XMiner: Mining XML Mediated Schemas

Hong-Quang Nguyen, Wenny Rahayu, Kinh Nguyen
Dept of Computer Science and Computer Engineering
La Trobe University, VIC 3086, Australia
{h20nguyen, wenny, kinh.nguyen}@cs.latrobe.edu.au

David Taniar
Clayton School of Information Technology
Monash University, Clayton, VIC 3800, Australia
david.taniar@infotech.monash.edu.au

Abstract

This paper presents a novel schema mediation approach, called XMiner, for mining a mediated schema from a set of XML schemas. XMiner addresses three main mediation problems resulting from the heterogeneous source schemas: nesting discrepancy, backward paths and schema discrepancy. It discovers frequent concepts and paths, which are used to construct the mediated schema. XMiner exploits structural context, forward/backward paths, and label semantics to preserve the hierarchical structure as the best as possible while avoiding information loss. Experimental evaluations on real and synthetic datasets show that XMiner offers acceptable performance for large-scale scenarios.

1 Introduction

A wealth of information available on the Internet and among various business systems has spurred much research effort in data integration and information retrieval [1, 3, 5, 7, 8]. In order to effectively and efficiently exploit useful information from such data sources, building mediated schemas, especially for XML schemas, for large-scale applications has become an important task.

Schema matching aims to reconcile heterogeneous data sources by finding semantic corresponding elements between two schemas. Various semi-automatic schema matching approaches have been proposed such as Cupid [8], Clio [11], COMA++ [3], and µBE [1]. These matchers generally produce semantic mappings between elements of two schemas; hence they are able to produce merged schemas rather than mediated schema. The merged schema is unable to capture the common structure from multiple schema sources [3, 10]. To solve those outstanding problems, we propose a schema mediation approach, called XMiner. Our key contributions are: (1) proposing XMiner to generate a mediated schema based on frequent subtrees; (2) emphasizing the importance of contextual semantics and hierarchical structure in schema mediation, including nesting discrepancy and backward paths. Execution time and scale-up performance are evaluated on both real and synthetic datasets.

2 Related Work and Motivation

Several surveys [12, 14, 4] on schema matching have been extensively conducted. Different schema matchers have been proposed in the literature, such as LSD/COMAP/GLUE [5]; COMA/COMA++ [3]; Cupid [8]; SemInt [7], Clio [11], µBE [1], S-Match [6], and RONDO [10]. They employ different techniques; e.g., machine learning, rule-based algorithms and partial schema mappings. Schema matching aims to find semantic correspondences of elements between two schemas. However, they do not fully exploit tree-like structure of XML schemas and often ignore the semantics conveyed in hierarchical nesting between elements [7, 3]. Fig. 1 presents a motivating example with a simplified schema repository consisting of three schema trees $D = \{T_1, T_2, T_3\}$ related to Movie domain.

![Figure 1. A Database of XML Schemas](image)

COMA++ [3] is one of the most recent composite schema matchers. It uses bottom-up strategy for finding correspondences between elements. The problems with COMA++ is to find correspondences of the containment between two nodes which carries the same meaning but is designed using either parent-child path $X/Y$ or ancestor-descendant path $X//Y$ (Fig. 2a). We refer this as nesting discrepancy. For example, COMA++ cannot match Movie/Year in $T_1$ with Movie/GenInfo/Year in $T_3$ in Fig. 1 due to an extra node GenInfo between Movie and Year. Further, COMA++ incorrectly outputs a match between Actor and Director because all of Actor’s children in $T_1$ are exactly matched with those of Director in $T_3$. To the best of our knowledge, PORSCHE [13] is a recent frequent subtree mining approach for building the mediated schema. However, PORSCHE is unable to discover the correspondences be-

DOI 10.1109/WIIAT.2008.301
3.1 Pre-processing Schema Sources

XMiner takes as input (1) a set of XML schemas, (2) a user-defined minimum support threshold (*minsup* for short) and (3) domain resources such as synonym dictionary and abbreviation table. XMiner first pre-processes the input schemas; then, it performs frequent substructure mining and constructs the mediated schema.

### 3 Pre-processing Schema Sources

#### 3.1 Pre-processing Schema Sources

**Resolving label conflicts.** As schema sources can be disparate, it is essential to resolve label conflicts between those schema sources and conceptualize their node labels. A node is attached with a label which conveys a meaning, or concept, in a particular domain. A concept can be conceptually represented by many labels; and a label in a single domain is assumed to represent only one concept. The same concept in the same schema source is seldom represented by two different labels. To conceptualize a label in a tree, we tokenize that label, expand its abbreviation and put it into relevant group of synonymous labels (steps 1-3 in Fig. 3). Abbreviated tokens are expanded into its unabbreviated forms using abbreviation table [13]. For instance, `TicketNo` and `ticket-num`, tokenized and expanded as `ticket, number`, are considered equivalent.

**Generating Tree-Grid.** We propose Tree-Grid for storing and manipulating tree structure of XML schemas. Tree-Grid is generated for each XML schema tree in step 4 of Fig. 3. A Tree-Grid is an artificial grid where node can be placed at the intersection of a row and a column in the grid. A node is a quadruple \( v_i = (lid, row, col, width) \), where *lid* is label identifier, *row* and *col* are row and column indices, and *width* is the number of leaf descendants of \( v_i \). Tree and Tree-Grid are equivalent: Tree-Grid \( T = \{ v_1, v_2, \ldots, v_n \} \) contains all the information about the tree structure. Given a Tree-Grid, we can reconstruct the original tree through the following properties: i) *Root*: \( x \) is root of \( T \) \( \iff (x.row = x.col = 0) \land (x.width = \max \{v.width\}, \forall v \in T) \). ii) *Leaf*: \( x \) is a leaf \( \iff x.width = 0 \). iii) *Non-Leaf*: \( x \) is a non-leaf \( \iff x.width > 0 \). iv) *Ancestor-Descendant*: \( x \) is ancestor of \( y \) \( \iff (x.row < y.row) \land (x.col < y.col + x.width) \). v) *Ancestor-Descendant Distance*: \( \exists x/y \in T \Rightarrow d(x, y) = y.row - x.row \). We develop our Tree-Grid to meet our own mining purposes. Tree-Grid is simple but efficient for tree traversal. It is also compact; hence, space-efficient. We can quickly calculate the distance between any two nodes based on *row* values.

\[
\begin{align*}
\text{Tree-Grid } T & \ni x \in T \Longrightarrow \text{Tree and Tree-Grid are equivalent: Tree-Grid } T = \{ v_1, v_2, \ldots, v_n \} \text{ contains all the information about the tree structure. Given a Tree-Grid, we can reconstruct the original tree through the following properties:}
\end{align*}
\]

\[
\begin{align*}
\text{i) Root: } x & \text{ is root of } T \iff (x.row = x.col = 0) \land (x.width = \max \{v.width\}, \forall v \in T). \\
\text{ii) Leaf: } x & \text{ is a leaf } \iff x.width = 0. \\
\text{iii) Non-Leaf: } x & \text{ is a non-leaf } \iff x.width > 0. \\
\text{iv) Ancestor-Descendant: } x & \text{ is ancestor of } y \iff (x.row < y.row) \land (x.col < y.col + x.width). \\
\text{v) Ancestor-Descendant Distance: } & \exists x/y \in T \Rightarrow d(x, y) = y.row - x.row.
\end{align*}
\]

**Figure 2. Schema Matching Problems**

**Figure 3. Pre-processing Schema Trees**

**3.2 Mining Frequent Substructures**

XMiner (1) starts with discovering frequent concepts represented by nodes; (2) discovers frequent paths for forming larger concepts — termed as composite concepts (CCs) — and relations between these CCs; and (3) constructs the mediated schema from the CCs and relations. We model an XML schema as a set of rooted unordered labeled schema trees. We define the frequency of a subtree \( T' \) in \( D \), denoted by \( f(T') \), is the percentage of \( m \) occurrences of \( T' \) in \( D \): \( f(T') = \frac{m}{|D|} \), where \( T' \) can be a single node or a path. \( T' \) is called frequent in \( D \) if \( f(T') \geq \text{minsup} \); otherwise, it is called infrequent.

**Mining frequent concepts.** Concept is the most essential constituent to construct a mediated schema. Thus, XMiner focuses on mining concepts and their semantic associations in schema trees. A concept is defined as a set of labels which have similar meaning in the same domain. A frequent concept is a concept whose occurrence in \( D \) is no less than *minsup*. A frequent concept is considered to be important and of interest in that domain. The level of user’s interest is determined by the support threshold. Frequent concepts are more likely to be included in the mediated schemas while infrequent concepts can be early removed.
due to their unimportance and irrelevance in the domain.

**Constructing composite concepts.** A frequent concept, carried by a single node, does not carry much information. Such isolated concept becomes more meaningful when it is semantically connected with other concepts to form a larger concept, called composite concept (CC), via the hierarchical containment. Let \( N \) be a set of concepts which occur at least one non-leaf in \( D \), and \( L \) a set of concepts which occur as leaf at least once in \( D \).

**Definition 2 (Composite Concept).** A composite concept \( X(x_1,x_2,...,x_p) \) is a tree of height \( l \) (where \( X \in N, x_i \in L \) ), which consists of \( X \) as the root and \( \{x_1,x_2,...,x_p\} \) as a set of \( X \)'s children such that \( f(X) \geq \text{minsup} \wedge f(x_i) \geq \text{minsup} \wedge f(X//x_i) \geq \text{minsup} \).

The purpose of a CC \( X(x_1,x_2,...,x_p) \) is to bear a coherent semantics representing a real world entity in the domain of interest. \( X \), when standing alone, is referred to as CC-root, and \( x_i \) (where \( i = 1,\ldots,p \) ) is called elementary concept (EC). The meaning of each EC (e.g. Title) provides more details about its CC (Movie); the CC, in turn, allows to identify the context for that EC (e.g. Title of a movie, instead of a person). Together, it is the structural context of \( x_i \) under \( X \) that helps clarify the meaning of both \( X \) and \( x_i \) in the hierarchy. The definition of the CC also describes what a CC is and how to mine such substructure. To avoid the explosion of ancestor-descendant combination of both \( X \) and \( x_i \), CC-root \( X \) must be a non-leaf concept \( (X \in N) \) and \( \text{EC} x_i \) must be a leaf concept \( (x_i \in L) \). Instead of using parent-child path, \( X \) and \( x_i \) are combined based on their ancestor-descendant path, i.e. \( X//x_i \), for the CC creation. Only meaningful frequent paths which have frequency exceeding the support threshold are retained for the mediated schema. This solves the problem of nesting discrepancy by exploring leaf descendants (instead of leaf children) of a non-leaf node.

**Mining Relations between CCs.** This step performs semantic association between two CCs mined from previous step, termed relation. Mined CCs from the previous step are disconnected from each other. In the real world, CCs make it existence clearer if they are in interaction with each other. Such semantics are captured in schema sources in form of hierarchical paths between non-leaves (i.e., roots) of CCs.

**Definition 3 (Relation).** Given two CCs \( X \) and \( Y \). A relation from \( X \) to \( Y \), denoted as \( X \rightarrow Y \), is defined as a frequent direct ancestor-descendant path from \( X \) to \( Y \). Let \( C \) denote a set of CCs and \( R \) a set of relations mined from \( D \). A relation must satisfy:

1. **Rule 1. Forward Path Only:** \( (X,Y \in C \land f(X//Y) \geq \text{minsup} \wedge f(Y//X) = 0) \Rightarrow X \rightarrow Y \in R \).
2. **Rule 2. Both Forward and Backward Paths:** \( (X,Y \in C \land f(X//Y) > 0 \land f(Y//X) > 0 \land f(Y//X) + f(X//Y) \geq \text{minsup}) \Rightarrow (X \rightarrow Y \in R \land Y \rightarrow X \in R) \).

The inclusion rules above define necessary conditions for an association between any two CCs to become a relation. We observe that same meaning between two concepts \( X \) and \( Y \) can be expressed in either in a top-down \( (X//Y) \) or bottom-up \( (Y//X) \) manner. For example, \( \text{Actor}/\text{Movie} \) meaning that “an actor casts a movie” is equivalent to \( \text{Movie}/\text{Actor} \) which means “a movie cast by an actor”. Ignoring the semantic similarity between those two ways of expression causes information loss. Thus, besides forward paths, it is critical to mine backward paths so that the final mediated schemas become more comprehensive. Rule 1 finds the relation between CCs \( X \) and \( Y \) based on forward paths only while Rule 2 includes both forward path \( X//Y \) and backward path \( Y//X \) as long as each of them exists at least once in the sources. The combination of both \( X//Y \) and \( Y//X \) mutually make the \( \text{minsup} \)-condition less strict and offer more chance for including relation between \( X \) and \( Y \). Such relaxation of inclusion rules allows us to solve the problem of backward paths faced by PORSCHE [13].

**Constructing Mediated Schema.** The mined CCs and relations are used to generate the mediated schema.

**Definition 4 (Mediated Schema).** Let \( C \) denote a set of CCs and \( R \) a set of relations. A mediated schema is defined as an ordered pair \( G = (C, R) \).

![Figure 4. Generated Mediated Schemas](image)

The mediated schema \( G \) contains all of the CCs in \( C \) and relations in \( R \). Frequent substructures of interest captured in the CCs are connected through frequent paths in form of relations. Each CC \( X(x_1,x_2,...,x_p) \in C \) is a subtree of height 1. For each relation \( X_i \rightarrow X_j \in R \), we establish a hierarchical containment relationship between \( X_i \) and \( X_j \) (where \( X_i \) becomes parent of \( X_j \)), jointly producing a larger subtree. For \( X_k \in C \) which do not participate in any relation in \( R \), each CC \( X_k \) remains a subtree on its own. As a result, there are two kinds of subtrees in \( G \): (1) the subtrees created from CCs and their relations; and (2) the subtrees of CCs which do not have any relation. The roots of these subtrees are connected to a virtual root. Fig. 4 presents the construction of two schema trees \( G_1 \) and \( G_2 \) based on a set of three CCs (Actor, Movie and Director) and their relations (Actor→Movie and Movie→Director).

### 4 Performance Evaluation

We evaluate XMiner w.r.t execution time and scale-up performance on two synthetic datasets, BOOKS [13] and
MOVIES 1, and a large real dataset on e-business, OAGIS 2 with characteristics in Table 1. Our settings are Intel P4, 2.4GHz, 1GB RAM with Java 1.6 and MySQL 5.1. Fig. 5a-b show execution times with \( \text{minsup} \) between 0 to 1.

Execution time is evaluated as the processing time required for producing mediated schemas against the minimum support threshold. It takes much more time to process MOVIES (1,312 schemas) than BOOKS (176) due to the number of schemas and their complexity. Yet MOVIES and BOOKS are not as large and complicated as OAGIS. To integrate 108 OAGIS schemas, it takes roughly 1,200s to process approximately 220,000 nodes when \( \text{minsup} \) is set high to 100%, and becomes 6.7 times slower when \( \text{minsup} = 0 \).

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>OAGIS</th>
<th>MOVIES</th>
<th>BOOKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Schema Trees</td>
<td>108</td>
<td>1,312</td>
<td>176</td>
</tr>
<tr>
<td>Number of Distinct Labels</td>
<td>925</td>
<td>87</td>
<td>19</td>
</tr>
<tr>
<td>Total Number of Nodes</td>
<td>218,762</td>
<td>64,706</td>
<td>1320</td>
</tr>
</tbody>
</table>

Table 1. Experimental Datasets

Figure 5. Performance Evaluation

Scale-up performance evaluates the processing speed required when size of schema sources increases. Fig. 5c-d show that XMiner scales well when the number of processed schemas/nodes scales up. Fig. 5c shows that BOOKS is processed with less than 400ms, only half of PORSCHE [13]’s speed (800ms). Compared to BOOKS, MOVIES dataset is slightly more complex: it contains more nodes and backward paths. Thus, it takes around 3 seconds to complete 176 MOVIES schemas. OAGIS has less schemas than MOVIES/BOOKS but contains much more nodes in each schema. Such complexity greatly impacts on the mediation performance. Fig. 5d shows a linear processing time when OAGIS scales up by thousands of nodes. Up to 80,000 nodes, XMiner’s speed is relatively comparable to PORSCHE’s; however, PORSCHE only processes almost half of the number of nodes that XMiner handles because it ignores all of the XML attributes. The largest schema COMA++ handles has 40,000 nodes [3]. XMiner performs more extensive experiments on very large schema instances of up to 220,000 nodes, and XMiner still scales up well.

5 Conclusion and Future Work

In this paper, we propose a schema mediation approach with an extension of frequent substructure mining [2]. Our approach reconciles the heterogeneous schema sources including nesting discrepancy, backward paths and schema discrepancy to generate mediated schemas. Our experiments show that XMiner is efficient enough to integrate hundreds or thousands of schema sources in large-scale scenarios without requiring too much human intervention [9]. We plan to extend our work by examining other XML features such as datatypes, cardinalities, and references.

References


1Available at: http://homepage.cs.latrobe.edu.au/h20nguyen/research/
2http://www.openapplications.org