# HIERARCHAL CLUSTERING ALGORITHM FOR LARGE XML DATA

BY

ABDIRAHMAN MOHAMED ABDI DAUD

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

**MASTER OF SCIENCE**

**In**

**COMPUTER SCIENCE**

**JUNE 2011**

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Abdirahman Mohamed Abdi Daud** under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of
**MASTER OF SCIENCE IN COMPUTER SCIENCE.**

**Thesis Committee**

[illegible]

Dr. Salahadin Adam Mohammed

[illegible]

Dr. Muhammed Saleh Al-Mulhem

[illegible]

Dr. Adel Fadhl Ahmed

[illegible]

Dr. Adel Fadhl Ahmed
(Department Chairman)

[illegible]

Dr. Salam A. Zummo
(Dean of Graduate Studies)

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

24/7/11

Date:

*I sincerely dedicate this thesis to my loving mum and amazing father...*

# ACKNOWLEDGMENTS

*In the name of Allah, Most Gracious, Most Merciful*

All praise is to Allah. I thank him for giving us the ability to learn and share knowledge. I also thank Allah for the golden chance of being taught by Dr. Salahadin Adam Mohammed, the most inspiring professor I ever met. Without Dr. Mohammed's teaching, guidance, and advice this thesis would not have been possible.

I would like also thank my thesis committee members Dr. Adel Fadhl Ahmed and Dr. Muhammed Saleh Al-Mulhem for their time and guidance. I would like to express special thanks to Dr.El-Sayed Mohamed El-Alfy for his valuable feedback and comments. I am grateful to King Fahd University's deanships and centers for providing students excellent utilities, tools, and support to conduct research.

I would like to express my appreciation to my friends who have helped in many ways. Namely, I would like to thank to Mohamed Osman, Awale Said, Waleed Al-Zu'bi, Mohamed Al-Qadi, Ayed Al-Qahtani, Mohammad Amro, and Raed Shammas. My greatest gratitude is due to Mohammad Ayar who we spend days reviewing the thesis book together. May Allah bless you.

My deepest thanks is due to my father who encouraged me from childhood to think independently and pursue my education. Without my mother's love and care, I would not be able to take this and every other step in my life. Thank you for everything.

Abdirahman

June 2011

# TABLE OF CONTENTS

**LIST OF TABLES** **ix**

**LIST OF FIGURES** **x**

**ABSTRACT (ENGLISH)** **xi**

**ABSTRACT(ARABIC)** **xiii**

**CHAPTER 1. INTRODUCTION** **1**

1.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.2 Thesis Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.3 Thesis Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.4 Research Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.5 Thesis Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

**CHAPTER 2. BACKGROUND** **6**

2.1 eXtensible Markup Language . . . . . . . . . . . . . . . . . . . . . . 6

2.2 Data Clustering . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.3 Related Concepts in Data Clustering . . . . . . . . . . . . . . . . . 12

2.3.1 Similarity Functions . . . . . . . . . . . . . . . . . . . . . 13

2.3.2 Null Values . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3.3 Normalization . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3.4 Data types . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.3.5 Quality of Data Clustering . . . . . . . . . . . . . . . . . 16

2.4 BIRCH Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 16

2.4.1 Scalability of BIRCH algorithm . . . . . . . . . . . . . . . 17

2.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

**CHAPTER 3. LITERATURE REVIEW 20**

3.1 Clustering Approaches . . . . . . . . . . . . . . . . . . . . . . 20

3.2 Schema-Based Clustering . . . . . . . . . . . . . . . . . . . . 21

3.2.1 Main Features . . . . . . . . . . . . . . . . . . . . . . . . 21

3.2.2 Advantages of Schema Based Clustering . . . . . . . . . . 23

3.2.3 Limitations of Schema Based Clustering . . . . . . . . . . 24

3.3 Structure-Based Clustering . . . . . . . . . . . . . . . . . . . 24

3.3.1 Main Features of Structure Based Clustering . . . . . . . 25

3.3.2 Advantages and Limitations of Structure Based Clustering 27

3.4 Structure and Content Based Clustering . . . . . . . . . . . . . 27

3.4.1 Main Features . . . . . . . . . . . . . . . . . . . . . . . . 28

3.4.2 Advantages and Limitations . . . . . . . . . . . . . . . . . 28

3.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

**CHAPTER 4. XML HIERARCHAL CLUSTERING ALGORITHMS 30**

4.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

4.1.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . 31

4.1.2 Preliminaries . . . . . . . . . . . . . . . . . . . . . . . . 31

4.1.3 Overview of the Proposed Algorithms . . . . . . . . . . . . 32

4.2 XML Structure Clustering Algorithm (XSC) . . . . . . . . . . . 35

4.2.1 Phase 1: Building XML Data Guide . . . . . . . . . . . . 36

4.2.2 Phase 2: XML Structure Index . . . . . . . . . . . . . . . 41

4.3 XML Hierarchal Content Clustering Algorithm (XHCC) . . . . . 45

4.3.1 Phase 3: XML Univariate Content Clustering . . . . . . . 47

4.3.2 Phase 4: Clustering Transactions by Structure and Content 49

4.4 XHCC Revisited . . . . . . . . . . . . . . . . . . . . . . . . . 52

4.4.1 Normalization . . . . . . . . . . . . . . . . . . . . . . . . . . 52
4.4.2 Threshold Values . . . . . . . . . . . . . . . . . . . . . . . . . 53
4.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54

**CHAPTER 5. EXTENDED BIRCH ALGORITHM (XBIRCH) 55**

5.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
5.2 Clustering Univariate Categorical Values . . . . . . . . . . . . . . 57
5.3 Multivariate Categorical Clustering . . . . . . . . . . . . . . . . . 58
5.4 Priority and Order of Datapoints' Attributes . . . . . . . . . . . . 60
5.5 The Sensitivity to Order . . . . . . . . . . . . . . . . . . . . . . . 64
5.6 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

**CHAPTER 6. EXPERIMENTAL RESULTS AND ANALYSIS 65**

6.1 Performance Parameters . . . . . . . . . . . . . . . . . . . . . . . 65
6.2 Experimental Setup . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.3 Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
6.4 Testing Methodology . . . . . . . . . . . . . . . . . . . . . . . . . 69
6.5 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
6.6 Experimental Analysis . . . . . . . . . . . . . . . . . . . . . . . . 72
6.7 Comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
6.8 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78

**CHAPTER 7. CONCLUSION AND FUTURE WORK 79**

7.1 Thesis Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
7.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

**REFERENCES 82**

**CURRICULUM VITA 91**

# LIST OF TABLES

4.1 Building the XSI for the university dataset . . . . . . . . . . . . . . 42
4.2 XSI for the university dataset . . . . . . . . . . . . . . . . . . . . . . 43
4.3 The possible data types of datapoints . . . . . . . . . . . . . . . . . 47

6.1 General specifications of the main datasets used for experiments . 69
6.2 Detailed specifications of sub-datasets . . . . . . . . . . . . . . . . 70
6.3 Results for Mondial dataset . . . . . . . . . . . . . . . . . . . . . . . 71
6.4 Results for Movies dataset. . . . . . . . . . . . . . . . . . . . . . . . 71
6.5 Results for DBLP dataset. . . . . . . . . . . . . . . . . . . . . . . . 71
6.6 Results for Cars dataset. . . . . . . . . . . . . . . . . . . . . . . . . 72
6.7 Recall per dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72

# LIST OF FIGURES

2.1 An XML document . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.2 An XML document tree representation (XML-tree) . . . . . . . . 9
2.3 A generic high-level architecture for XML data clustering . . . . . 11
2.4 A CF-Tree structure . . . . . . . . . . . . . . . . . . . . . . . . . 18
2.5 A CF -entry for three numbers . . . . . . . . . . . . . . . . . . . . 18

3.1 A scheme to classify XML clustering algorithms into approaches . 22
3.2 A new scheme to categorize XML Clustering approaches . . . . . 22

4.1 Universities dataset before preprocessing . . . . . . . . . . . . . . 33
4.2 Universities after preprocessing . . . . . . . . . . . . . . . . . . . 34
4.3 Overview of XSC . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.4 university XDG . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.5 Overview of XHCC . . . . . . . . . . . . . . . . . . . . . . . . . . 46
4.6 The University XDG in Phase 3 . . . . . . . . . . . . . . . . . . . 46

5.1 Effect of attribute order on XBIRCH tree . . . . . . . . . . . . . . 63

6.1 Space Usage of XSC . . . . . . . . . . . . . . . . . . . . . . . . . 76
6.2 Space Usage of XHCC . . . . . . . . . . . . . . . . . . . . . . . . 76
6.3 Running Time of XSC . . . . . . . . . . . . . . . . . . . . . . . . 77
6.4 Running Time of XHCC . . . . . . . . . . . . . . . . . . . . . . . 77
6.5 Recall of INEX 2008 participants . . . . . . . . . . . . . . . . . . 78

# THESIS ABSTRACT

| | |
|---|---|
| **NAME:** | Abdirahman Mohamed Abdi Daud |
| **TITLE OF STUDY:** | HIERARCHAL CLUSTERING ALGORITHM FOR LARGE XML DATA |
| **MAJOR FIELD:** | COMPUTER SCIENCE |
| **DATE OF DEGREE:** | June 2011 |

Data clustering algorithms are widely applied in areas of business, science, and engineering. Examples include marketing, bioinformatics, genetics, medicine, and education. XML data clustering is a hot research area because on the Internet, XML is the most popular format for data exchange. Furthermore, XML data clustering improves the query processing efficiency of the new generation of databases, the Native XML databases. In this thesis we present a survey of existing XML clustering algorithms. Then we propose a new XML clustering algorithm that clusters XML data based on its structure and content. The BIRCH algorithm, a popular hierarchal clustering algorithm used by our algorithm, is extended to work with categorical values. Experiments showed that our algorithm requires only two scans to cluster XML data. The experiments also showed that the phases of the proposed algorithm have linear time complexity and sub-linear space complexity. On the average, the recall of the our algorithm is 89.5 % which is high recall value in the field of XML data clustering. To the best of our knowledge, this is the first algorithm which produces hierarchical clusters of XML data by both structure and content for large homogeneous XML datasets.

**Keywords:** *XML, Data Clustering, XML Clustering, Data Mining, Algorithms, Web Mining, XML databases*

## خلاصة الرسالة

**الاسم:** عبدالرحمن محمد عبدي داود

**عنوان الرسالة:** خوارزمية تجميع عنقودي لبيانات لغة اكس-ام-ال ذات الحجم الكبير

**مجال التخصص:** علوم الحاسب و المعلومات

**تاريخ التخرج:** يونيو، 2011

يتم استخدام خوارزميات تجميع البيانات في مجالات واسعه منها الأعمال التجارية ، والمجالات العلمية والهندسية. ومن الأمثلة على ذلك التسويق ، والمعلوماتية الحيوية وعلم الوراثة و الطب و التعليم. تجميع بيانات لغة اكس-ام-ال هي من البحوث النشطة لان لغة اكس-ام-ال هي التنسيق الأكثر شعبية لتبادل البيانات على شبكة الانترنت. علاوة على ذلك، فان تجميع بيانات لغة اكس-ام-ال يحسن من كفاءة معالج استعلام الجيل الجديد من قواعد البيانات ، قواعد بيانات لغة اكس-ام-ال.

في هذه الرسالة نقدم دراسة استقصائية لخوارزميات تجميع البيانات الغير منظمة و الشبة منظمة. ثم نقترح خوارزمية جديدة لتجميع بيانات لغة اكس-ام-ال والتي تستخدم لحفظ هذه النوعية من البيانات. أيضا في هذه الرسالة سنقوم بتوسيع خوارزمية بيرش ذات الخاصية الهرمية لتشمل القيم الغير رقمية. تبين التجارب ان الخوارزمية المقترحة في هذه الرسالة تقوم بمسح البيانات مرتين فقط للقيام بالتجميع العنقودي لهيكل و محتوى لغة اكس-ام-ال. و تبين التجارب أيضا أن اجزاء الخوارزمية المقترحة لديها نمو خطي في التحليل الزمني و نمو دون المستوى الخطي في تحليل المساحات، الأمر الذي يجعلها مناسبة للبيانات ذات الحجم الكبير. وقد بينت التجارب أن معدل دقة الخوارزمية المقترحة هو 89.5 بالمئة وهي نسبة عالية في مجال تجميع البيانات الغير منظمة و الشبه منظمة.

الكلمات الرئيسية : لغة الرقم القابلة للامتداد، التحليل العنقودي، تحليل البيانات ، تحليل لغة الرقم القابلة للامتداد ، استخراج البيانات، الخوارزميات ، تحليل الشبكة العنكبوتية، تحليل قواعد البيانات

# CHAPTER 1

# INTRODUCTION

Researchers have extensively studied the clustering of structured data, but the clustering of semi-structured and unstructured data was given very little attention. This is due to the lack of file structure, before XML, that can store semi-structured and unstructured data. Today XML data clustering is a young and an active research area that attracts many researchers because on the Internet, XML is the most popular format for data exchange, and to improve the query processing efficiency of Native XML databases (NXD). Unlike Relational Databases (RDB), NXD can natively store and manage semi-structured and unstructured data. RDBs are limited to structured data only. Like in RDB, mining data in NXD using efficient techniques is essential. However, mining XML data is different than mining RDB data. This is mainly due to the fact that XML combines both structure and content of data in a combined manner that makes it harder to process.

In addition to the common benefits of data clustering, clustering XML data can be used to improve the performance of Information Retrieval (IR) and database systems. For example in databases, instead of searching the whole database, only a portion of it (a cluster) will be searched.

There are different approaches for XML data clustering. In the past, the focus was to cluster XML data based on structure only. Today, researches are looking at the content of XML data to increase the cluster quality, and to support more applications.

The rest of this chapter is organized as follows. The thesis problem statement is presented in Section 1.1. Section 1.2 states the thesis objectives. The thesis contributions are presented in Section 1.3. Section 1.4 discusses the thesis research methodology. Finally, the thesis outline is presented in Section 1.5.

## 1.1 Problem Statement

There are already a number of XML data clustering algorithms. While most algorithms are designed for heterogeneous datasets, a few target homogeneous ones and most of these require high computation resources. Currently, there are two applications where large homogeneous datasets are processed. These are XML databases (NXD) and clustering XML streams. Our study focuses on the problem of designing a scalable clustering algorithm for large homogeneous XML datasets.

## 1.2 Thesis Objectives

Up to the date of writing this thesis, there is no clustering algorithm that has all the following four properties:

- An algorithm that clusters by structure AND content
- A hierarchal clustering algorithm
- An algorithm that clusters homogeneous data
- An algorithm that can cluster any size of XML data with few scans

The objective of this thesis is to develop an XML clustering algorithm which has the above four properties.

## 1.3 Thesis Contributions

The contributions of our thesis are as follows:

1. We present a survey of existing XML clustering algorithms. The survey is presented in Chapter 3.
2. We propose a new clustering algorithm which clusters XML data by structure. We call it: XML Structure Clustering Algorithm (XSC). XSC is explained in detail in Chapter 4.

3. We propose a new XML Hierarchal Content Clustering Algorithm (XHCC). XHCC extends XSC to cluster the content of XML data. It is also presented in Chapter 4.

4. We propose a new clustering algorithm called XBIRCH. XBIRCH extends the famous BIRCH algorithm. The BIRCH algorithm only clusters numerical data, while XBIRCH can cluster numerical and categorical data. XBIRCH is explained in Chapter 5.

5. We implemented XSC, XHCC and XBIRCH algorithms.

6. We performed many experiments to study the performance of the proposed algorithm. We present the experimental results and analysis in Chapter 6.

## 1.4 Research Methodology

In order to achieve the thesis objectives, our research went though six phases. These are as follows:

**Phase One: Literature Survey**

A complete literature survey was conducted in the area of XML clustering. At the end of this phase, a survey was presented.

**Phase Two: Specification of the Problem Statement**

In this phase the problem statement was formally specified and analyzed. The output of this phase was the thesis proposal.

**Phase Three: Specification of the Proposed Algorithms**

In this phase the proposed algorithms were specified and analyzed before implementation. The output of this phase was the pseudo-code of the proposed algorithms.

**Phase Four: Implementation of Proposed Algorithms**

In this phase all the algorithms were implemented. The output of this phase was a complete running application that implements the proposed algorithms.

**Phase Five: Testing**

In this phase we tested the proposed algorithms using large XML benchmark datasets.

**Phase Six: Performance Analysis**

After experiments have been conducted, the results were gathered and analyzed in order to study the performance of the proposed algorithms.

## 1.5 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 gives background information on XML, data clustering, and related topics. The literature review is discussed in Chapter 3. Chapter 4 explains XHCC and XSC. Chapter 5 presents XBIRCH. After that, the results and analysis of the experiments are explained in Chapter 6. Finally, Chapter 7 concludes our work.

# CHAPTER 2

# BACKGROUND

This chapter offers the reader background information that is needed to fully understand the thesis work. This chapter is organized in the following way. First, a background on XML is given in Section 2.1. This is followed by an introduction to data clustering in Section 2.2. Section 2.3 discusses some related data clustering concepts. Section 2.4 briefly describes the BIRCH algorithm, a popular hierarchal clustering algorithm. Finally, Section 2.5 concludes the chapter with a summary of the previous sections.

## 2.1 eXtensible Markup Language

XML stands for eXtensible Markup Language which is a language for storing semi-structured and structured data. A small XML document is shown in Figure 2.1.

An XML document basically consists of the following components:

- Elements: Each element represents a logical component of a document. Elements can contain other elements and/or text (character data). The boundary of each element is marked with a start tag and an end tag. A start tag starts with the "<" character and ends with the ">" character. An end tag starts with "</" and ends with ">". The root element contains all other elements in the document. In the XML document shown in Figure 2.1, the root element of the document is the "paper" element. Children of an element are elements that are directly contained in that element. For example, in Figure 2.1 the *title* element is a child of the *paper* element. In some XML documents, the element is not enough to describe its content. Such documents are called text-centric documents.

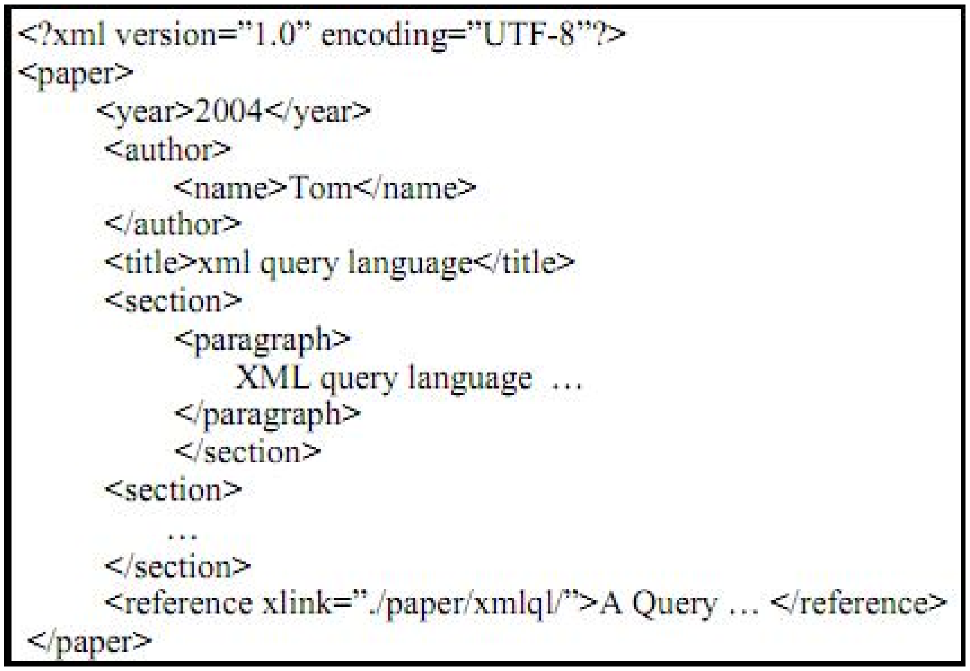
- Attributes: Attributes are descriptive information attached to elements. The values of attributes are set inside the start tag of an element. For example, in Figure 2.1, the expression <reference xlink="./paper/xmlql" > sets the value of the attribute xlink to "./paper/xmlql". The main differences between elements and attributes is that attributes cannot contain other attributes or elements.

- Values: Values are sequences of characters which appear between elements' start-tag and end-tag. Like attributes, values cannot contain elements. In Figure 2.1, the expressions "2004" and "Tom" are examples of values.

  Due to its nested structure, XML is commonly modeled as a rooted and labeled tree. Nodes of the tree correspond to elements, attributes and text in XML documents. Edges represent element-subelement, element-attribute and element-text relationships. This tree model reflects the logical structure of an XML document and can be used to store and query XML data [1]. For the sample XML document shown in Figure 2.1, its tree representation is shown in Figure 2.2. A path is a series of ordered nodes between the root node and an internal or a leaf node. An example of a path in Figure 2.2 is the path "/paper/author/name". For detailed information about XML, please refer to the W3C XML specification in [2].

An XML document is a self-describing document. XML elements can either be simple or complex. Simple elements contain only values or attributes. On the other hand, complex elements can additionally contain other elements and therefore a nesting structure is formed. This structure can have any level of nesting.

Some XML documents have to conform to a Document Type Definition (DTD). DTD specifies the elements, the attributes and the structure of an XML document. Unlike relational database tables, XML documents are semi-structured. A newer specification for XML documents is the XML schema. The XML schema can impose more constraints on an XML document than the DTD. It also has a hierarchal structure that specifies the name and the data type of XML elements. The flexibility of defining the XML structure makes XML able to represent any kind of data but it also makes it more difficult to process.



```
<?xml version="1.0" encoding="UTF-8"?>
<paper>
        <year>2004</year>
        <author>
               <name>Tom</name>
        </author>
        <title>xml query language</title>
        <section>
               <paragraph>
                     XML query language  …
               </paragraph>
               </section>
        <section>
               …
        </section>
        <reference xlink="./paper/xmlql/">A Query … </reference>
 </paper>
```

Figure 2.1: An XML document

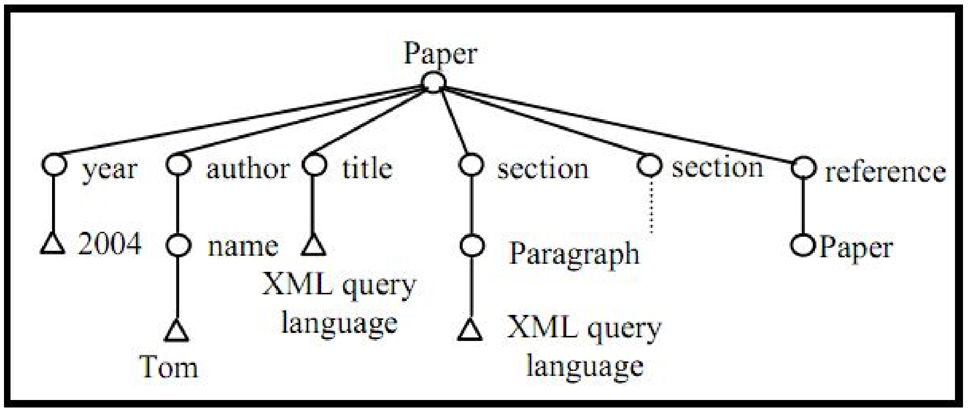


Figure 2.2: An XML document tree representation (XML-tree)

## 2.2 Data Clustering

Data clustering is defined as the problem of grouping similar objects such that similarity between objects of the same group is higher than the similarity between objects of other groups.

There are several algorithms for clustering XML data. Figure 2.3 shows a generic high-level architecture of XML data clustering algorithms [3]. Nearly all XML clustering algorithms follow this architecture. In this architecture, first the XML dataset is read. The dataset can be XML documents or XML schema or both. Secondly and optionally, the data is represented in a model such as a tree model or Vector Space Model (VSM). After that, a similarity function measures the distance between any two XML objects, or parts of the model. Finally, these objects are grouped as an array of clusters or as a hierarchy structure.

In this section we introduce the main approaches of clustering algorithms. Also, we introduce three concepts related to data clustering, namely, similarity functions, null values, and scalability. The main data clustering approaches are as follows:

**Partitioning Approach**: Algorithms that follow this approach start by taking n data points and then classifying them into k (k>n) partitions. Examples of this approach are k-means, k-medoids and CLARANS [4].

**Hierarchical Approach**: a hierarchical approach creates a hierarchical decomposition of the given set of data objects. It can either be done from top-down (divisive) or bottom-up (agglomerative). Hierarchical approaches result in creating a tree that holds a cluster of clusters.

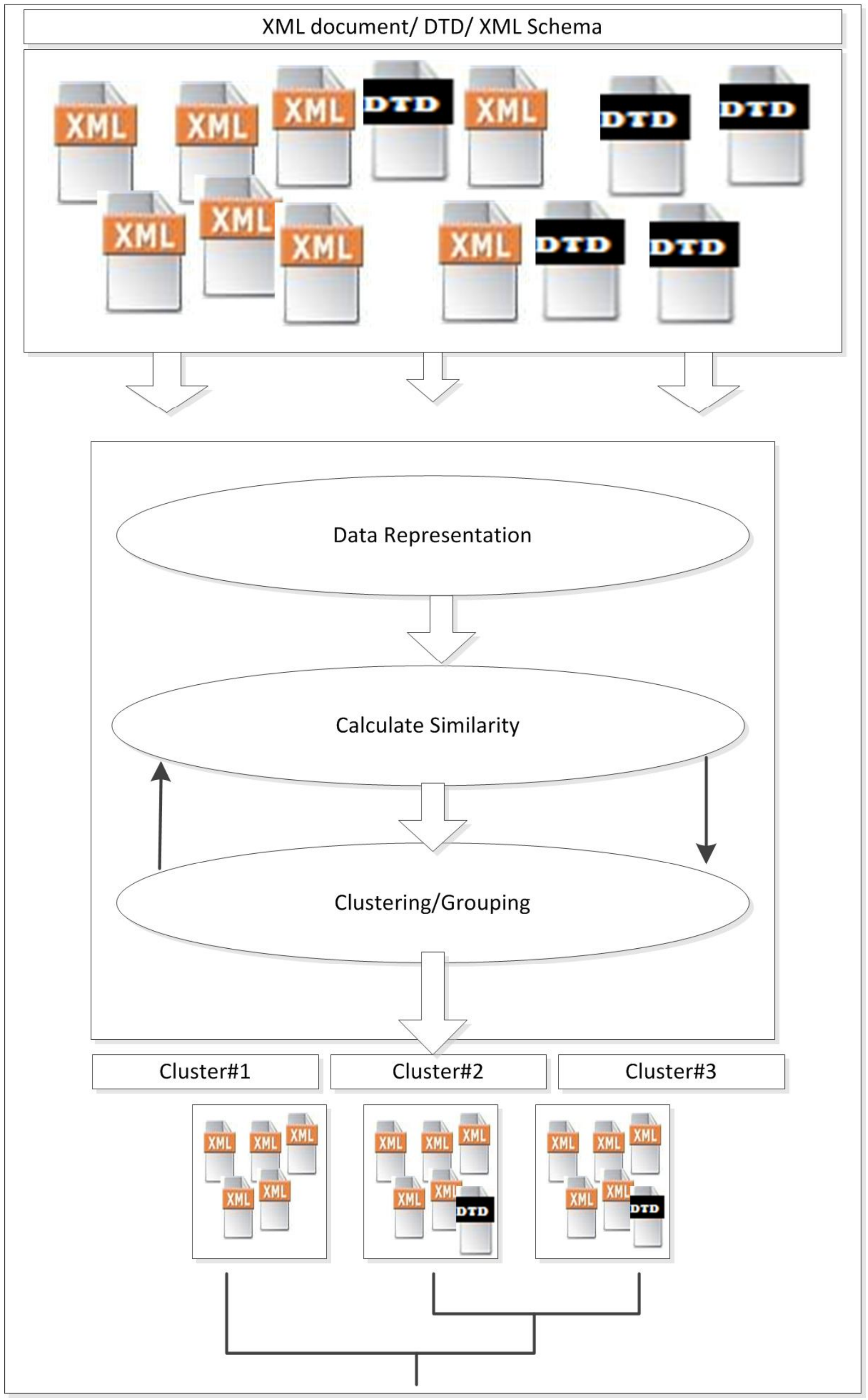


Figure 2.3: A generic high-level architecture for XML data clustering

One example of the hierarchical approach is the BIRCH algorithm (Balanced Iterative Clustering using Hierarchies). BIRCH in its first phase creates a tree that summarizes the input data. This tree is called the Clustering-Feature tree ( CF-tree). A single node in the BIRCH tree has a few attributes that summarize the statistical features of its descendant nodes [5].

**Density-based approach**: The idea of this approach is to continue growing a given cluster as long as the density (number of objects or data points) in the neighborhood does not fall below a certain threshold. Examples of this approach include DBSACN, OPTICS and DenClue [4].

**Grid-based approach**: The algorithms of this approach rely on creating a grid structure. This grid is finite and created by quantizing the data object space. This approach is known to be efficient [4].

**Model-based approach**: The algorithms of this approach use machine learning techniques that learn from the distribution of data points. Examples of this approach are self-organizing feature map and COBWEB [4].

## 2.3 Related Concepts in Data Clustering

In this section, we will describe similarity functions, null values, normalization, data types, and quality of data clustering.

### 2.3.1 Similarity Functions

A similarity or distance function finds the similarity between two objects. There are several similarity functions. The choice of a similarity function depends on the type of data. For example, in the case of numerical values, Manhattan or Euclidean distance functions might be applied (see Equations 2.1 and 2.2). In the case of categorical values, a hamming distance function is used (see Equations 2.3).

$$Euclidean(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2} \tag{2.1}$$

$$Manhattan - Distance(x, y) = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n| \tag{2.2}$$

$$Hamming(x, y) = the\ number\ of\ components\ that\ x\ and\ y\ differ \tag{2.3}$$

### 2.3.2 Null Values

One issue with data clustering is handling null values. In literature if two values are null, they are equal [6]. Subsequently, a null value and a non-null value are dissimilar. Therefore, the distance functions mentioned above are modified to include Equation 2.4 below:

$$distance(i, j) = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are both null} \\ 1 & \text{if either } i \text{ or } j \text{ is null and } i \neq j \\ c & \text{if } i \text{ and } j \text{ are both not null} \end{cases} \tag{2.4}$$

where c is a value between 0 and 1.

### 2.3.3 Normalization

Normalization refers to the process of scaling a value to fall within a specified range. For example, before normalization, *age* ranges between 0 and 150 and *salary* ranges between 1000 and 50,000. After normalization, both *salary* and *age* will fall into the same range, 0 to 1.

Clustering without normalizing the input data results in bad quality clusters [7]. For example, assume Ali's age is 60 and his salary is SAR 10,000; Sami's salary is SAR 9800 and his age is 20; Bandar's age is 60 and his salary is SAR 9800. Ali and Bander have similar age and similar salary, so they should be in the same cluster. Sami who is 40 years younger than the older guys should be in a different cluster. However,However if clustering is done before normalization, then Sami and Bandar will be in the same cluster while Ali will be in a separate cluster. Putting Sami who is 20 years old, and Bandar who is 60 years old in the same cluster doesn't make any sense. However, if clustering was done after normalization, then the older men will be in the same cluster while the young Sami will be in a separate cluster. So normalization before clustering results in better quality clusters.

### 2.3.4 Data types

XML data can be categorized into four types:

- *Continuous:* Variables of a continuous data type are mainly numeric. They draw their values from an infinite domain. These are represented in decimal format.

- *Categorical:* Variables of a categorical data type draw their values from a distinct and finite domain. There are two types of categorical data types, namely, nominal and ordinal. For example, the values: “male” and “female” are of the categorical type.
  - *Nominal:* A variable of this type draws its values from a finite, distinct and unordered list of values. For example, the values “blue”, “red” and “green” are of the nominal type.
  - *Ordinal:* A variable of this type draw its values from a finite, distinct and ordered list of values. The difference between nominal and ordinal values is that ordinal values can be put in order. For example, the values “freshmen”, “junior” and “senior” are of the ordinal type.
- *Textual:* Textual Values are readable text which can vary from one sentence to a whole paragraph. A title of a book such as “Introduction to Machine Learning” is an example of a textual value.

The proposed algorithms deal with all these data types except for textual data. Ordinal values are treated like numbers. The mapping from ordinal values to numbers can be done either by the help of the user or by XML schema files which specify values of an element with their proper order.

### 2.3.5 Quality of Data Clustering

There are different measurements to evaluate the quality of data clusters. Two of these measurements, precision and recall are shown in Equations 2.5 and 2.6. In these equations, there are four terms: TP, FP, TN and FN. They stand for: True Positive, False Positive, True Negative, and False Negative respectfully.

$$Precision = \{\frac{TP}{TP + FP}\} \tag{2.5}$$

$$Recall = \{\frac{TP}{TP + FN}\} \tag{2.6}$$

Precision measures what percentage of the results was correct while recall measures what percentage of correct results was found. Both measurements should be high to indicate a high quality of the final clusters.

## 2.4 BIRCH Algorithm

BIRCH stands for Balanced Iterative Clustering using Hierarchies. As explained in the previous section, BIRCH clusters incoming multi-dimensional data points to produce quality clusters with the available memory. BIRCH uses the concept of Cluster Feature (CF) to condense information about sub-clusters of points. The Cluster Features are organized in a height-balanced tree called the CF-tree. The algorithm makes full use of available memory and requires at-most two scans of the input data. BIRCH clusters only numeric data and as a result it uses similarity functions like Euclidean or Manhattan.

A CF-tree node corresponds to a cluster and is represented by a CF entry. A

CF entry consists of three numbers, namely, N, LS and SS where N is the count of the data points in the cluster, LS is their summation, and SS is their squared summation. These three numbers summarize the features of data points in a cluster [5]. For Example, assume Cluster C contains the numbers 3, 4 and 5. The CF entry of C is as shown in Figure 2.5.

CF tree has two parameters: branching factor B and threshold T. B is the maximum number of clusters that can be clustered in a a non-leaf node of the CF-tree. In other words, each nonleaf node contains at most B entries of the form [$CF_i$;$child_i$], where i =1, 2 . . . B. $child_i$ points to its i-th child node whereas $CF_i$ is the CF entry of the cluster represented by this child. A CF entry in a nonleaf node summarizes all the CF entries of one of its child nodes. The threshold parameter, T, corresponds to the maximum distance allowed between any two data points of the same cluster. An example of a CF-tree is shown in Figure 2.4.

### 2.4.1 Scalability of BIRCH algorithm

The CF-tree size is a function of T. The larger T is, the smaller the CF-tree. That is why BIRCH is a scalable algorithm. If the memory of a system is low, the threshold is increased and thus the tree can fit the available memory.

Like a B+-tree, a CF-tree is built dynamically when new data objects are inserted. It guides a new insertion into the correct cluster for clustering purposes just like a B+-tree guides a new insertion into the correct position for sorting purposes. For more details on the BIRCH algorithm please refer to [5].

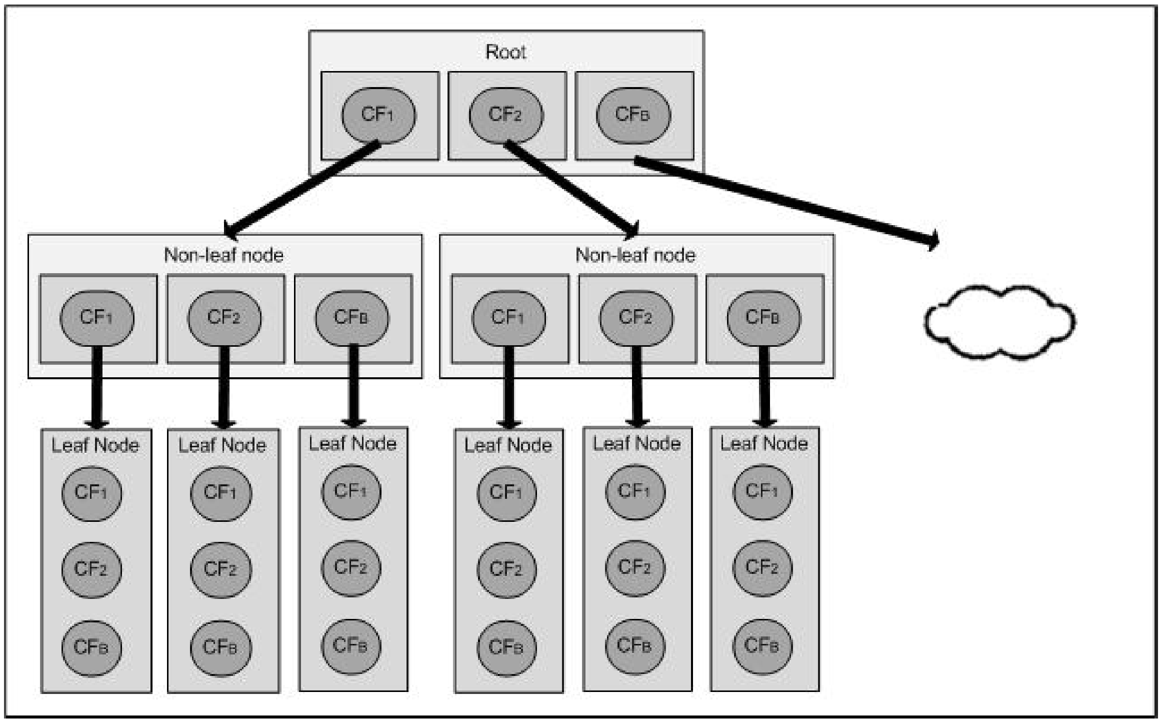


Figure 2.4: A CF-Tree structure

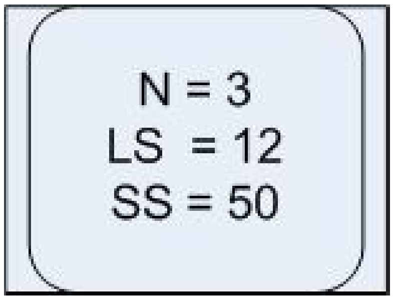


Figure 2.5: A CF -entry for three numbers

## 2.5 Conclusion

In this chapter we presented background information needed to understand the rest of the thesis. The chapter covered the basics of XML and data clustering. In addition, the famous BIRCH algorithm was explained in detail because we are going to extend it and use it in the proposed algorithms.

# CHAPTER 3

# LITERATURE REVIEW

This chapter is a survey of XML clustering algorithms. In the survey, we classify the algorithms into three clustering approaches and then we discuss each approach in a separate section. This chapter is organized as follows. In Section 3.1 we introduce the proposed three clustering approaches. Then the features, advantages, and limitations of each approach is discussed in the next three sections: 3.2, 3.3 and 3.4. The chapter concludes by comparing the approaches with our proposed clustering algorithms.

## 3.1 Clustering Approaches

Up to the writing of this thesis, there was only one survey of XML clustering algorithms [3]. The surveyors classified XML clustering algorithms based on two parameters: the type of XML file ( schema or document) and the way the data is represented (tree or VSM). Figure 3.1 shows their scheme.

In this thesis we use one parameter to classify XML clustering algorithms. Our classification parameter is *the type of XML information that is processed by an algorithm.* This classification parameter makes the classification clearer and easier to use. One advantage of this classification criteria is that it helps us to choose the best clustering approach for any application. This will be shown in subsequent sections.

The proposed clustering approaches are:

- Schema-Based Clustering
- Structure-Based Clustering
- Structure and Content-Based Clustering

The proposed approaches are shown in Figure 3.2. As it can be seen from the figure, the approaches are ordered from top to bottom based on their complexity. The most complex approach is structure and content based-clustering. Each approach can optionally include the one above it. For example, an algorithm of the second approach is based on the XML structure. This algorithm can also use schema information as well.

## 3.2 Schema-Based Clustering

### 3.2.1 Main Features

The fastest way to cluster XML data is by using its schema only. As explained in Chapter 2, a schema file contains only the definition of an XML document.

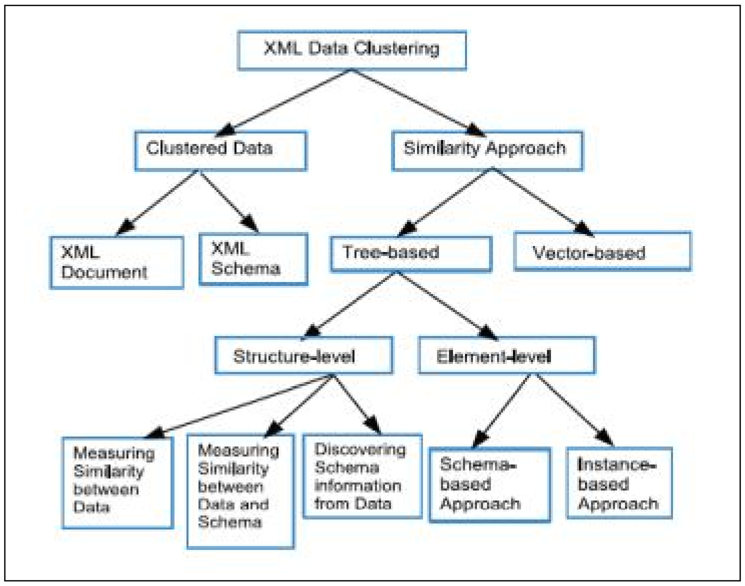


Figure 3.1: A scheme to classify XML clustering algorithms into approaches

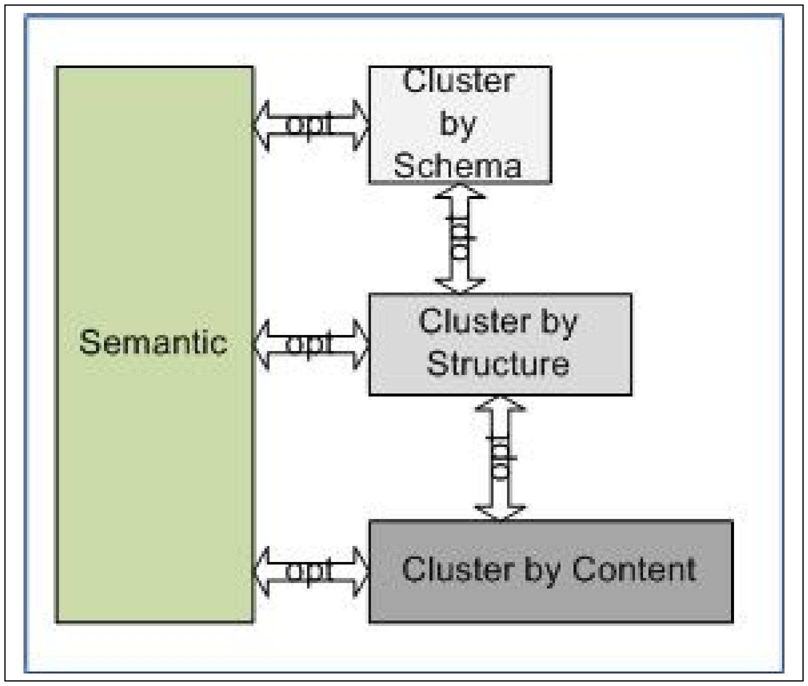


Figure 3.2: A new scheme to categorize XML Clustering approaches

Some algorithms of this approach calculate the similarity between an XML document and an XML schema to put the document in the right cluster. Nearly all the algorithms in schema based clustering use semantic analysis to measure the distance between XML instances [8–15]. There is a need to include and test clustering XML schema without the use of semantic analysis, because some XML datasets use labels which do not have a semantic meaning at all. This issue has been studied in [16].

The algorithms in this approach can further be classified into two classes: those that use XML documents to gain more insight on the data and those which only use XML schema. For example, "Clustering XML Documents Based on Structural Similarity" is the title of a paper by Yang et al. which uses a schema based approach [17]. Their argument is that the schema defines the structure of the XML document. Therefore, instead of comparing two XML documents, each XML document is compared to a schema. Since many XML datasets do not have a schema, they introduced algorithms to generate a schema for schema-less XML files. A Similar approach has been taken by [18, 19].

### 3.2.2 Advantages of Schema Based Clustering

The main advantage of this approach is the small amount of data that is processed. The size of an XML schema is vastly less than the size of an XML document. Since the amount of processed data is small, it is justifiable to use complex algorithms and tools such as semantic measurements. Another advantage is that once an

XML schema is classified into a cluster, all current and future instances of the schema will be grouped into the same cluster [3]. For example, suppose $a$ and $b$ are two XML documents. Suppose both $a$ and $b$ are defined by a schema file $i$. Once $i$ is clustered, both $a$ and $b$ will have the same label of $i$.

An application of XML schema clustering is grouping an extensively heterogeneous environment such as the web. It can also be a pre-processing step for the other two approaches in order to boost their efficiency. In conclusion, clustering by schema is a suitable option for large XML datasets provided that they are heterogeneous.

### 3.2.3 Limitations of Schema Based Clustering

It is impossible to use this approach in a homogeneous environment such as an XML database. This is because in a homogeneous environment, all XML datasets will have a single schema. Another major disadvantage is that this approach aims to cluster the roots of XML datasets only, not within an XML file. It only separates and groups $n$ XML documents into $x$ number of clusters but does not cluster or process XML elements within a document.

## 3.3 Structure-Based Clustering

Clustering by structure is to group XML documents by the similarity of its structural information.

### 3.3.1 Main Features of Structure Based Clustering

Structural similarity can be any or all of: path similarity, parent/child relationships or the data types and names of XML elements. Sometimes semantics of XML element titles are also used. The only data which is not used is the content of XML elements. Among the three approaches of clustering XML data, structure based clustering is the most popular [20].

The algorithms in this approach can further be classified into three subcategories, namely tree-edit based, similarity based and unique algorithms. The tree-edit based algorithms deal with XML documents as a tree while the other two transform XML into different representations. Some of the algorithms of this approach consider the semantic information of the element names [21, 22].

In tree-edit based algorithm, to find how much two XML trees Tree1 and Tree2 are similar, Tree1 is edited until it becomes identical to Tree2 [1,23–31]. The similarity between Tree1 and Tree2 is measured by the number of edit operations needed to make Tree1 structurally identical to Tree2. The fewer operations measured, the more similar Tree1 is to Tree2. These operations can be inserting, updating, or removing a node or a sub-tree [32].

Tree-edit based algorithms have polynomial running time complexity. To reduce the number of compared nodes (XML elements in a tree), [33] makes the edit operation on sub-trees instead of single nodes. This has better performance than manipulating the individual nodes alone. It also removes repetitive sub trees to further reduce comparison time. Still, the running time complexity is polynomial since it is a pair-wise operation. A complete list of edit-based algorithms is found in [32].

The second category is similarity based which focuses on some or all parts of the XML document. While some algorithms deal with path similarity [34–36], others deal with level similarity [37]. More similarity based algorithms can be found in [38–43].

There are unique algorithms that cannot be grouped together. For example, there are solutions that perform sequence mining [44,45]. Another unique solution uses the Fuzzy C-means algorithm and clusters XML data in a multilevel format [46]. Other solutions in this category focus on solving a sub-problem in structural clustering such as in [47]. This work addresses the problem of high dimensionality of XML structural data and proposes a method to focus on important XML elements of a dataset.

The matching technique in [48] is a unique solution since it is based on the concept of entropy. Entropy is a measurement of how random or regular a list of points is. The solution first encodes the XML files and then finds the entropy between the XML documents. This entropy-based solution has similar accuracy to the tree-edit based algorithms. However, it is more scalable since it runs in linear time.

### 3.3.2 Advantages and Limitations of Structure Based Clustering

The complexity of the structural-based approach is higher than the schema based approach. Its complexity ranges from linear time [1, 8, 27, 28, 39, 43] to quadratic time or higher which is the case in tree-edit based approach.

Structural based approaches are useful when the tag names of XML document are meaningful and the document is not text-centric. In a text centric XML document, the semantic information is located in the content of the elements (in the values). Similar to the schema-based approach, structural based clustering does not perform well well in clustering homogeneous data [20].

Generally, when schema based clustering fails to accurately cluster heterogeneous XML datasets, structural-based clustering should be used.

## 3.4 Structure and Content Based Clustering

To solve the problem of clustering homogeneous documents, the content of XML documents have to be processed and not only the structural information [20]. This is because content carries most of the information especially in text-centric XML documents.

### 3.4.1 Main Features

The algorithms in this approach are relatively new and most of the research started in year 2006. One example of content based clustering is the work in [17]. It simply extends the vector space model to handle content data and then applies a hierarchal clustering algorithm used in document clustering.

Some works incorporate semantic information which might increase the accuracy but take longer computation times [11, 49–52]. The work in [53] is unique since it uses Self-Organizing-Maps which is computationally an expensive algorithm. Similarly, the work in [54] applies Latent Semantic Kernel which has a high computational complexity. This is because it performs a pair-wise comparison.

A novel approach is presented in [50] where XML data is clustered using an XML-summarization model, named XCLUSTER. The algorithm effectively clusters XML elements based on both structure and content. Then the values are compressed using histograms.

### 3.4.2 Advantages and Limitations

A common drawback of the algorithms of this approach is their high time and space complexity when compared to the algorithms of the other approaches. Mostly, algorithms in this approach have an expensive preprocessing step where data has to be processed to perform stop and stemming tasks especially in textual XML documents. Very few algorithms in this approach have a low complexity [17, 55, 56]. However, it is generally assumed that near-linear-time-complexity algorithms have lower accuracy [3]. Therefore, the problem of XML clustering is an optimization problem where our aim is to reduce the time and space complexity and get higher accuracy.

It is important to note that some algorithms in this approach are designed for heterogeneous datasets and not homogeneous ones [20, 50, 51, 54, 57].

## 3.5 Conclusion

In this chapter we presented a survey of XML clustering algorithms. We classified the algorithms into three approaches according to the type of XML information that is processed by the algorithm. The approaches are: schema based clustering, structure based clustering, and structure and content based clustering. Our proposed clustering algorithm belongs to the last approach.

Clustering XML by content has always been ignored until recently. As a result, up to the date of writing this thesis, there is no single work that has all of the following properties:

1. Clustering by content and structure
2. Clustering using hierarchal clustering
3. Clustering homogeneous data collections
4. Clustering large XML datasets

Our aim of this thesis is to introduce a new XML clustering algorithm that has all the four properties above.

CHAPTER 4

# XML HIERARCHAL CLUSTERING ALGORITHMS

The main objective of our thesis is to introduce a scalable clustering algorithm for large XML datasets. Since our algorithm consists of several algorithms, it will be referred to as the *proposed algorithms*. In this chapter, we will explain the proposed algorithms in details. The proposed algorithms are XML Structure Clustering Algorithm (XSC) and XML Hierarchal Content Clustering Algorithm (XHCC). The rest of this chapter is organized in the following order. First, the proposed algorithms are introduced in Section 4.1. Then we highlight the main phases of each algorithm. Each of these phases is explained in depth with examples in Section 4.2 and Section 4.3. Finally, in Section 4.4 we revisit XHCC and discuss the following issues: categorical data, normalization, and threshold values.

## 4.1 Introduction

Before describing the proposed algorithms, we shall first introduce our problem statement. Then the related assumption and definitions are explained. Finally, we introduce the phases of the proposed algorithms.

### 4.1.1 Problem Statement

Given a large homogeneous XML dataset, we would like to cluster its content and structure. The output should be in a form of hierarchal clusters.

To understand the limitation and scope of our problem statement, a number of assumptions and definitions are explained next in preliminaries.

### 4.1.2 Preliminaries

The proposed algorithms have the following assumptions:

**Large XML documents:** Since we are targeting large XML documents, we need to define what is meant by large documents. Large documents are datasets that are too big to be stored in a system's memory.

**Homogeneous XML dataset:** Homogeneous XML datasets share the same schema file. As mentioned in Section 3.1, the problem of classifying heterogeneous XML datasets has been studied in the literature [20]. The challenge nowadays is to efficiently and accurately cluster homogeneous XML documents.

**Targeted Data Types:** In Section 2.3.4, different types of data have been explained. The proposed algorithms process all data types except for textual values.

**XML Transaction:** An XML transaction is an instance of an XML element which is important to the user. For example, in the university dataset shown in Figure 4.1, the XML element "university" is the XML transaction for the dataset and there are three XML transactions. These are as follows: Southern University, Northern University, and Eastern University.

### 4.1.3 Overview of the Proposed Algorithms

The proposed algorithms are: XML Structure Clustering Algorithm (XSC) and XML Hierarchal Content Clustering Algorithm (XHCC). There is a third algorithm, XBIRCH, which is explained in Chapter 5 while XSC and XHCC are further discussed in the subsequent sections. The XBIRCH plays an important role in XHCC algorithm.

XSC algorithm is composed of two phases. In the first phase, XSC extracts the structural features of an XML document. This is achieved by building the XML Data Guide. In the second phase, it creates XML Structure Index (XSI). XSI is a special data structure to group the structural features of an XML document.

The second algorithm, XHCC, is composed of four phases. The first two phases are the same as the two phases of XSC. The third phase of XHCC clusters simple XML elements values while the fourth phase clusters complex XML elements. Both algorithms, XSC and XHCC, require a prepossessing step which is explained next.

```
<university >
  <ID id="1" />
  <name >Southern University </name >
</university >
```

```
<university >
  <ID id="2" />
  <name >Northern University </name >
  <numberOfStudents >150 </numberOfStudents >
</university >
```

```
<university >
<ID id="3" />
  <name >Eastern University </name >
  <numberOfStudents >170 </numberOfStudents >
</university >
```

Figure 4.1: Universities dataset before preprocessing

```
<Universities >
  <university >
    <ID id="1" />
    <name >Southern University </name >
  </university >
  <university >
    <ID id="2" />
    <name >Northern University </name >
    <numberOfStudents >150 </numberOfStudents >
  </university >
  <university >
  <ID id="3" />
    <name >Eastern University</name >
    <numberOfStudents >170 </numberOfStudents >
  </university >
</Universities >
```

Figure 4.2: Universities after preprocessing

### Preprocessing Step

The goal of the preprocessing step is to turn several XML documents into one document with a single root. If the dataset is a single root, then this step is skipped. The step is straightforward and computationally inexpensive.

To elaborate more, suppose we want to cluster the university dataset shown in Figure 4.1. In this dataset, there are three XML documents and thus there are three roots. The XML documents are: University of Southern University, Northern University, and Eastern University. From these three documents, a new XML document is created with a new and single root, *Universities*. This root has three child elements which are the roots of the previous three XML documents. The new dataset is shown in Figure 4.2.

## 4.2 XML Structure Clustering Algorithm (XSC)

The goal of XSC is to cluster XML elements by structure only. In XSC, XML elements with similar structure will be grouped together. The clustering algorithm is composed of two phases. The phases are as follows:

- Phase 1: Building the XML Data Guide (XDG)
- Phase 2: Creating the XML Structure Index (XSI)

Figure 4.3 shows an overview of the XSC algorithm. As can be seen from the figure, XSC scans the XML dataset only once. In the the first phase, an XML

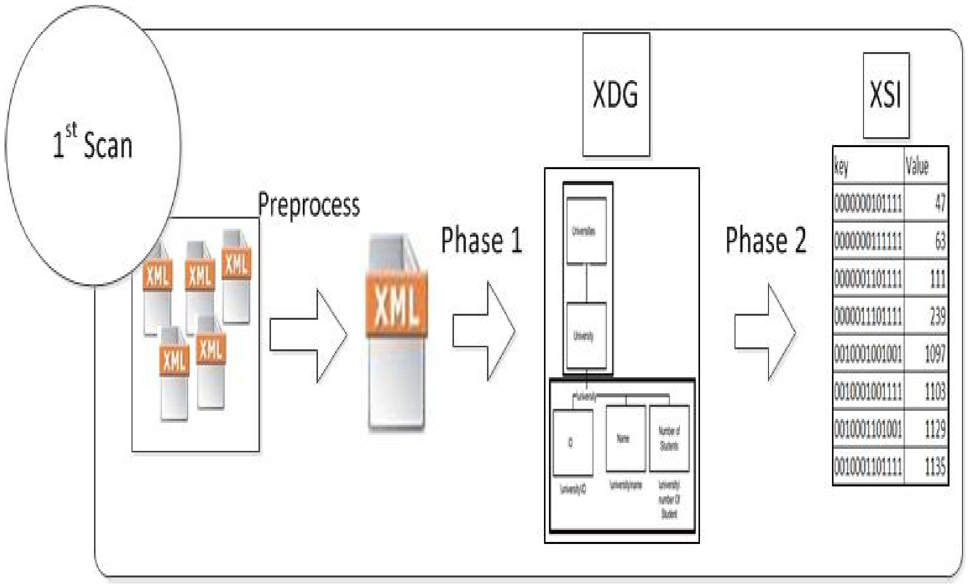
Data Guide (XDG) is created. XDG is a tree that summarizes the XML structure. It is used by the second phase in order to get the list of unique paths of an XML document. In the second phase, the XML Structure Index (XSI) is created. XSI is a data structure that identifies the different or structure information that XML transactions have. Finally, each entry in the XSI is considered a cluster and therefore XSI is the output of the algorithm. Next, all the phases are explained in details.

### 4.2.1 Phase 1: Building XML Data Guide

The XML Data Guide (XDG) is an unbalanced tree that consists of several nodes. The XDG summarizes the structure of the XML document. The XDG preserves the parent-child relationship between XML elements. When an XML element is read from a database or a document, its path is mapped into the nodes of the XDG. For each XML dataset, a single XDG tree is created. Figure 4.4 shows the XDG for the university dataset.

The main purpose of XDG is to store and label the distinct root to leaf paths (DRLP) in an XML document. This information is important for Phase 2: Building the XSI, as we we will see in Section 4.2.2.

Instead of building an XDG, another alternative is to store the DRLP as a list. However, the XDG is more efficient to update and search its structure than a list. Another alternative is to generate XDG from XML DTD or schema. However, around 52% of the XML files are schema-less.



| key | Value |
|---|---|
| 0000000101111 | 47 |
| 0000000111111 | 63 |
| 0000001101111 | 111 |
| 0000011101111 | 239 |
| 0010001001001 | 1097 |
| 0010001001111 | 1103 |
| 0010001101001 | 1129 |
| 0010001101111 | 1135 |

Figure 4.3: Overview of XSC

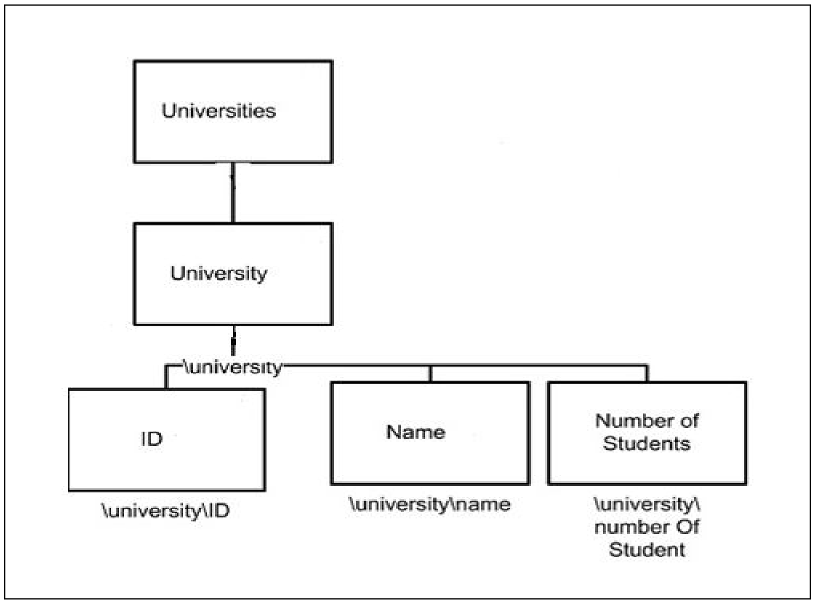


Figure 4.4: university XDG

In addition, a schema file specifies the set of possible DRLP and not the actual set of DRLP that exists in an XML file [58]. Figure 4.4 shows an XDG for the university dataset. This XDG contains five nodes: universities, university, ID, Name and number of students. The set of DRLP of this dataset is as follows:

- DRLP(1): /universities / University / ID
- DRLP(2):/ universities / University / Name
- DRLP(3): / universities / University / NumberOfStudents

Algorithms 1, 2 and 3 listed below explain in detail how the XDG tree is created while XML data is read. If the XDG tree is empty it is initialized with the first XML element: the root. Each node of the XDG has a title which is equal to the path of the corresponding element in the XML document. For example, in the case of the university dataset, the first XML element is “universities”. Therefore, a new XDG node with title “universities” is created. After initialization, the *XDG insert algorithm* is called on the root of XDG tree until all XML elements are processed. The insert algorithm is a recursive procedure. It takes an XML path as an input, say $x_i$. Since $x_i$ is a path, it is composed of a several XML elements' names separated by backslash characters. The algorithm matches each name in $x_i$ with its corresponding nodes in XDG tree. If a matching XDG node is found for the first name in $x_i$, say the node $n_j$, then the insert algorithm returns and starts finding a match for the second name in in the children of the XDG node $n_j$. If any of the elements names of $x_i$ did not match with nodes of XDG tree, a new node is created in XDG. If $x_i$ is a DRLP, the corresponding XDG node is marked as DRLP. This label helps to retrieve the list of DRLP from the XDG tree.

In contrast to algorithms which transform XML data to another representation such as VSM, the XDG tree preserves the structural information, such as parent-child relationships of an XML document.

**Algorithm 1** XDG tree Algorithm

```
ALGORITHM: XDG
  BEGIN
  i ← an XML element
  if i is first element then
    Initialize(T,root,i)
  else
    while i is not null do
      insert (T,i,root)
      i ← next XML element
    end while
  end if
  END
```

**Algorithm 2** XDG Initialize Algorithm

```
ALGORITHM: Initialize (T, i)
  BEGIN
  path ← path of element i
  C ← create new node
  C[title] ← i
  END
```

## Time and Space Complexity of Phase 1

Let us assume that an XDG tree has a branching factor of *"b"* and a depth of *"d"*. The worst case scenario is when each XML element finds a matching XDG node

**Algorithm 3** XDG Insert Algorithm

```
ALGORITHM: Insert (T,startAT,i)
  BEGIN
  path ← path of i
  R ← startAT
  if path equals R[title] then
    return path
  else
    if path contains R[title] then
      Insert (T,ChildrenOfR,i)
    end if
    if no child is matched then
      C ← create new Node
      C[title] ← i
      C is child of R
      if C is a leaf in XML file then
        Mark C as DRLP
      end if
    end if
  end if
  END
```

and thus continues to search its children of the matched XDG node. In this case, the number of comparisons for a single XML element is "b * d". Therefore in this scenario, there will be , $b * d * n$ comparisons where $n$ stands for the number of XML elements in XML document. Thus, the worst case scenario for Phase 1 is as follows:

$$XDG()time = O(b * d * n)$$

To find the space complexity of phase one, we need to find the size of the XDG tree. The number of XDG tree nodes is related to the number of leaf nodes in an XDG tree. Usually, the levels above the leafs share the nodes and therefore are less than the number leaf nodes. Therefore, a worst case scenario is when upper levels do not share any nodes except for the root element. In such a case, the number of XDG nodes is equal to “(p * d) +1” where $p$ is the number of leaf nodes of XDG tree . Therefore, the space complexity is as follows:

$$XDG()space = O(p * d)$$

### 4.2.2 Phase 2: XML Structure Index

In phase 1, we explained how the structure of an XML document is summarized by the XDG tree. The next step to is classify XML elements based on structure. This is achieved by building XML Structure Index (XSI). An XSI identifies the different clusters of XML transactions based on their structure.

An XSI is a hash table with a number of columns. The most important column is the binary number. The binary number represents the structure of a transaction. A question is: how can a binary number represent the structural features of a transaction? From the XDG tree we can obtain the DRLP of a dataset. For example, in the university dataset the path “/universities/university/ID” is a DRLP. However, the path “/universities/university” is not a DRLP as we have explained in Section 4.2.1. In our example, the university dataset has three DRLPs. Thus, each university transaction can be represented by a binary value of three bits. The order of the bits is the same order of the DRLP in the XDG tree. If the first bit is equal to 1, it means DRLP (1) exists in the transaction and 0 indicates otherwise. In our university dataset example, we have three transactions as shown in Figure 4.2. The names of these transactions are as follows: *Southern University*, *Eastern*

*University* and *Northern University.*

Next, the XSI algorithm, shown in Algorithm 4, will find the binary values of university transactions. Table 4.1 shows how the binary values are calculated. For example, the Southern University transaction has two DRLP which are:

- DRLP(1): /universities / University / ID
- DRLP(2):/ universities / University / Name

However, it does not have the following DRLP

- DRLP(3): / universities / University / NumberOfStudents

Therefore, the Southern University transaction can be represented by the binary value “110” which means it has the first two DRLP and not the last one.

An XSI consists of a number of entries. The XSI of the university dataset, shown in Table 4.2, has two entries. Each entry is composed of a *binary representation*, a *decimal number*, a *hash value* and a *count.* XSI stores only the unique binary representations, not the binary representations for all transactions.

Table 4.1: Building the XSI for the university dataset

| Transaction | DBLP(1) | DBLP(2) | DBLP(3) | Binary Number |
|---|---|---|---|---|
| Southern Uni. | Exists | Exists | Does Not Exist | 110 |
| Eastern Uni. | Exists | Exists | Exists | 111 |
| Northern Uni. | Exists | Exists | Exists | 111 |

Table 4.2: XSI for the university dataset

| Binary Number (key) | Decimal Number (key) | Hashing Value (value) | Count |
|---|---|---|---|
| 110 | 6 | 10 | 1 |
| 111 | 7 | 11 | 2 |

The hash value enables fast access to XSI entries. The decimal value is calculated from the binary value. For example, in Table 4.2, "6" and "7" are the decimal values for "110" and "111" respectively. To save space, we can optionally discard the binary values and store only the decimal numbers in XSI entries. The *count* in XSI entry is useful to know the sizes of the clusters as shown in Table 4.2.

The result of the XSC algorithm is the XSI. Each XSI entry represents a cluster of the XML document. Therefore, in the university dataset, we have two clusters.

**Algorithm 4** XML Structure Index Algorithm

```
ALGORITHM: XSI
  BEGIN
  t ← get a transaction
  DRLP ← unique paths from XDG
  BinaryValue ← Calculate Binary value from DRLP and t
  DecimalValue ← Get Decimal value from BinaryValue
  HashValue ← Get HashValue value from DecimalValue
  if HashValue does not exist in XSI then
    NewXSIEntry ← DecimalValue, HashValue
  else
    Increment the count of the matching XSI entry
  end if
  END
```

## Time and Space Complexity of The XSI Algorithm

The XSI algorithm extracts the binary value from each transaction. The time complexity of XSI algorithm is as follows:

$$XSI()time = O(t * p)$$

where $t$ is the number of transactions in an XML document and p is the number of DRLP entries. Note that $p$ is vastly smaller than $n$. Also note that the number of transactions, $t$, is usually a fraction of, $n$, the number of all XML elements. This is because a single transaction is composed of $c$ XML elements. Therefore, $t$ is as follows:

$$t = n/c$$

In phase 2, the worst case scenario for space usage is when each transaction has a unique structure. In this scenario, the XSI contains $t$ entries where $t$ is the number of transactions in an XML document. Therefore the space complexity of phase two is:

$$XSI()space = O(t)$$

## Overall Time and Space Complexity of XSC Algorithm

XML structure clustering algorithm (XSC) is composed of phase 1 and phase 2. Thus, the over all time and space complexity of the algorithm is as shown below:

- Time complexity of XSC

$$XSC()time = O(b * d * n) + O(p * t) = O(b * d * n)$$

- Space complexity of XSC

$$XSC()space = O(p * d) + O(t) = O(t)$$

Note that $t$ is less than $n$.

## 4.3 XML Hierarchal Content Clustering Algorithm (XHCC)

The XML Hierarchal Content Clustering algorithm (XHCC) extends XSC to cluster XML data by both structure and content. The phases of XHCC are as follows:

- Phase 1: Building the XML Data Guide (XDG)
- Phase 2: Creating the XML Structure Index (XSI)
- Phase 3: XML Univariate Content Clustering (XUCC)
- Phase 4: Clustering transactions by structure and content

Figure 4.5 shows an overview of XHCC. XHCC can be viewed as an extension to XSC since the first has all the phases of the latter. Phases 3 and 4 are added to XHCC. In Phase 3, the extended BIRCH algorithm, XBIRCH, is used to cluster the content of XML elements. In the last phase, XHCC clusters the transactions by both structure and content.

Next, phases 3 and 4 are explained in detail.

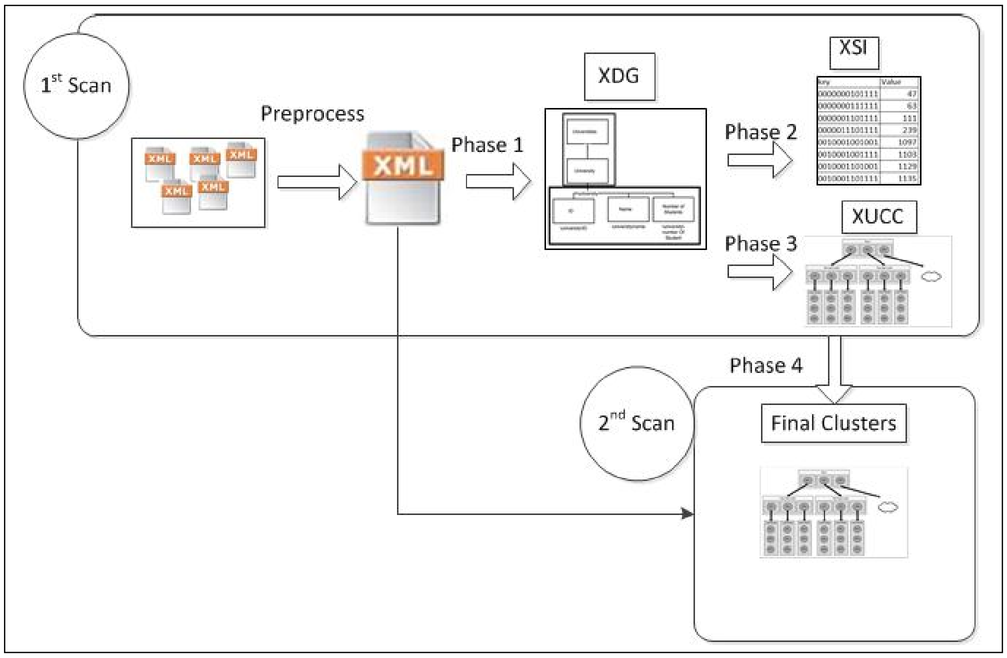


Figure 4.5: Overview of XHCC

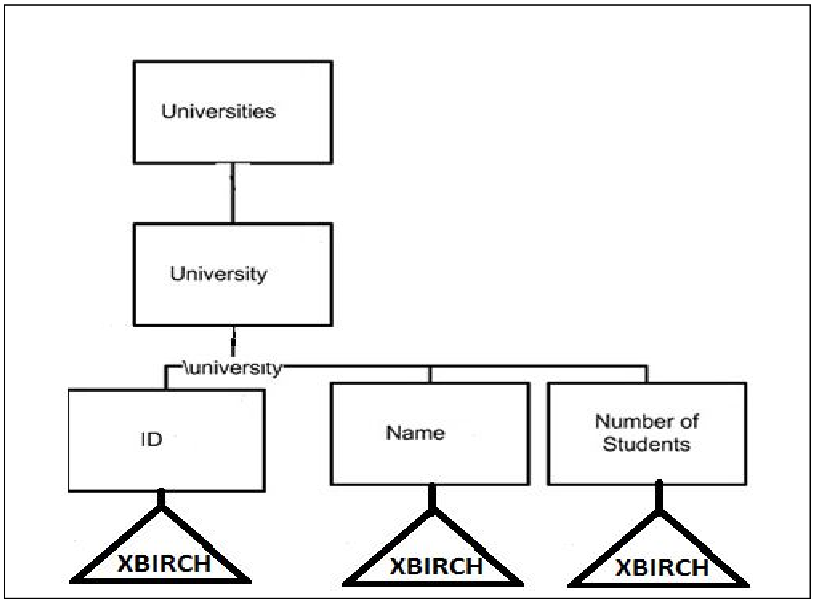


Figure 4.6: The University XDG in Phase 3

## 4.3.1 Phase 3: XML Univariate Content Clustering

In XHCC, building the XDG (phase 1) and XML Univariate Content Clustering (phase 3) are preformed simultaneously. For every XDG node labeled as DRLP, an XBIRCH tree is created. XBIRCH is an extension to the BIRCH algorithm and it is explained in Chapter 5.

For example, Figure 4.6 shows the XDG tree for universities dataset. Since they are marked as DRLP, the XML elements *name* and *ID* are clustered by using XBIRCH trees.

Table 4.3: The possible data types of datapoints

| Case | Example | Solution |
|---|---|---|
| Continuous | 12, 1.5, 8.2 | BIRCH |
| Ordinal | Freshman, Sophomore, Junior | Converting, BIRCH |
| Nominal | Red, Green, Blue | XBIRCH |
| Textual | Book titles, abstracts and articles | Future work |

**Algorithm 5** XML Univariate Content Clustering Algorithm

```
ALGORITHM: XUCC
  BEGIN
  while there is new datapoint, d  do
    n ← matching node in XDG
    v ← value of d
    Insert v into a BIRCH tree whose root is n
  end while
  END
```

XUCC Algorithm, listed in Algorithm 5, takes place whenever a new datapoint is inserted to a DRLP labeled node of the XDG tree. A datapoint is a paired value composed of a path and value. The path is used by phase 1 to find the matching XDG node for a datapoint. After that, the value of the datapoint is inserted into an XBIRCH tree which is a child node of the matching XDG node.

The Values of DRLP labeled nodes are one of four cases: continuous, nominal, textual, or ordinal values. As shown in Table 4.3, each case is handled differently. In the case of continuous values, a BIRCH tree is used to cluster these values as has been explained in Section 2.4.

Ordinal values are handled similarly, however, the values are first converted into numbers as explained in Section 2.3.4 and then they are clustered by a BIRCH tree. The nominal values are clustered using XBIRCH algorithm as we will see in Section 5.2.

## Time and Space Complexity of XUCC Algorithm

Let $p$ be the number of DRLP in an XDG tree. For each DRLP labeled node, an XBIRCH tree is created. Time complexity of XBIRCH is O(n), where $n$ is number of XML elements [5]. Therefore, the time complexity of XUCC is as follows (Note that $p$ is vastly less than $n$):

$$XUCC(n)time = O(p * n)$$

The space requirement of XUCC is dependent on the space requirement of BIRCH algorithm which is not covered in the original work [5]. One reason behind this is the flexibility of the BIRCH algorithm. Depending on the threshold value, the size of the BIRCH tree changes. The smaller the threshold, the bigger the BIRCH tree.

### 4.3.2 Phase 4: Clustering Transactions by Structure and Content

Phase 4 takes place after all other phases are complete. In this phase, a second scan is performed on the XML dataset.

Phase 4 has one algorithm: Transaction Clustering Algorithm (TCA) which is listed in Algorithm 6. The goal of TCA is to cluster XML transactions by content and structure. This is achieved by building an XBIRCH tree which will be used to cluster XML transactions.

To explain more, Let $y$ be a transaction with three components (XML children): $x_1$ , $x_2$, and $x_3$. Also let $X_1$, $X_2$, and $X_3$ be DRLP labeled XDG nodes. Since XUCC is complete, $X_i$ has an XBIRCH tree. Each of these XBIRCH trees have cluster features (CF). Suppose the closest CF entry to our transaction components $x_1$ , $x_2$, and $x_3$ are CF labels $a$, $b$, and $c$ respectively. These CF labels are found in the XBIRCH trees located in XDG nodes: $X_1$, $X_2$, and $X_3$ respectively.

For the purpose of creating a new XBIRCH tree in $y$, we create a vector of the cluster labels $a$, $b$, and $c$:

$$y = \{a, b, c\}$$

Lets give an example using the universities dataset. In this dataset, *university* is our transaction node. Next, we will create the datapoint for Northern University. First, suppose there are 3 clusters in the XBIRCH tree of the *ID* node and 2 clusters in the XBIRCH tree of the *number-of-students* node. Suppose the ID value of Northern University was found in the first cluster of the *ID* XBIRCH tree. Meanwhile the *number-of-students* value of Northern University was found in the second cluster of the *number-of-students*'s XBIRCH tree. A datapoint for *Northern University* transaction will be in the following format:

$$Datapoint_{NorthernUniversity} = \{1, 2\}$$

The values 1 and 2 are categorical values and therefore we will use the XBIRCH algorithm to cluster all the *university* datapoints. The new XBIRCH tree is inserted as a child for the *university* node in XDG tree.

Since our aim is to cluster XML data by both structure and content, the *university* XDG node will have more than one XBIRCH tree. This is because *university* transactions will be first clustered by structure using XSI and then by content using TCA. This means if that we have two entries in XSI, TCA will create two XBIRCH trees for each entry. Optionally, we can opt to only create one XBIRCH tree for all transactions. In such a case, we are clustering XML by content only.

**Algorithm 6** Transaction Clustering Algorithm

```
ALGORITHM: TCA
  BEGIN
  t ← new transaction
  for all c children of t do
    C ← the XBRICH tree of the XDG node with title c
    datapoint[tc] ← the label of the closest CF entry in C
  end for
  insert datapoint[t] into new XBIRCH rooted at the XDG node with title t
  END
```

### Time and Space Complexity of Phase 4

The Transaction Clustering Algorithm (TCA) is mainly composed of two steps. In the first step, TCA searches for the matching cluster labels of a transaction's children. Then in the next step, a new XBIRCH tree is built to cluster transactions. Therefore, the time complexity of TCA is as follows:

$$TCA(t) = O(t * c * log(t)) + O(l * t)$$

where $c$ is the number of leafs/children of a transaction and $l$ is the number of XSI entries. The values of $c$ and $l$ are vastly less than $t$, the number of transactions in an XML document. In the equation above, the first term is for finding the closest CF label and the second is for building an XBIRCH tree.

The space complexity of phase 4 and phase 3 are similar. For the space complexity of phase 3 please refer to Section 4.3.1.

### The Overall Time Complexity of the XHCC Algorithm

The XHCC algorithm is composed of four phases. Therefore the time complexity of the algorithm is as follows:

$$XHCC(n) = O(b * d * n) + O(p * t) + O(p * n) + O(t * c * log(t)) + O(l * t) = O(n)$$

Note that $p$, $d$, $b$, $l$ and $e$ are vastly less than $n$.

## 4.4 XHCC Revisited

In this section, we highlight some parts of XHCC in order to further explain the algorithm. These areas are: normalization and threshold values.

### 4.4.1 Normalization

In Section 2.3.3, normalization was discussed as a general problem in data clustering where it is a necessary preprocessing step. In our proposed algorithms, a new normalization technique is introduced. We normalize the output of phase 3 (XML Univariate Content Clustering) before executing phase 4. Instead of normalizing based on individual values, we normalize based on the number of clusters. Lets review the example given in Section 2.3.3.

Suppose the number of clusters for age and salary is 5 and 100 respectively. As we explained in phase 4, the transactions Ali (A), Bandar (B) and Sami (S) are as follows:

A= { 5 , 3 }

B = { 5 ,3 }

S= { 1, 3 }

As we explained, the values of attributes in the transactions above are the cluster labels and not the actual values of salary and age. Next, we want to normalize these values and make them lie in the same range. This is done by dividing each attribute by the number of clusters it has. Therefore, the three transactions become as follows:

A= { 5 /5, 3 /100 }

B = { 5/5 ,3 /100 }

S= { 1/5, 3 /100 }

Thus, the distance between Bander and Sami is as follows:

The difference in age is (5-1)/5 = 0.8

The difference in salary is (3-3)/100 = 0

This result is logical because the difference in age between B and S is greater than their difference in terms of their salaries.

In conclusion, normalization is handled differently in the proposed algorithms and there is no need for a preprocessing step to scale input values.

### 4.4.2 Threshold Values

This section describes how threshold values are set in the proposed algorithms. As mentioned in Chapter 2, the BIRCH algorithm has a threshold value that specifies the maximum distance allowed between a data point and cluster feature (CF) [5]. The most accurate threshold value is zero which builds a large BIRCH tree. If a smaller memory footprint is required, the threshold value is increased which makes the tree smaller but with a lower accuracy.

In our proposed algorithms there is a threshold value per DRLP labeled XDG node. Therefore, the problem of finding the most appropriate threshold value for each node is a challenge. A study of this problem is part of our future work.

## 4.5 Conclusion

The proposed algorithms, XSC and XHCC, can be divided into two and four main phases respectively. These phases are Building the XDG tree, Building the XSI, XUCC, and Transaction Clustering Algorithm (TCA). These phases have been explained with examples using a small dataset. Experiments and analysis on large datasets is the topic of Chapter 6. Before that, since BIRCH only works with numerical data, there is a need to extend BIRCH to handle categorical data as well. This is further discussed in the next chapter.

CHAPTER 5

# EXTENDED BIRCH ALGORITHM (XBIRCH)

A typical XML document consists of elements with different data types. However, the BIRCH algorithm one of, the building block of our proposed algorithms, only clusters numerical values. Therefore, in this chapter we will modify the BIRCH algorithm to cluster categorical (nominal) values as well.

The chapter is organized in the following order. First, Section 5.1 highlights the difficulty of clustering categorical values with the BIRCH algorithm. Then in Section 5.2 we explain how to cluster univariate categorical values. After that, in Section 5.3, we extend the BIRCH algorithm to cluster multivariate categorical values, and name the algorithm extended BIRCH (XBIRCH). After that, the issue of attribute priority and order is analyzed in Section 5.4. Then the issue of sensitivity to order is highlighted in Section 5.5. Finally, we introduce an optimizing algorithm for XBIRCH to produce more accurate results.

## 5.1 Overview

Clustering categorical values is different from clustering numerical ones. In the case of clustering numerical values with BIRCH, statistical measurements are used to represent sets of numbers. However, in the case of clustering categorical values, it is not possible to use a statistical measurement. For example, what is the mean for red, black, and blue? There is no statistical measurement that can capture a set of categorical values which is why BIRCH cannot cluster them. Since many XML datasets contain categorical values, we cannot ignore clustering them. For example, a dataset of world countries can be expressed in terms of datapoints with the following variables or attributes: *weather*, *language*, and *political system*. Thus, *Australia* can be represented as [Sunny, English, multi-party] while *China* is represented as [rainy, Chinese, Single-party]. Similarly [Windy, English, Multi-party] can represent the *USA*. Imagine that these three countries were in the same cluster. What measurement or model can summarize this cluster? As we add more countries, how can they be represented in hierarchy of clusters similar to a BIRCH tree?

For this purpose, we introduce a clustering feature along with a distance function. The clustering feature is the Nominal Clustering Feature (NCF). The most suitable distance function for NCF is the hamming distance which will be explained in Section 5.3. The main difference between BIRCH and XBIRCH is that BIRCH uses CF entries to cluster values while XBIRCH uses NCF entries to cluster categorical values. For more details about BIRCH, please refer to Section 2.4.

## 5.2 Clustering Univariate Categorical Values

In this section, we explain how univariate categorical values are clustered. For example, in the countries dataset there are three categorical values one of them is *weather*. It is possible to have clusters such as: “rainy”, “cloudy”, “hot”, etc. If the number of the distinct values of weather is ten, then ten clusters will be created.

To group the identical values together, one approach is to compare their string values. This is a pair-wise operation and it is computationally expensive. A better mechanism is to use hashing functions. However, there is a possibility that a hashing function can produce the same value for two different inputs. Therefore, we can combine multiple hash functions to reduce that possibility.

For example, we conducted an experiment where all entries of an English lexicon were clustered. The lexicon is composed of 519,752 words. We represented each word with a vector of two values. Each value is the product of a hashing function. In our experiment, we used two hashing functions: the Microsoft .NET GetHashValue() and a custom hashing function that we implemented to convert strings into numbers. After that, each vector was inserted to a BIRCH tree with threshold zero. As a result we obtained 519752 clusters which is equal to the number of words in the lexicon.

In summary, clustering univariate categorical values is achieved by using hashing functions and a BIRCH tree. This will produce a number of clusters equal to the number of distinct values of the categorical variable.

## 5.3 Multivariate Categorical Clustering

In this section, we describe how multivariate categorical values are clustered. It is done through the modification of the BIRCH algorithm.

Let us assume that we have a vector that contains only categorical values. For example, in the countries dataset, China, USA, and Australia are datapoints which can be represented as follows:

$$Datapoint_{USA} = \{c_1 = 2, c_2 = 2\ , c_3 = 3\}$$
$$Datapoint_{Australia} = \{c_1 = 3, c_2 = 2\ , c_3 = 3\}$$
$$Datapoint_{China} = \{c_1 = 1, c_2 = 1\ , c_3 = 2\}$$

where $c_1$, $c_2$, and $c_3$ represent the attributes *weather*, *language*, and *political system* respectively. Note here that the values for these attributes have no meaning other than being different. Value 3 does not mean it is greater than 2. It is just a label for a categorical value. Instead of inserting these datapoints into a CF of BIRCH tree, they are inserted into an NCF. An NCF captures the clustering feature for categorical values. Instead of a statistical representation, a vector of characters, called a header is used. For example, a country's NCF will be composed of a vector of three characters. Each character represents one attribute. An NCF for a country will be as follows:

$$NCF_{Country} = \{c_1, c_2, c_3\}$$

Where the value of $c_i$ can be from 1 to $k$, where $k$ is the number of clusters in attribute $i$. Additionally, $c_i$ can have asterisk value (*) which means any value. In our countries dataset, the NCF header after inserting the USA datapoint becomes as follows:

$$NCF_{USA} = \{2, 2, 3\}$$

Note that the NCF header equals to the USA datapoint. Secondly, we insert Australia's datapoint and the NCF header becomes as follows:

$$NCF_{Australia+USA} = \{*, 2, 3\}$$

In this case, USA and Australia share two attribute values: *language* and *political system* while they are different in terms of the first attribute, *weather*. For that reason, we mark the *weather* variable in NCF with the asterisk value (*). Lastly, we add China to the NCF and it becomes as follows:

$$NCF_{China+USA+Australia} = \{*, *, *\}$$

After adding China, the NCF header has lost all its numeric values. This means that the inserted datapoints have nothing in common. To restrict the membership to an NCF, a threshold value and distance function are used to see if a datapoint can be inserted to an NCF or not. A datapoint is inserted if the hamming distance between the header of an NCF and the datapoint is less than the threshold value. For example, if we set the threshold value for countries to 2, China can't join the USA and Australia cluster. It will form a new cluster by itself. This is because the hamming distance (see Equation 2.3) between the China datapoint and $NCF_{USA+Australia}$ is three which is larger than 2. Similar to the BIRCH tree, a threshold value will control the size of an XBIRCH tree. The smaller the threshold value, the bigger the XBIRCH tree.

In an XBIRCH tree, the branching factor cannot be controlled by the user but rather the data itself. The advantage of this is that clusters will be more natural. However if the attributes have a large number of distinct values, the XBIRCH tree will have huge width. To minimize its width, the NCF vector's attributes are ordered by their number of distinct values. This means if *weather* has less number of distinct values, it appears before *language*. This will make the width of the tree as small as possible. This problem is further analyzed in the following section.

## 5.4 Priority and Order of Datapoints' Attributes

Herein, we study the issue of priority and order of datapoints' attributes. In section 4.3.2, we studied the university dataset where one of the *university* transactions was composed of two attributes: *ID* and *Number-Of-Students*. Suppose there is another attribute, *size-of-university*, which is similarly a DRLP labeled node in the XDG tree. Lets assume all these attributes are categorical values. As a result, we can have the following datapoints:

$$Datapoint_{SouthernUniversity} = \{2, 3, 3\}$$

$$Datapoint_{NorthernUniversity} = \{1, 3, 3\}$$
$$Datapoint_{EasternUniversity} = \{2, 5, 3\}$$

As we explained, these three datapoints will be inserted into an XBIRCH tree where the hamming distance is used to calculate the distance between these datapoints and the NCFs. In our example above, the distance between Southern University and either of Northern University or Eastern University is equal to 1. However, to group these datapoints there are two solutions. One solution is to group Southern University with Northern University while Eastern University is a cluster by itself. Another solution is to instead group Southern University with Eastern University and make Northern University a cluster by itself. Generating two solutions for the same input is undesirable. Ideally, if we run a deterministic algorithm multiple times on the same dataset, we should have the same solution. To overcome this problem, we will specify an order and a priority when creating a datapoint. First, attributes are ordered by the number of clusters they have or in other words, the number of their distinct values. For example in university dataset, the XBRICH trees for attributes: *ID*, *number-of-students*, and *university-size* have 2, 3, and 4 clusters respectively. Thus, a university datapoint is as follows:

$$Datapoint_{SouthernUniversity} = \{ID_i, numberofstudents_j, universitysize_k\}$$

As can been seen, the attributes of the Southern University datapoint are ordered by the number of clusters (ascending). The second step is to set a priority for these attributes. When calculating the distance function between two datapoints, we can set the following rule: Finding a match in the first attribute shall have higher priority than finding a match in the second attribute and so on. As a result, the algorithm will produce the same results even if it runs several times. In our example, Southern University will be grouped with Eastern University since they are similar in the first attribute (ID).

Shall we arrange the attributes of a datapoint in ascending or descending order? Before answering this question, we need to see the effect of order on the XBIRCH tree.

Changing the order of attributes has an effect on the shape of an XBIRCH tree. For example, in our university dataset, we have *ID* and *number of students* as DRLP labeled nodes that have 2 and 4 clusters respectively. Both *ID* and *number of students* are attributes of the *university* datapoints. If these attributes are arranged in an ascending order, we obtain tree T2 in Figure 5.1. The branching factor in tree T2 increases from top to bottom. On the other hand, if attributes are arranged in a descending order, tree T1 is obtained. In tree T1, the branching factor decreases from top to bottom. Tree T2 is useful in database query applications where smaller tree enables faster access to leaf nodes. On the other hand, in data clustering applications tree T1 is more useful. To achieve natural clusters, finding a match between attributes with larger number of clusters should have higher weight. Thus, the decreasing order of attributes is more preferable in data clustering applications.

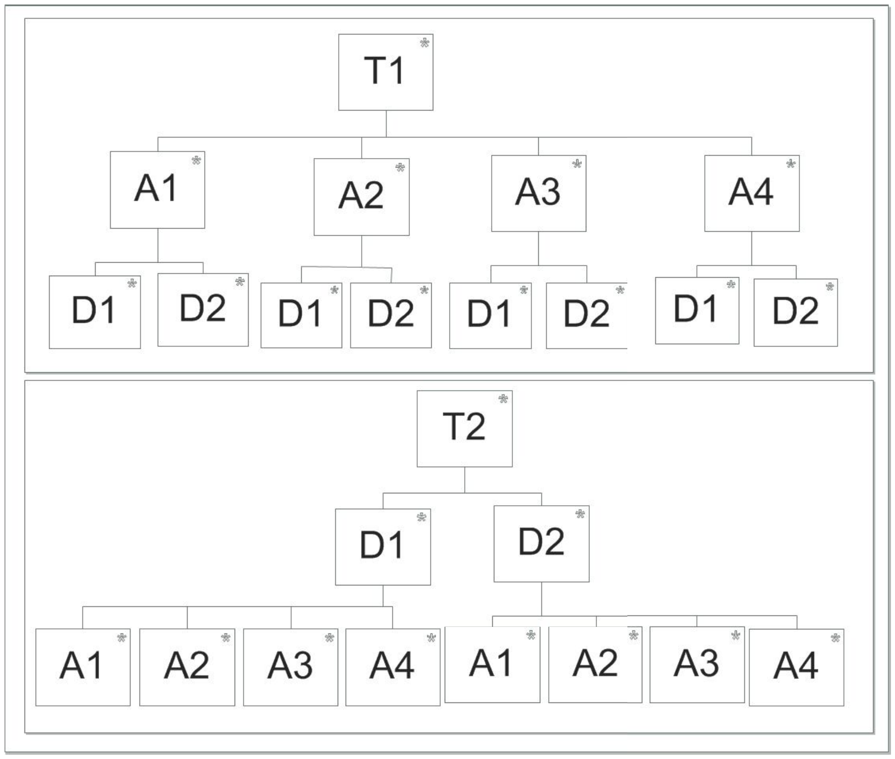


Figure 5.1: Effect of attribute order on XBIRCH tree

## 5.5 The Sensitivity to Order

Another issue is the sensitivity of XBIRCH to the order of datapoints. Similar to BIRCH, the order of the datapoints (for example inserting China first) will change the final results. This is overcome by a second scan of datapoints (not the XML data). In this second scan a different algorithm is used where old NCFs are kept and datapoints are re-inserted into the XBIRCH tree.

**Algorithm 7** Optimize- XBIRCH clusters

```
ALGORITHM: Optimize-XBIRCH
  BEGIN
  Delete all datapoints in clusters
  Keep NCF headers as they are
  Cluster datapoints again
  Delete empty NCFs
  Return XBIRCH tree
  END
```

It is an optimization algorithm which outputs more accurate results. The optimization algorithm is listed in Algorithm 7.

## 5.6 Conclusion

Clustering categorical values requires a modification of the BIRCH algorithm. This is achieved by introducing the Nominal Clustering Feature (NCF). In this chapter we explained how to cluster univariate and multivariate categorical values. The problem of datapoints order and the order of attributes has been studied. Finally, an optimization algorithm for XBRICH has been introduced to output more accurate results.

CHAPTER 6

# EXPERIMENTAL RESULTS AND ANALYSIS

This chapter describes the experimental results and analysis of the proposed algorithms. The goal of the experiments is to measure the performance and recall of XHCC and XSC algorithms. The chapter is organized as follows. First, performance parameters are described in Section 6.1. Then in Section 6.2 we specify the experimental setup. Datasets are then described in Section 6.3. After that, in Section 6.4 we explain our experiments methodology. Then we discuss the experimental results and analysis in Section 6.6. Finally, we compare our algorithms with INEX 2008 in Section 6.7.

## 6.1 Performance Parameters

Our performance parameters are measurements used to analyze the proposed algorithms from three aspects. These aspects are: time, space, and recall of the clustering results.

**Time:** To measure the scalability of the proposed algorithms, the time to run the algorithms is measured. The CPU time of the running application is measured in seconds.

**Space:** To test the capability of handling large XML datasets, the space used by proposed algorithms' data structures is measured. We analyze the growth of space usage used as the size of datasets increases. Space is measured in kilobytes(KB).

**Recall:** In section 2.3.5, we presented ways to evaluate the quality of data clusters. We will use the recall measurement to test the quality of the proposed algorithms. To calculate the recall, the following procedure is followed:

- Set all BIRCH and XBIRCH threshold values to zero
- All XML transactions are clustered either by XSC or XHCC
- XML transactions are clustered again
- Check the clusters and see if they have duplicate values

If each two identical XML transactions fall in the same leaf node, the recall is 100%. Measured in percentage, the recall is the ratio of identical point found in the same leaf node over the total number of identical pairs as shown in Equation 5.1 below.

$$Recall = \{\frac{I}{A}\} \tag{6.1}$$

where $I$ is the count of identical pairs grouped in the same node and $A$ is the number of XML transactions multiplied by 2.

## 6.2 Experimental Setup

This section lists the specifications of the testing environment. First, the proposed algorithms have been implemented in the C# language and the application runs on the Microsoft .NET 4 framework. A 64-bit Windows Server 2008 R2 machine has been used with 8 GB of RAM. The processor of the machine is Intel Core 2 with speed of 2.4 GHZ.

## 6.3 Datasets

Following is a description of the datasets used to test the proposed algorithms. Our data source is real life data taken from the XML Data Repository [59] and Wisconsins XML data bank [60]. Six different datasets have been used. The criteria for choosing the datasets are as follows:

- Two datasets which mostly contain continuous values
- Two or more datasets which contain mixed types of values (categorical and continuous values)
- All datasets should not mainly be composed of textual XML elements

Based on these criteria, the following datasets have been chosen: Mondial, Movies, Parts, Orders, DBLP, and Cars. Their description is as follows.

**Mondial:** Mondial is part of the XML Data Repository. It is an XML dataset containing information about the countries of the world integrated from the CIA World Fact book, the International Atlas, and the TERRA database among other sources. It contains information about nearly every country in the world. This dataset contains mostly numeric values.

**Movies:** IMDB stands for Internet Movie Database. It is listed also in the XML data Repository. It was retrieved from the IMDB website. It has information related to movies, television shows, actors, and production crew personnel.

**DBLP:** DBLP stands for Digital Bibliography and Library Project. It is a computer science bibliography website which indexes more than one million articles and contains more than 10,000 links to home pages of computer scientists.

**Cars:** The cars dataset has been taken from the Wisconsins XML data bank. It has specifications for several car models. This dataset mostly contains continuous values.

**Parts and Orders:** Parts and Orders datasets are listed also in the XML Data Repository. It was generated by the Transaction Processing Performance Council [60]. The first dataset, Parts, contains manufactured metal products while Order's dataset is composed of a list of orders made by a number of customers and served by some clerks. These two sets cover the case when data is composed of both numerical and categorical data.

Table 6.1 shows general properties of the datasets. These properties are: Dataset name, Data Type, Average and Maximum Depth and the XML transaction.

Furthermore, for each dataset, four sub-datasets have been generated. Therefore, there are 24 datasets in total. The reason for these sub-datasets is to test the scalability of the proposed algorithms. Table 6.2 shows the specifications for each sub-dataset. The specifications are: number of elements, attributes and the size of the dataset. Note that within a dataset, the ratio of the size of two consecutive sub-datasets is 1:10 for most of the sub-datasets. For example in Movies dataset, Mov2 has the size of Mov1 multiplied by 10. As a result, Mov1 has a size of 0.87 MB and Mov2 has size of 8.7 MB. Note that for the first two sub-datasets, M1 and M2, the ratio is around 1:13.

Table 6.1: General specifications of the main datasets used for experiments

| Dataset Name | Data Type | Average Depth | Maximum Depth | Transaction Name |
|---|---|---|---|---|
| Mondial | Numerical | 3.6 | 5 | Country and Province |
| Movies | Mixed (Categorical & Numerical) | 2.89 | 3 | movie |
| DBLP | Mixed (Categorical & Numerical) | 2.9 | 6 | article |
| Cars | Numerical | 3.88 | 5 | Specs |
| Parts | Mixed (Categorical & Numerical) | 2.98 | 3 | T |
| Orders | Mixed (Categorical & Numerical) | 2.98 | 3 | T |

## 6.4 Testing Methodology

Testing takes place in two steps: First, we test the time, the space, and the recall of running the XSC algorithm on each dataset. Then the experiment is repeated for the XHCC algorithm.

Table 6.2: Detailed specifications of sub-datasets

| Dataset Name | Sub-dataset Name | Number of elements & attributes | Size |
|---|---|---|---|
| Mondial | M1 | 2781 | 0.07 MB |
| | M2 | 37496 | 1.16 MB |
| | M3 | 374960 | 11.6 MB |
| | M4 | 3749600 | 116.06 MB |
| Movies | mov1 | 25500 | 0.87 MB |
| | mov2 | 255000 | 8.7 MB |
| | mov3 | 2550000 | 78 MB |
| | mov4 | 25500000 | 782.2 MB |
| DBLP | D1 | 34214 | 1.3 MB |
| | D2 | 342140 | 13 MB |
| | D3 | 3421400 | 133 MB |
| | D4 | 34214000 | 1338.9 MB |
| Cars | C1 | 15000 | 1.2 MB |
| | C2 | 150000 | 12.6 MB |
| | C3 | 1500000 | 126.8 MB |
| | C4 | 15000000 | 1260 MB |
| Parts | P1 | 200 | 0.06 MB |
| | P2 | 2000 | 0.6MB |
| | P3 | 200000 | 6MB |
| | P4 | 200000 | 60 MB |
| Orders | O1 | 150 | 0.052 MB |
| | O2 | 1500 | 0.525 MB |
| | O3 | 15000 | 5.2 MB |
| | O4 | 150000 | 52.5 MB |

## 6.5 Results

This section lists the results obtained from running our experiments. These results are shown in Tables 6.3, 6.4, 6.5, and 6.6. Each table represents the results per dataset. For each table the following is given: clustering time and space of running XSC and XHCC algorithms.

Table 6.3: Results for Mondial dataset

| Data set | Clustering time for structure | Clustering time for content and structure | Memory Size (structure only) | Memory Size (structure and content) |
|---|---|---|---|---|
| M1 | 0.625 Sec | 0.875 Sec | 609.725 KB | 1269.06 KB |
| M2 | 3.688 Sec | 7.125 Sec | 609.725 KB | 1674.55 KB |
| M3 | 32.969 Sec | 62.766 Sec | 609.725 KB | 1816.56 KB |
| M4 | 333.469 Sec | 599.984 Sec | 609.725 KB | 1958.57 KB |

Table 6.4: Results for Movies dataset.

| Data set | Clustering time for structure | Clustering time for content and structure | Memory Size (structure only) | Memory Size (structure and content) |
|---|---|---|---|---|
| Mov1 | 2.594 Sec | 5.25 Sec | 5578.90 KB | 11174.92 KB |
| Mov2 | 22.656 Sec | 47.203 Sec | 5578.98 KB | 11175.07 KB |
| Mov3 | 199.984 Sec | 411.203 Sec | 5578.98 KB | 11177.93 KB |
| Mov4 | 2063.156 Sec | 3997.766 Sec | 5578.98 KB | 11177.93 KB |

Table 6.5: Results for DBLP dataset.

| Data set | Clustering time for structure | Clustering time for content and structure | Memory Size (structure only) | Memory Size (structure and content) |
|---|---|---|---|---|
| D1 | 3.922 Sec | 7.031 Sec | 7488.23 KB | 14996.42 KB |
| D2 | 35.313 Sec | 64.406 Sec | 7488.28 KB | 15416.84 KB |
| D3 | 405.625 Sec | 741.5 Sec | 7497.90 KB | 19231.52 KB |
| D4 | 4059.828 Sec | 7091.016 Sec | 7497.90 KB | 31788.66 KB |

Table 6.6: Results for Cars dataset.

| Data set | Clustering time for structure | Clustering time for content and structure | Memory Size (structure only) | Memory Size (structure and content) |
|---|---|---|---|---|
| C1 | 2.063 Sec | 2.563 Sec | 3286.81 KB | 6584.47 KB |
| C2 | 17.016 Sec | 21.25 Sec | 3286.81 KB | 6587.32 KB |
| C3 | 165.641 Sec | 212.203 Sec | 3286.81 KB | 6587.90 KB |
| C4 | 1614.953 Sec | 2121.75 Sec | 3286.81 KB | 6591.88 KB |

The recall for each dataset is shown in Table 6.7. Generally, the average recall is between 85 % and 94 % which is equal to 89.5 %. The recall value varies as the size and type of data change. The Orders dataset shows the highest recall value. This is because this dataset is the smallest in size. The bigger the dataset, the more errors are seen. This is because bigger datasets have more clusters based on content and structure and therefore the probability of error is higher.

Table 6.7: Recall per dataset

| Data set | Recall for First sub-dataset | Recall for sub-dataset | Average |
|---|---|---|---|
| Mondial | 85% (M1) | 85% (M2) | 85% |
| Parts | 90% (P1) | 85% (P2) | 87.5% |
| Orders | 92 % (O1) | 96 % (O2) | 94 % |
| Cars | 88 % (C1) | 86 % (C2) | 87 % |

## 6.6 Experimental Analysis

Figure 6.1 shows a chart for the space measurement for XSC. The horizontal axis represents the count of XML elements in a dataset. The vertical axis represents the measured space used by each algorithm. From Figure 6.1, we note that the space complexity is almost constant. Regardless of the increasing size of the dataset, the space remains almost the same. In Chapter 4, we expected a linear growth for the space usage of both the XSC and XHCC algorithms, yet we have found in experiments that the space usage is almost constant with some exceptions. The reason behind this is that when no more new distinct values or new DRLP are found, the size of the data structures stay the same. The second reason is that in our analysis we studied the worst case scenario and therefore our results have to be equal or less than the complexity analysis estimations.

Figure 6.1 shows that the DBLP dataset has the highest space usage followed by Movies, Cars, and lastly Mondial. This order is the same order if datasets are ordered by the number of XML elements. Therefore, the more XML elements in a dataset, the more space XSC needs for structural clustering.

Note that the space usage of the algorithm is larger than the size of some of the small datasets. For example the size of D1 dataset is 1.3 MB while the size used by XSC is 7.4 MB. This suggests that the space usage of XSC is not efficient for small XML documents. Nevertheless, the space usage is relatively in case of large datasets as the space usage becomes almost constant.

Similarly Figure 6.2 shows the space usage of the XHCC algorithm. The space usage for most datasets is constant. An exception is DBLP, where there is sub-linear growth. The reason is that DBLP contains more XML elements with categorical values compared to other datasets. In addition, these categorical values have high number of distinct values. Each unique categorical value is represented by an array of products of two hashing functions. As a result, when more data is processed, more distinct values are seen and thus more space is required.

To analyze the time complexity for XHCC and XSC, please see figures 6.3 and 6.4. Both figures show that the time complexity is linear for both algorithms regardless of the XML type. Similar to space usage, the order of the datasets is in the same descending order if datasets are ordered by the number of XML elements.

## 6.7 Comparison

Related published papers in this area have focused on datasets that mostly contain textual values. This does not satisfy our third dataset criterion. To the best of our knowledge and based on our literature review in Chapter 2, there is no work that is similar to the proposed algorithms. Solutions to clustering XML by content and structure are mainly addressing the problem from an Information Retrieval (IR) point of view and not from a data mining or database perspective.

The problem of determining cluster quality is an open problem [61]. Therefore, comparing algorithms is a challenging work. There are many measurements to find quality of clusters. The closest work to ours are those found in INEX 2008 [61]. INEX is is an international campaign that provides a means of evaluating retrieval systems that provide access to XML content. The similarities and differences between our algorithm and INEX 2008 are as follows:

- Similarities: Algorithms in INEX 2008 target large homogeneous datasets similar to our algorithm.

- Differences: INEX 2008 algorithms focus on text-centric datasets such as Wikipedia. Our algorithm XHCC deals with data types and ignores textual data. The proposed algorithms in INEX 2008 transform XML to other formats while our algorithm keeps the XML structure using XDG tree.

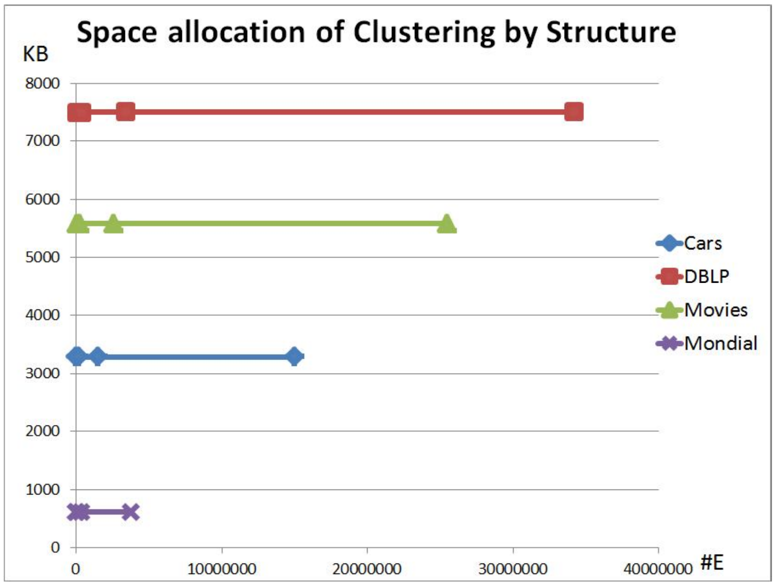


Figure 6.1: Space Usage of XSC

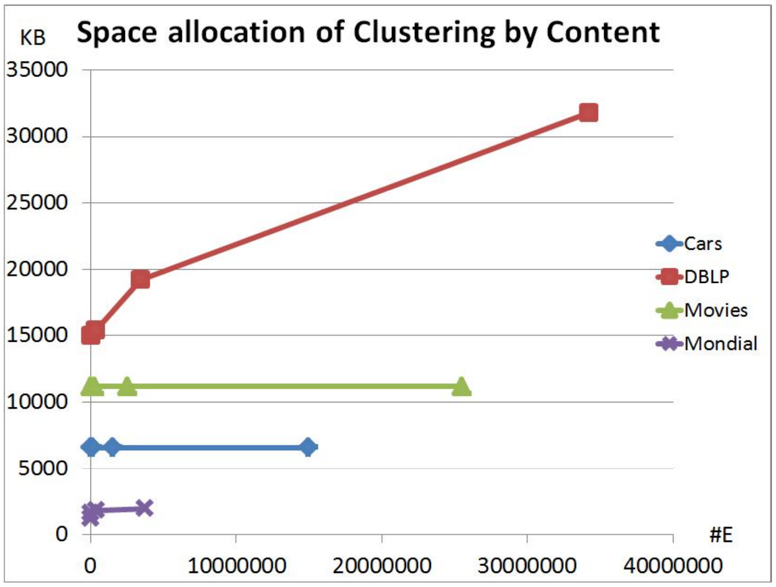


Figure 6.2: Space Usage of XHCC

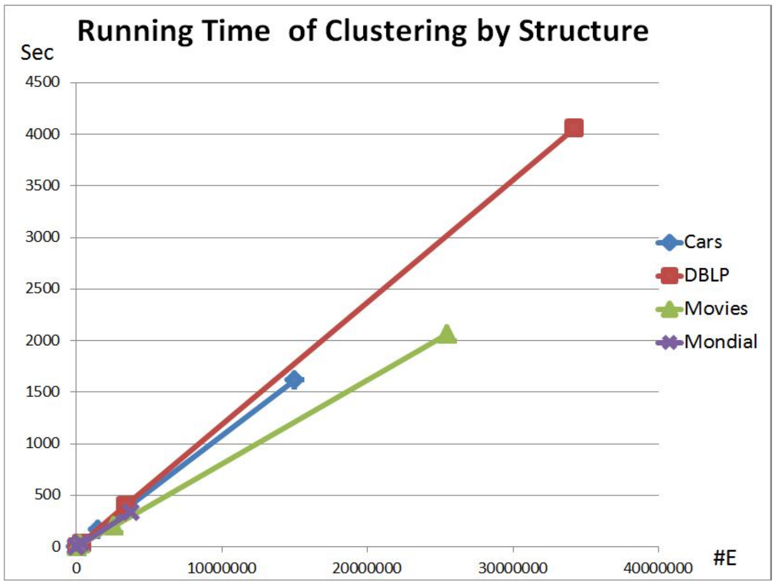


Figure 6.3: Running Time of XSC

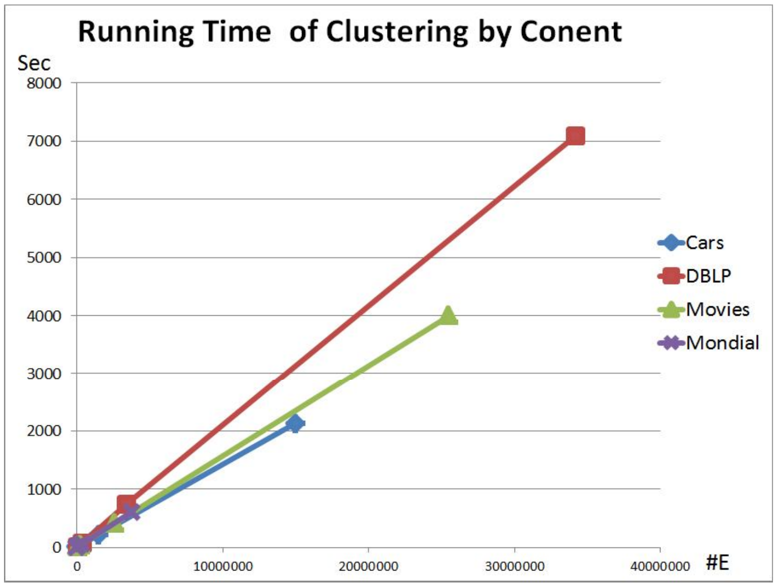


Figure 6.4: Running Time of XHCC

Another major difference is the type of experiments being conducted by INEX 2008 and our algorithm. As we explained in Section 6.1, we cluster the data twice and see if each identical objects are grouped together. Meanwhile, INEX 2008 clusters the Wikipedia entries and assigns a label (such as subject) to each cluster. Nevertheless, we can still look at the results to have a general understanding about the recall values in this area. Figure 6.5 shows the recall measurements for participants of the classification task at INEX 2008 [61]. The highest recall was 78% achieved by Gery et al. Meanwhile our average recall is 89.5% . This is to give an idea about the recall values in the area of XML clustering.

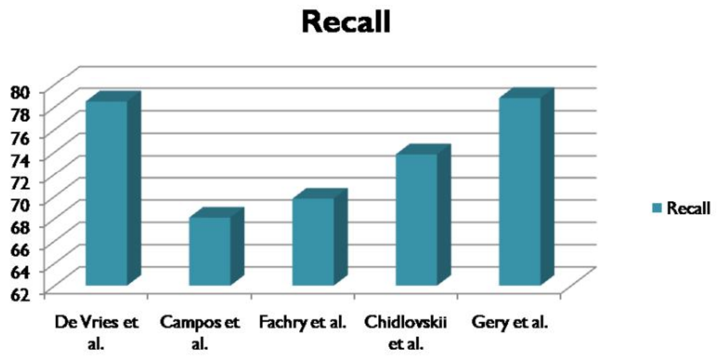


Figure 6.5: Recall of INEX 2008 participants

## 6.8 Conclusion

This section has presented the analysis of the proposed algorithms (XSC and XHCC) based on experiments on real XML benchmark datasets. Time and space complexities show that the proposed algorithms are scalable, fast, and can handle large XML data. The experiments also have shown that the proposed algorithms have a high recall of 89.5% on average.

CHAPTER 7

# CONCLUSION AND FUTURE WORK

In this chapter, Section 7.1 summarizes the thesis work. After that, a list of future work is given in Section 7.2.

## 7.1 Thesis Summary

The main objective of our thesis was to introduce a scalable clustering algorithm for large XML data. To achieve this objective, first we presented a literature review of XML clustering in Chapter 3. The review showed that there are only a few algorithms that can cluster XML data by both content and structure. The proposed algorithms have been designed to achieve the following goals:

- Ability to cluster homogeneous XML Data
- Ability to cluster large XML data

- Hierarchal Clustering by content and structure

While working toward achieving these goals, a number of problems have been solved. These problems are:

- The BIRCH algorithm has been extended to handle categorical data
- A new scaling technique was proposed that does not require preprocessing
- An optimization algorithm was implemented to increase the accuracy of categorical data

To validate our algorithm design and analysis, experiments were conducted using real life XML benchmark datasets. The results of these experiments are as follows:

- The space complexity of XSC is constant
- The space complexity of XHCC is sub-linear
- The time complexity of XSC is linear
- The time complexity of XHCC is linear
- XSC require the data to be scanned only once
- XHCC require the data to be scanned two times
- The recall of XHCC is 89.5% on average.

## 7.2 Future Work

There are three tasks that extend this thesis. They are: clustering the textual data type, finding the best threshold values for the XBIRCH trees, and feature selection.

***Textual Data Type***: Currently our approach deals with categorical and continuous data. There is a need to include textual data such as book titles and messages.

***Finding Best Parameters***: As explained in Section 2.4, a CF tree has a number of parameters. For achieving the highest accuracy, the threshold value is set at 0. However, in case where memory size is limited, the size of the CF tree can be reduced by a higher threshold value.

In the XHCC algorithm, several XML elements are clustered. The values of these elements have different ranges. In case of a limited memory scenario, this means that there are different combinations of threshold values to be set. In such a case, there is a need to find an algorithm that assigns optimal threshold values in order to reduce the size of the XBIRCH tree without major reduction in accuracy.

***Feature Selection***: In machine learning, the accuracy of classification can be increased when the number of features (attributes) is reduced to the most influential features. Similarly, the accuracy of clustering can be increased by choosing the most discriminating XML elements. An algorithm to solve this problem is an important extension to our work.

# References

[1] J. H. Hwang and K. H. Ryu, “A weighted common structure based clustering technique for xml documents,” *J. Syst. Softw.*, vol. 83, no. 7, pp. 1267–1274, 2010.

[2] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, “Extensible markup language (xml),” *World Wide Web Journal*, vol. 2, no. 4, pp. 27–66, 1997.

[3] G. S. A Algergawy, “A classification scheme for xml data clutering techniques,” in *Forth International Conference on Intelligent Computing and Information Systems*, (Cairo, Egypt), pp. 550–555, ACM, 2009.

[4] P.-N. Tan, M. Steinbach, and V. Kumar, “Introduction to data mining,” in *Introduction to Data Mining*, 2005.

[5] T. Zhang, R. Ramakrishnan, and M. Livny, “Birch: an efficient data clustering method for very large databases,” *SIGMOD Rec.*, vol. 25, pp. 103–114, June 1996.

[6] A. Bouchachia and M. Hassler, “Classification of xml documents,” in *CIDM*, pp. 390–396, 2007.

[7] P. Perner, “Data mining - concepts and techniques,” *KI*, vol. 16, no. 1, p. 77, 2002.

[8] A. Algergawy, E. Schallehn, and G. Saake, “A schema matching-based approach to xml schema clustering,” in *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*, (New York, NY, USA), pp. 131–136, ACM, 2008.

[9] H.-J. Moon, S. Kim, J. Moon, and E. ser Lee, “An effective data processing method for fast clustering,” in *ICCSA (2)*, pp. 335–347, 2008.

[10] M. L. Lee, L. H. Yang, W. Hsu, and X. Yang, “Xclust: clustering xml schemas for effective integration,” in *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, (New York, NY, USA), pp. 292–299, ACM, 2002.

[11] R. Nayak and W. Iryadi, “Xml schema clustering with semantic and hierarchical similarity measures,” *Know.-Based Syst.*, vol. 20, no. 4, pp. 336–349, 2007.

[12] L. Yang, J. Gu, and H. Chen, “Clustering algorithm based on semantic distance for xml documents,” in *DBTA '09: Proceedings of the 2009 First International Workshop on Database Technology and Applications*, (Washington, DC, USA), pp. 549–552, IEEE Computer Society, 2009.

[13] H.-J. Moon, J.-W. Yoo, and J. Choi, “An effective detection method for clustering similar xml dtds using tag sequences,” in *ICCSA (2)*, pp. 849–860, 2007.

[14] P. D. Meo, G. Quattrone, G. Terracina, and D. Ursino, “An approach for clustering semantically heterogeneous xml schemas,” in *OTM Conferences (1)*, pp. 329–346, 2005.

[15] P. D. Meo, G. Quattrone, G. Terracina, and D. Ursino, “Semantics-guided clustering of heterogeneous xml schemas,” *J. Data Semantics*, vol. 9, pp. 39–81, 2007.

[16] G. Xing, Z. Xia, and J. Guo, “Clustering xml documents based on structural similarity,” in *DASFAA*, pp. 905–911, 2007.

[17] G. Yongming, C. Dehua, and L. Jiajin, “Clustering xml documents by combining content and structure,” in *ISISE ’08: Proceedings of the 2008 International Symposium on Information Science and Engieering*, (Washington, DC, USA), pp. 583–587, IEEE Computer Society, 2008.

[18] E. Bertino, G. Guerrini, and M. Mesiti, “Measuring the structural similarity among xml documents and dtds,” *J. Intell. Inf. Syst.*, vol. 30, no. 1, pp. 55–92, 2008.

[19] J. Tekli, R. Chbeir, and K. Yétongnon, “Structural similarity evaluation between xml documents and dtds,” in *WISE*, pp. 196–211, 2007.

[20] T. Tran, R. Nayak, and P. Bruza, "Combining structure and content similarities for xml document clustering," in *AusDM*, pp. 219–226, 2008.

[21] X. Wan and J. Yang, "Using proportional transportation similarity with learned element semantics for xml document clustering," in *WWW '06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 961–962, ACM, 2006.

[22] J. Yang, W. K. Cheung, and X. Chen, "Integrating element and term semantics for similarity-based xml document clustering," in *Web Intelligence*, pp. 222–228, 2005.

[23] Z. Lin, H. Wang, S. McClean, and C. Liu, "All common embedded subtrees for measuring tree similarity," in *ISCID '08: Proceedings of the 2008 International Symposium on Computational Intelligence and Design*, (Washington, DC, USA), pp. 29–32, IEEE Computer Society, 2008.

[24] M. H. Chehreghani, M. Rahgozar, C. Lucas, and M. H. Chehreghani, "A heuristic algorithm for clustering rooted ordered trees," *Intell. Data Anal.*, vol. 11, no. 4, pp. 355–376, 2007.

[25] J. H. Hwang and M. S. Gu, "Clustering xml documents based on the weight of frequent structures," in *ICCIT '07: Proceedings of the 2007 International Conference on Convergence Information Technology*, (Washington, DC, USA), pp. 845–849, IEEE Computer Society, 2007.

[26] T. yang Lv, X. zhe Zhang, W. Zuo, and Z. Wang, "Xml clustering based on common neighbor," in *APWeb Workshops*, pp. 137–141, 2006.

[27] W. Liang and H. Yokota, "Lax: An efficient approximate xml join based on clustered leaf nodes for xml data integration," in *BNCOD*, pp. 82–97, 2005.

[28] J. H. Hwang and K. H. Ryu, "Clustering and retrieval of xml documents by structure," in *ICCSA (2)*, pp. 925–935, 2005.

[29] T. Dalamagas, T. Cheng, K.-J. Winkel, and T. Sellis, "A methodology for clustering xml documents by structure," *Inf. Syst.*, vol. 31, no. 3, pp. 187–228, 2006.

[30] W. Lian, D. W.-l. Cheung, N. Mamoulis, and S.-M. Yiu, "An efficient and scalable algorithm for clustering xml documents by structure," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 1, pp. 82–96, 2004.

[31] G. Costa, G. Manco, R. Ortale, and A. Tagarelli, "A tree-based approach to clustering xml documents by structure," in *PKDD*, pp. 137–148, 2004.

[32] J. Tekli, R. Chbeir, and K. Yétongnon, "An overview on xml similarity: Background, current trends and future directions," *Computer Science Review*, vol. 3, no. 3, pp. 151–173, 2009.

[33] L. Liu, Y. Zheng, B. Ding, and H. Liu, "A methodology for clustering xml documents based on labeled tree," in *FSKD (1)*, pp. 397–401, 2009.

[34] J. Yuan, X. Li, and L. na Ma, "An improved xml document clustering using path feature," in *FSKD (2)*, pp. 400–404, 2008.

[35] I. Choi, B. Moon, and H.-J. Kim, "A clustering method based on path similarities of xml data," *Data Knowl. Eng.*, vol. 60, no. 2, pp. 361–376, 2007.

[36] H. pong Leung, K. F.-L. Chung, S. C. fai Chan, and R. W. P. Luk, "Xml document clustering using common xpath," in *WIRI*, pp. 91–96, 2005.

[37] R. Nayak, "Fast and effective clustering of xml data using structural information," *Knowl. Inf. Syst.*, vol. 14, no. 2, pp. 197–215, 2008.

[38] B. Zhao, Y.-S. Zhang, and H.-X. Zhang, "A robust clustering method for xml documents," in *ICIII '08: Proceedings of the 2008 International Conference on Information Management, Innovation Management and Industrial Engineering*, (Washington, DC, USA), pp. 19–23, IEEE Computer Society, 2008.

[39] M. Alishahi, M. Ravakhah, B. Shakeriaski, and M. Naghibzade, "Xml document clustering based on common tag names anywhere in the structure," in *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pp. 588 –595, 20-21 2009.

[40] P. Antonellis, C. Makris, and N. Tsirakis, "Xedge: clustering homogeneous and heterogeneous xml documents using edge summaries," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, (New York, NY, USA), pp. 1081–1088, ACM, 2008.

[41] S. Iyer and D. A. Simovici, "Multisets and clustering xml documents," in *ICTAI '07: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, (Washington, DC, USA), pp. 267–274, IEEE Computer Society, 2007.

[42] R. Nayak and S. Xu, "Xcls: A fast and effective clustering algorithm for heterogenous xml documents," in *PAKDD*, pp. 292–302, 2006.

[43] T. Wang, D. xin Liu, X.-Z. Lin, W. Sun, and G. Ahmad, "Clustering large scale of xml documents," in *GPC*, pp. 447–455, 2006.

[44] J. H. Hwang and K. H. Ryu, "A new xml clustering for structural retrieval," in *ER*, pp. 377–387, 2004.

[45] J. H. Hwang and K. H. Ryu, "A new sequential mining approach to xml document clustering*," in *APWeb*, pp. 266–276, 2005.

[46] M. Kozielski, "Multilevel conditional fuzzy c-means clustering of xml documents," in *PKDD*, pp. 532–539, 2007.

[47] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. Zaki, "Xproj: a framework for projected structural clustering of xml documents," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 46–55, ACM, 2007.

[48] S. Helmer, "Measuring the structural similarity of semistructured documents using entropy," in *VLDB*, pp. 1022–1032, 2007.

[49] T. Tran, S. Kutty, and R. Nayak, "Utilizing the structure and content information for xml document clustering," in *INEX*, pp. 460–468, 2008.

[50] N. Polyzotis and M. Garofalakis, "Xcluster synopses for structured xml content," in *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, (Washington, DC, USA), p. 63, IEEE Computer Society, 2006.

[51] A. Tagarelli and S. Greco, "Toward semantic xml clustering," in *SDM*, pp. 188–199, 2006.

[52] A. Tagarelli and S. Greco, "Semxclust: A system for semantic xml clustering," in *SEBD*, pp. 72–79, 2006.

[53] M. Hagenbuchner, A. C. Tsoi, A. Sperduti, and M. Kc, "Efficient clustering of structured documents using graph self-organizing maps," in *INEX*, pp. 207–221, 2007.

[54] T. Tran, R. Nayak, and P. Bruza, "Document clustering using incremental and pairwise approaches," in *INEX*, pp. 222–233, 2007.

[55] S. Kutty, R. Nayak, and Y. Li, "Hcx: an efficient hybrid clustering approach for xml documents," in *DocEng '09: Proceedings of the 9th ACM symposium on Document engineering*, (New York, NY, USA), pp. 94–97, ACM, 2009.

[56] S. Kutty, R. Nayak, and Y. Li, "Xcfs: an xml documents clustering approach using both the structure and the content," in *CIKM '09: Proceeding*

*of the 18th ACM conference on Information and knowledge management*, (New York, NY, USA), pp. 1729–1732, ACM, 2009.

[57] N. Nagwani and A. Bhansali, "Clustering homogeneous xml documents using weighted similarities on xml attributes," in *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, pp. 369 –372, 19-20 2010.

[58] Y. Shen and B. Wang, "Clustering schemaless xml documents," in *CoopIS/DOA/ODBASE*, pp. 767–784, 2003.

[59] washington, "Xml data reprository," Dec. 2010.

[60] Wisconisn, "Wisconisns xml data bank," Dec. 2010.

[61] L. Denoyer and P. Gallinari, "Overview of the inex 2008 xml mining track," in *INEX*, pp. 401–411, 2008.

# CURRICULUM VITA

**ABDIRAHMAN MOHAMED ABDI DAUD**

Current and Permanent Address: P.O.Box 278 Dhahran 31261, Saudi Arabia

Tel: +966561235267

Email: mrabdi@gmail.com, daud@kfupm.edu.sa

Nationality: Somali

**Computer skills**

**Languages**

- Proficient in: Microsoft .NET Framework 4 more specifically C# and ASP.NET
- Familiar with: Java and PHP

**Software**

- Systems: Proficient in Microsoft SharePoint 2010 and SQL Server 2008. Familiar with Oracle 9i
- Design and testing Tools: Proficient in Enterprise Architect and NUnit

**Experience**

**Research Assistant (full time)**

Oct 2008-Present

*King Fahd University of Petroleum and Minerals (KFUPM)*

- Conducted funded research in artificial intelligence (Sentiment analysis on Islamic Hadith)
- Maintained and developed website of Deanship of Graduate Studies
- Helped the department in teaching undergraduate students.

**Manager**

Oct 2007-Sep 2008

*Al-Manal Company*

- In addition to managerial tasks, developed financial software similar to QuickBooks (using VB & MS ACCESS).

**Training**

**Software Developer**

Summer of 2007

*Futureware,* Khobar, Saudi Arabia

- Worked on programming problems on Qt C++ framework under Linux OS

*Paramount, Riyad, Saudi Arabia*

- Successfully configured/installed two solutions for a company in Riyadh

**Education**

**King Fahd University of Petroleum and Minerals**

2010 *Dhahran, Saudi Arabia*

- M.S., Computer Science (Artificial Intelligence –Data mining)
- Conducted a research with title: *HIERARCHAL CLUSTERING ALGORITHM FOR LARGE XML DATA*
- Gradation Date Jun 2011
- **First Honor** GPA: **3.87 out of 4**

**King Fahd University of Petroleum and Minerals**

*Dhahran, Saudi Arabia*

- B.S., Software Engineering
- Second Honor GPA: **3.4/4**
- Gradation Date: June 2007

**Latest Projects**

Web application for ERD Modeling (2009-2010)

*King Fahd University of Petroleum and Minerals*

- Implemented: Entity Relationship Diagram Tool: a KFUPM funded project to teach students how to create database schema
- Implemented using SQL2005/ASP.NET3.5 and C#
- Accessible at www2.kfupm.edu.sa/erdtool

**Latest Awards**

**Microsoft Intl. Imagine Cup (2007)**

- With a team of three, Won First place (out of KSA universities)
- Designed and implemented Smart Pal: a **Question-Answer Multi-agent system** that aims to replace search engines by creating a knowledge-based system from text
- Implemented using **.NET /XAML/ C#**

**Outstanding Awards**

*King Fahd University of Petroleum and Minerals*

- (2007) A Creativity certificate from **Saudi National Competition** of Computer Skills
- (2006) Sixth Place in KFUPM Creativity Competition (over 30 participants)
- (2002) Best Award for outstanding performance in KFUPM English Program
- (2002-2007) Several KFUPM Student Honor Awards

**More Information**

**Languages**

- Arabic (Native Speaker)
- English **(Fluent with TOEFL score 100 out of 120)**
- Somali (Fluent)

**Residence**

- Dhahran, Saudi Arabia ( as of June 2011)