XML Data Clustering: An Overview

Alsayed Algergawy
Magdeburg University,
Marco Mesiti
University of Milano,
Richi Nayak
Queensland University of Technology
and
Gunter Saake
Magdeburg University

In the last few years we have observed a proliferation of approaches for clustering XML documents and schemas based on their structure and content. The presence of such a huge amount of approaches is due to the different applications requiring the XML data to be clustered. These applications need data in the form of similar contents, tags, paths, structures and semantics. In this paper, we first outline the application contexts in which clustering is useful, then we survey approaches so far proposed relying on the abstract representation of data (instances or schema), on the identified similarity measure, and on the clustering algorithm. This presentation leads to draw a taxonomy in which the current approaches can be classified and compared. We aim at introducing an integrated view that is useful when comparing XML data clustering approaches, when developing a new clustering algorithm, and when implementing an XML clustering component. Finally, the paper moves into the description of future trends and research issues that still need to be faced.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications— Data mining;—*documentation*; H.3 [**Information Systems**]: Information Search and Retrieval, Online Information Services

General Terms: Documentation, Algorithms, Performance

Additional Key Words and Phrases: XML data, Clustering, Tree similarity, Schema matching, Semantic similarity, Structural similarity

1. INTRODUCTION

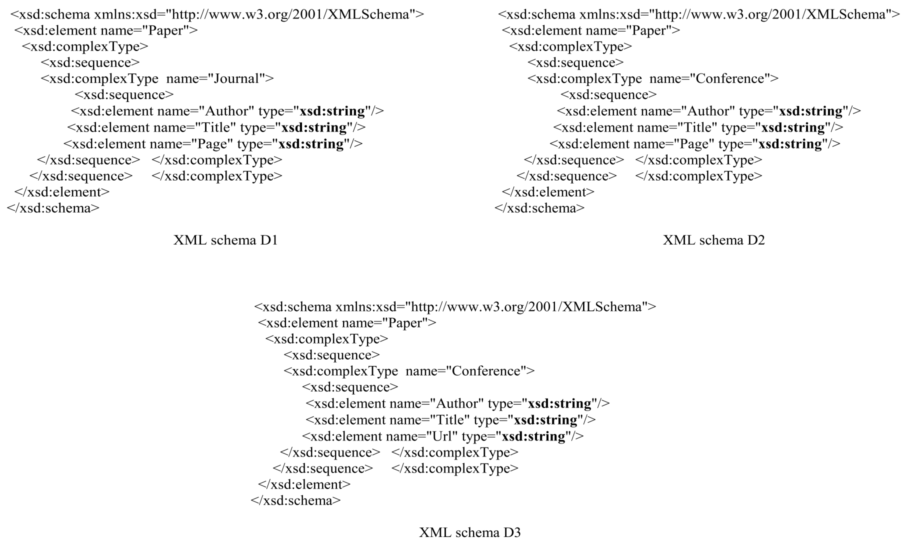
The eXtensible Markup Language (XML) has emerged as a standard for information representation and exchange on the Web and the Intranet [Wilde and Glushko 2008]. Consequently, a huge amount of information is represented in XML and several tools have

Author's address: A. Algergawy, Magdeburg University, Computer Science Department, 39106 Magdeburg, Germany, email:alshahat@ovgu.de; Marco Mesiti, DICO - Universita' di Milano, via Comelico 39/41, 20135 Milano (Italy) e-mail: mesiti@dico.unimi.it; Richi Nayak, Queensland University of Technology, Faculty of Science and Technology, GPO Box 2434 Brisbane, Australia, email: r.nayak@qut.edu.au; G. Saake, Magdeburg University, Computer Science Department, 39106 Magdeburg, Germany, email:saake@iti.cs.uni-magdeburg.de

been developed to deliver, store, integrate, and query XML data [Wang et al. 2004; Bertino and Ferrari 2001; Florescu and Kossmann 1999]. It becomes inevitable to develop high-performance techniques for efficiently managing and analyzing extremely large collections of XML data. One of the methods that many researchers have focused on is *clustering* that groups similar XML data according to their content and structures. The clustering process of XML data plays a crucial role in many data application domains, such as information retrieval, data integration, document classification, Web mining, and query processing.

Clustering, in general, is a useful technique for grouping data objects within a single group/cluster that share similar features, while placing objects in different groups that are dissimilar [Jain et al. 1999; Berkhin 2002; Xu and Wunsch 2005]. Specific research on clustering XML data is gaining momentum [Lee et al. 2002; Lian et al. 2004; Leung et al. 2005; Dalamagasa et al. 2006; Nayak and Tran 2007; Aggarwal et al. 2007; Choi et al. 2007; Nayak 2008] both for clustering XML documents and XML schemas according to their content and structures. Several XML schema languages have been proposed [Lee and Chu 2000] for the description of the structure and the legal building blocks of an XML document. Among them, XML DTD and XML Schema Definition (XSD) are commonly used. Since the document definition outlined in a schema holds true for all document instances of that schema, the result produced from the clustering of schemas is able to group together documents that present similar characteristics. However, in practice, some XML documents do not have an associated schema and schema instances might present different structures due to the employment of the choice operator. Therefore, algorithms for clustering both XML documents and XML schemas have attracted attention from researchers.

Clustering XML data is an intricate process and it differs significantly from clustering of flat data and text. The difficulties of clustering XML data are due to the following reasons [Aggarwal et al. 2007]. First, clustering algorithms require the computation of similarity between different sets of XML data, which is itself a difficult research problem (the heterogeneity in XML data presents many challenges to identify the ideal similarity function). Second, the structural organization of XML data increases implicit dimensionality that a clustering algorithm needs to handle, which leads to meaningless clusters. XML data have several features, such as semantic, structure, and content, each containing a set of sub-features. Clustering XML data considering one feature while ignoring the others fails to achieve accurate cluster results. For example, Fig. 1a shows three XML schemas representing journal and conference papers in the DBLP database. The data set has common elements such as “Author” and “Title”. Even if D1 and D2 have only one different element, they should be in two different clusters according to usual semantics that give different relevance to publications in journals and conferences. In contrast, even if D2 and D3 have only one different element, they should be in the same cluster because both refer to conference papers. Furthermore, the need to organize XML documents according to their content and structure has become challenging, due to the increase of XML data heterogeneity. Fig. 1b depicts the fragments of six XML documents from the publishing domain: the XML fragments shown in (a), (b), (c) and (f) share a similar structure, and the fragments in (d) and (e) share a similar structure. It can be observed that the fragments in (a) and (f) share a similar structure to fragments in (b) and (c), however, these two sets of fragments differ in their content. These documents will be grouped in two clusters about “Books” and “Conference Articles” if structural similarity is considered as a criterion for clustering. However, this kind of grouping will fail to further distinguish the documents in



```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Paper">
  <xsd:complexType>
     <xsd:sequence>
     <xsd:complexType  name="Journal">
           <xsd:sequence>
          <xsd:element name="Author" type="xsd:string"/>
         <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Page" type="xsd:string"/>
    </xsd:sequence>   </xsd:complexType>
   </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML schema D1

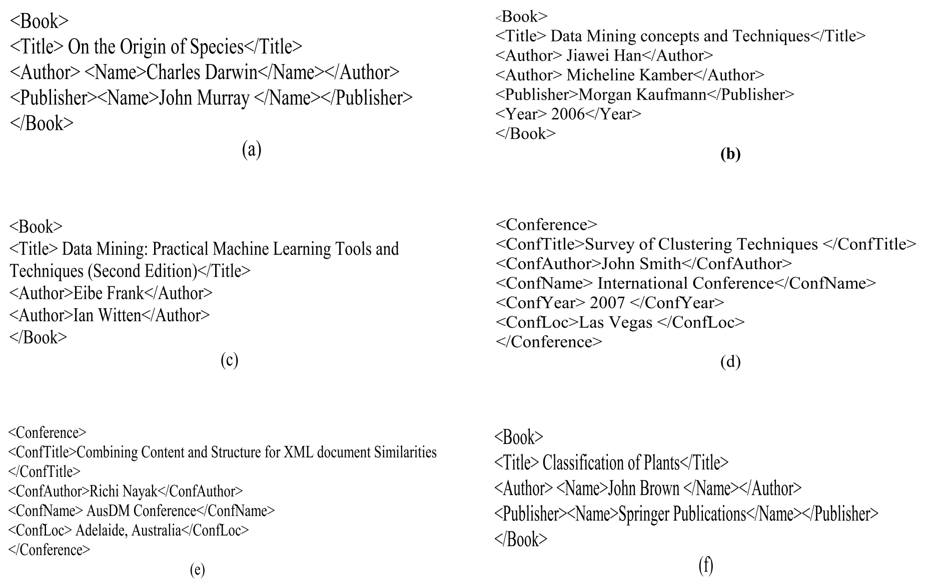
```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Paper">
  <xsd:complexType>
     <xsd:sequence>
     <xsd:complexType  name="Conference">
           <xsd:sequence>
          <xsd:element name="Author" type="xsd:string"/>
         <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Page" type="xsd:string"/>
    </xsd:sequence>   </xsd:complexType>
   </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML schema D2

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Paper">
  <xsd:complexType>
     <xsd:sequence>
     <xsd:complexType  name="Conference">
        <xsd:sequence>
         <xsd:element name="Author" type="xsd:string"/>
         <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Url" type="xsd:string"/>
    </xsd:sequence>   </xsd:complexType>
   </xsd:sequence>    </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

XML schema D3

(a) A set of XML schemas.



```
<Book>
<Title> On the Origin of Species</Title>
<Author> <Name>Charles Darwin</Name></Author>
<Publisher><Name>John Murray </Name></Publisher>
</Book>
```

(a)

```
<Book>
<Title> Data Mining concepts and Techniques</Title>
<Author> Jiawei Han</Author>
<Author> Micheline Kamber</Author>
<Publisher>Morgan Kaufmann</Publisher>
<Year> 2006</Year>
</Book>
```

(b)

```
<Book>
<Title> Data Mining: Practical Machine Learning Tools and
Techniques (Second Edition)</Title>
<Author>Eibe Frank</Author>
<Author>Ian Witten</Author>
</Book>
```

(c)

```
<Conference>
<ConfTitle>Survey of Clustering Techniques </ConfTitle>
<ConfAuthor>John Smith</ConfAuthor>
<ConfName> International Conference</ConfName>
<ConfYear> 2007 </ConfYear>
<ConfLoc>Las Vegas </ConfLoc>
</Conference>
```

(d)

```
<Conference>
<ConfTitle>Combining Content and Structure for XML document Similarities
</ConfTitle>
<ConfAuthor>Richi Nayak</ConfAuthor>
<ConfName> AusDM Conference</ConfName>
<ConfLoc> Adelaide, Australia</ConfLoc>
</Conference>
```

(e)

```
<Book>
<Title> Classification of Plants</Title>
<Author> <Name>John Brown </Name></Author>
<Publisher><Name>Springer Publications</Name></Publisher>
</Book>
```

(f)

(b) A set of XML documents.

Fig. 1: Examples of XML data.

the "Books" cluster that contains books of several genres. On the other hand, clustering of documents based only on content features similarity will fail to distinguish between conference articles and books that follow two different structures. In order to derive a meaningful grouping, these fragments should be analyzed in terms of both their structural and content features similarity. Clustering the XML documents by considering the structural and content features together will result in three clusters, namely "Books on Data Mining (DM)", "Books on Biology (Bio)" and "Conference articles on Data Mining".

In order to conduct a good survey and to construct a fair base for comparing existing XML data clustering approaches, a high-level architecture for a generic framework of XML data clustering is proposed. Inspired from data clustering activity steps [Jain et al. 1999], Fig. 2 depicts the framework of XML data clustering with three basic phases.

(1) *Data representation.* XML data are represented using a common data model that captures semantic and structure information inherent in XML data. This phase includes two subphases that are *feature selection* and *feature extraction*. Feature selection chooses distinctive features from a set of candidates, while feature extraction employs rules to generate useful and novel features from the original ones. We elaborate on XML data representation in Section 3.

(2) *Similarity computation.* The proximity functions to measure the similarity between pairs of data objects are determined. XML data are grouped according to the similarity of the extracted/selected features. Performance of the clustering solution mainly depends upon the similarity measure employed. Based on the type of data model used to represent XML data, several XML similarity measures have been proposed. We discuss XML similarity measures in Section 4.

(3) *Clustering/grouping.* Similar XML data are grouped together based on a proximity function using a proper clustering algorithm. The majority of clustering algorithms are implicitly or explicitly linked to the similarity measures employed. In Fig. 2, the thin arrows between the "similarity computation" and the "clustering/grouping" boxes indicate that the grouping process can be interleaved with the similarity computation phase. Finally, the output of the clustering framework can be represented either as a set of clusters or as nested sets of data (hierarchies) depicted as dotted lines in Fig. 2. We make a detailed discussion on the clustering approaches and the evaluation of the quality of their application in Section 5.

This paper presents an overview of XML data clustering methodologies and implementations in order to draw a road map of using clustering algorithms in XML data management. The paper starts from the application contexts where XML data clustering is useful, and then surveys the current approaches, and presents a taxonomy that explains their common features. The paper also includes a discussion on the challenges and benefits that the field of XML data clustering brings forward. It is hoped that the survey would be helpful both to developers of new approaches and to users who need to select a method from a library of available approaches.

The remainder of the paper is organized as follows. Section 2 surveys different application domains which utilize the output of XML data clustering. The three phases of the generic clustering framework are discussed in Sections 3, 4, and 5, respectively. In Section 6, the existing approaches are presented and compared according to the introduced framework. Concluding remarks and open research directions are presented in Section 7.

2. APPLICATION DOMAINS

To motivate the importance of clustering XML data, we summarize its use in several data application domains.

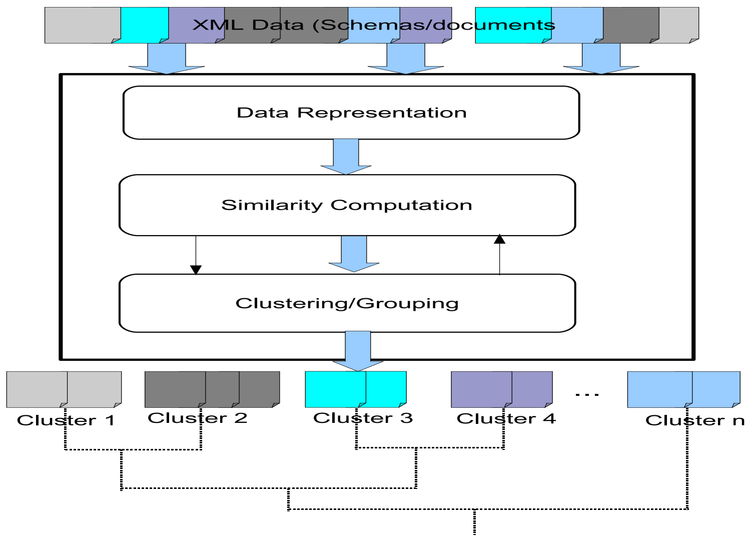


Fig. 2: A Generic XML data clustering framework.

2.1 XML Query Processing

Although XML is mainly perceived as a standard medium of information exchange, storing, indexing and querying of XML data are still important issues and have become research hotspots both in the academic and in the industrial communities [Wang et al. 2004; Melton and Buxton 2006; Gou and Chirkova 2007]. Both XML native (e.g., Tamino, eXist, TIMBER) and enabled (e.g., Oracle, IBM DB2, SQL Server) Database Management Systems (DBMSs) have been so far proposed [Bourret 2009] for storing and querying XML documents. Native DBMSs rely on a data model specifically conceived for the management of XML, whereas enabled DBMSs are relational or object-relational ones that have been extended for the treatment of XML. Enabled XML DBMSs are more mature than the native ones because supported by big vendors and the integration of XML data with other company data is easier. Some enabled DBMSs support XML Schema for the specification of a mapping between an XSD and internal relational or object-relational representation of XML documents [Florescu and Kossmann 1999; Shanmugasundaram et al. 1999].

Query languages like Xpath and XQuery have been developed for accessing and manipulating XML documents in their native representation and well as extension to the SQL standard have been conceived to handle XML data besides relational data [Melton and Buxton 2006]. All the standards so far proposed do not deal with data heterogeneity. To deal with heterogeneous queries, approximation techniques have been proposed to evaluate twig patterns [Gou and Chirkova 2007]. Twig patterns are simple tree-structured queries for XML that include three basic language elements, namely node conditions, parent-child edges, and ancestor-descendant edges. Twig patterns are applicable to information-retrieval (IR) as well as database settings. Database-style queries return all results that precisely match (the content and structure requirements) the query, while, IR-style queries allow “fuzzy” results, which are ranked based on their query relevance.

Although twig pattern matching has become an important research area and several approaches have been developed to tackle it, it suffers from several drawbacks especially in large scale XML data and complex twig patterns, where data related to the query appear in a small part of the whole XML document. So if we can access only parts of the data that we need, the query processing can be conducted more efficiently because the search space is reduced by skipping unnecessary data during the query processing. A good solution is thus to consider clustering approaches in order to partition the whole XML data based on their common content, semantics and structures [Lian et al. 2004; Choi et al. 2007].

2.2 XML Data Integration

XML is widely used as the medium of data exchange among Web applications and enterprises. Integration of distributed XML data is thus becoming a research problem. This is due to the large number of business data appearing on the Web and a large number of service-oriented architecture is being adapted in the form of Web services. XML data integration includes the construction of a global view for a set of independently developed XML data [Batini et al. 1986; Le et al. 2006; Bertino and Ferrari 2001].

Since XML data are engineered by different people, they often have different structural and terminological heterogeneities. The integration of heterogeneous data sources requires many tools for organizing and making their structure and content homogeneous. XML data integration is a complex activity that involves reconciliation at different levels: (1) at *schema level*, reconciling different representations of the same entity or property, and (2) at *instance level*, determining if different objects coming from different sources represent the same real-world entity. Moreover, the integration of Web data increases the integration process challenges in terms of heterogeneity of data. Such data come from different resources and it is quite hard to identify the relationship with the business subjects. Therefore, a first step in integrating XML data is to find clusters of the XML data that are similar in semantics and structure [Lee et al. 2002; Viyanon et al. 2008]. This allows system integrators to concentrate on XML data within each cluster. We remark that reconciling similar XML data is an easier task than reconciling XML data that are different in structures and semantics, since the later involves more restructuring.

There are two directions of using clustering in XML data integration. (1) A similarity matrix across XML data is determined and a clustering algorithm is applied to the computed similarity matrix producing clusters of similar XML data. Then, the XML data within each cluster are integrated [Lee et al. 2002]. (2) Each XML data tree is clustered into subtrees, which reduces the number of comparisons dramatically. The similarity degree based on data and structures of each pair of subtrees is then measured. The data tree similarity degree is calculated from the mean value of similarity degrees of matched subtrees. If the data tree similarity degree is greater than a given threshold, the two XML documents can be integrated [Viyanon et al. 2008].

2.3 XML Information Retrieval

Traditional Information Retrieval (IR) systems [Singhal 2001] rely either on the Boolean model or the Vector Space model to represent the flat structure of documents as a bag of words. Extensions of these models have been proposed, e.g., the fuzzy Boolean model and the knowledge-aware model. However, all of these indexing models do ignore the structural organization of text. XML documents have a hierarchical structure defined by a DTD or an XML schema. While this structure allows documents to be represented with hierarchical levels of granularity in order to achieve better precision by means of focused retrieval, it implies more requirements on the representation and retrieval mechanisms. The retrieval of XML documents using IR techniques is known as XML-IR. Growing enthusiasm centering around XML retrieval led to the formation of the “Initiative for the Evaluation of XML retrieval” (or INEX in short)[1]. Organized each year since 2002, INEX is a TREC[2]-like forum where participating researchers can discuss and evaluate their retrieval techniques using uniform scoring procedures over a reasonably large relevance-assessed test collection. With the growing popularity of XML clustering techniques, INEX 2009 includes a clustering track in which clustering is used to organize a very large data set in minimal number of clusters that need to be searched to satisfy a given query and/or reorganize results furnished by an initial search system as response to a users query.

An XML-IR process starts when a user submits a query into the system. Executing the user query on the huge amount of XML documents is a time-consuming and error-prone process. A nice solution is to first cluster together semantically and structurally similar XML documents. Then, the user query is executed on one or more related clusters [Lian et al. 2004; Tagarelli and Greco 2006]. Another way clustering can improve the XML-IR process is using the clustering results for ranking the documents to be returned to the user. The process is to take the search results of an XML search engine, cluster them, and present them to the user in semantically distinct groups [Vutukuru et al. 2002]. This assists in obtaining a unique cluster containing all relevant documents or a set of clusters addressing the different aspects of the relevant information.

2.4 Web Mining

With the huge amount of information available online, the Web is a fertile area for data mining research. Clustering XML data is a relevant problem in Web mining and consists of the process of organizing data circulated over the Web into groups/clusters in order to facilitate data availability and accessing, and at the same time to meet user preferences. In an effort to keep up with the tremendous growth of the Web, many approaches and systems have been proposed in order to organize their contents and structures to make it easier for the users to efficiently and accurately find the information they want. According to the type of mined information and the goal of the mining process, these methods can be broadly classified into three categories. [Vakali et al. 2004; Pal et al. 2002]: (1) *Web structure mining* referring broadly to the process of uncovering interesting and potentially useful knowledge about the Web, (2) *Web usage mining* using the Web-log data coming from users’ sessions to group together a set of users’ navigation sessions having similar characteristics, and (3) *Web content mining* clustering methods to try to identify inherent groupings of pages so that a set of clusters is produced in which relevant pages (to a specific topic) and irrelevant pages are separated in the clustering process. XML clustering techniques can help in conceptualizing Web structure, usage and content mining with a better use of structural and content information represented in XML documents.

2.5 Bioinformatics

Bioinformatics represents a new field of scientific inquiry, devoted to answering questions about life and using computational resources to answer those questions. A key goal of bioinformatics is to create database systems and software platforms capable of storing and analyzing large sets of biological data. To that end, hundreds of biological databases are

[1] http://inex.is.informatik.uni-duisburg.de/
[2] http://trec.nist.gov/

now available and provide access to a diverse set of biological data. Given the diversity and the exponential growth of biological data sets, and the desire to share data for open scientific exchange, the bioinformatics community is continually exploring new options for data representation, storage, and exchange. In the past few years, many in the bioinformatics community have turned to XML to address the pressing needs associated with biological data [Cerami 2005].

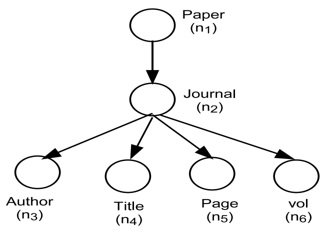
XML-like presentations have been proposed for the following bio-molecular data types. (1) Principal bio-molecular entities (DNA, RNA, and proteins), for example, the Bioinformatic sequence markup language (BSML) [Hucka and et al. 2003] has been used to describe biological sequences (DNA, RNA, protein sequences), while ProXML [Hanisch et al. 2002] is used to represent protein sequences. (2) Biological expression (microarray), the MAGE project [3] provides a standard for the representation of microarray expression data to facilitate their exchange among different data systems. (3) System biology, the need to capture the structure and content of biomolecular and physiological systems has led to develop the System Biology Markup Language (SBML)[4].

In general, clustering analysis can be used in two main directions: gene clustering [Andreopoulos et al. 2009; Tamayo et al. 1999; Eisen et al. 1998] and DNA or protein sequences clustering [Sasson et al. 2002; Somervuo and Kohonen 2000]. Results of gene clustering may suggest that genes in the same group have similar features and functionalities, or they share the same transcriptional mechanism. The authors in [Jeong et al. 2006] propose a scheme for grouping and mining similar elements with structural similarities from an XML schema for biological information, in which a number of elements and attributes are defined. cluML, a free, open, XML-based format, is a new markup language for microarray data clustering and cluster validity assessment. This format has been designed to address some of the limitations observed in traditional formats, such as inability to store multiple clustering (including biclustering) and validation results within a data set [Bolshakova and Cunningham 2005]. On the other hand, several clustering techniques have been proposed and applied ti organize DNA or protein sequence data. CONTOUR is a new approach that mines a subset of high-quality subsequences directly in order to cluster the input sequences [Wang et al. 2009].

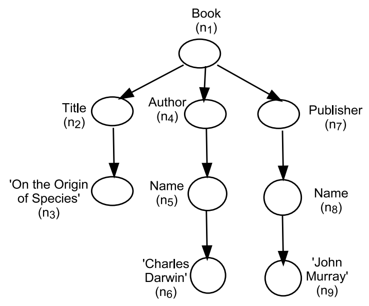
3. DATA REPRESENTATION

XML data can be represented using a common data model, such as rooted labeled trees, directed acyclic graphs, or vector-based techniques. The data model should capture both content and structure features of XML data and it is the basis for the identification of the features to be exploited in the similarity computation (see Section 4). Data representation starts with parsing XML data using an XML parsing tool, such as the SAX parser[5]. In case of XML schema clustering, the parsing process may be followed by a normalization process to simplify the schema structure according to a series of predefined transformation procedures similar to those in [Lee et al. 2002]. In the remainder of the section the two most commonly used models to represent XML data are discussed.

[3]MAGE project: http://www.mged.org/Workgroups/MAGE/mage.html
[4]http://sbml.org/
[5]`http://www.saxproject.org`



(a) Data tree of schema D1.



(b) Data tree of XML document.

Fig. 3: Tree representation of XML data.

3.1 Tree-based representation

XML data can be represented as a data tree. A *data tree* (DT) is a rooted labeled tree defined as a 3-tuple *DT = (N$_T$, E$_T$, Lab$_{NT}$)*, where:

—$N_T = \{n_{root}, n_2, ..., n_n\}$ is a finite set of nodes, each of them is uniquely identified by an object identifier (OID), where n_{root} is the tree root node. Three types of nodes in a data tree can basically occur:
  (1) *Element nodes*. They correspond to element declarations or complex type definitions in XML schemas or to tags in XML documents.
  (2) *Attribute nodes*. These correspond to attribute declarations in XML schemas or to attributes associated to tags in XML documents.
  (3) *Text nodes*. These correspond to values in XML documents and basic types in XML schemas.

—$E_T = \{(n_i, n_j)|n_i, n_j \in N_T\}$ is a finite set of edges, where n_i is the parent of n_j. Each edge represents a relationship between two nodes.

—*Lab*$_{NT}$ is a finite set of node labels. These labels are strings for describing the properties of the element and attribute nodes, such as *name, data type*, and *cardinality*, or they are the data values associated with text nodes.

Fig. 3 illustrates the tree representation of XML data. Specifically, Fig. 3a shows the data tree of schema $D1$ represented in Fig. 1a, while Fig. 3b represents an XML document depicted in Fig. 1b. Each node in the data tree is associated with the *name* label, (such as "Author" and "Title") as well as its OID, (such as n_1 and n_2). In Fig 3a, the nodes n_1, n_2 and n_4 represent examples of element nodes, node n_6 is an attribute node. In Fig 3b node n_9 is a text node. A data tree DT is called an *ordered labeled tree* if a left-to-right order among siblings in DT is given, otherwise it is called an *unordered tree*.

Given a tree representation of XML data, an *object* (OP) is a single portion to be exploited for the computation of similarity. The property set associated to each object is called the *object feature*. We classify objects in a data tree into: *complex objects*, which include the whole data tree, subtrees, and paths, and *simple objects*, which include element, attribute, and text nodes. Furthermore, there exist many relationships among (simple) objects which reflect the hierarchical nature of the XML data tree. These relationships include:

—*parent-child (induced)* relationships, that is the relationship between each element node and its direct subelement/attribute/text node;

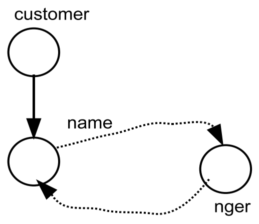
—*ancestor-descendant (embedded)* relationships, that is the relationship between each element node and its direct or indirect subelement/attribute/text nodes;

—*order* relationships among siblings.

The tree-based representation of XML data introduces some limitations for the presentation of these relationships. For example, association relationships that are structural relationships specifying that two nodes are conceptually at the same level, are not included in the tree representation. Association relationships essentially model key/keyref and substitution group mechanisms. As a result, another tree like structure, such as directed acyclic graphs, is used [Boukottaya and Vanoirbeek 2005]. Fig. 4(a) represents a substitution group mechanism, which allows customer names to be in an English or a German style. This type of relationship can not be represented using the tree representation. It needs a bi-directional edge, as shown in Fig. 4(b).

```
...
<xs:element name="name" type="xs:string"/>
<xs:element name="nger" substitutionGroup="name"/>
<xs:element name="customer" >
<xs:complexType >
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>
<xs:element/>
...
```

(a) Substitution group example.



(b) Data graph.

Fig. 4: Graph representation of an XML schema part.

3.2 Vector-based representation

The Vector Space Model (VSM) model [Salton et al. 1975] is a widely used data representation for text documents. In this model each document is represented as a feature vector of the words that appear in documents of the data set. The term weights (usually term frequencies) of the words are also contained in each feature vector. VSM represents a text document, doc_x, using the document feature vector, d_x, as [Yang et al. 2009]

$$d_x = [d_{x(1)}, d_{x(2)}, ..., d_{x(n)}]^T, d_{x(i)} = TF(\rho_i, dox_x).IDF(\rho_i)$$

where $TF(\rho_i, doc_x)$ is the frequency of the term ρ_i of doc_x, $IDF(\rho_i) = \log(\frac{|D|}{DF(\rho_i)})$ is the inverse document frequency of the term ρ_i for discounting the importance of the frequently appearing terms, $|D|$ is the total number of documents, $DF(\rho_i)$ is the number of documents containing ρ_i, and n is the number of distinct terms in the document set. Applying VSM directly to represent semi-structured documents is not desirable, as the document syntactic structure tagged by their XML elements will be ignored.

XML data generally can be represented as vectors in an abstract n-dimensional feature space. A set of objects can be extracted from an XML data. Each object has a set of features, where each feature is associated with a domain to define its allowed values. The

```
  <Paper>
   <Journal>
        <Author>John</Author>
        <Title> XML Data Clustering</Title>
         <Page>25-36</Page>
    </Journal>
</Paper>
```

(a) XML document instance of D1.

$$d_x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{array}{c} \text{John} \\ \text{XML} \\ \text{Data} \\ \text{Clustering} \\ 25 \\ 36 \end{array} \qquad \Delta_x = \begin{bmatrix} \text{author} & \text{title} & \text{page} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Feature vector (d_x) and feature matrix (Δ_x) of D1.

XML doc. / path	D_1	D_2	D_3	D_4	D_5	D_6
P_1(Book/Title)	1	1	1	0	0	1
P_2(Book/Author/Name)	1	0	0	0	0	1
P_3 (Book/Publisher/Name)	1	0	0	0	0	1
P_4(Book/Author)	0	2	2	0	0	0
P_5(Book/Publisher)	0	1	0	0	0	0
P_6(Book/Year)	0	1	0	0	0	0
P_7(Conference/ConfTitle)	0	0	0	1	1	0
P_8(Conf/ConfAuthor)	0	0	0	1	1	0
P_9(Conf/ConfName)	0	0	0	1	1	0
P_{10}(Conf/ConfYear)	0	0	0	1	0	0
P_{11}(Conf/ConfLoc)	0	0	0	1	1	0

(c) A structure-based representation for Fig.1(b).

XML doc. / path	D_1	D_2	D_3	D_4	D_5	D_6
P_1	1	1	1	0	0	1
P_2	1	0	0	0	0	1
P_3	1	0	0	0	0	1
P_4	0	1	1	0	0	0
P_5	0	1	0	0	0	0
P_6	0	1	0	0	0	0
P_7	0	0	0	1	1	0
P_8	0	0	0	1	1	0
P_9	0	0	0	1	1	0
P_{10}	0	0	0	1	0	0
P_{11}	0	0	0	1	1	0

(d) A bitmap index for Fig. 1(b).

Fig. 5: Vector representation of an XML document.

level of an XML element is a feature whose domain are the positive integers (0 for the root, 1 for the first level, and so on), while the name of the element is another feature whose domain is string. This representation model introduces a challenge to incorporate the structural relationships between elements. In the following, we elaborate on different methods used to model XML data based on their either content, structure, or content & structure.

—*Content-based representation.* Each XML document, doc_x, is represented using a vector, called the feature vector d_x, that stores the frequency of (distinct) words in the document. Fig. 5(a,b) shows the content-based representation of D1 depicted in Fig. 1a.

—*Structure-based representation.* In this case, the feature vector capture the structure information of the XML document instead of the content information. Several methods have proposed to achieve this task. Most of them are based on exploiting path information. The approach in [Tran et al. 2008] models each XML document as a vector $\{P_1, P_2, ..., P_n\}$, where each element, P_i, of the vector represents the frequency of the path that appears in the document, as shown in Fig. 5c. A path, P_i, contains element names from the root element to a leaf element. The leaf element is an element that contains the textual content. The bitmap index technique is also used to represent XML data [Yoon et al. 2001]. A set of XML documents is represented using a 2-dimensional matrix. As illustrated in Fig. 5d, if a document has $path$, then the corresponding bit in the bitmap index is set to 1. Otherwise, all bits are set to 0.

—*Content and structure-based representation.* Representing XML documents using either the content or the structure feature is not sufficient to effectively model them. To this,

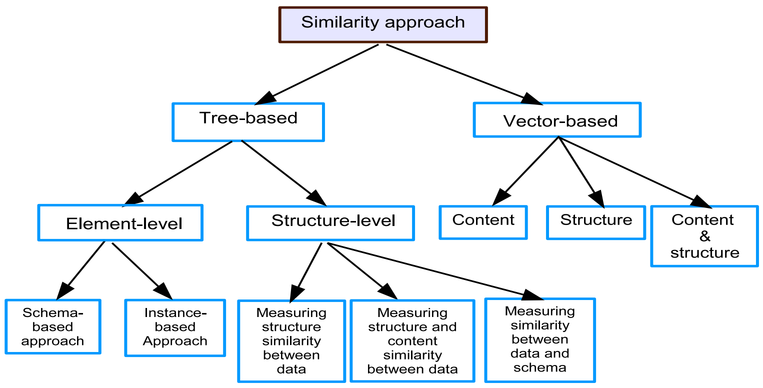


Fig. 6: Similarity measures.

both features should be taken into account. An extended version of the vector space model called *structural link vector model (SLVM)* is used to capture syntactic structure tagged by XML elements [Yang et al. 2009]. SLVM represents an XML document doc_x using a document feature matrix $\Delta_x \in R^{n\times m}$, given as

$$\Delta_x = [\Delta_{x(1)}, \Delta_{x(2)}, ..., \Delta_{x(m)}]$$

where m is the number of distinct XML elements, $\Delta_{x(i)} \in R^n$ is the TFIDF feature vector representing the ith XML element, $(1 \leq i \leq m)$, given as $\Delta_{x(i)} = TF(\rho_j, doc_x.e_i).IDF(\rho_j)$ for all $j = 1$ to n, where $TF(\rho_j, doc_x.e_i)$ is the frequency of the term w_j in the element e_i of doc_x. The SLVM representation of an XML document instance for D1 depicted in Fig. 1a is reported in Fig. 5b, which illustrates, for example, that the term *XML* appears one time in D1 (from the document feature vector d_x) under the element *title* (from the document feature matrix Δ_x).

Another vector-based representation that captures both structure and content of the XML data is represented in [Yoon et al. 2001]. The bitmap indexing technique, shown in Fig. 5d is extended, where a set of XML documents is represented using a 3-dimensional matrix, called BitCube. Each document is defined as a set of $(path, word)$, where $path$ is a root-to-leaf path, and $word$ denotes the word or content of the path. If a document has $path$, then the corresponding bit in the bitmap index is set to 1. Otherwise, all bits are set to 0 (and if $path$ contains a word, the bit is set to 1, and 0 otherwise).

4. SIMILARITY MEASURES AND COMPUTATION

Starting from the representation model of objects and their features, the similarity between XML data can be identified and determined by exploiting objects, objects' features, and relationships among them. There are various aspects that allow the description and categorization of XML data similarity measures, such as the kind of methodology being used, the kind of XML data representation, and the planned application domain [Tekli et al. 2009]. In the following, for homogeneity of presentation, we survey several XML similarity measures based on the used data representation, as shown in Fig. 6.

4.1 Tree-based similarity approaches

The computation of similarity among XML data represented as data trees depends on the exploited objects on which similarity functions to be applied. A function, *Sim*, is a similarity measure that exploits the features of objects as well as the relationships among them, in order to determine the proximity between objects. It is represented as $Sim(OP_1, OP_2)$, and its value ranges between 0 and 1, when the measure is normalized. The value 0 means strong dissimilarity between objects, while the value 1 means strong similarity. Based on the objects used to compute the similarity among XML data trees, we classify the similarity measures into: *element-level measures* and *structure-level measures*.

4.1.1 *Element-level Measures.* These measures, also known as *schema matching-based* methods, consider (simple) objects' details such as name, data type as well as the relationships between objects. In element-level measures, the similarity between XML data is based on the computed similarities among their elements. The similarity between two simple objects in two document trees $OP_1 \in DT_1$ and $OP_2 \in DT_2$ can be determined using the following equation:

$$Sim(OP_1, OP_2) = w_s \times SSim(OP_1, OP_2) + w_x \times CSim(OP_1, OP_2)$$

where $SSim(OP_1, OP_2)$ represents the simple similarity measure between two objects exploiting their features, such as name, data type, and constraint, while $CSim(OP_1, OP_2)$ represents the complex similarity measure between them exploiting the relationships of each object, and w_s and w_x are weights to quantify the importance of each measure, $(w_s, w_x \geq 0, w_s + w_x = 1)$. These computed similarities are then aggregated to determine the semantic similarities among paths and XML data trees themselves [Rahm and Bernstein 2001; Kade and Heuser 2008].

—*Simple Similarity Measures:* These measures determine the similarity between two objects $OP1 \in DT1$ and $OP2 \in DT2$ using their features.

(1) *Name Similarity*. Object names can be semantically similar (e.g. person, people) or syntactically similar (e.g. name, fname). Hence, both semantic and syntactic measures are included to determine the degree of similarity between objects' names. Semantic measures rely heavily on external or user-defined dictionaries such as WordNet. In the syntactic measures, each object names is decomposed into a set of tokens T_1 and T_2 using a customizable tokenizer using punctuation, upper case, special symbols, and digits, e.g. LastName $\rightarrow$ $\{$Last, Name$\}$. The name similarity between the two sets of name tokens T_1 and T_2 is determined as the average best similarity of each token with a token in the other set. It is computed as follow:

$$Nsim(T_1, T_2) = \frac{\sum_{t_1 \in T_1}[\max_{t_2 \in T_2} sim(t_1, t_2)] + \sum_{t_2 \in T_2}[\max_{t_1 \in T_1} sim(t_2, t_1)]}{|T1| + |T2|}$$

To measure the string similarity between a pair of tokens, $sim(t_1, t_2)$, several string similarity functions, such as the edit distance and n-grams, can be used [Cohen et al. 2003].

(2) *Data Type Similarity*. The Built-in XML data types hierarchy[6] can be used in order to compute data type similarity. Based on the XML schema data type hierarchy, a

[6]`http://www.w3.org/TR/xmlschema-2/`

data type compatibility table is built, as the one used in [Madhavan et al. 2001], as shown in Fig. 7(a). The figure illustrates that elements having the same data types or belonging to the same data type category have the possibility to be similar and their type similarities ($Typesim$) are high. The type similarity between two objects using the type compatibility table is:

$$Typesim(type1, type2) = TypeTable(type1, type2)$$

(3) *Constraint Similarity*. Another feature of an object that makes a small contribution in determining the simple similarity is its cardinality constraint. The authors of XClust [Lee et al. 2002] have defined a cardinality table for DTD constraints, as shown in Fig. 7(b). Also, the authors of PCXSS [Nayak and Tran 2007] have adapted the cardinality constraint table for constraint matching of XSDs. The cardinality similarity between two objects using the constraint cardinality table is:

$$Cardsim(card1, card2) = CardinalityTable(card1, card2)$$

(4) *Content Similarity*. To measure the similarity between *text nodes* of XML documents, the content information of these nodes should be captured. To this end, a token-based similarity measure can be used. According to the comparison made in [Cohen et al. 2003], the TFIDF ranking performed best among several token-based similarity measures. TFIDF (Term Frequency Inverse Document Frequency) is a statistical measure used to evaluate how important a word is to a document or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. It is used here to assess the similarity between text nodes. In this case, the content of these nodes can be considered as multisets (or bags) of words. Given the content of two *text nodes* represented as texts $cont_1$ and $cont_2$, The TFIDF measure between them is given by [Cohen et al. 2003]:

$$Contsim(cont_1, cont_2) = \sum_{w \in cont_1 \cap cont_2} V(w, cont_1).V(w, cont_2)$$

where,

$$V(w, cont_1) = \frac{V'(w, cont_1)}{\sqrt{\sum_{w'} V'(w, cont_1)^2}}$$

and,

$$V'(w, cont_1) = \log(TF_{w,cont_1} + 1). \log(IDF_w)$$

where, $TF_{w,cont_1}$ is the frequency of the word w in $cont_1$, and IDF_w is the inverse of the fraction of names in the corpus that contain w.

These simple similarity measures are then aggregated using an aggregation function, such as the weighted sum, to produce the simple similarity between two objects.

type1	type2	$Typesim$
string	string	1.0
string	decimal	0.2
decimal	float	0.8
float	float	1.0
float	integer	0.8
integer	short	0.8

(a) Type compatibility table.

	*	+	?	none
*	1	0.9	0.7	0.7
+	0.9	1	0.7	0.7
?	0.7	0.7	1	0.8
none	0.7	0.7	0.8	1

(b) Cardinality constraint table.

Fig. 7: Type compatible & cardinality tables.

$$\begin{aligned} SSim\,(OP_1, OP_2) = \quad & w_n \times Nsim(name1, name2) \\ & + w_t \times Typesim(type1, type2) \\ & + w_c \times Cardsim(card1, card2) \\ & + w_{co} \times Contsim(cont1, cont2) \end{aligned}$$

where w_n, w_t, w_c and w_{co} are weights to quantify the importance of each similarity measure, and $w_n + w_t + w_c + w_{co} = 1$. In XML schema, the value for w_{co} is set to 0, while in the XML document context $w_t = w_c = 0$. w_n is assigned a larger value w.r.t. w_t and w_c while comparing XML schema elements. In fact, the tuning of weight values is a challenge and needs more attention [Lee et al. 2007].

—*Complex Similarity Measures*. These measures determine the similarity between two objects $OP1 \in DT1$ and $OP2 \in DT2$ using their relationships, and exploiting the computed simple similarities between objects. In general, these measures depend on the object (node) context, which is reflected by its ancestors and its descendants, as shown in Fig. 8. The descendants of an object include both its immediate children and the leaves of the subtrees rooted at the element. The immediate children reflect its basic structure, while the leaves reflect the element's content. As a sequence, these measures may depend on:

(1) *Child Context*. The child context of an object (node) is the set of its immediate children. To compute the child context similarity between two objects OP_1 and OP_2, the child context set is first extracted for each node, then the simple similarity between each pair of children in the two sets is determined, the matching pairs with maximum similarity values are selected, and finally the average of best similarity values is computed.

(2) *Leaf Context*. The leaf context of an object (node) is the set of leaf nodes of subtrees rooted at the node. To determine the leaf context similarity between two objects OP_1 and OP_2, the leaf context set is extracted for each node, then a suitable comparison function between two sets is applied.

(3) *Ancestor Context*. The ancestor context of an object (node) is the path extending from the root node to the node itself. To measure the ancestor context similarity between the two objects OP_1 and OP_2, each ancestor context , say path P_1 for OP_1 and P_2 for OP_2, has to be extracted. Then, the two paths are compared using a suitable path comparison function.

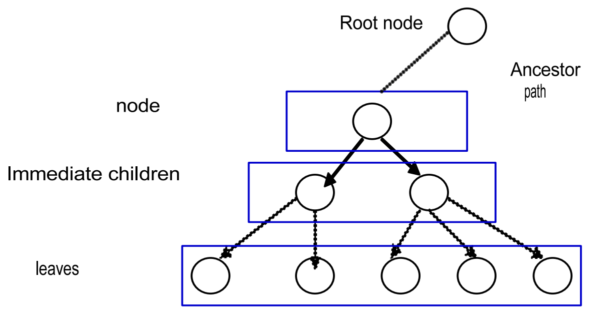


Fig. 8: The context of an object.

These similarity measures are then aggregated using an aggregation function, such as the weighted sum, to produce the complex similarity between two objects.

$$\begin{aligned} CSim\,(OP_1, OP_2) = \; & Combine_C(Child(OP_1, OP_2), \\ & Leaf(OP_1, OP_2), \\ & Ancestor(OP_1, OP_2)) \end{aligned}$$

where $Child(OP_1, OP_2), Leaf(OP_1, OP_2)$ and $Ancestor(OP_1, OP_2)$ are the similarity functions that compute the child, leaf, and ancestor contexts between the two objects, respectively, and $Combine_C$ is the aggregation function to combine the similarity values.

The schema matching community, in general, classifies approaches for XML schema matching (element-level) into two main approaches [Rahm and Bernstein 2001], as shown in Fig. 6.

—*Schema-based Approaches*. These approaches only consider schema information, not instance data. Schema-based approaches exploit the object features and their relationships. These properties are then exploited using either *individual matchers* or *combining matchers* in a *rule-based* fashion to determine semantic similarity among schema elements. An individual matcher exploits only one type of element properties in a single matcher, while a combining matcher can be one of two types: *hybrid matchers*, which exploit multiple types of element properties in a single algorithm and *composite matchers*, which combine the results of independently executed matchers. The element level similarities are then aggregated to compute the total similarity between schemas [Do and Rahm 2002; Melnik et al. 2002; Giunchiglia et al. 2007; Bonifati et al. 2008; Saleem et al. 2008; Algergawy et al. 2009]. They are easy to implement and do not need to be trained before put in use.

—*Instance-based Approaches*. These approaches consider data instances as well as schema-based information. The instance-level data give important insight into the contents and meaning of schema elements. The main advantage of these approaches is that they can empirically learn the similarities among data relying on their instance values. Hence, many *learner-based* schema matching systems have been developed to determine the element-level similarity [Li and Clifton 2000; Doan et al. 2004]. These systems depend largely on pre-match phases such as the training phase using unsupervised learning in SemInt [Li and Clifton 2000], using machine learning in GLUE [Doan et al. 2004], or using neural network-based partial least squares in [Jeong et al. 2008]. However, the main disadvantage of using learner-based approaches is that instance data is generally available in very vast quantity. Hence, the computational cost is very expensive, which affects the schema matching performance.

4.1.2 *Structure-level Measures.* On the other hand, *structure-level measures*, also known as *tree-editing* methods, exploit complex objects without taking into account the detailed object components in the data tree. The tree-editing problem is the generalization of the problem of computing the distance between two strings to labeled trees. As usual, the edit distance relies on three elementary edit operations: insertion, deletion, and re-labeling of a node.

Let DT_1 and DT_2 be two data trees and assume that we are given a cost function defined on each edit operation. An edit script (ES) between the two data trees DT_1 and DT_2 is a sequence of edit operations turning DT_1 into DT_2. The cost of ES is the sum of the costs of the operations in ES. An optimal edit script between DT_1 and DT_2 is an edit script between DT_1 and DT_2 of minimum cost. This cost is called the tree edit distance, denoted by $\delta(DT_1, DT_2)$ [Bille 2005], that is:

$$\delta(DT_1, DT_2) = min\{\gamma(ES) | \textit{ES is an edit operation sequence transforming } DT_1 to\, DT_2\}$$

where γ is a cost function defined on each edit operation.

Several algorithms have been proposed to solve the tree edit distance problem. The first non-exponential algorithm that allows the insertion, deletion, and relabeling of inner and leaf nodes has a complexity of $O(|DT_1| \times |DT_2| \times depth(DT_1)^2 \times depth(DT_2)^2)$ [Tai 1979], where $|DT_1|$ and $|DT_2|$ are data tree cardinalities. Another set of approaches has been proposed allowing the edit operations of nodes anywhere in the tree [Zhang and Shasha 1989; Shasha and Zhang 1995]. These early attempts to solve the tree edit distance problem were not mainly developed in the context of XML data similarity, and thus might yield results that are not completely fitting to XML data. The work proposed in [Chawathe 1999] has been considered as the mainstone for various XML-related structural comparison approaches [Tekli et al. 2009]. Chawathe suggests a recursive algorithm to calculate the tree edit distance between two rooted ordered labeled trees, using a shortest path detection technique on an edit graph. The author restricts the insertion and deletion operations to leaf nodes, and allows the relabeling of nodes anywhere in the data tree. The Chawathe's algorithm has an overall complexity of $O(|DT_1| \times |DT_2|)$.

Even if the Chawathe's algorithm is considered as a starting point for recent XML tree edit distance approaches, the algorithm lacks sub-tree similarity computation. Due to the frequent presence of repeated and optional elements in XML data (especially XML documents), there is a growing need to identify sub-tree structural similarities in the XML data tree comparison context. As a result, several approaches have been proposed to address the sub-tree structural similarity [Nierman and Jagadish 2002; Dalamagasa et al. 2006]. The work provided in [Nierman and Jagadish 2002] extends the Chawathe's approach by adding two new operations: insert tree and delete tree to discover sub-tree similarities. While this algorithm outperforms, in quality, the Chawathe's algorithm, the authors in [Nierman and Jagadish 2002] show that their algorithm is more complex than its predecessor, requiring a pre-computation phase for determining the costs of tree insert and delete operations. The algorithm provided in [Dalamagasa et al. 2006] proposes the usage of tree structural summaries that have minimal processing requirements instead of the

Table I: Tree edit distance algorithms.

Method	Edit operations	Where	Time complexity	Comparison context
[Tai 1979]	insert, delete, relabel	anywhere in the data tree	$O(\|DT_1\| \times \|DT_2\| \times depth(DT_1)^2 \times depth(DT_2)^2)$	similarity between ordered labeled trees
[Zhang and Shasha 1989]	insert, delete, relabel	anywhere in the data tree	$O(\|DT_1\| \times \|DT_2\|) \times min(depth(DT_1), leaf(DT_1)) \times min(depth(DT_2), leaf(DT_2))$	similarity between ordered labeled trees
[Shasha and Zhang 1995]	insert, delete, relabel	anywhere in the data tree	$O(\|DT_1\| \times \|DT_2\|) \times depth(DT_1) \times depth(DT_2)$	similarity between ordered labeled trees
[Chawathe 1999]	insert, delete relabel	leaf nodes anywhere in the data tree	$O(\|DT_1\| \times \|DT_2\|)$	hierarchical structured data such as object class hierarchies, HTML, XML
[Nierman and Jagadish 2002]	insert, delete relabel insert tree delete tree	leaf nodes anywhere in the data tree	$O(2 \times \|DT\| + \|DT_1\| \times \|DT_2\|)$	Structural similarity in XML documents
[Dalamagasa et al. 2006]	insert, delete relabel	leaf nodes anywhere in the data tree	$O(\|DT_1\| \times \|DT_2\|)$	Structural similarity in XML document clustering

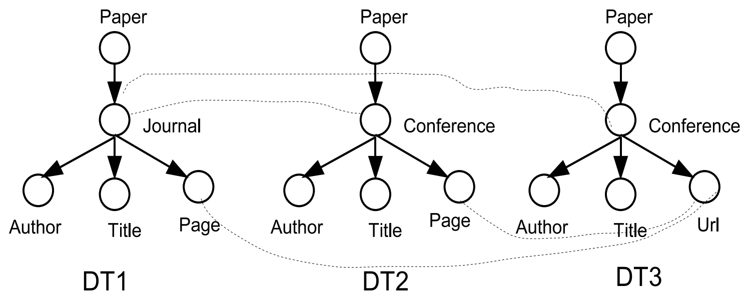


Fig. 9: Tree distance between XML data.

original trees representing the XML documents. Those summaries maintain the structural relationships between the elements of an XML document, reducing repetition and nesting of data tree elements. The authors present a new algorithm to calculate tree edit distances and define a structural distance metric to estimate the structural similarity between the structural summaries of two rooted ordered labeled trees. The algorithm has a complexity of $O(|DT_1| \times |DT_2|)$.

Table I reports the mentioned tree edit distance algorithms representing their time complexity and their comparison contexts. The table shows that the last three methods fit for measuring the similarity between XML data. However, they only consider the structure of XML data and ignore their semantic and content. Fig. 9 illustrates that the edit operations required to transform $DT1$ to $DT2$ equal to that required to transform $DT2$ to $DT3$ because only one relabeling operation is required in both cases to transform the source tree into the target tree. A dotted line from a node in a data tree, such as $DT1$, to a node in another data tree, such as $DT2$, indicates that a relabeling operation is required. Assigning a constant cost for the edit operations results in an equal tree distance between $DT1$ and $DT2$ and $DT2$ and $DT3$. This simple example shows that the tree editing method may not be able to distinguish the hierarchical difference in some situations and a combination of simple and complex similarity measures should be applied.

A number of approaches specific to measure the similarity between XML data have been proposed. These approaches can be classified into three directions [Nayak and Iryadi

2007], as shown in Fig.6.

—*Measuring the structure similarity between XML documents.* To measure the structure similarity of heterogenous XML documents, XML element/attribute values are generally ignored. Different types of algorithms have been proposed to measure the document similarity based on their structure, including tree-edit distance similarity, path similarity, set similarity, etc. [Buttler 2004; Tekli et al. 2009]. The authors in [Buttler 2004; Rafiei et al. 2006] represent the structure of XML documents as a set of paths from the root to a leaf. They also consider any partial path, the path from the root to any node of the data tree. XML documents are then compared according to their corresponding sets of paths. The experimental results in [Buttler 2004; Rafiei et al. 2006] show that the path similarity method provides accurate similarity results compared with the tree-edit distance results. While, the authors in [Candillier et al. 2005] transform data trees into sets of attribute-values, including: the set of parent-child relations, the set of next-sibling relations, and the set of distinct paths. The authors can then apply various existing methods of classification and clustering on such data using the structural description of XML documents alone. However, the set similarity method is not compared to similarly existing methods, such as the tree-edit similarity or the path similarity.

—*Measuring the structure and content similarity between XML documents*. While several methods have been proposed to measure the similarity between XML documents based on their structural features, others consider the content feature in their similarity computation. Research along this direction has been an active area of research and it is fundamental to many applications, such as document clustering, document change detection, integrating XML data sources, XML dissemination and approximate querying of documents [Leung et al. 2005]. A number of approaches have been used to measure the document similarity considering both structure and content features of XML documents, such as leaf node clustering [Liang and Yokota 2005; Viyanon et al. 2008], Baysian networks [Leito et al. 2007], pattern matching [Dorneles et al. 2004], etc. The authors in [Liang and Yokota 2005] propose LAX (Leaf-clustering based Approximate XML join algorithm), in which two XML document trees are clustered into subtrees representing independent items and the similarity between them is determined by calculating the similarity degree based on the leaf nodes of each pair of subtrees. The experimental results in [Liang and Yokota 2005] illustrate that LAX is more efficient in performance and more effective for measuring the approximate similarity between XML documents than the tree edit distance. The authors state that when applying LAX to large XML documents, the hit subtrees selected from the output pair of fragment documents that have large tree similarity degrees might not be the proper subtrees to be integrated. SLAX [Liang and Yokota 2006] is therefore an improved LAX to solve this problem. To address the drawbacks of LAX and SLAX, the authors in [Viyanon et al. 2008] develop a three-phase approach. Each data tree is first clustered into a set of subtrees by considering leaf-node parents as clustering points. The clustered subtrees are then considered independent items that will be matched. The best matched subtrees are integrated in the first XML tree as a resulting XML tree. The experimental results in [Viyanon et al. 2008] shows that the developed approach performs better than LAX and SLAX.

—*Measuring the structural similarity between data and schema*. Evaluating similarity between XML documents and their schemas can be exploited in various domains, such as for classifying XML documents against a set of DTDs/schemas declared in an XML data, XML document retrieval via structural queries, as well as the selective dissemination of XML documents wherein user profiles are being expressed as DTDs/schemas against which the incoming XML data stream is matched [Bertino et al. 2004; 2008; Tekli et al. 2007]. Measuring the structural similarity between XML documents and DTDs/XSDs is not a trivial task. Many factors should be taken into account in the evaluation, like the hierarchical and complex structure of XML documents and schemas, and the tags used to label their semantics. These factors limit developing approaches to measures this kind of structural similarity [Bertino et al. 2008; Tekli et al. 2007]. The similarity approach in [Tekli et al. 2007] relies on transforming both XML documents and DTDs into ordered label trees and then applying the tree editing distance. The approach in [Bertino et al. 2008] also represents XML documents and DTDs as trees, and then computes the structural similarity between a tree and an intensional representation of a set of trees (generated from the DTD). This result could be achieved through an extensional approach, that is, by making the set of document structures described by the DTD explicit, and computing the minimal distance between the document and each document structure. However, this approach is unfeasible because of the high number of document structures that can be generated from a DTD.

4.2 Vector-based similarity approaches

The similarity between XML data represented using vector-based approaches is mainly based on the exploited features. As shown in Fig. 6, the vector-based similarity approaches can exploit either content, structure, or both of compared objects. In general, once the features have been selected, the next step is to define functions to compare them. Given a domain $\mathcal{D}_i$ a comparison criterion for values in $\mathcal{D}_i$ is defined as a function $C_i : \mathcal{D}_i \times \mathcal{D}_i \longrightarrow G_i$, where G_i is a totally ordered set, typically the real numbers. A similarity function $Sim : (\mathcal{D}_1, ..., \mathcal{D}_n) \times (\mathcal{D}_1, ..., \mathcal{D}_n) \longrightarrow L$, where L is a totally ordered set, can now be defined to compare two objects represented as feature vectors and returns a value that corresponds to their similarity [Guerrini et al. 2007]. If feature vectors are real vectors, metric distances induced by norms are typically used. The best-known examples are the $\mathcal{L}_1$ (Manhattan) and $\mathcal{L}_2$ (Euclidean) distances. Other measures have been proposed based on the geometric and probabilistic models. The most popular geometric approach to distance is the vector space model used in Information Retrieval [Salton et al. 1975]. Other popular similarity measures are the cosine ($cos(v_1, v_2) = \frac{v_1 v_2}{|v_1||v_2|}$), Dice ($Dice(v_1, v_2) = \frac{2 v_1 v_2}{|v_1|^2 |v_2|^2}$), and Jaccard ($Jac(v_1, v_2) = \frac{v_1 v_2}{|v_1|^2 |v_2|^2 - v_1 v_2}$) coefficients. In the following, we elaborate on vector-based similarity approaches guided by the taxonomy shown in Fig. 6.

—*Content-based similarity measures.* In this case, each XML document is represented as a feature vector exploiting the content feature of the document elements. The similarity between XML document pairs can be determined using IR measures. For example, to compute the similarity between two document instances of D1 and D2 (see Fig. 5 and Fig. 10), where d_x is the feature vector of D1 and d_y is the feature vector of D2, the cosine measure can be used as given [Yang et al. 2009], $sim(D1, D2) = cos(d_x, d_y) = \frac{d_x . d_y}{|d_x||d_y|} = 0.9258$. Although the similarity value is high, the two document instances, as stated before, should be in separate clusters. This conflict arises from the vector representation of document instances which considers only the content feature and ignores the structural features of the documents. To consider the semantic associated to doc-

```
  <Paper>
    <Conference>
        <Author>John</Author>
       <Author>Marco</Author>
        <Title> XML Data Indexing</Title>
        <Page>43-51</Page>
    </Journal>
</Paper>
```

$$d_y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{array}{c} John \\ Marco \\ XML \\ Data \\ Indexing \\ 43 \\ 51 \end{array}$$

Fig. 10: XML document instance of D2 and its feature vector.

ument contents, the approach in [Tran et al. 2008] makes use of the latent semantic kernel [Cristianini et al. 2002]. Given two XML documents, doc_x and doc_y represented as vectors, d_x and d_y, the semantic similarity of the documents content is measured as:

$$Sim(doc_x, doc_y) = contSim(d_x, d_y) = \frac{d_x^T \mathcal{P}\mathcal{P}^T d_y}{|\mathcal{P}^T d_x||\mathcal{P}^T d_y|}$$

where $\mathcal{P}$ is a matrix used as a mapping function to transform the two documents, d_x and d_y, into concept space to determine the semantic association of document contents.

—*Structure-based similarity measures.* To measure the similarity between XML data based on their structures requires exploiting the structure information of XML data in the feature vectors. The xOR is used to determine the distance between two XML documents represented in the BitCube [Yoon et al. 2001]. Given two XML documents (two bitmap rows), doc_x and doc_y, the similarity is given by

$$Sim(doc_x, doc_y) = 1 - \frac{|xOR(doc_x, doc_y)|}{max(|doc_x||doc_y|)}$$

where xOR is a bit-wise exclusive OR operator, and $|doc_x|$ and $|doc_x|$ are the cardinalities of the documents.

—*Structure& content-based similarity measures.* To measure the similarity between XML documents exploiting both the content and structure features, two strategies can be utilized:

(1) This strategy relies on representing the content and structure of XML documents using feature vectors. The content similarity and the structure similarity are then computed separately, and the two computed values are combined using an aggregation function [Tran et al. 2008].

(2) The second strategy is based using the representation of XML documents as feature matrices that captures both the content and structure information of them. A suitable similarity measure can be proposed. For example, to compute the similarity between two XML documents represented in n-dimensional matrix is presented by introducing the kernel matrix

$$Sim(doc_x, doc_y) = \sum_{i=1}^{n} d_{x_i}^T \bullet M_e \bullet d_{y_i}$$

where d_x and d_y are the normalized document feature vectors of doc_x and doc_y, respectively, M_e is a kernel matrix that capture the similarity between pairs of matrix elements, and $\bullet$ indicates the vector dot product.

Experimental evaluation reported in [Tran et al. 2008; Yang et al. 2009] shows that combining the structure and content features to compute the similarity between XML documents outperforms the other two methods, especially when documents belong to different structural definitions.

5. CLUSTERING/GROUPING

XML data that are similar in structures and semantics are grouped together to form a cluster using a suitable clustering algorithm [Jain et al. 1999; Berkhin 2002; Xu and Wunsch 2005]. Clustering methods are generally divided into two broad categories: *hierarchical* and *non-hierarchical (partitional)* methods. The non-hierarchical methods group a data set into a number of clusters using a pairwise distance matrix that records the similarity between each pair of documents in the data set, while the hierarchical methods produce nested sets of data (hierarchies), in which pairs of elements or clusters are successively linked until every element in the data set becomes connected. Many other clustering techniques have been developed considering the more frequent requirement of tackling large-scale and high-dimensional data sets in many recent applications.

5.1 Hierarchical Methods

Hierarchical clustering builds a hierarchy of clusters, known as a dendrogram. The root node of the dendrogram represents the whole data set and each leaf node represents a data item. This representation gives a well informative description and visualization of the data clustering structure. Strategies for hierarchical clustering generally fall into two types: *agglomerative* and *division*. The divisive clustering algorithm builds the hierarchical solution from top toward the bottom by using a repeated cluster bisectioning approach. Given a set of N XML data to be clustered, and an $N \times N$ similarity matrix, the basic process of the agglomerative hierarchical clustering algorithm is:

(1) treat each XML data of the data set to be clustered as a cluster, so that if you have N items, you now have N clusters, each containing just one XML data.

(2) find the most similar pair of clusters and merge them into a single cluster.

(3) compute similarities between the new cluster and each of the old clusters.

(4) repeat steps 2 and 3 until all items are clustered into a single cluster of size N.

Based on different methods used to compute the similarity between a pair of clusters (Step 3), there are many agglomerative clustering algorithms. Among them are the *single-link* and *complete-link* algorithms [Jain et al. 1999; Manning et al. 2008]. In the single-link method, the similarity between one cluster and another cluster is equal to the maximum similarity from any member of one cluster to any member of the other cluster. In the complete-link method, the similarity between one cluster and another cluster has to be equal to the minimum similarity from any member of one cluster to any member of the other cluster. Even if the single-link algorithm has a better time complexity ($O(N^2)$) than the time complexity ($O(N^2 \log N)$) of the complete-link algorithm, it has been observed that the complete-link algorithm produces more useful hierarchies in many applications than the single-link algorithm. Further, the high cost for most of the hierarchical algorithms limit their application in large-scale data sets. With the requirement for handling large-scale data sets, several new hierarchical algorithms have been proposed to improve the clustering performance, such as BIRCH [Zhang et al. 1996], CURE [Guha et al. 1998], and ROCK [Guha et al. 2000].

5.2 Non-hierarchical Methods

The non-hierarchical clustering methods produce a single partition of the data set instead of a clustering structure, such as the dendrogram produced by a hierarchical technique. There are a number of such techniques, but two of the most prominent are k-means and K-medoid. The non-hierarchical clustering methods also have similar steps as follows:

(1) select an initial partition of the data with a fixed number of clusters and cluster centers;

(2) assign each XML data in the data set to its closest cluster center and compute the new cluster centers. Repeat this step until a predefined number of iterations or no reassignment of XML data to new clusters; and

(3) merge and split clusters based on some heuristic information.

The K-means algorithm is popular because it is easy to implement, and it has a time complexity of $O(NKd)$, where N is the number of data items, K is the number of clusters, and d is the number of iterations taken by the algorithm to converge. Since K and d are usually much less than N and they are fixed in advance, the algorithm can be used to cluster large data sets. Several drawbacks of the K-means algorithm have been identified and well studied, such as the sensitivity to the selection of the initial partition, and the sensitivity to outliers and noise. As a result, many variants have appeared to defeat these drawbacks [Huang 1998; Ordonez and Omiecinski 2004].

5.3 Other Clustering Methods

Despite the widespread use, the performance of both hierarchal and non-hierarchal clustering solutions decreases radically when they are used to cluster a large scale and/or a large number of XML data. This becomes more crucial when dealing with large XML data as an XML data object is composed of many elements and each element of the object individually needs to be compared with elements of another object. Therefore the need for developing another set of clustering algorithms arises. Among them are the incremental clustering algorithms [Can 1993; Charikar et al. 2004] and the constrained agglomerative algorithms [Zhao and Karypis 2002b].

Incremental clustering is based on the assumption that it is possible to consider XML data one at a time and assign them to existing clusters with the following steps. (1) Assign the first XML data to a cluster. (2) Consider the next XML data object, either assign this object to one of the existing clusters or assign it to a new cluster. This assignment is based on the similarity between the object and the existing clusters. (3) Repeat step 2 until all the XML data are assigned to specific clusters. The incremental clustering algorithms are non-iterative, i.e., they do not require to compute the similarity between each pair of objects. So, their time and space requirements are small. As a result, they allow to cluster large sets of XML data efficiently, but they may produce a lower quality solution due to the avoidance of computing the similarity between every pair of documents.

Constrained agglomerative clustering is a trade-off between hierarchical and non- hierarchical algorithms. It exploits features from both techniques. The constrained agglomerative algorithm is based on using a partitional clustering algorithm to constrain the space over which agglomeration decisions are made, so that each XML data is only allowed to merge with other data that are part of the same partitionally discovered cluster [Zhao and Karypis 2002b]. A partitional clustering algorithm is first used to compute a k-way clustering solution. Then, each of these clusters, referred to as constraint clusters, is treated as a separate collection, and an agglomerative algorithm is used to build a tree for each one of them. Finally, the k different trees are combined into a single tree by merging them using an agglomerative algorithm that treats the documents of each subtree as a cluster that has already been formed during agglomeration. The constrained agglomerative algorithms has two main advantages. First, it is able to benefit from the global view of the collection used by partitional algorithms and the local view used by agglomerative algorithms. An additional advantage is that the computational complexity of the algorithm is $O(k(\frac{N}{k})^2 \log(\frac{N}{k}))$, where k is the number of constraint clusters. If k is reasonably large, e.g., k equals $\sqrt{N}$, the original complexity of $O(N^2 \log N)$ for agglomerative algorithms is reduced to $O(N^{\frac{2}{3}} \log N)$.

5.4 Data Clustering Evaluation Criteria

The performance of (XML) data clustering approaches can be evaluated using two major aspects: the *quality* and the *efficiency* of clusters.

5.4.1 *Quality Evaluation.* The quality of clusters can be measured by *external* and *internal* criteria [Guerrini et al. 2007]. The external quality measures, such as the *entropy*, *purity*, and *FScore*, use an (external) manual classification of the documents, whereas the internal quality measures are evaluated by calculating the average inter- and intra-clustering similarity.

FScore is a trade-off between two popular information retrieval metrics, precision P and recall R [Baeza-Yates and Ribeiro-Neto 1999]. Precision considers the rate of correct matches in the generated solution, while recall considers the rate of correct matches in the model solution. Given a cluster C_i, let TP be the number of XML data in C_i which are similar (correctly clustered), FP be the number of documents C_i which are not similar (misclustered), FN be the number of documents which are not in C_i but should be, N be the total number of XML data, and N_i be the number of XML data in C_i. The precision P_i and recall R_i of a cluster C_i are defined as follows:

$$P_i = \frac{TP}{TP + FP}, R_i = \frac{TP}{TP + FN} \tag{1}$$

FScore combining precision and recall with equal weights for the given cluster C_i is defined as

$$FScore_i = 2 \times \frac{P_i \times R_i}{P_i + R_i} \tag{2}$$

Hence, the FScore of the overall clustering approach is defined as the sum of the individual class FScores weighted differently according to the number of XML data in the class

$$FScore = \frac{\sum_{i=1}^{k} N_i \times FScore_i}{N} \tag{3}$$

where k is the number of clusters. A good clustering solution has the FScore value closer to one. A number of methods use two standard measures derived from FScore to reflect the quality of each cluster and inter-cluster: micro F1 and macro F1 measures. Micro-average FScore is calculated by summing up the TP, the FP, and the FN values from all the categories; FScore value is then calculated based on these values. Macro-average FScore, on the other hand, is derived from averaging the FScore values over all the categories. The best clustering solution for an input data set is the one where micro- and macro-average FScore measures are close to 1.

Purity, in general, is a quantitative assessment of homogeneity. Hence, purity measures the degree to which a cluster contains XML data primarily from one class. The purity of cluster C_i is defined as [Zhao and Karypis 2002a]

$$Pur(C_i) = \frac{1}{N_i} \max(N_i^r), \tag{4}$$

which is nothing more than the fraction of the overall cluster size (N_i) that represents the largest class of documents (N_i^r) assigned to that cluster. The overall purity of the clustering solution is obtained as a weighted sum of the individual cluster purities and it is:

$$purity = \sum_{i=1}^{k} \frac{N_i}{N} Pur(C_i). \tag{5}$$

In general, the larger the values of purity, the better the clustering solution is.

Entropy is a widely used measure for clustering solution quality, which measures how the various classes of the XML data are distributed within each cluster. The entropy of a cluster C_i is defined as

$$E(C_i) = -\frac{1}{\log q} \sum_{r=1}^{q} \log \frac{N_i^r}{N_i}, \tag{6}$$

where q is the number of classes in the XML dataset, and N_i^r is the number of XML data of the rth class that is assigned to the cluster ith. The entropy of the entire clustering solution is then defined to be the sum of the individual cluster entropies weighted according to the cluster size. That is,

$$Entropy = \sum_{i=1}^{k} \frac{N_i}{N} E(C_i). \tag{7}$$

A perfect clustering solution will be the one that leads to clusters that contain documents from only a single class, in which case the entropy will be zero. In general, the smaller the entropy values, the better the clustering solution is.

The internal clustering solution quality is evaluated by calculating the average inter- and intra-clustering similarity. The intra-clustering similarity measures the cohesion within a cluster, how similar the XML data within a cluster are. This is computed by measuring the similarity between each pair of items within a cluster, and the intra-clustering similarity of a clustering solution is determined by averaging all computed similarities taking into account the number of XML data within each cluster

$$IntraSim = \frac{\sum_{i=1}^{k} IntraSim(C_i)}{N}. \tag{8}$$

In general, the larger the values of intra-clustering similarity ($IntraSim$), the better the clustering solution is.

The inter-clustering similarity measures the separation among different clusters. It is computed by measuring the similarity between two clusters. A good clustering solution has lower inter-clustering similarity values.

5.4.2 *Efficiency Evaluation.* Efficiency (scalability) is one the most important aspects of today's software applications. As XML data grow rapidly, the systems that support XML data management need to grow. As they grow, it is important to maintain their performance (both quality and efficiency). The scalability of a system, such as XML data clustering, is a measure of its ability to cost-effectively provide *increased throughput*, *reduced response time*, and/or support more users when hardware resources are added. From this definition, efficiency (scalability) is mainly evaluated using two properties: speed (the time it takes for an operation to complete), and space (the memory or non-volatile storage used up by the construct). In order to obtain a good efficiency measure for XML data clustering approaches, we should consider two factors [Algergawy et al. 2008a]: The first factor is the identification of the critical phase of a clustering process: data representation, similarity computation, or grouping/clustering. Intuitively, data representation is a basic phase, while both similarity computing and grouping are two critical phases which affect the XML data clustering efficiency. This leads to the second factor that is the type of methodology used to perform the required task. Is the similarity computation performed pairwise or holistically? Is the clustering algorithm hierarchical, non-hierarchical or incremental?

6. XML DATA CLUSTERING PROTOTYPES: A COMPARISON

In this section we present current approaches for clustering XML data. The description relies on the kind of data to be clustered (documents or schemas) and focus on the adopted data representation (among those discussed in Section 3), the employed similarity measure (among those discussed in Section 4), and the used clustering algorithm and quality evaluation criteria (among those discussed in Section 5).

6.1 Clustering XML Document Prototypes

6.1.1 ***XCLS***. XML documents Clustering with Level Similarity (XCLS) [Nayak 2008] represents an incremental clustering algorithm that groups XML documents according to their structural similarity. Following our generic framework, XCLS has the following phases.

—*Data Representation*. XML documents are first represented as ordered data trees where leaf nodes represent content values. XCLS considers only document elements not document attributes. Each node (simple object) is ranked by a distinct integer according to the pre-order traversal of the data (document) tree. A level structure format is introduced to represent the structure of XML documents for efficient processing, as shown in Fig. 11(b). The level structure groups the distinct XML nodes (distinguished by a number associated to each node as shown in Fig. 11(a)) for each level in the document.

—*Similarity Computation*. XCLS proposes a new algorithm for measuring the structural similarity between XML documents represented as level structures called Level Similarity. The level similarity measures the occurrences of common elements in each corresponding level. Different weights are associated to elements in different levels. Consider two level representations $L_1$and L_2 with N_1 and N_2 levels, respectively and a positive integer $a > 0$. The level similarity, Sim_{L_1,L_2}, between two level representations is:

$$Sim_{L_1,L_2} = \frac{0.5 \times \sum_{i=0}^{M-1} c_i^1 \times a^{M-i-1} + 0.5 \times \sum_{j=0}^{M-1} c_j^2 \times a^{M-j-1}}{\sum_{k=0}^{M-1} t_k \times a^{M-k-1}}$$

where c_i^1 denotes the number of common elements in level i of L_1 and some level of L_2, where c_j^2 denotes the number of common elements in level j of L_2 and some level of L_1 during the matching process, t_k denotes the total number of distinct elements in the level k of L_1, and M is the number of levels in the first level structure. Since the level similarity measure is not transitive, XCLS determines the structural similarity from the first XML document to the second one and viceversa and chooses the highest value between the two.

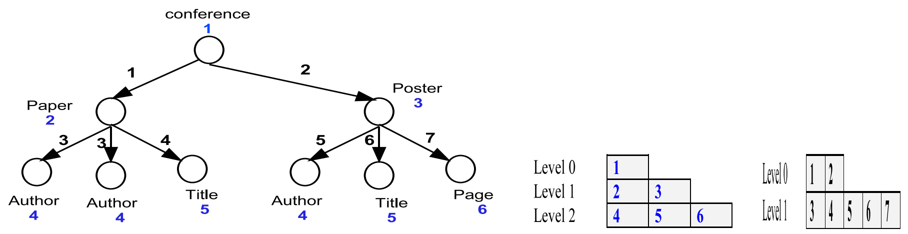
—*Clustering/Grouping*. In this phase, an incremental clustering algorithm is used to group the XML documents within various XML sources considering the level similarity. The clustering algorithm progressively places each coming XML document into a new cluster or into an existing cluster that has the maximum level similarity with it.

—*Evaluation Criteria.* The quality of XCLS is evaluated using the standard criteria, such as the intra- and inter-cluster similarity, purity, entropy, and FScore. The scalability of the XCLS system is also evaluated to validate its space and time complexity. Since XSLC uses an incremental clustering algorithm that avoids the need of pairwise comparison, it has a time complexity of $O(NKdc)$, where c is the number of distinct elements in clusters. Experimental evaluation indicates that XCLS achieves a similar quality result as the pairwise clustering algorithms in much less time. However, XCLS lacks a unified clustering framework that can be applied efficiently both to homogeneous and heterogenous collections of XML documents.

6.1.2 ***XEdge***. XML documents Clustering using Edge Level Summaries (XEdge) [Antonellis et al. 2008] represents a unified clustering algorithm for both homogeneous and heterogenous XML documents. Based on the type of the XML documents, the proposed method modifies its distance metric in order to adapt to the special structure features of homogeneous and heterogeneous documents.

—*Data Representation*. Like XCLS, XEdge represents XML documents as ordered data trees where leaf nodes represent content values. Instead of summarizing the distinct nodes as XCLS, XEdge introduces the LevelEdge representation, which summarizes all distinct parent/child relationships (distinguished by a number associated to each edge as shown in Fig. 11(a)) in each level of the XML document, as shown in Fig. 11(c). The distinct edges are first encoded as integers and those integers are used to construct the LevelEdge representation.

—*Similarity Computation*. In order to compute the similarity among XML documents, XEdge proposes two similarity measures to distinguish between similarity computation among homogeneous XML documents and similarity computation among heterogeneous ones. Assuming that homogeneous XML documents are derived from sub-DTDs of the same DTD, XEdge only searches for common edges in the same levels in both documents. To measure the similarity between two homogeneous XML documents represented as LevelEdge L_1 and L_2, it uses the following similarity function:



(a) XML document (b) Level Structure (c) Level Edge

Fig. 11: Example of Level Structure and LevelEdge.

$$Sim_{L_1,L_2} = \frac{\sum_{i=0}^{m-1} c_i \times a^{m-i-1}}{\sum_{k=0}^{M-1} t_k \times a^{M-k-1}}$$

where c_i is the number of common distinct edges in the level i of L_1 and L_2, while t_k is the total number of distinct edges in the level k of both L_1 and L_2, $m = min(N_1, N_2)$ and $M = max(N_1, N_2)$. For heterogeneous XML documents, common edges in different levels should be identified and a similarity function similar to the one used in XCLS is used.

—*Clustering/Grouping*. In this phase, a partitional clustering algorithm based on a modified version of k-means is used.

—*Evaluation Criteria*. The quality of the XEdge system is only evaluated using the external criteria, such as precision, recall, and FScore. The experimental results show that both XEdge and XCLS fail to properly cluster XML documents derived from different DTDs, although XEdge outperforms XCLS both in case of homogeneous and heterogeneous XML documents. However, scalability of the approach has not been checked.

6.1.3 ***PSim***. Path Similarity (PSim) [Choi et al. 2007] proposes a clustering method that stores data nodes in an XML document into a native XML storage using path similarities between data nodes. The proposed method uses path similarities between data nodes, which reduce the page I/Os required for query processing. According to our generic framework, PSim presents the following phases.

—*Data Representation*. XML documents are modeled as data trees, including both elements and attributes. No more normalization/transformation process is needed.

—*Similarity Computation*. PSim uses the Same Path (SP) clusters, where all nodes (simple objects) with the same absolute path are stored in the same cluster, as the basic units for similarity computation. The PSim method identifies the absolute path for each SP cluster and then compares between SP clusters utilizing their absolute paths as identifiers. It computes the path similarity between every SP cluster pair using the edit distance algorithm. Given two absolute paths $P_1 = /a_1/a_2.../a_n$ and $P_2 = /b_1/b_2/.../b_m$, the edit distance between two paths, $\delta(P_1, P_2)$, can be computed using a dynamic programming technique. Using the edit distance between two absolute paths, a path similarity between two SP clusters sp_1 and sp_2 is computed as follows :

$$path\_similarity(sp_1, sp_2) = 1 - \frac{\delta(P_1, P_2)}{max(|P_1||P_2|)}$$

—*Clustering/Grouping*. The path similarity matrix is represented as a weighted graph, named the path similarity graph, where nodes represent the set of SP clusters and edges connect them with weights. The weight of each edge is the path similarity between two nodes which are connected by that edge. The greedy algorithm is used to partition the path similarity graph.

—*Evaluation Criteria.* To validate the performance of the PSim system, it is evaluated with 1000 randomly generated path queries over an XML document with 200,000 nodes. The total sum of page I/Os required for query processing is used as the performance criterion. Experimental results show that the PSim clustering method has good performance in query processing, however, the proposed method assumes that document updates are infrequent. Further studies on the clustering method are needed when updates are frequent.

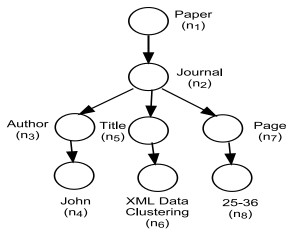
6.1.4 ***XProj***. XProj [Aggarwal et al. 2007] introduces a clustering algorithm for XML documents that uses substructures of the documents in order to gain insight into the important underlying structures.

—*Data Representation*. XML documents are modeled as ordered data trees. XProj does not distinguish between attributes and elements of an XML document, since both are mapped to the label set. For efficient processing, the pre-order depth-first traversal of the tree structure is used, where each node is represented by the path from the root node to itself. The content within the nodes is ignored and only the structural information is being used.

—*Similarity Computation*. XProj makes use of frequent substructures in order to define similarity among documents. This is analogous to the concept of projected clustering in multi-dimensional data, hence the name XProj. In the projected clustering algorithm, instead of using individual XML documents as representatives for partitions, XProj uses a set of substructures of the documents. The similarity of a document to a set of structures in a collection is the fraction of nodes in the document that are covered by any structure in the collection. This similarity can be generalized to similarity between a set of documents and a set of structures by averaging the structural similarity over the different documents. This gives the base for computing the frequent sub-structural self-similarity between a set of XML documents. To reduce the cost of similarity computation, XProj selects the top-K most frequent substructures of size l.

—*Clustering/Grouping.* The primary approach is to use a sub-structural modification of a partition-based approach in which the clusters of XML documents are built around groups of representative substructures. Initially, the sets of XML documents are divided into K partitions with equal size, and the sets of substructure representatives are generated by mining frequent substructures from these partitions. These structure representatives are used to cluster the XML documents using the self-similarity measure.

—*Evaluation Criteria.* The two external criteria, precision and recall, are used to evaluate the quality of the XProj clustering method. XProj is also compared with the Chawathe algorithm [Chawathe 1999] and the structural algorithm [Dalamagasa et al. 2006]. The comparison results indicate that both the XProj and the structural algorithms work well and have a higher precision and recall than Chawathe's tree edit distance-based algorithm when dealing with heterogeneous XML documents. In case of homogeneous XML documents, XProj outperforms the other two systems.



(a) XML data tree of D1 in Fig. 4(a)

path	Answer	node ID
Paper.Journal.Author	John	n_4
Paper.Journal.Title	XML Data Clustering	n_6
Paper.Journal.page	25-36	n_8

(b) tree tuples

transaction	set
tr_1	n_4, n_6, n_8

(c) Transaction set

Fig. 12: Transaction representation of an XML document.

6.1.5 ***SemXClust.*** The Semantic XML Clustering (SemXClust) method [Tagarelli and Greco 2006] investigates how to cluster semantically related XML documents through in-depth analysis of content information extracted from textual elements and structural information derived from tag paths. Both kinds of information are enriched with knowledge provided by a lexical ontology.

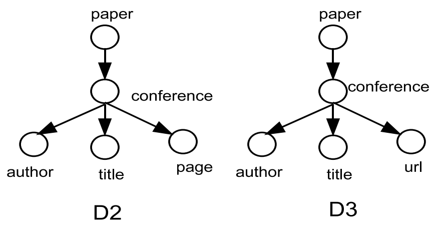
—*Data Representation*. The XML documents are represented as data trees. SemXClust defines an XML tree tuple as the maximal subtree of the document tree satisfying the path/answer condition. This means that tree tuples extracted from the same tree maintain an identical structure while reflecting different ways of associating content with structure. For efficient processing, especially in the huge amount of available structured data, a relevant portion is represented by variable-length sequences of objects with categorical attributes, named *transactional data*, as shown in Fig. 12.

—*Similarity Computation*. Since SemXClust represents and embeds XML features in tree tuple items, therefore the notion of similarity between tree tuple items is a function (weighted sum) of the similarity between their respective structure and content features. The structural similarity between two tree tuple items is determined by comparing their respective tag paths and computing the average similarity between the senses of the respective best matching tags. Furthermore, the content similarity is determined using the text extracted from any leaf node of an XML tree. The extracted text is represented as a bag-of-words model and then subjected to both the lexical and semantic analysis.

—*Clustering/Grouping*. SemXClust applies a partitional algorithm devised for the XML transactional domain, called *TrK-means* [Giannotti et al. 2002]. The clustering algorithm has two phases: (1) working as a traditional centroid-based method to compute $k+1$ clusters, and (2) recursively splitting the $(k+1)$th cluster into a small number of clusters. SemXClust adapts the *TrK-means* clustering algorithm focusing on conceiving suitable notions of proximity among XML transactions. The resulting algorithm is called *XTrK-means*.

—*Evaluation Criteria.* The intra- and inter-clustering similarity criteria are used to validate the quality of the clustering solution. Both criteria are based on the pairwise similarity between transactions. No comparison with other clustering algorithms has been done. Furthermore, SemXClust is not tested against the clustering scalability.

6.1.6 ***S-GRACE.*** S-GRACE [Lian et al. 2004] proposes a hierarchical algorithm for clustering XML documents based on the structural information in the data. A distance



(a) XML data tree of D2 and D3

edges	entry 1	entry 2
paper $\rightarrow$ conference	1	1
conference $\rightarrow$ author	1	1
conference $\rightarrow$ title	1	1
conference $\rightarrow$ page	1	0
conference $\rightarrow$ url	0	1
	DT2	DT3

(b) SG

Fig. 13: S-Graph encoding.

metric is developed on the notion of the structure graph which is a minimal summary of edge containment in the documents. According to our generic framework, S-GRACE has the following phases.

—*Data Representation.* An XML document is represented as a directed graph called the structure graph (or *s-graph*), where the node set includes the set of all elements and attributes of the document. The s-graphs of all documents are analyzed and stored in a data structure called SG. Each entry in the SG contains two information fields, as shown in Fig. 13: (1) a bit string representing the edges of an s-graph and (2) a set containing the IDs of documents whose s-graphs are represented by this bit string. Therefore, the problem of clustering XML documents is transformed into clustering a smaller set of bit strings.

—*Similarity Computation.* S-GRACE computes the distance between all pairs of s-graphs in SG using a distance metric proposed for s-graphs and given by the following equation:

$$\delta(DT1, DT2) = 1 - \frac{|sg(DT1) \cap sg(DT2)|}{max\{|sg(DT1)|, |sg(Dt2)|\}}$$

where $|sg(DT1)|$ is the number of edges in $sg(DT1)$, and $|sg(DT1) \cap sg(DT2)|$ is the set of common edges of $sg(DT1)$ and $sg(DT2)$. Instead of using a pure distance-based algorithm, S-GRACE, as ROCK [Guha et al. 2000], considers the number of the common neighbors. The distance between all pairs of s-graphs in SG is computed and stored in a distance array.

—*Clustering/Grouping.* S-GRACE is a hierarchical clustering algorithm, which applies ROCK [Guha et al. 2000] on the s-graphs. Initially, each entry in SG forms a separate cluster. The clustering technique builds two heaps — a local heap $q[i]$ for each cluster i and a global heap Q containing all clusters. The clustering algorithm iterates until $\alpha \times K$ clusters remains in the global heap, where α is a small number controlling the merging process.

—*Evaluation Criteria.* The quality of S-GRACE is evaluated using non-standard criteria, such as the closeness between clusters and the average similarity over all pairs of clusters, while the scalability is validated using the response time. The time complexity of the S-GRACE method is $O(m^2 \log m)$, where m is the number of distinct s-graphs in SG. Experimental results using both synthetic and real-world data sets show that S-GRACE is effective in identifying clusters, however it does not scale well. The system needs 4500 seconds to clusters a set of XML documents with size of $200KB$.

6.1.7 ***SumXClust.*** The Clustering XML documents exploiting structural summaries (SumXClust) [7]method [Dalamagasa et al. 2006] represents a clustering algorithm for grouping structurally similar XML documents using the tree structural summaries to improve the performance of the edit distance calculation. To extract structural summaries, SumXClust performs both nesting reduction and repetition reduction.

—*Data Representation.* XML documents are represented as ordered data trees. SumXClust proposes the usage of compact trees, called tree structural summaries that have minimal processing requirements instead of the original trees representing XML documents. It performs *nesting reduction* and *repetition reduction* to extract structural summaries for data trees. For nesting reduction, the system traverses the data tree using pre-order traversal to detect nodes which have an ancestor with the same label in order to move up their subtrees. For repetition reduction, it traverses the data tree using pre-order traversal ignoring already existing paths and keeping new ones using a hash table.

—*Similarity Computation.* SumXClust proposes a structural distance S between two XML documents represented as structural summaries. The structural distance is defined as

$$S(DT_1, DT_2) = \frac{\delta(DT_1, DT_2)}{\delta'(DT_1, DT_2)}$$

where $\delta'(DT_1, DT_2)$ is the cost to delete all nodes from DT_1 and insert all nodes from DT_2. To determine edit distances between structural summaries $\delta(DT_1, DT_2)$, the system uses a dynamic programming algorithm which is close to [Chawathe 1999].

—*Clustering/Grouping.* SumXClust implements a hierarchical clustering algorithm using Prim's algorithm for computing the minimum spanning tree. It forms a fully connected graph with n nodes from n structural summaries. The weight of an edge corresponds to the structural distance between the nodes that this edge connects.

—*Evaluation Criteria.* The performance as well as the quality of the clustering results is tested using synthetic and real data. To evaluate the quality of the clustering results, two external criteria, precision and recall, have been used, while the response time is used to evaluate the efficiency of the clustering algorithm. SumXClust is also compared with the Chawathe algorithm [Chawathe 1999]. Experimental results indicate that with or without summaries, the SumXClust algorithm shows excellent clustering quality, and improved performance compared to Chawathe's.

6.1.8 ***VectXClust.*** Clustering XML documents based on the vector representation for documents (VectXClust)[8] [Yang et al. 2005] proposes the transformation of tree-structured data into an approximate numerical multi-dimensional vector, which encodes the original structure information.

—*Data Representation*. XML documents are modeled as ordered data trees, which are then transformed into full binary trees. A full binary tree is a binary tree in which each node

[7]We give the tool this name for easier reference
[8]We give the tool this name for easier reference

has exactly one or two children. The binary trees are then mapped to a numerical multi-dimensional vectors, called binary branch vectors, where the features of the vectors retain the structural information of the original trees. The binary branch vector can be extended to the characteristic vector, which includes all the elements in the q-level binary branch space.

—*Similarity Computation*. Based on the vector representation, VectXClust defines a new distance, called the binary branch distance, of the tree structure as the L_1 distance between the vector images of two trees. The methodology is to embed the defined distance function, that is, the lower bound of the actual tree edit distance into a filter-and-refine framework. This allows filtering out very dissimilar documents and computing the tree edit distance only with a restricted number of documents

—*Clustering/Grouping*. In fact, VectXClust does not conduct any clustering step. However, the measured distances between XML documents can be used later for any clustering process.

6.1.9 ***DFTXClust***. Clustering XML documents based on the time series representation for XML documents (DFTXClust)[9] [Flesca et al. 2005] proposes the linearization of the structure of each XML document, by representing it as a numerical sequence and, then, comparing such sequences through the analysis of their frequencies. The theory of Discrete Fourier Transform is exploited to compare the encoded documents in the domain of frequencies.

—*Data Representation.* DFTXClust is only interested in the structure of XML documents, hence it limits the attention to start tags and end tags. Each tag instance is denoted by a pair composed by its unique identifier and its textual representation. Moreover, it is necessary to consider the order of appearance of tags within a document. To this end, given an XML document doc_x, the authors define its skeleton as the sequence of all tag instances appearing with doc_x. The system relies on the effective encoding of the skeleton of an XML document into a time series summarizing its features. To this purpose, the authors develop two encoding functions: *A tag encoding function*, which assigns a real value to each tag instances, and *a document encoding function*, which associates a sequence of reals with the skeleton of the document. In order to measure the impact of encoding functions in detecting dissimilarities among documents, several encoding schemes have been proposed.

—*Similarity Computation.* Based on the pre-order visit of the XML document starting at initial time t_0, the authors assume that each tag instance occurs after a fixed time interval $\triangle$. The total time spent to visit the document is $n\triangle$, where n is the size of $tags(doc_x)$. During the visit, the system produces an impulse which depends on a particular encoding tag function and the document encoding function. Instead of using Time Warping to compare sequences, Discrete Fourier Transform (DFT) is used to transform document signals into vectors whose components correspond to frequencies in the interval $[-0.5, 0.5]$. Based on this representation, the authors define a metric distance called the Discrete Fourier Transform distance as the approximation of the difference of the magnitudes of the DFT of the two encoded documents. The result of this phase is a similarity matrix representing the degree of structural similarity for each pair of XML documents in the data set.

[9] We give the tool this name for easier reference

6.2 Clustering XML Schema Prototypes

6.2.1 ***XClust.*** XClust [Lee et al. 2002] proposes an approach of clustering the DTDs of XML data sources to effectively integrate them. It is defined by using the phases of the generic framework as follows.

—*Data Representation.* DTDs are analyzed and represented as unordered data trees. To simplify schema matching, a series of transformation rules are used to transform auxiliary OR nodes (choices) to AND nodes (sequences) which can be merged. XClust makes use of simple objects, their features, and relationships between them.

—*Similarity Computation.* To compute the similarity of two DTDs, XClust is based on the computation of the similarity between simple objects in the data trees. To this end, the system proposes a method that relies on the computation of semantic similarity exploiting semantic features of objects, and on the computation of the structure and context similarity exploiting relationships between objects. The output of this phase is the DTD similarity matrix.

—*Clustering/Grouping.* The DTD similarity matrix is exploited by a hierarchical clustering algorithm to group DTDs into clusters. The hierarchical clustering technique can guide and enhance the integration process, since the clustering technique starts with clusters of single DTDs and gradually adds highly similar DTDs to these clusters.

—*Evaluation Criteria.* Since the main objective of XClust is to develop an effective integration framework, it uses criteria to quantify the goodness of the integrated schema. No study concerning the clustering scalability has been done.

6.2.2 ***XMine.*** XMine [Nayak and Iryadi 2007] introduces a clustering algorithm based on measuring the similarity between XML schemas by considering the semantics, as well as the hierarchical structural similarity of elements.

—*Data Representation.* Each schema is represented as an ordered data tree. A simplification analysis of the data (schema) trees is then performed in order to deal with the nesting and repetition problems using a set of transformation rules similar to those in [Lee et al. 2002]. XMine handles both the DTD and XSD schemas, and, like XClust, makes use of simple objects, their features, and relationships between objects.

—*Similarity Computation.* XMine determines the schema similarity matrix through three components. (1) *The element analyzer*, it determines the linguistic similarity by comparing each pair of elements of two schemas primarily based on their names. It considers both the semantic relationship as found in the WordNet thesaurus and the syntactic relationship using the string edit distance function. (2) *The maximally similar paths finder*, it identifies paths and elements that are common and similar between each pair of tree schemas based on the assumption that similar schemas have more common paths. Moreover, it adapts the sequential pattern mining algorithm [Srikant and Agrawal 1996] to infer the similarity between elements and paths. (3) *The schema similarity matrix processor*, the similarity matrix between schemas is computed based on the above measured criteria. This matrix becomes the input to the next phase.

—*Clustering/Grouping.* The constrained hierarchical agglomerative clustering algorithm is used to group similar schemas exploiting the schema similarity matrix. XMine makes use of the wCluto[10] web-enabled data clustering applications to form a hierarchy of

[10]http://cluto.ccgb.umn.edu/cgi-bin/wCluto

schema classes.

—*Evaluation Criteria.* XMine is tested using real-world schemas collected from different domains. To validate the quality of XMine, the standard criteria including FScore, intra- and inter-clustering similarity have been used. The evaluation shows that XMine is effective in clustering a set of heterogenous XML schemas. However, XMine is not subject to any scalability test and a comparison with another clustering method is missing.

6.2.3 ***PCXSS***. The Progressively Clustering XML by Semantic and Structural Similarity (PCXSS) [Nayak and Tran 2007] method introduces a fast clustering algorithm based on a global criterion function CPSim (Common Path Coefficient) that progressively measures the similarity between an XSD and existing clusters, ignoring the need to compute the similarity between two individual XSDs. The CPSim is calculated by considering the structural and semantic similarity between objects.

—*Data Representation.* Each schema is represented as an ordered data tree. A simplification analysis of the data (schema) trees is then performed in order to deal with the composite elements. The XML tree is then decomposed into path information called node paths. A node path is an ordered set of nodes from the root node to a leaf node.

—*Similarity Computation.* PCXSS uses the CPSim to measure the degree of similarity of nodes (simple objects) between node paths. Each node in a node path of a data tree is matched with the node in a node path of another data tree, and then aggregated to form the node path similarity. The node similarity between nodes of two paths is determined by measuring the similarity of its features such as name, data type and cardinality constraints.

—*Clustering/Grouping.* PCXSS is motivated by the incremental clustering algorithms. It first starts with no clusters. When a new data tree comes in, it is assigned to a new cluster. When the next data tree comes in, it matches it with the existing cluster. PCXSS rather progressively measures the similarities between a new XML data tree and existing clusters by using the common path coefficient (CPSim) to cluster XML schemas incrementally.

—*Evaluation Criteria.* PCXSS is evaluated using the standard criteria. Both the quality and the efficiency of PCXSS are validated. Furthermore, the PCXSS method is compared with a pairwise clustering algorithm, namely wCluto [Zhao and Karypis 2002b]. The results show that the increment in time with the increase of the size of the data set is less with PCXSS in comparison to the wCluto pairwise method.

6.2.4 ***SeqXClust***. The Sequence matching approach to cluster XML schema (SeqXClust) [Algergawy et al. 2008b] method introduces an efficient clustering algorithm based on the sequence representation of XML schema using the Prüfer encoding method.

—*Data Representation.* Each schema is represented as an ordered data tree. A simplification analysis of the data (schema) trees is then performed in order to deal with the composite elements. Each data tree is then represented as sequence using a modified Prüfer encoding method. The semantic information of the data tree is captured by Label Prüfer sequences (LPSs), while the structure information is captured by Number Prüfer sequences (NPSs).

—*Similarity Computation.* SeqXClust uses a sequence-based matching approach to measure the similarity between data trees. It first measures the similarity between nodes (simple objects) exploiting their features such as name, data type, and relationships between them, such as the node context. The output of this phase is the schema similarity matrix.

—*Clustering/Grouping.* SeqXClust, like XMine, makes use of the constrained hierarchical agglomerative clustering algorithm to group similar schemas exploiting the schema similarity matrix.

—*Evaluation Criteria.* The SeqXClust method is tested using real-world data sets collected from different domains. The quality of the method is evaluated using the standard evaluation criteria. However, neither a scalability study nor a comparison evaluation has been conducted.

6.3 Comparison of Discussed Clustering Techniques

Table II shows how the discussed XML data clustering prototypes fit the classification criteria introduced in the paper. Furthermore, the table shows which part of the solution is covered by which prototypes, thereby supporting a comparison of the illustrated approaches. It also indicates the supported clustered data types, the internal data representations, the proximity computation methodologies, the exploited clustering algorithms, and the performance evaluation aspects.

The table shows that all systems support data representation, most of them in the form of data trees, only one, S-GRACE, as a directed graph, and two, VectXClust and DFTXClust, as a vector representation. XML schema-based prototypes perform an extra normalization step on the schema trees, while some document-based prototypes extend document trees to handle them efficiently. Moreover, schema-based prototypes make use of schema matching-based approaches to compute the schema similarity matrix. Tree matching-based approaches are used for document-based prototypes. On the other hand, metric distances such as L_1 (Manhattan), cosine measures, and Discrete Fourier Transform distance are used for vector-based approaches. To represent the similarity across schemas/documents, some prototypes use the array (matrix) data structure and another one makes use of fully connected graphs such as SumXClust.

It is worth noting that some prototypes first complete the similarity computation process and then apply the clustering algorithm, while others perform similarity computation during clustering. For example, XClust performs inter-schema similarity computation to get the schema similarity matrix, and then applies the hierarchical clustering algorithm to form a hierarchy of schema classes, while XCLS computes the level similarity to quantify the structural similarity between an XML document and existing clusters and groups the XML document to the cluster with the maximum level similarity.

From the performance point of view, nearly all XML data clustering prototypes evaluate their performance according to the quality aspect considering the internal measures, such as SemXClust, or the external measures, such as SumXClust and XProj, or both, such as XMine and XCLS. The XClust prototype evaluates its quality according to the quality of the integrated schema based on the context of the application domain. To evaluate their scalability, the prototypes use the time response as a measure of the efficiency aspect. Since most of these prototypes make use of pairwise clustering algorithms, their time complexity is at least $O(N^2)$, where N is the number of elements of XML data. This is infeasible for

Table II: Features of XML Data Clustering Approaches

Criteria \ Prototypes		XClust	XMine	PCXSS	SeqXClust	XCLS	XEdge	PSim	XProj	SemXClust	S-GRACE	SumXClust	VectXClust	DFTXClust
Clustered data type		DTD schema	DTD& XSD schema	XSD schema	XSD schema	XML documents	XML documents	XML documents	XML documents	XML documents	XML documents	XML documents	XML documents	XML documents
Data representation		rooted unordered labeled tree	rooted ordered labeled tree	rooted ordered labeled tree	rooted ordered labeled tree	rooted ordered labeled tree	rooted ordered labeled tree	rooted labeled tree	rooted ordered labeled tree	rooted labeled tree	directed graph	rooted ordered labeled tree	binary branch vector	time series
Exploited objects		elements, attributes, paths	elements, attributes, paths	node paths	elements, attributes, node context	level information elements	edge information	paths	substructures	subtrees (tree tuple)	whole XML tree (s-graph)	whole XML tree (structural summaries)	whole XML tree	whole XML tree
Similarity approach	*Schema matching*	name, cardinality, path context	linguistic, hierarchical structure, path similarity	name, data type, constraint	name, data type, node context	–	–	–	–	–	–	–	–	–
	Tree matching	–	–	–	–	level similarity	edge similarity	path similarity	frequent substructure	content & structural similarity	distance metric	structural distance	–	–
	Vector based	–	–	–	–	–	–	–	–	–	–	–	binary branch distance	DFT distance
Similarity representation		DTD similarity matrix	schema similarity matrix	–	schema similarity matrix	–	edge similarity matrix	path similarity matrix	–	–	distance array	fully connected graph	–	–
Clustering algorithm		hierarchical	hierarchical agglomerative	incremental	constrained hierarchical agglomerative	incremental	K-means	graph partitioning greedy algorithm	partition-base approach	partitional XTrK-means algorithm	hierarchical ROCK approach	single link hierarchical	–	–
Cluster representation		hierarchy of schema classes (dendrogram)	hierarchy of schema classes (dendrogram)	K-clusters	hierarchy of schema classes (dendrogram)	K-clusters	K-clusters	K-clusters	K-clusters	K-clusters	K-clusters	K-clusters	–	–
Quality criteria		quality of integrated schema	inter & intra-cluster FScore	inter & intra-cluster FScore	inter & intra-cluster FScore	external& internal measures purity, entropy, FScore inter & intra cluster	FScore	–	Precision recall	inter & intra-cluster	closeness standard deviation outlier ratio	precision recall	–	–
Efficiency evaluation		–	–	response time	–	time& space complexity	–	page I/O	response time	–	response time	response time	response time	–
Application area		XML data integration	general XML management	general XML management	general XML management	general XML management	general XML management	XML storage for query processing	XML data mining	XML data mining	XML query processing	hierarchical structural management	Query processing	Storage & retrieval of large documents

large amounts of data. However, no evaluation work has been done considering a trading-off between the two performance aspects.

Finally, clustering XML data is useful in numerous applications. Table II shows that some prototypes have been developed and implemented for specific XML applications, such as XClust for DTDs integration and S-GRACE for XML query processing, while other prototypes have been developed for generic applications, such as XMine and XCLS for XML management in general.

7. SUMMARY AND FUTURE DIRECTIONS

Clustering XML data is a basic problem in many data application domains such as Bioinformatics, XML information retrieval, XML data integration, Web mining, and XML query processing. In this paper, we conducted a survey about XML data clustering methodologies and implementations. In particular, we proposed a classification scheme for these methodologies based on three major criteria: the type of clustered data (clustering either XML documents or XML schemas), the used similarity measure approach (either the tree similarity-based approach or the vector-based approach), and the used clustering algorithm. As well as, in order to base the comparison between their implementations, we proposed a generic clustering framework.

In this paper, we devised the main current approaches for clustering XML data relying on similarity measures. The discussion also pointed out new research trends that are currently under investigation. In the remainder we sum up interesting research issues that still deserve attention from the research community classified according to four main directions: Semantics, performance, trade-off between clustering quality and clustering scalability, and application context directions.

—**Semantics**. More semantic information should be integrated in clustering techniques in order to improve the quality of the generated clusters. Examples of semantic information are: Schema information that can be associated with documents or extracted from a set of documents; Ontologies and Thesauri, that can be exploited to identify similar concepts; Domain knowledge, that can used within a specific context to identify as similar concepts that in general are retained different (for example, usually papers in proceedings are considered different from journal papers, however a paper in the VLDB proceedings has the same relevance as a paper in an important journal). The semantics of the data to be clustered can also be exploited for the selection of the measure to be employed. Indeed, clustering approaches mainly rely on well-known fundamental measures (e.g., tree edit distance, cosine and Manhattan distances) that are general purpose and cannot be easily adapted to the characteristics of data. A top-down or bottom-up strategy can be followed for the specification of these kinds of functions. Following the top-down strategy, a user interface can be developed offering the user a set of building-blocks similarity measures (including the fundamental ones) that can be applied on different parts of XML data, and users can compose these measures in order to obtain a new one that is specific for their needs. The user interface shall offer the possibility to compare the effectiveness of different measures and choose the one that best suits their needs. In this strategy, the user is in charge of specifying the measure to be used for clustering. In the context of approximate retrieval of XML documents, the multi-similarity system arHex [Sanz et al. 2006] has been proposed. Users can specify and compose their own similarity measures and compare the retrieved results depending on the applied measures. Analogous approaches should be developed for clustering XML data, also for incrementally checking the correctness of the specified measure, and for changing the clustering algorithm.

Following the bottom-up strategy, the characteristics of the data that need to be clustered are considered to point out the measure (or the features of the measure) that should return the best result ([Muller et al. 2005b] and [Muller et al. 2005a] developed a similar approach in other contexts). Exploiting functions that extract different kinds of information from the data (e.g., element tags, relevant words of elements, attributes, links), the features of the similarity measures to be applied on the data can be pointed out. For example, if attributes occur seldomly in a collection of documents, a similarity measure that relies on attributes is not really useful; by contrast, if the internal structure of the documents is mostly identical, a content-based measure is expected. In this strategy, thus, the data characteristics suggest the measure to apply for clustering. [Sanz et al. 2008] propose the use of entropy-based measures to evaluate the level of heterogeneity of different aspects of documents. This information can be used to choose the more adequate indexing structure, and to find out whether it is better to use an exact or approximate retrieval approach. The integration of the two strategies and its employment in XML data clustering is thus a very interesting research direction.

Finally, most approaches are based on equality comparisons concerning the evaluation of the similarity at single elements/nodes. A more semantic approach, relying on ontologies and Thesauri for allowing multilingual data handling and concept-based clustering, would certainly be useful.

—**Performance**. Another research issue to be faced is the performance of the similarity evaluation process. When the size and the number of the data to be clustered increase, the current approaches do not scale well. A possible solution that needs to be investigated is to employ two measures in the evaluation of similarity similarity (in the same spirit of [Yang et al. 2005]). The first one should be able to quickly identify the documents that are dissimilar in order to identify data that presumably belong to the same cluster. Then, using a finer (and time-consuming function) the previous results should be revised on a reduced number of documents/schemas. Key issue is the choice of the two measures that should be compatible, that is, the two measures should return comparable results even if with different performances (currently in [Yang et al. 2005] there is a factor 5 between the two measures that is too high). Orthogonally to this approach, a ranking on the performances of the approaches proposed so far should be provided eventually bound to their theoretical complexity.

—**Trade-off Between Clustering Quality and Clustering Efficiency**. Many real-world problems such as the XML data clustering problem, involve multiple measures of performance, which should be optimized simultaneously. Optimal performance according to one objective, if such an optimum exists, often implies unacceptably low performance in one or more of the other objective dimensions, creating the need for a compromise to be reached. In the XML data clustering problem, the performance of a clustering system involves multiple aspects, among them clustering quality and clustering efficiency, as mentioned before. Optimizing one aspect, for example, clustering quality will affect the other aspects such as efficiency, etc. Hence, we need a compromise between them, and we could consider the trade-off between quality and efficiency of the clustering result as a multi-objective optimization problem [Algergawy et al. 2008a]. In practice, multi-objective problems have to be re-formulated as a single objective problem. This representation enables us to obtain a combined measure as a compromise between them. A critical scenario in which a trade-off between clustering quality and clustering efficiency should be taken into account can be stated as follows: Let we have a set of XML data to be clustered, and we have two clustering systems A and B. System A is more effective than system B, while system B is more efficient than system A. The question arising here is which system will be used to solve the given problem?

—**Application Context**. Another interesting research direction is the employment of the data "context" in the evaluation of similarity. The context can be any information about the data that is not contained in the file. For example, the authors of the data, the creation date, the geospatial position of the file. This information can be integrated in the similarity measure in order to obtain more precise results.

Acknowledgements: *Alsayed Algergawy* has been supported by the Egyptian Ministry of Higher Education and Tanta University, Egypt. We wish to thank *Stefanie Quade* for improving the quality of the paper. We are grateful to the anonymous reviewers for significantly improving the article by the detailed and deeply knowledgeable comments.

REFERENCES

AGGARWAL, C. C., TA, N., WANG, J., FENG, J., AND ZAKI, M. J. 2007. Xproj: a framework for projected structural clustering of XML documents. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. 46–55.

ALGERGAWY, A., SCHALLEHN, E., AND SAAKE, G. 2008a. Combining effectiveness and efficiency for schema matching evaluation. In *First International Workshop Model-Based Software and Data Integration (MBSDI 2008)*. Berlin, Germany, 19–30.

ALGERGAWY, A., SCHALLEHN, E., AND SAAKE, G. 2008b. A schema matching-based approach to XML schema clustering. In *10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008)*. Linz, Austria, 131–136.

ALGERGAWY, A., SCHALLEHN, E., AND SAAKE, G. 2009. Improving XML schema matching using Prufer sequences. *Data & Knowledge Engineering 68,* 8, 728–747.

ANDREOPOULOS, B., AN, A., WANG, X., AND SCHROEDER, M. 2009. A roadmap of clustering algorithms: finding a match for a biomedical application. *Briefings in Bioinformatics 10,* 3, 297–314.

ANTONELLIS, P., MAKRIS, C., AND TSIRAKIS, N. 2008. XEdge: Clustering homogeneous and heterogeneous XML documents using edge summaries. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*. Brazil, 1081–1088.

BAEZA-YATES, R. AND RIBEIRO-NETO, B. 1999. *Modern Information Retrieval*. ACM Press/Addison-Wesley.

BATINI, C., LENZERINI, M., AND NAVATHE, S. 1986. A comparative analysis of methodologies for database schema integration. *ACM Comput Surv 18,* 4, 323–364.

BERKHIN, P. 2002. Survey of clustering data mining techniques. In *Accrue Software, Inc.* 1–56.

BERTINO, E. AND FERRARI, E. 2001. XML and data integration. *IEEE Internet Computing 5,* 6, 75–76.

BERTINO, E., GUERRINI, G., AND MESITI, M. 2004. A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications. *Information Systems 29,* 1, 23–46.

BERTINO, E., GUERRINI, G., AND MESITI, M. 2008. Measuring the structural similarity among XML documents and DTDs. *Intelligent Information Systems 30,* 1, 55–92.

BILLE, P. 2005. A survey on tree edit distance and related problems. *Theoretical Computer Science 337,* 1-3, 217–239.

BOLSHAKOVA, N. AND CUNNINGHAM, P. 2005. cluML: a markup language for clustering and cluster validity assessment of microarray data. Tech. Rep. TCD-CS-2005-23, The University of Dublin.

BONIFATI, A., MECCA, G., PAPPALARDO, A., RAUNICH, S., AND SUMMA, G. 2008. Schema mapping verification: the spicy way. In *Proceedings of the 11th international conference on Extending database technology (EDBT'08)*. 85–96.

Boukottaya, A. and Vanoirbeek, C. 2005. Schema matching for transforming structured documents. In *Proceedings of the 2005 ACM Symposium on Document Engineering (DocEng'05)*. 101–110.

Bourret, R. 2009. XML database products. http://www.rpbourret.com/xml/XMLDatabaseProds.htm.

Buttler, D. 2004. A short survey of document structure similarity algorithms. In *International Conference on Internet Computing*. Nevada, USA, 3–9.

Can, F. 1993. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems (TOIS) 11,* 2, 143–164.

Candillier, L., Tellier, I., and Torre, F. 2005. Transforming XML trees for efficient classification and clustering. In *INEX 2005*. Dagstuhl, Germany, 469–480.

Cerami, E. 2005. *XML for Bioinformatics*. Springer New York.

Charikar, M., Chekuri, C., Feder, T., and Motwani, R. 2004. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing 33,* 6, 1417–1440.

Chawathe, S. S. 1999. Comparing hierarchical data in external memory. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*. Edinburgh, Scotland, 90–101.

Choi, I., Moon, B., and Kim, H.-J. 2007. A clustering method based on path similarities of XML data. *Data & Knowledge Engineering 60*, 361–376.

Cohen, W. W., Ravikumar, P., and Fienberg, S. E. 2003. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (II-Web'03)*. 73–78.

Cristianini, N., Shawe-Taylor, J., and Lodhi, H. 2002. Latent semantic kernels. *Journal of Intelligent Information Systems (JIIS) 18,* 2-3, 127–152.

Dalamagasa, T., Cheng, T., Winkel, K.-J., and Sellis, T. 2006. A methodology for clustering XML documents by structure. *Information Systems 31,* 187–228.

Do, H. H. and Rahm, E. 2002. COMA- A system for flexible combination of schema matching approaches. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*. 610–621.

Doan, A., Madhavan, J., Domingos, P., and Halevy, A. 2004. *Handbook on Ontologies*. Springer, Chapter Ontology matching: A machine learning approach, 385–404.

Dorneles, C. F., Heuser, C. A., Lima, A. E. N., da Silva, A. S., and de Moura, E. S. 2004. Measuring similarity between collection of values. In *Sixth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2004)*. Washington, DC, USA, 56–63.

Eisen, M. B., Spellman, P. T., Brown, P. O., and Botstein, D. 1998. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA 95,* 1486314868.

Flesca, S., Manco, G., Masciari, E., Pontieri, L., and Pugliese, A. 2005. Fast detection of XML structural similarity. *IEEE Trans. KDE 17,* 2, 160–175.

Florescu, D. and Kossmann, D. 1999. Storing and querying XML data using an RDMBS. *IEEE Data Eng. Bull. 22,* 3, 27–34.

Giannotti, F., Gozzi, C., and Manco, G. 2002. Clustering transactional data. In *PKDD*. 175–187.

Giunchiglia, F., Yatskevich, M., and Shvaiko, P. 2007. Semantic matching: Algorithms and implementation. *Journal on Data Semantics 9*, 1–38.

Gou, G. and Chirkova, R. 2007. Efficiently querying large XML data repositories: A survey. *IEEE Trans. on Knowledge and Data Engineering 19,* 10, 1381–1403.

Guerrini, G., Mesiti, M., and Sanz, I. 2007. *An Overview of Similarity Measures for Clustering XML Documents. Web Data Management Practices: Emerging Techniques and Technologies*. IDEA GROUP.

Guha, S., Rastogi, R., and Shim, K. 1998. CURE: An efficient clustering algorithm for large databases. In *Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD'98)*. Washington, USA, 73–84.

Guha, S., Rastogi, R., and Shim, K. 2000. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems 25,* 5, 345–366.

Hanisch, D., Zimmer, R., and Lengauer, T. 2002. ProML-the protein markup language for specification of protein sequences, structures and families. *Silico Biol. 2,* 3, 313–324.

Huang, Z. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery 2,* 3, 283–304.

HUCKA, M. AND ET AL. 2003. The systems biology markup language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics 19,* 4, 524–531.

JAIN, A., MURTY, M., AND FLYNN, P. 1999. Data clustering: A review. *ACM Computing Surveys 31,* 3, 264–323.

JEONG, B., LEE, D., CHO, H., AND LEE, J. 2008. A novel method for measuring semantic similarity for XML schema matching. *Expert Systems with Applications 34*, 1651–1658.

JEONG, J., SHIN, D., CHO, C., , AND SHIN, D. 2006. Structural similarity mining in semi-structured microarray data for efficient storage construction. In *OTM Workshops 2006, LNCS 4277*. Montpellier, France, 720729.

KADE, A. M. AND HEUSER, C. A. 2008. Matching XML documents in highly dynamic applications. In *Proceedings of the 2008 ACM Symposium on Document Engineering (DocEng'08)*. 191–198.

LE, D. X., RAHAYU, J. W., AND PARDEDE, E. 2006. Dynamic approach for integrating web data warehouses. In *Computational Science and Its Applications (ICCSA'06)*. 207–216.

LEE, D. AND CHU, W. W. 2000. Comparative analysis of six XML schema languages. *SIGMOD Record 9,* 3, 76–87.

LEE, M. L., YANG, L. H., HSU, W., AND YANG, X. 2002. Xclust: Clustering XML schemas for effective integration. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management (CIKM'02)*. 292–299.

LEE, Y., SAYYADIAN, M., DOAN, A., AND ROSENTHAL, A. 2007. etuner: tuning schema matching software using synthetic scenarios. *VLDB J. 16,* 1, 97–132.

LEITO, L., CALADO, P., AND WEIS, M. 2007. Structure-based inference of XML similarity for fuzzy duplicate detection. In *Sixteenth ACM Conference on Information and Knowledge Management, (CIKM 2007)*. Lisbon, Portugal, 293–302.

LEUNG, H.-P., CHUNG, F.-L., AND CHAN, S. C.-F. 2005. On the use of hierarchical information in sequential mining-based XML document similarity computation. *Knowledge and Information Systems 7,* 4, 476–498.

LI, W. AND CLIFTON, C. 2000. SemInt: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Konwledge Engineering 33*, 49–84.

LIAN, W., LOK CHEUNG, D. W., MAMOULIS, N., AND YIU, S.-M. 2004. An efficient and scalable algorithm for clustering XML documents by structure. *IEEE Trans. on KDE 16,* 1, 82–96.

LIANG, W. AND YOKOTA, H. 2005. LAX: An efficient approximate XML join based on clustered leaf nodes for XML data integration. In *22nd British National Conference on Databases*. Sunderland, UK, 82–97.

LIANG, W. AND YOKOTA, H. 2006. SLAX: An improved leaf-clustering based approximate XML join algorithm for integrating XML data at subtree classes. *IPSJ Digital Courier 2*, 382–392.

MADHAVAN, J., BERNSTEIN, P. A., AND RAHM, E. 2001. Generic schema matching with cupid. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*. Roma, Italy, 49–58.

MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

MELNIK, S., GARCIA-MOLINA, H., AND RAHM, E. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*.

MELTON, J. AND BUXTON, S. 2006. *Querying XML: XQuery, XPath, and SQL/XML in Context*. Morgan Kaufmann/Elsevier.

MULLER, T., SELINSKI, S., AND ICKSTADT, K. 2005a. Cluster analysis: A comparison of different similarity measures for SNP data. In *2nd IMS-ISBA Joint Meeting.*

MULLER, T., SELINSKI, S., AND ICKSTADT, K. 2005b. How similar is it? towards personalized similarity measures in ontologies. In *7. International Tagung Wirschaftinformatik.*

NAYAK, R. 2008. Fast and effective clustering of XML data using structural information. *Knowledge and Information Systems 14,* 2, 197–215.

NAYAK, R. AND IRYADI, W. 2007. XML schema clustering with semantic and hierarchical similarity measures. *Knowledge-based Systems 20*, 336–349.

NAYAK, R. AND TRAN, T. 2007. A progressive clustering algorithm to group the XML data by structural and semantic similarity. *International Journal of Pattern Recognition and Artificial Intelligence 21,* 4, 723–743.

NIERMAN, A. AND JAGADISH, H. V. 2002. Evaluating structural similarity in XML documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB'02)*. 61–66.

ORDONEZ, C. AND OMIECINSKI, E. 2004. Efficient disk-based k-means clustering for relational databases. *IEEE Trans. Knowl. Data Eng 16,* 8, 909–921.

PAL, S., TALWAR, V., AND MITRA, P. 2002. Web mining in soft computing framework: Relevance, state of the art and future directions. *IEEE Trans. on Neural Networks 13,* 5, 1163–1177.

RAFIEI, D., MOISE, D. L., AND SUN, D. 2006. Finding syntactic similarities between XML documents. In *the 17th International Conference on Database and Expert Systems Applications (DEXA)*. Krakow, Poland, 512–516.

RAHM, E. AND BERNSTEIN, P. A. 2001. A survey of approaches to automatic schema matching. *VLDB Journal 10,* 4, 334–350.

SALEEM, K., BELLAHSENE, Z., AND HUNT, E. 2008. PORSCHE: Performance oriented schema mediation. *Information Systems 33,* 7-8, 637–657.

SALTON, G., WONG, A., AND YANG, C. 1975. A vector space model for automatic indexing. *Communications of the ACM 18,* 11, 613–620.

SANZ, I., MESITI, M., BERLANGA, R., AND GUERRINI, G. 2008. An entropy-based characterization of heterogeneity of XML collections. In *3rd International Workshop on XML Data Management Tools and Techniques, (XANTEC 2008)*. Torino, Italy, 238–242.

SANZ, I., MESITI, M., GUERRINI, G., AND BERLANGA, R. 2006. Arhex: An approximate retrieval system for highly heterogeneous XML document collections. In *10th International Conference on Extending Database Technology (EDBT'06)*. Munich, Germany, 1186–1189.

SASSON, O., LINIAL, N., AND LINIAL, M. 2002. The metric space of proteins-comparative study of clustering algorithms. *Bioinformatics 18,* 1, 14–21.

SHANMUGASUNDARAM, J., TUFTE, K., HE, G., ZHANG, C., DEWITT, D., AND NAUGHTON, J. 1999. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*. 302–314.

SHASHA, D. AND ZHANG, K. 1995. Approximate tree pattern matching. In *Pattern Matching in Strings, Trees, and Arrays*. Oxford University Press.

SINGHAL, A. 2001. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull. 24,* 4, 35–43.

SOMERVUO, P. AND KOHONEN, T. 2000. Clustering and visualization of large protein sequence databases by means of an extension of the self-organizing map. In *Discovery Science, Third International Conference*. Kyoto, Japan, 76–85.

SRIKANT, R. AND AGRAWAL, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *the 5th International Conference on Extending Database Technology (EDBT'96)*. France, 3–17.

TAGARELLI, A. AND GRECO, S. 2006. Toward semantic XML clustering. In *Proceedings of the Sixth SIAM International Conference on Data Mining (SDM'06)*. 188–199.

TAI, K.-C. 1979. The tree-to-tree correction problem. *Journal of the ACM 26,* 3, 422–433.

TAMAYO, P., SLONIM, D., MESIROV, J., ZHU, Q., KITAREEWAN, S., DMITROVSKY, E., LANDER, E. S., AND GOLUB, T. R. 1999. Interpreting patterns of gene expression with self-organizing maps: Methods and application to hematopoietic differentiation. *Proc. Natl. Acad. Sci. USA 96,* 6, 2907–2912.

TEKLI, J., CHBEIR, R., AND YETONGNON, K. 2009. An overview on XML similarity: background, current trends and future directions. *Computer Science Review 3,* 3.

TEKLI, J., CHBEIR, R., AND YTONGNON, K. 2007. Structural similarity evaluation between XML documents and DTDs. In *Proceedings of 8th International Conference on Web Information Systems Engineering (WISE'07)*. 196–211.

TRAN, T., NAYAK, R., AND BRUZA:, P. 2008. Combining structure and content similarities for XML document clustering. In *the Seventh Australasian Data Mining Conference (AusDM 2008)*. Australia, 219–226.

VAKALI, A., POKORN, J., AND DALAMAGAS, T. 2004. An overview of web data clustering practices. In *EDBT Workshops*. 597–606.

VIYANON, W., MADRIA, S. K., AND BHOWMICK, S. S. 2008. XML data integration based on content and structure similarity using keys. In *OTM Conferences (1)*. Mexico, 484–493.

VUTUKURU, V., PASUPULETI, K., KHARE, A., AND GARG, A. 2002. Conceptemy : An issue in XML information retrieval. In *WWW2002 Posters*.

WANG, G., SUN, B., LV, J., AND YU, G. 2004. RPE query processing and optimization techniques for XML databases. *J. Comput. Sci. Technol. 19,* 2, 224–237.

WANG, J., ZHANG, Y., ZHOU, L., KARYPIS, G., AND AGGARWAL, C. C. 2009. CONTOUR: an efficient algorithm for discovering discriminating subsequences. *Data Min Knowl Disc 18,* 129.

WILDE, E. AND GLUSHKO, R. J. 2008. XML fever. *Communications of the ACM 51,* 7, 40–46.

XU, R. AND WUNSCH, D. 2005. Survey of clustering algorithms. *IEEE Transactions on Neural Networks 16,* 3, 645–678.

YANG, J., CHEUNG, W. K., AND CHEN, X. 2009. Learning element similarity matrix for semi-structured document analysis. *Knowledge and Information Systems 19,* 1, 53–78.

YANG, J., WANG, H., WANG, W., AND PHILIP, S. 2005. An improved biclustering method for analyzing gene expression profiles. *Artificial Intelligence Tools 14,* 5, 771–790.

YANG, R., KALNIS, P., AND TUNG, A. K. H. 2005. Similarity evaluation on tree-structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD Conference'05)*. 754–765.

YOON, J. P., RAGHAVAN, V., CHAKILAM, V., AND KERSCHBERG, L. 2001. Bitcube: A three-dimensional bitmap indexing for XML documents. *Journal of Intelligent Information Systems (JIIS) 17,* 2-3, 241–254.

ZHANG, K. AND SHASHA, D. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing 18,* 6, 1245–1262.

ZHANG, T., RAMAKRISHNAN, R., AND LIVNY, M. 1996. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD Conference'96)*. Quebec, Canada, 103–114.

ZHAO, Y. AND KARYPIS, G. 2002a. Criterion functions for document clustering: Experiments and analysis. Tech. Rep. #01-40, Department of Computer Science, University of Minnesota.

ZHAO, Y. AND KARYPIS, G. 2002b. Evaluation of hierarchical clustering algorithms for document datasets. In *the ACM International Conference on Information and Knowledge Management (CIKM'02)*. Virginia, USA, 515–524.