, day, month}>\) with \(<1, *, *>, <2, *, *>, \cdots, <24, *, *>\); \(<\text{hour, day, month}>\) with \(<*, 1, *>, <*, 2, *>, \cdots, <*, 31, *>\); and \(<\text{day, week}>\) with \(<1, *>, <2, *>, \cdots, <7, *>\). In the following experiments, we use the 1000 sampled query pairs for evaluation.

Figure 6.6 shows the comparison of quality about the similarity values obtained using the incremental approach and the time-dependent model. Note that, the size of the training data is varied from 1/4 of the dataset to 7/8 of the dataset as well, while the rest is used for testing. It can be observed that when the intervals in the calendar schema become larger, the quality of the time-dependant model decreases. This is due to the fact that we only use a click-through data of 32 days, which can produce satisfactory results with the hourly and daily based calendar patterns. However, in general the quality of the time-dependent model is better than the incremental approach. However, it can be observed that for some calendar schema based time-dependent query similarity model, the accuracy of the model
CHAPTER 6. BUILDING TIME-DEPENDENT SEMANTIC QUERY SIMILARITY MODEL

Figure 6.6: Quality of the Time-dependent model (3)

decreases dramatically when the size of the training data decrease. Especially for the daily based calendar schema. The reason is that size of our data collection is not large enough such that when the size of the training data decreases it cannot cover every possible day in one month (requires at least 31 days of training data).

6.5 Summary

With the availability of massive amount of click-through data in current commercial search engines, it becomes more and more important to exploit the click-through data for improving the performance of the search engines. In this chapter, we attempt to extract the semantic similarity information between queries by exploring the historical click-through data collected from the search engine. Different from previous works, we proposed a time-dependent semantic similarity model by studying the temporal information associated with the query terms in the click-through data. We formulated the time-dependent semantic similarity model into the format of kernel functions using the marginalized kernel technique, which can discover the explicit and implicit similarity similarities effectively. We conducted the experiments on the click-through data from a real-world commercial search engine. Our results show that term similarity does evolve from time to time and our semantic similarity model is effective in modelling the similarity information between queries. Finally, we observed an interesting phenomenon that the evolution of query similarity from time to time may reflect the evolution patterns and events happening in different periods of time.
Chapter 7

Event Detection From Evolution of Click-Through Data

In this chapter, we focus on the issue of semantic extraction from evolution of semi-structured Web data. Specifically, we propose to detect events from the evolution of Web search engine click-through data. The intuition behind event detection from click-through data is that click-through data is often event-driven. As a result, each event can be represented as a set of query-page pairs that are not only semantically similar but also have similar evolution pattern over time. Note that in previous work events are defined as clusters of Web documents [188]. To the best of our knowledge, this is the first approach to detect event from the visitor-centric data. Previous efforts on event detection from the web have focused primarily on author-centric data such as Web content and structure data.

The rest of this chapter is organized as follows. Firstly, we present the background of event detection from Web data and existing event detection approaches in Section 7.1. Then, the motivation of detecting event from the evolution of click-through data is presented by highlighting limitations of existing works in Section 7.2. Section 7.3 presents our event detection approach in detail. The performance studies are discussed in Section 7.4 using real data collection obtained from MSN search engine. Section 7.5 concludes this chapter. A preliminary version of this chapter was appeared in the ACM Knowledge Discovery and Data Mining Conference 2006 [213].
7.1 Background of Event Detection Problem

The web has invaded our lives. Web data now covers almost every object and event in the real world. That is, in some sense, the web is a sensor of the real world. This sensor-centric view of the web has recently triggered research efforts to extract knowledge such as topics, events, and stories from large volumes of web data\[13, 112, 187, 190, 199\]. These approaches can be classified into two groups: *structure-based* extraction and *content-based* extraction. In the structure-based approaches, the website structures, hyperlink structures, and URLs are used to extract sets of web pages corresponding to events and objects \[153, 111\]. In the content-based extraction, content of web pages are segmented and categorized into subgroups that correspond to different topics, events, and stories using techniques such as natural language processing and probability models \[187, 190, 199\]. At the same time, such extraction results have been proved useful in many applications such as organizing the website structure \[153\], restructuring the web search results \[111\], terrorism event detection \[155\], and *Photo Story* and *Chronicle* \[112\]. Here, specifically, we review the existing works on topic detection and tracking (TDT).

The TDT research project was initially a DARPA sponsored project concerned with finding groups of stories on the same topic. It consists of three major issues: segmenting the text corpus into events, tracking the development of the detected events, and detecting new events \[187\]. Existing TDT research focuses on analyzing news oriented textual materials from a variety of broadcast news media. As for event detection, different approaches have been proposed \[13, 112, 190, 199\]. In \[190\], a two-phase novel topic detection algorithm is developed. The idea is to first classify the incoming news into predefined categories and then use the topic-conditioned heuristics to identify the new events. In \[13\], the authors compare the TDT problem with a more difficult two-part task defined by the TREC 2002 novelty track: given a topic and a group of documents relevant to that topic, 1) find the relevant sentences from the documents, and 2) find the novel sentences from the collection of relevant sentences. More recently, in \[199\], a general English language model is used as
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

the base distribution to handle the generation of new events. In [112], retrospective new event detection (RED) approach is proposed to discovery novel events in the news corpus by taking both the content and the time information.

7.2 Motivation

In the context of event detection from Web data, the types of data that are used play an important role. Recall that Web data can be broadly classified into two types: author-centric and visitor-centric. We observed that in most of the existing event detection approaches only the author-centric data is considered while the rich collection of visitor-centric data is ignored. We believe that in many real life cases, the visitor-centric data plays equally important role for event detection. We believe that events can be detected from the visitor-centric data. Note that visitor-centric data can be either Web access patterns of visitors reflecting their browsing activities or it can be click-through data. In this approach, we focus our attention to detect events from the click-through data. To the best of our knowledge, this is the first effort to detect events from click-through data. We show that it is indeed possible to produce high quality event detection results without analyzing the content and structure of Web pages (author-centric data).

As previous works on event detection from Web data have primarily focused on detecting events from a collection of hyperlinked pages, we first justify the reasons for choosing the Web search engine click-through data as the source for event detection. First, the increasing popularity of Web search engines has given rise to a large number of search engine users issuing huge volumes of queries. These queries often return links to high quality Web pages. Consequently, there is a large volume of click-through data that can be potentially exploited for event detection. Second, as shown in Table 6.1, the click-through data contains the query keywords that are created by users and links to Web pages that often describe real world events. Specifically, as we shall see later, these keywords and the corresponding pages clicked by the users often reflect their response to recent real world events.
Besides exploiting visitor-centric data, another key distinguishing feature of our proposed event detection approach is that we take into account the *dynamic* nature of the visitor-centric data in the event detection process. Here, *dynamic* nature refers to the evolving nature of the queries and pages in the click-through data over time. For instance, visitors may formulate new queries that were not formulated before, or some previously executed queries may be executed more or less frequently due to some recent events. Furthermore, new Web pages that were not available earlier may now appear in the search results and hence they may be clicked now by visitors. Similarly, previously clicked pages may be clicked more or less frequently in recent times. Also, previously executed queries may lead to a different set of Web pages when the queries are formulated during a specific time period. As a result, the frequencies of queries being issued and pages being clicked may change over time as discussed in Chapter 1.1 with examples shown in Figure 1.3.

Our investigation with real click-through data revealed that the above dynamic behavior of the click-through data is often driven by real world events. Specifically, the frequencies of the issued queries and clicked pages as well as the co-occurrences of the queries and pages often depend on real world events that have happened, are happening, or are going to happen. For example, reconsider Figure 1.3(c). In September 2004, the frequency of the query “911” and corresponding Web page $P_1$ being issued and accessed increased dramatically. At the same time, the co-occurrence of “911” and $P_2$ increased substantially. Further investigation revealed that this is primarily due to the occurrences of two events during that time. One is the three year anniversary for the “911” event in 2001 (represented by $P_1$). The other is the new movie “Fahrenheit 911” (represented by $P_2$) which was released in June 2004 in the United States. Observe that co-occurrence of “911-$P_2$” started increasing from end of June and reached a maximum value in September 2004.

By incorporating such dynamic nature of the click-through data in the event detection process, the quality of events detected can be improved. Let us illustrate this with the following examples.
When a new event occurs, the number of related queries being issued and the number of related Web pages being visited may increase dramatically. At the same time, the co-occurrence relationships between these queries and pages are surprisingly strong. For example, we observed from the click-through data of a commercial search engine that the query of “Shania Twain”, a singer, was surprisingly popular from November 2004 to December 2004 as shown in Figure 1.3(a). Also, there is a set of related Web pages that have high co-occurrences with this keyword. It turns out that “Shania Twain” was expected to have a series of concerts in the first two weeks of December, 2004 in the US. Observe that this event can be detected by analyzing the dynamic behavior of the frequency of the issued query, clicked Web pages, and the query-page relationships.

For a group of similar queries, the corresponding Web pages being clicked can be substantially different from time to time. In another word, the same query may represent different events depending on the set of Web pages clicked by the users in the query result set. Similarly, the same page may represent different events depending on the query that directed to this page. Let us reconsider the example in Figure 1.3(c). The query of “9/11” was issued frequently in September 2004 and a lot of pages about the 9/11 anniversary event and the new movie Fahrenheit 9/11 were clicked. Using existing event detection approaches, which take the click-through data as snapshot data, it is difficult to distinguish the two events as they share a list of similar queries and the Web pages being clicked may be connected via hyperlinks. However, by incorporating the dynamics of the query-page relationship over time, as shown in Figure 1.3(c), the two events can be easily distinguished.

7.3 Event Detection From Evolution of Click-Through Data

In this section, we first present an overview of event detection problem from evolution of click-through data in Section 7.3.1. Section 7.3.2 addresses the issue of representing the
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

Given the click-through data and the user-specified time granularity such as hour, day, week, month, etc., the data is segmented into a sequence of collections such that the user sessions within each collection are within the same time interval. Note that the time granularity is user-defined and application-dependent. For example, if the click-through data is segmented on an hourly basis, then the click-through data for a particular day will be partitioned into a sequence of 24 collections. Then, each collection is represented as a bipartite graph $G = (V, E)$, where nodes in $V$ represent queries and Web pages and edges in $E$ represent the strengths of the query-page pairs within the time interval [177]. As a result,
the input click-through data is transformed into a sequence of bipartite graphs.

To represent the evolution of the query-page relationships, we propose to merge the sequence of bipartite graphs into a *vector-based graph*, where the edge in the graph is a vector that represents the sequence of strengths of the query-page relationship over time as shown in Figure 7.1 and the vertices represent the queries and Web pages. For clarity, we only show the strength vector of one query-page pair in the figure. Note that the strength vector in the vector-based graph enables us to model the evolution pattern of the strength of query and page relationship. However, we cannot directly detect events from the vector-based graph by partitioning it into subgraphs for the following two reasons.

- First, real world events are expected to be represented as sets of query-page pairs, while the vertices in the vector-based graph represent individual queries or Web pages. More specifically, to detect events, we need to know the similarity between query-page pairs rather than the similarity between queries and pages.

- Second, some queries or pages may belong to two or more different events, as we discussed in the previous section (Figure 1.3(c)). If we partition the vector-based graph, then the query term(s) can only belong to one of the events as it is represented as a single vertex. Hence, queries or pages that represent multiple events pose a problem if we attempt to detect events directly from the vector-based graph.

Consequently, we transform the vector-based graph into its *dual graph*, which is well-defined in graph theory [80]. We shall elaborate on the transformation later. The basic idea is that the edges in the vector-based graph are transformed into vertices and the vertices in the vector-based graph are transformed into edges in the dual graph. Note that the vectors in the vector-based graph are represented as attributes of the vertices in the dual graph as shown in Figure 7.1. Note that for clarity, we only show the attribute of vertex (1, 1) in Figure 7.1. The dual graph addresses the above two issues in the following ways. In the dual graph, each vertex is a *query-page pair* and each edge represents the *similarity* between the
query-page pairs. Moreover, the same query can be included in more than one vertex in the dual graph.

Then, the event detection problem can now be formulated as the problem of partitioning the dual graph based on the similarity values between query-page pairs. In our approach, we consider two types of similarities, the semantic-based similarity and the evolution pattern-based similarity. The evolution pattern-based similarity values are computed from the co-occurrence vectors while the semantic-based similarity values are computed from the structure of the dual graph. We shall elaborate on these two metrics. Based on the state-of-the-art graph cut algorithm [147] and the two similarity measures, a two-phase graph cut algorithm is proposed to partition the dual graph. Firstly, the dual graph is cut into subgraphs that correspond to groups of semantically related events using the semantic-based similarity. After that, the groups of semantically related events are partitioned again into individual events using the evolution pattern-based similarity. By looking into the event detection results, the users can further refine the time granularity with respect to the application requirements to detect different types of events.

7.3.2 Representing Click-Through Data

In the literature, there are two different ways to represent the click-through data. However, we observed that previous representations of the click-through data, to the best of our knowledge, do not take the timestamps of the query-page pairs into account. That is, occurrences of a query-page pair are taken as equally important and meaningful even though the corresponding timestamps are different. This assumption may not always be true in real life applications. For instance, reconsider the examples in Figure 1.3(c). The co-occurrences of a query-page pair at different timestamps may be taken as different indicators of the underlying events. More importantly, the evolution patterns of the query-page relationships may indicate the evolution of the corresponding events. In our approach, we propose to monitor their evolutionary patterns and use such patterns for accurate event detection as well.
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

To represent the evolutionary patterns, the occurrences of the query-page pairs in the click-through data are partitioned into collections based on their timestamps and the user-defined time granularity. Note that the time intervals can be defined according to users' requirement and the temporal properties of the interesting events. By doing this, the click-through data can be represented as a sequence of collections, where each sub-group can be represented as a bipartite graph, which is commonly used in click-through data analysis [177].

Formally, given a click-through data \( C \), it is partitioned into a sequence of collections based on user-defined time granularity, denoted as \( < c_1, c_2, \ldots, c_n > \). We define the strength of a query-page pair \( P_{s,t} = (q_s, p_t) \) in \( c_i \), denoted as \( s_i(P_{s,t}) \), as \( s_i(P_{s,t}) = \frac{|P_{s,t}(c_i)|}{\sum_{i=1}^n |P_{s,t}(c_i)|} \) where \( 1 \leq i \leq n \). Note that the numerator is the number of co-occurrences of the query-page pair in the collection \( c_i \) and the denominator is the total number of co-occurrences of the query-page pair in \( C \). Observe that the strength measure is the ratio of the co-occurrences of a query-page pair in a specific collection \( c_i \) against the total number of co-occurrences of the query-page pair in the entire collection \( C \).

If we consider the absence of a query and a page as a relationship of strength 0, then we can assume that the sizes of the sequence of bipartite graphs are equal and fixed, and the only difference between these graphs are the weights of the edges. Thus, the representation of the click-through data can be formulated as a merged bipartite graph, called vector-based graph. Formally,

**Definition 7.46 Vector-based Graph:** Let \( < c_1, c_2, \ldots, c_n > \) be the sequences of click-through data segmented from click-through data \( C \) using the user-defined time granularity. A vector-based graph is a bipartite graph, \( G = (V, E) \), where \( V \) is a set of queries and pages; \( E \) is a set of labelled edge \( (q, l, p) \) where \( q, p \in V \) and \( l \) is a vector denoted as \( [s_1, s_2, \ldots, s_n] \) that represents the sequence of strengths of the query-page pair \( (q, p) \) where \( s_i \) is the strength of \( (q, p) \) in \( c_i \).
For example, suppose the vector corresponding to the query-page pair \((q_1, p_1)\) is \([0.42, 0.37, 0.21]\). It means that the strengths of the query-page pair \((q_1, p_1)\) are 0.42, 0.37, and 0.21, respectively in the three consecutive time intervals. Note that different from previous approaches, the query-page relationships are decomposed into a vector. Each individual value represents the corresponding strength of the query-page relationship during a specific time period. As a result, we successfully embed the temporal and evolutionary information of the click-through data in the vector-based graph representation.

As discussed in the framework, we have to transform the vector-based graph to its dual graph. Basically, in the dual graph, each vertex is a query-page pair, which is an edge in the bipartite graph; and each edge is a query/page that links the two query-page pairs, which is a vertex in the bipartite graph. Formally, dual graph is defined as follows.

**Definition 7.47 Dual Graph:** Given a vector-based graph \(G=(V, E)\), a graph \(G'=(V', E')\) is the dual graph of \(G\) if and only if 1) \(V'\) is a set of vertex labelled as \((p, l, q)\) and for each vertex \(v'_i \in V'\) there is a corresponding edge in \(E\); 2) \(E'\) is a set of edges and for each \(v'_{ij} \in V'\) there is a corresponding vertex in \(V\); 3) There exists an edge \(e_{ij}\) that connects \(v'_i\) and \(v'_j\) if and only if \(q_i = q_j\) or \(p_i = p_j\).
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

An example of the vector-based bipartite graph and the corresponding dual graph is shown in Figure 7.2. Note that the dotted edges in the dual graph represent the query-based edge, while the solid edges represent the page-based edge in the vector-based graph. Here, a query-based edge connects two query-page pairs that share the same query, while a page-based edge connects two query-page pairs that share the same page. Moreover, in the dual graph, the vectors that represent the historical strengths of the query-page pairs are now represented as attributes of the nodes in the dual graph.

7.3.3 Similarity Measures

As mentioned in the preceding section, there are two types of similarities concerning the query-page pairs. In this section, we elaborate on this. The intuition behind semantic-based similarity is that the query-page pairs sharing the query or page are similar to each other. One of the key features of the similarity is that it can be propagated. For example, suppose we have two query-page pairs \( P_{i,j} = (q_i, p_j) \) and \( P_{s,t} = (q_s, p_t) \) that share neither the query nor the page. However, \( P_{i,j} \) and \( P_{s,t} \) can be similar if there exist another query-page pair \( P_{k,l} = (q_k, p_l) \), which shares the query with \( P_{i,j} \) and the page with \( P_{s,t} \). The second similarity is the evolution pattern-based similarity. That is, query-page pairs sharing similar evolution patterns are expected to be similar. Specifically, evolution pattern refers to the pattern how the strength of the query-page pair relationships change over time in the history as shown in Figure 1.3(c). Typical evolution patterns include periodic change, burst, increasing or decreasing change, and other model-based changes. To represent the different types of similarities, we decompose the dual graph into two different graphs: semantic-based graph and evolution pattern-based graph, and then define the two types of similarities, respectively.

Semantic-based Graph and Similarity

The semantic-based graph is structurally the same graph as the dual graph except that the vector attributes in the original dual graph are not included in the semantic-based graph. Each edge \( e_{ij} \), if exists, in the semantic-based graph denotes the strength of the relationship between two query-page pairs that are connected by \( e_{ij} \). Recall that in the
dual graph there are two types of edges the query-based edge and the page-based edge. The semantic relationship between nodes is actually the fusion of this page-based and query-based relationship. For simplicity, we represent such a fusion as a linear combination of the two types of relationships, where the weights of the page-based and query-based edge are denoted as \( \alpha \) and \( \beta \) respectively (\( \alpha + \beta = 1 \)). In the following, we set both \( \alpha \) and \( \beta \) to 0.5. Then, the semantic-based similarity between two nodes \( a \) and \( b \) in the dual graph, denoted as \( S^*(a,b) \), is defined as follows.

**Definition 7.48 Semantic-based Similarity:** Let \( a \) and \( b \) be two nodes in the dual graph. The semantic-based similarity between them, denoted as \( S^*(a,b) \), is defined as:

\[
S^*(a,b) = \frac{C}{|N(a)||N(b)|} \sum_{i=1}^{\max(|N(a)|,|N(b)|)} \sum_{j=1}^{\min(|N(a)|,|N(b)|)} S^*(N_1(a), N_j(b))
\]

where \( C \) is the constant decay factor between 0 and 1, \( |N(a)| \) and \( |N(b)| \) are the numbers of neighbors for node \( a \) and \( b \), respectively. \( N_i(a) \) and \( N_i(b) \) are the \( i^{th} \) neighbors of \( N(a) \) and \( N(b) \), respectively.

From the above definition, it is obvious that the similarity value has a range between 0 and 1. In this following, the decay factor is set to 0.7. Also, the equation is recursive and the similarity between query-page pairs can be propagated at the next recursion. It has been proved that the value of \( S^*_k \), which is based on the value of \( S^*_{k-1} \), is non-decreasing and will converge eventually [87]. In our approach, we start with \( S^*_0 \):

\[
S^*_0 = \begin{cases} 
\alpha & \text{if } a \text{ and } b \text{ are connected via a query edge} \\
\beta & \text{if } a \text{ and } b \text{ are connected via a page edge} \\
0 & \text{otherwise}
\end{cases}
\]

**Evolution Pattern-based Graph and Similarity**

Similar to the semantic-based graph, in the evolution pattern-based graph, each vertex also represents a query-page pair. However, the edges in the evolution pattern-based graph are different. Specifically, an edge between two query-page pairs represents the evolution pattern-based similarity calculated based on the affiliated vectors. As a result, the evolution
pattern-based graph is a fully connected graph. Since the dimension of the vectors can be huge, we propose to summarize the vector using histogram with different granularity such that the dimensions can be reduced. Then, similarity is defined based on the histogram representation of the vectors using dynamic time warping [94].

Suppose the click-through data is segmented into a sequence of \( n \) collections based on the user-defined time granularity. Then there will be a sequence of \( n \) strength values represented as a vector \( S = [s_1, s_2, \ldots, s_n] \). Given the window size of the histogram representation, \( w \), each vector will then be compressed into histogram representation \( H = (H_1, H_2, \ldots, H_{n/w}) \), where \( H_i = \sum_{j=(i-1)\times n/w}^{i\times n/w} s_j \). Based on different window size, the histogram representation may show different characteristics of the data in different granularity. In the following sections, we will show that different types of events can be detected using different width.

Given two query-page pairs \((q_i, p_j)\) and \((q_s, p_t)\), denoted as \(P_{i,j}\) and \(P_{s,t}\), the histogram representations of their historical strengths are denoted as \(H^{i,j}\) and \(H^{s,t}\). In the dynamic time warping approach, an \( n \times n \) matrix is built, where the \((x^{th}, y^{th})\) element denotes the distance between \(H_{x}^{i,j}\) and \(H_{y}^{s,t}\) (typically the Euclidean distance is used). A warping path, \(W\), is a set of continuous elements in the matrix that defines a mapping between \(H^{i,j}\) and \(H^{s,t}\). The \(k^{th}\) element of \(W\) is defined as \(w_k = (x, y)_k\), so we have \(W = \langle w_1, w_2, \ldots, w_k \rangle\), where \(n \leq k \leq 2n - 1\). The warping path is typically subjected to the following three constraints.

- **Boundary conditions:** the warping path should starts from \((1,1)\) and ends at \((n, n)\).
- **Continuity:** given \(w_k = (a, b)\) and \(w_{k-1} = (a, b')\), then \(a - a' \leq 1\) and \(b - b' \leq 1\).
- **Monotonicity:** given \(w_k = (a, b)\) and \(w_{k-1} = (a, b)\), then \(a - a' \geq 0\) and \(b - b' \geq 0\).

Then, formally the evolution pattern-based similarity is defined as follows:

**Definition 7.49 Evolution Pattern-based Similarity:** Let \(P_{i,j}\) and \(P_{s,t}\) be two query-page pairs with dynamic warping \(W = \langle w_1, w_2, \ldots, w_k \rangle\). The evolution pattern-based similarity between \(P_{i,j}\) and \(P_{s,t}\), denoted as \(S^P(P_{i,j}, P_{s,t})\), is defined as:
\[ S^P(\mathcal{P}_{ij}, \mathcal{P}_{ab}) = 1 - \min \left\{ \frac{1}{k} \sum_{m=1}^{k} w_m \right\} \]

It can be observed that if two query-page pairs have exactly the same evolution pattern, then the evolution pattern-based similarity will be 1. From the definition, we can observe that the range of the similarity is between 0 and 1.

### 7.3.4 Event Detection Algorithm

In this section, we present the algorithm for event detection from the dual graph representation of the click-through data. Based on the normalized graph cut algorithm [147] reviewed in Chapter 5.5.2, a two-phase clustering algorithm is proposed to cluster the dual graph based on the semantic-based similarity and evolution pattern-based similarity. The algorithm of event detection from the click-through data is shown in Algorithm 7.1. Basically, as highlighted in the framework in Figure 7.1, the event detection algorithm consists of four major steps. Given the click-through data, the first step is to segment the query-page sessions into collections based on the user-defined time granularity. Each collection is represented as a bipartite graph and the click-through data is represented as a sequence of bipartite graph as shown in Lines 1-3 in the algorithm. Then, the sequence of bipartite graphs is merged into a single graph called vector-based graph. As a result, all the evolution patterns of the query-page relationships are stored. After that, the vector-based graph is transformed into its dual graph, where each vertex is a query-page pair. The goal is then to partition the dual graph into subgraphs that represent real world events. The partition algorithm is based on the above normalized graph cut as shown in Lines 5-10 in the algorithm. The graph cut algorithm is applied to the dual graph twice. Firstly, the graph is cut into subgraphs based on the semantic-based similarity such that each subgraph may contain a set of semantically related events. Then, the subgraphs are cut again based on the evolution pattern-based similarity.

Note that here we focused on designing an algorithm that can detect sub-groups of click-through data corresponding to real world events. As we shall see in Section 6.3, there are
Algorithm 16 Event Detection from Click-Through Data

Input: A set of query-page sessions: $D$  
Output: A set of query-page clusters: $C_1$, $C_2$, ..., $C_n$  
1: Partition the query-page sessions into a sequence of groups $(G_1, G_2, ..., G_n)$ based on the user-defined time interval  
2: Construct the corresponding bipartite graphs $(B_1, B_2, ..., B_n)$  
3: Construct the vector-based bipartite graph $B_v$  
4: Construct the dual graph $D_B$ from $B_v$  
5: Calculate the semantic-based similarity among vertexes in $D_B$  
6: Perform the semantic-based normalized graph cut on $D_B$  
7: Store the intermediate clusters $C' = \{C_1, C_2, ..., C_m\}$  
8: Calculate the pattern-based similarity for vertexes within the same cluster in $C'$  
9: Perform the pattern-based normalized graph cut to individual cluster in $C'$  
10: Return the updated clustering results $C_1, C_2, ..., C_n$

different types of events such as periodic events (national holidays), burst events (unpredictable accidents), etc. The issue of classifying the detected events further into different categories will be part of our future work.

7.4 Performance Evaluation

In this section, we study the performance of our event detection approach. Firstly, the characteristics of the dataset are presented. Then, quality of the experimental results is evaluated under different scenarios. Moreover, a list of events detected from the dataset is presented.

7.4.1 Dataset

A real click-through dataset that is extracted from a commercial Web search engine is used in the following experiments. The click-through data contains 15 million records of query-page pairs over 32 days from June 16, 2005 to July 17, 2005. The entire click-through data is partitioned into 768 collections, where each collection consists of the query-page pairs occurred during an interval of one hour. The average number of query-page pairs in each group is around 2,100,000.

7.4.2 Quality of Detected Events

To evaluate the quality of the event detection results, we manually labelled a list of 30 representative real world events as test cases. The reason is that for event detection from
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

Click-through data there is no benchmark dataset. In our experiment, each labelled event consists of a set of query-page pairs. The labelling process is as follows.

1. Select a list of key queries, denoted as $K_q$, which was frequently issued and corresponded to a list of specific events.

2. For each key query $q_i \in K_q$, a set of key pages, denoted as $K_p$, that is led to by $q_i$ is extracted.

3. For each key page $p_i \in K_p$, the list of queries that led to $p_i$ is extracted and inserted into $K_q$.

4. Recursively execute step 2 and 3 to update $K_p$ and $K_q$ till the frequencies of the queries and pages extracted reach the minimum threshold.

5. Based on the frequency of the queries and pages, together with their relationships in the click-through data, we manually select a list of query-page pairs to represent each event referring to the related news archives that record real world event.

Some of the selected events and corresponding queries are shown in Table 7.1. Here a key query is a query that has been issued most frequently and lead to most of the event relevant pages. The expanded queries are these queries that were issued frequently and lead to relevant pages in the labelled events. The timestamp represents the time interval when the queries were issued. For example, the key query “fireworks” is the most frequently issued query relevant to the firework show during the National Day of the United States. From the expanded queries, it can be observed that people are preparing for that event. For another example, the key query “Lohan” is about an entertainment star in the US, during the week she attended the “Jay Leno show”. As a result, many related queries were issued most likely by her fans. It can be observed that there are two types of scenarios. First, the queries may be issued frequently with many visits to the pages before scheduled events such as National Day of the US. Second, the queries are issued after certain unpredictable events such as the recent earthquakes and hurricanes.
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

<table>
<thead>
<tr>
<th>Key query</th>
<th>Expanded queries</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>fireworks</td>
<td>July-4th show picture foto locations buy cheap black market safety injuries</td>
<td>June 21-27, 2005</td>
</tr>
<tr>
<td>Wimbledon</td>
<td>tennis championships results women’s 2005 live schedule</td>
<td>June 21-27, 2005</td>
</tr>
<tr>
<td>Lohan</td>
<td>Lindsey boob jay-leno tonight-show picture</td>
<td>June 21-27, 2005</td>
</tr>
<tr>
<td>battlefield</td>
<td>2 player joystick cracked server code keygen download review demo tips</td>
<td>June 21-27, 2005</td>
</tr>
<tr>
<td>earthquake</td>
<td>California center current recent picture locations</td>
<td>June 13-20, 2005</td>
</tr>
<tr>
<td>batman begins</td>
<td>Wallpaper review picture download scarecrow cast movie walkthrough</td>
<td>June 13-20, 2005</td>
</tr>
<tr>
<td>Live 8</td>
<td>concert song pink pictures floyd download locations list</td>
<td>27-July 04, 2005</td>
</tr>
<tr>
<td>War of the world</td>
<td>movie photo rating review game download cruise wells</td>
<td>27-July 04, 2005</td>
</tr>
<tr>
<td>hurricane dennis</td>
<td>forecast update path 2005 weather projected report track</td>
<td>July 04-11, 2005</td>
</tr>
<tr>
<td>kelly monaco</td>
<td>naked playboy-news movie pics nude gallery biography</td>
<td>July 04-11, 2005</td>
</tr>
<tr>
<td>british open</td>
<td>odds golf history score tiger results leader coverage 2005</td>
<td>July 11-18, 2005</td>
</tr>
<tr>
<td>harry potter</td>
<td>half-blood-prince party shop book song goblet spoilers release summary</td>
<td>July 11-18, 2005</td>
</tr>
<tr>
<td>Wedding Crashers</td>
<td>movie quotes review script premier trailer diona</td>
<td>July 11-18, 2005</td>
</tr>
</tbody>
</table>

Table 7.1: Examples of events.

<table>
<thead>
<tr>
<th>in cluster</th>
<th>not in cluster</th>
<th>in event</th>
<th>not in event</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.2: The cluster-event contingency table.

Note that the experiment is conducted with the entire click-through data, which may produce many other events that are not listed in the table. The reason that we do not select only the query-page pairs related to the selected event is that, in most case, these events are not connected in the bipartite representation of the click-through data.

With the partitioning results of the click-through data, the clusters that best match the 30 labeled events are used for evaluation. Note that both the clusters in the event detection results and the labelled events are represented as sets of query-page pairs. A best match of a labelled event is a cluster of query-page pairs in the event detection results that share the same key query and has the maximum overlap between the labelled event in terms of the number of common query-page pairs. Then, the evaluation is based on the contingency tables.

Table 7.2 shows the two-by-two contingency table for a cluster-event pair, where a, b, c, and d are the numbers of query-page pairs in the corresponding cases. For example, given a labelled event with its best matched cluster, a represents the number of query-page pairs that are included in both the labelled event and the detected cluster; b represents the number of query-page pairs that are in the detected cluster but not in the labelled event; c represents the number of query-page pairs that are in the labelled event but not in the
detected cluster; while \( d \) represents the number of query-page pairs that are not included in the detected cluster and the labelled event. Five evaluation measures, \( \text{miss} \), \( \text{false alarm} \) (\( f \)), \( \text{recall} \) (\( r \)), \( \text{precision} \) (\( p \)), and the \( F_1 \) measure (\( F_1 \)), are defined as follows based on the contingency table.

- \( \text{miss} = \frac{c}{a+c} \) if \((a+c) > 0\), otherwise undefined;
- \( f = \frac{b}{b+d} \) if \((b+d) > 0\), otherwise undefined;
- \( r = \frac{a}{a+c} \) if \((a+c) > 0\), otherwise undefined;
- \( p = \frac{a}{a+b} \) if \((a+b) > 0\), otherwise undefined;
- \( F_1 = \frac{2rp}{r+p} = \frac{2a}{2a+b+c} \) if \((2a+b+c) > 0\), otherwise undefined.

To measure the global quality of the event detection results, two average metrics, the micro-average \( F_1 \) (\( Mic \ F_1 \)) and macro-average \( F_1 \) (\( Mac \ F_1 \)) are used [190]. Given \( n \) events, the \( Mic \) and \( Mac \ F_1 \) are defined as follows.

\[
Mic \ F_1 = \frac{2 \times \sum_{i=1}^{n} a_i}{2 \times \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} c_i}
\]

\[
Mac \ F_1 = \frac{1}{n} \times \sum_{i=1}^{n} \frac{2 \times a_i}{2 \times a_i + b_i + c_i}
\]

Table 7.3 shows the quality of the event detection results, where the size of the window in the histogram representation varied from 1 to 3, which is denoted as \( s_i \) where \( i \) is the window size. By combining the window size and the time granularity, \( s_1 \), \( s_2 \), and \( s_3 \) represent the hourly, daily, and weekly interval, respectively. Also, the reverse order of graph cut algorithm is evaluated, where the evolution-similarity based graph cut is performed first and followed by the semantic-similarity based graph cut. At the same time, the quality of the corresponding event detection results that only use the semantic-based similarity are denoted as \( s'_i \) in the table. In general, the event detection results are satisfactory and encouraging. Also, it shows that the window size of the histogram representation may affect the quality of the results.
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

<table>
<thead>
<tr>
<th></th>
<th>With Evolution</th>
<th>Reverse Order</th>
<th>W/O Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$r$ (%)</td>
<td>76</td>
<td>68</td>
<td>79</td>
</tr>
<tr>
<td>$p$ (%)</td>
<td>91</td>
<td>83</td>
<td>90</td>
</tr>
<tr>
<td>miss (%)</td>
<td>24</td>
<td>32</td>
<td>21</td>
</tr>
<tr>
<td>$f$ (%)</td>
<td>0.52</td>
<td>1.41</td>
<td>0.82</td>
</tr>
<tr>
<td>$\text{Mic } P_1$</td>
<td>0.83</td>
<td>0.75</td>
<td>0.84</td>
</tr>
<tr>
<td>$\text{Mac } P_1$</td>
<td>0.85</td>
<td>0.79</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Table 7.3: Event detection results.

Figure 7.3: Example of events detected from the real dataset.

For example, when the window size is set to $s_2$, the quality of the results is not as good as others. The reason is that different events may happen and last for different time. For instance, with the window size of $s_2$, most of the events that only last for a window size of $s_1$ cannot be discovered. From the experimental results, we can observe that changing the order of evolution-similarity based graph cut and semantic-similarity based graph cut does not affect the performance significantly. This observation indicts that the information embedded in the semantic similarity and the evolutionary similarity are complementary to each other. Moreover, it can be observed that the evolution pattern-based similarity improve the event detection results substantially. Note that in the above approaches only the click-through data is used. We do not analyze the content or structure of the Web pages in the click-through data. This indicates that it is indeed possible to detect events using only visitor-centric data.

7.4.3 Examples of Detected Events

In this section, we present a list of detected events and the corresponding semantics in real world. As the size of the click-through data is prohibitively large, we only take as example
CHAPTER 7. EVENT DETECTION FROM EVOLUTION OF CLICK-THROUGH DATA

a collection 32 days query log that is more than 22 GB. Hence, here we only show some representative events.

Figure 7.3(a) shows the related queries about the firework event that happened on the US National day (July 4th). It can be observed that the keyword “firework” and related pages are becoming more popular one week before the event and reach the peak on July 4th. While other related keywords and Web pages such as “firework + buy” and “firework + show” become popular and reach their peaks a few days before July 4th. At the same, the related keywords and Web pages such as “firework + injuries” and “firework + picture” have a little delay in terms of the number of times being issued and visited. This observation explains why we use the dynamic time warping-based similarity measure. Note that this event can be most effectively detected based on a daily basis when we vary the window size in the histogram representation.

Figure 7.3(b) shows the hurricane events that happened in July 2005. It can be observed that the keyword “hurricane” and related pages are becoming more popular during the third, fourth, and fifth weeks starting from June 15, 2005. Observe that related keywords and Web pages such as “hurricane + dennis” and “dennis” become popular and reach their peaks in the fourth week. Also, other related keywords and Web pages such as “hurricane + emily” and “emily” reached their peak in the fifth week. Note that this experiment is done on a week-based time granularity. In this case, we detected the hurricanes “emily” and “dennis” as a single event because the five queries in the figure have similar evolution patterns. However, the hurricanes “emily” and “dennis” are actually two events that happened within two weeks. If we detect the events on a daily-basis time granularity, then the two events can be successfully separated. This is shown in Figure 7.3(c) where the five queries have different evolution patterns. More importantly, “hurricane+emily” and “emily” have similar evolution patterns, while “hurricane+dennis” and “dennis” have similar evolution patterns.
7.5 Summary

In this chapter, we address the issue of semantic extraction from evolution of semi-structured Web data by detecting events from the evolution of click-through data. This work is motivated by the fact that existing event detection approaches only take the author-centric data into account for detecting events, while ignoring the rich collection of visitor-centric data. We believe that the visitor-centric data plays an important role in event detection.

In this chapter, we proposed a novel approach to detect events from the web by analyzing the click-through data of web search engines. A key feature of our approach is that the dynamic nature of the click-through data is incorporated in the event detection process. We have conducted extensive experiments using real click-through data from a commercial web search engine. Results show that our event detection approach can successfully detect many events with high quality.
Chapter 8

Conclusions and Future Work

In this chapter, we conclude this dissertation by summarize our contributions and proposing several directions for future research.

8.1 Summary and Contributions

Web mining has been widely used recently to extract hidden knowledge behind the massive amount of Web data. However, we observed that none of the existing approaches has systematically incorporating the dynamic nature of Web data. In this dissertation, we thoroughly exploited the problem of mining evolution of semi-structured Web data. In particular, we proposed the MONETA framework (Mining evOlutioN of sEmi-strucTured web dAta).

Under this framework, we focused on discovering novel knowledge that cannot be efficiently and accurately discovered using existing Web mining techniques.

The contributions of this dissertation can be summarized as follows.

- We proposed the MONETA framework for mining evolution of semi-structured Web data. Under this framework, three major research issues were identified: representation of historical semi-structured Web data, measurement of dynamic behaviors for parts or segments of Web data, and discovering novel knowledge under the representation and measurement proposed. Note that, this framework can be used to incorporate the temporal and dynamic nature of any data source into the corresponding mining process to discover novel knowledge.
CHAPTER 8. CONCLUSIONS AND FUTURE WORK

- We proposed the general model to represent the historical semi-structured data. We begin with the tree structured Web data and proposed the $H$-DOM representation, which is later extended to the $H$-DOM$^+$ and $H$-QPG-tree structure for tree structured data. Subsequently, graph fusion and vector-based graph are proposed to represent the graph structured data.

- A set of general dynamic metrics were proposed to measure the evolution behavior of Web data segments. Two types of metrics were proposed: the micro-pattern based metrics and the macro-pattern based metrics. Also, the evolution pattern-based similarities were proposed for clustering segments of Web data.

- Different algorithms for discovering novel knowledge from the evolution of semi-structured Web data have been proposed for different types of data. Note that both author-centric data and visitor-centric data are used to evaluate our mining techniques and usefulness of the discovered novel knowledge.

- We have applied the novel knowledge discovered from the evolution of semi-structured Web data in some real applications such as extracting active products in the bidding Website, dynamic-conscious cache for XML query evaluation, and event detection from click-through data. Experimental results show that the novel knowledge we extracted can be used in many business intelligence applications.

Specifically, we focus on discovering the following four types of novel knowledge in the MONETA framework.

- **Substructures with specific evolution patterns:** Similar to existing Web mining approaches, which extract substructures that occurred frequently in the data collection, we propose to extract substructures with specific evolution patterns. These specific evolution patterns can be patterns such as frequently changing patterns, increase/decrease changing patterns, and motif patterns. Note that different types of evolution patterns may be defined as useful in different applications or domains. In
CHAPTER 8. CONCLUSIONS AND FUTURE WORK

This dissertation, algorithms and dynamic metrics have been proposed to extract the three types of substructures. Frequently changing substructures refer to substructures that changed frequently and significantly in the history. Increase/decrease changing substructures refer to substructures that changed in increasing/decreasing manner in terms of frequency and/or significance over time. Motif substructures refer to substructures that never changed or did not change significantly most of the time (if not always) in the history [204, 206, 205, 208, 209].

- Clusters of substructures based on evolution patterns: While existing Web mining approaches focus on clustering Web data based on the content and snapshot structures, we propose to cluster Web data based on the evolution patterns of the corresponding substructures. The intuition is that if two substructures follow similar evolution patterns in the history, then they are expected to be similar to some extent. For example, in this dissertation, we propose to cluster Web access sequences based on their evolution patterns. Moreover, we propose to cluster Website data by integrating the evolution patterns of Web usage data with structure and content information [207, 210].

- Evolution model for substructures: Rather than identifying substructures with specific evolution patterns, we explore the issue of modeling the evolution patterns of substructures. In this dissertation, we propose to build a time-dependent model for the evolution patterns of the query similarity in Web search engine log using the marginalized kernel [212].

- Semantics extracted from evolution patterns: Beside extracting and monitoring evolution patterns of Web data, we move a step forward to explore the underlying semantics behind their evolutions such as real world events. Specifically, we proposed to detect real world events by analyzing the evolution patterns of Web access sequences in the Web usage log data and query and page co-occurrences in Web search engine log data [213].
8.2 Future Research Directions

We believe that our work in this dissertation opens up several important directions for future research on mining semi-structured Web data, especially to incorporate features in other dimensions such as the temporal dimension we handled in this dissertation. We now outline a few questions which we believe are important and will be our future work.

- **Mining Evolution of Web Content and its Combination with Structure**: In this dissertation, we focus on mining the evolution of Web structural data, specifically semi-structured Web data. Another direction is to mining the evolution of Web content and the combination of Web content and structure. To address this issue, different text mining and natural language processing techniques are required to summarized the content in such a way that the massive amount of content can be manipulated efficiently. Knowledge extracted from the evolution of content and its combination with structure can be used in many applications such as biology literature analysis and dynamic-conscious Web search engines. Note that the MONETA framework can also be used for mining evolution of Web content and its combination with structure.

- **Mining Evolution of Multi-Dimensional Graph**: Other than the graphs used in this dissertation, which is one dimensional graph, there are multi-dimensional graphs in many other applications. For instance, for the Web search engine click-through data, currently we model the strength of the query page relationship using the number of clicks and hidden semantics. There are information in the other dimensions such as location, user profile, and context, etc, which can be incorporated to make the time-dependent model more accurate. In this case, we need to address the evolution of multi-dimensional graph. Different evolution patterns in different dimensions and the correlations among the evolution patterns in different dimensions will be critical for mining evolution of multi-dimensional graph. The appropriate dimension selection for different types of applications will also be interesting.
• Mining Evolution of Heterogeneous Graph: In real world applications, to accurately model the semi-structured data, heterogeneous graph is required. That is, the vertices in the graph can be of different object type and the relations between different objects can be different as well. For example, social network can be modelled as an heterogeneous network where there are different objects such as people, company, and non-profit organization and relations such as friends, colleagues, and competitors. Be exploring the evolution of vertices or subgroups in the graph, we can monitor how it evolves and simulate the evolution and predict the future graph. It is useful for improving existing Web search engines and other business intelligence applications.

• Atomic Semantic Change Detection and Change Decomposition: As we mentioned and proved in this dissertation, there are semantics embedded in the evolution of historical Web data and we have discovered some of the semantics. However, the question is that "can we have a knowledge base for a specific domain that includes all the possible semantic change patterns?". As we know that the semantics can be represented in hierarchical structures, we have atomic semantic objects and they can further form larger semantic objects and so on. The objective is to find the hierarchy of semantic changes. To do this firstly, we have to identify all possible atomic semantic changes from the evolution of historical Web data by comparing between versions. Then, by decomposing the existing change patterns using these atomic semantics changes, we can further discover more atomic semantic changes. This process can iterate for many times till we find all the possible atomic semantic changes. Such knowledge base can make the existing change detection work more meaningful and can be used for business intelligence.

• Explore MONETA Framework in Other Vertical Areas: In this dissertation, we focus on different types of semi-structured Web data. But as we mentioned before that in many other areas such as bioinformatics, chemistry, and image and graphic, the data sources can also be modelled as graph structured data. By exploring the
MONETA framework to such vertical areas, with different domain dependent knowledge and requirement, we should be able to make the MONETA framework capable of exploring others types of novel knowledge that we did not addressed. At the same time, some obvious techniques in the Web data mining area may work perfect for other domains but are ignored due to the limitation of individuals knowledge base.

- **Reasoning of Web Dynamics:** From extracting pattern-based substructures, clustering of substructures, to semantic extraction from substructures, the next direction is to reason the factors behind Web dynamics. This research issue is important as if we can model the behaviors of the hidden factors of Web dynamics, then problems such as prediction and outlier detection can be more accurate. However, this issue is challenging for two reasons. Firstly, there may be large amount of noises in the data sources, which may deteriorate the quality of the mining results. Secondly, these factors are hidden variables and we do not even know the exact number of factors and the relations between them.
References


REFERENCES


REFERENCES


REFERENCES

[40] Y. Chi, S. Nijssen, R. R. Muntz, and J. N. Kok. Frequent subtree mining: An overview. *Fundamenta Informaticae, Special Issue on Graph and Tree Mining*, Accepted in 2005.


REFERENCES


REFERENCES


[73] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu. Next Generation Data Mining, chapter Mining Frequent patterns in Data Streams at Multiple Time Granularities. AAAI/MIT, 2003.


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


237
REFERENCES


Appendix A

List of Patents and Publications

The author has contributed a total of 15 international publications with a few waiting review results. The author also has 2 United States patents pending. The list is given below.

Patents


Journal and Book Chapter Publications


Conference and Workshop Publications

2. Qiankun Zhao and Sourav S. Bhowmick and Gruenwald Le. CLEOPATRA: Clustering of Evolutionary Pattern-based Web Access Sequences. To appear in PAKDD, 2006. (accept rate: 14%)


4. Qiankun Zhao, Sourav S. Bhowmick, and Gruenwald Le. WAMMiner: In the Search of Web Access Motifs from Historical Web Log Data. In ACM CIKM, 2005. (accept rate: 18%)


11. Qiankun Zhao and Tie-Yan Liu and Sourav S. Bhowmick and Wei-Ying Ma. Event Detection from Search Engine Click-through Data via Vector-based Graph Cut. In ACM KDD, 2006. (accept rate: 11%)