

# Automatic Complex Schema Mapping Discovery and Validation by Structurally Coherent Frequent Mini-Taxonomies

Khalid Saleem  
LIRMM - UMR 5506 CNRS  
University Montpellier 2  
161 Rue Ada, F-34392 Montpellier  
saleem@lirmm.fr

Zohra Bellahsene  
LIRMM - UMR 5506 CNRS  
University Montpellier 2  
161 Rue Ada, F-34392 Montpellier  
bella@lirmm.fr

## ABSTRACT

Match cardinality aspect in schema matching is categorized as *simple element level matching* and *complex structural level matching*. Simple matching comprises of 1:1, 1:n and n:1 match cardinality, whereas n:m match cardinality is considered to be complex matching. Most of the existing approaches and tools give good 1:1 local and global match cardinality but lack the capabilities for handling the complex cardinality issue. In this paper we demonstrate an automatic approach for creation and validation of n:m schema mappings. Our technique is applicable to hierarchical structures like XML Schema. Basic idea is to propose an n:m nodes mapping between children (leaf nodes) of two matching non-leaf nodes of two schemas. The similarity computation of the two non-leaf nodes is based upon the syntactic and linguistic similarity of node labels; supported by similarity among the ancestral paths from nodes to the root. The n:m mapping proposition is then verified with the help of mini-taxonomies extracted from a large set of same domain schema trees. The mini-taxonomies are automatically extracted using frequent sub-tree mining approach; higher the frequency, higher the confidence of reliability. The verification algorithm performs comparison between the mini-taxonomies and the subtrees rooted at non-leaf nodes which guide the system for authenticity of proposed n:m mapping.

## Keywords

Simple Matching, Complex Matching, Mini-taxonomies, Collaboration, Tree Mining, Large scale

## 1. INTRODUCTION

Schema matching relies on discovering correspondences between similar elements of two schemas. Several different types of schema matching tools [19, 22] have been studied, demonstrating their benefit in different scenarios. In data integration schema matching is of central importance [2]. The

need for information integration arises in data warehousing, OLAP, data mashups [14], and work flows. Omnipresence of XML as a data exchange format on the web and the presence of metadata available in that format force us to focus on schema matching, and on matching for XML schemas in particular.

Match cardinality aspect in schema matching is categorized as *simple element level matching* [19] and *complex structural level matching*. Simple matching comprises of 1:1, 1:n and n:1 match cardinality, whereas n:m match cardinality is considered to be complex matching. Research literature shows numerous works [3, 7, 9, 11, 13, 18, 21] on simple matching. Most of the existing approaches and tools give good 1:1 local and global match cardinality but lack the capabilities for handling the complex cardinality issue. Therefore, in this paper, we try to propose a technique for complex match discovery.

In this work, we consider schemas to be rooted, labeled trees. This supports the computation of contextual semantics in the tree hierarchy. The contextual aspect is exploited by tree-mining, making it feasible to use almost automated approximate schema matching [8], even in a large-scale scenario (schemas have large number of nodes). The individual semantics of node labels have their own importance. We utilize linguistic matchers, based on tokenisation, and synonym and abbreviation tables, to extract the concepts hidden within them.

Tree mining techniques extract similar subtree patterns from a large set of trees and predict possible extensions of these patterns. A pattern starts with one node and is incrementally augmented. There are different techniques [5] which mine rooted, labeled, embedded or induced, ordered or unordered sub-trees. The function of tree mining [24] is to find sub-tree patterns that are frequent in the given set of trees. We have utilised this aspect of tree mining for computing the mini-taxonomies for domain specific concepts, from a given set of schemas, input as trees [20]. The method is a combination of concept terms analysis, using syntactic, lexical and contextual meanings of terms and tree mining algorithm.

## Our Contributions

In this paper, we focus particularly, though not exclusively, on n:m complex mappings. We present a new methodology

for matching two schemas, covering the element level and structural level mappings discovery. The main features of our approach are as follows:

1. The approach is almost automatic and hybrid in nature.
2. It is based on a tree mining technique supporting large size schema matching. To support tree mining, we model schemas as rooted ordered (depth-first) labelled trees.
3. It extends the tree mining data structure proposed in [24]. It uses ancestor scope properties (integer logical operations) on schema nodes to enable fast calculation of contextual (hierarchical) similarity between them.
4. Our methodology discovers semantically approximate 1:1, 1:n, n:1 simple mappings between nodes of the two schema trees.
5. It suggests perspective n:m complex mappings between the two schemas and then validates these propositions using already available mini-taxonomies, representing schema concepts.
6. It utilises a new ancestor similarity measure for contextual matching, which overcomes the problem of inverted parent-child mapping (if in source schema a node  $x_s$  is parent of node  $y_s$  and in target schema node  $x_t$  is parent of node  $y_t$ , and there exists matches from  $x_s$  to  $y_t$  and  $y_s$  to  $x_t$ ).
7. It intuitively highlights the use of automatically extracted mini-taxonomies from a large set of domain specific schemas, supporting the collective intelligence aspect.
8. The approach is implemented as a prototype. We report on experiments using different real<sup>1</sup> and synthetic scenarios.

## Outline

The remainder of the paper is organized as follows. Section 2 presents the preliminaries of our matching approach, related definitions and concepts. Section 3 gives the related work in schema matching, specifically complex schema matching. In Section 4 we describe our approach for discovering and validating complex schema matching between schemas processed as tree structures. Section 5 demonstrates a running example to support our approach. Section 6 presents the experimental evaluation along with discussion on the results. Section 7 outlines future perspective and concludes.

## 2. PRELIMINARIES

Schema matching finds similarities between elements in two or more schemas. We model the schemas as tree structures for fast contextual semantics extraction.

**Definition 1 (Schema Tree):** A schema  $S = (V, E)$  is a rooted, labelled tree[24], consisting of nodes  $V = \{0, 1, \dots, n\}$ ,

<sup>1</sup><http://www.metaquerier.cs.uiuc.edu/repository/>,  
<http://www.cise.ufl.edu/project/thalia.html>

and edges  $E = \{(x, y) \mid x, y \in V\}$ . One distinguished node  $r \in V$  is called the root, and for all  $x \in V$ , there is a unique path from  $r$  to  $x$ . Further,  $lab: V \rightarrow L$  is a labelling function mapping nodes to labels in  $L = \{l_1, l_2, \dots\}$ .•

## 2.1 Match Cardinality

There can be several types of correspondences between elements of two schemas, depending upon the number of elements participating in a match, as schema match cardinality [19]. The match cardinality is mainly categorized as *global* or *local*. Global cardinality is related to the number of appearances of an element in the total match results between two input schemas. Local schema match cardinality characterises the number of elements participating in a certain match. Local schema match cardinality can be of four types.

There are three simple element level match cardinalities and one structural level complex match cardinality [19]. Since we are matching schema tree structures (elements are nodes), where the leaf nodes hold data (XML schemas), we place more emphasis on leaf node matching. Our categorisation of node match cardinalities is driven by the node's leaf or non-leaf (inner node) status.

- i) 1:1 - one node of source schema corresponds to one node in the target schema; leaf:leaf or non-leaf:non-leaf.
- ii) 1:n - one node in the source schema is equivalent to a composition of n leaves in the target schema; leaf:non-leaf, where a source leaf node is mapped to a set of n leaf nodes of the subtree rooted at the non-leaf node in the target schema.
- iii) n:1 - n leaves in source schema compositely map to one leaf in the target schema; non-leaf:leaf, allowing a set of n leaf nodes of the subtree rooted at the non-leaf node in the source schema, to be mapped to a target leaf.
- iv) n:m - n leaves in source schema compositely map to a composition of m leaves in the target schema; non-leaf:non-leaf, allowing a set of n leaf nodes of the subtree rooted at the non-leaf node in the source schema, to be mapped to a set of n leaf nodes, of subtree rooted at the non-leaf node in target schema.

The first three types of local match cardinality are normally evaluated using the element level information. Whereas the n:m cardinality also requires structural information of the input schemas.

**Example 1 (Match Cardinality):** For 1:1, cardinality is straightforward. Let us consider Figure 1 for 1:n and n:1 match. A match is found between  $S_{source}name[2, 2]^2$ , child of *author*, and  $S_{target}name[2, 4]$ , child of *writer*, with children *first* and *last*. We have a 1:2 mapping  $(name)_{source} : (name - first, name - last)_{target}$ <sup>3</sup>. Also, there is a match between  $S_{source}publisher[3, 4]$  and  $S_{target}publisher[6, 6]$ , implying a n:1 mapping of  $(publisher - name)_{source} : (publisher)_{target}$ , i.e., 1:1 mapping.•

Semantically speaking, a match between two nodes is fuzzy.

<sup>2</sup>In figure 1, for each node [x,y], x denotes the depth-first traversal number and y denotes the last node number of subtree rooted at x.

<sup>3</sup>- delimiter is used to show the downward traversal within the tree.

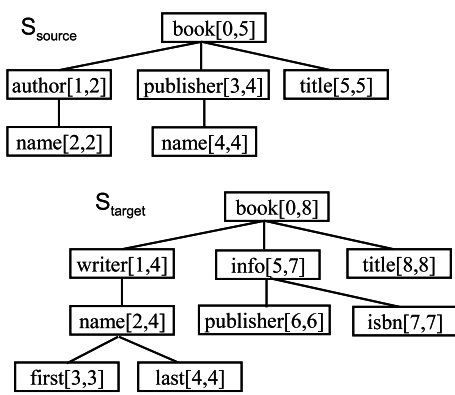


Figure 1: Two schemas with Node Scope values for each node.

It can be either an equivalence or a partial equivalence. In a partial match, the similarity is partial. It is highlighted in the example 2.

## 2.2 Label Semantics

Label semantics correspond to the meaning of the label (irrespective of the node it is related to). It is a composition of meanings attached to the tokens making up the label. As shown by Examples 3 and 4, different labels can represent similar concepts.

**Definition 2 (Label Semantics)** A label  $l$  is a composition of  $m$  strings, called tokens. We apply the tokenisation function  $tok$  which maps a label to a set of tokens  $T_l = \{t_1, t_2, \dots, t_m\}$ . Tokenisation [11] helps in establishing similarity between two labels.

$tok : L \rightarrow \mathcal{P}(T)$ , where  $\mathcal{P}(T)$  is a power set over  $T$ .

**Example 2 (Partial Match):** In source schema Name = ‘John M. Brown’, is partially matched to LastName = ‘Brown’ and FirstName = ‘John’ in the target, because Name also contains the MiddleInitial = ‘M’. •

**Example 3 (Label Equivalence):** ‘FirstName’, tokenised as {first,name}, and ‘NameFirst’, tokenised as {name, first}, are equivalent, with 100 % similarity. •

**Example 4 (Synonymous Labels):** ‘WriterName’, tokenised as {writer,name}, and ‘AuthorName’, tokenised as {author, name} are equivalent (they represent the same concept), since ‘writer’ is a synonym of ‘author’. •

## 2.3 Node Semantics

Semantics of node are given with combination of the semantics of the node’s label and its contextual placement in the tree [11]. Contextual placement of a node within the tree is calculated with the help of the concept *Node Scope*.

**Definition 3 (Node Scope)** In schema  $S$  each node  $x \in V$  is numbered according to its order in the depth-first traversal of  $S$  (the root is numbered 0). Let  $SubTree(x)$  denote the sub-tree rooted at  $x$ , and  $x$  be numbered  $X$ , and let  $y$  be the rightmost leaf (or highest numbered descendant) under  $x$ , numbered  $Y$ . Then the scope of  $x$  is  $scope(x)=[X, Y]$ . In-

tuitively,  $scope(x)$  is the range of nodes under  $x$ , and includes  $x$  itself, see Figure 1. The count of nodes in  $SubTree(x)$  is  $Y - X + 1$ .

## Scope Properties

Scope properties help in discovering the contextual placement of a node [24]. Property testing involves simple integer comparisons. We utilise these properties in frequent subtree detection. Example 5 gives an overview of the use of scope values.

### Node Scope Properties

Given  $x[X, Y]$ ,  $xa[Xa, Ya]$ , and  $xr[Xr, Yr]$ .

**Property 1 Leaf Node**( $x$ ):  $X = Y$ .

**Property 2 Non-Leaf Node**( $x$ ):  $X < Y$ .

**Property 3 Ancestor** ( $x, x_a$ ):  $x_a$  is an ancestor of  $x$ ;  $X_a < X$  and  $Y_a \geq Y$ .

**Property 4 RightHandSideNode** ( $x, x_r$ ):  $x_r$  is Right Hand Side Node of  $x$  with Non-Overlapping Scope;  $X_r > Y$ .

**Example 5 (Scope Properties Use) :** Let us consider schema tree  $S_{source}$  in Figure 1. The nodes  $name[2, 2]$ ,  $name[4, 4]$  and  $title[5, 5]$  are leaf nodes according to Property 1. Whereas  $book[0, 5]$ ,  $author[1, 2]$  and  $publisher[4, 5]$  satisfy Property 3, thus these are the non-leaf nodes. There are two nodes with label *name*. If we have to check for the *name* of *author*, we can perform the Ancestor check (Property 3) on each of the two nodes with respect to node *author*[1, 2]. The check returns true for node *name*[2, 2] and false for node *name*[4, 4]. Consider node *author*[1, 2] and node *publisher*[3, 4]. The two nodes have non-overlapping scope according to Property 3, and can be considered as cousin nodes.

## 3. RELATED WORK

Previous work on schema matching was developed in the context of schema translation and integration [3, 1], knowledge representation [11], machine learning, and information retrieval [8]. All these approaches aimed to provide a *good* quality matching but require significant human intervention. The direct human participation in large scale scenarios requires very user friendly interfaces, and abates the overall performance of the tool. Therefore we need to automate the schema matching approaches to the maximum.

Simple matching with quality has been successfully demonstrated in [1, 11, 17] by utilising element level and structural level schema knowledge. There has been very limited work on complex schema mapping. Use of data instance and machine learning techniques for simple and complex schema matching has been shown in [8, 6] for relational and XML schemas. [23] presents a user-centric approach using fixed point iteration algorithm similarity flooding (SF) [17] and other linguistic techniques to calculate complex mappings.

Authors in [6] demonstrate a semi-automatic tool called iMAP, which discovers 1:1 and 1:n mappings between two relational schemas. The approach utilises machine learning algorithms on data instances to predict 1:n mapping expressions. It employs knowledge like past complex matches, domain integrity constraints, and overlap data. In [12], He et al. workout a mining technique for integrating a large number of web query interfaces, considered as schemas. The

**Table 1: A brief survey of schema matching techniques with local cardinality**

Authors	Year	Characteristics	Card.
Saleem et al. [21]	2008	PORSCHÉ: Provides a large scale schema matching and integration method using tree mining technique	1:1,1:n
Wang et al. [23]	2007	SCIA: Presents an interactive schema matching tool with the ability to create executable mappings create executable mappings	n:m
Lee et al. [15]	2006	Computes similarity based upon domain ontology (evolves on user feedback), and ancestral path similarity	1:1,1:n
Boukottaya et al. [4]	2005	The approach exploits the ancestor, child and leaf context of elements to compute similarity	1:1
Do et al. [7]	2007	COMA++: Finds similarities between two large schemas/ontologies, using several lexical and contextual algorithms, compositely	1:1
Bernstein et al. [3]	2004	PROTOPLASM: Presents a interactive framework to incorporate state of the art matching techniques, ranging from string similarity to data mining algorithms	1:1
He et al. [12]	2004	MGS/DCM: Proposes automatic correlational data mining techniques for schema matching and integration, holistically	n:m
Dhamankar et al. [6]	2004	iMAP: Computes complex match expressions using data instance based machine learning techniques	n:m
Ehrig et al. [9]	2004	QOM: Matches large ontologies with time performance in view, employing iteratively lexical match algorithms to reduce the search space for an element with each iteration	1:1
Giunchiglia et al. [11]	2004	S-Match is a hybrid matcher, which utilizes Wordnet and tackles schema matching as propositional satisfiability problem	1:1
Embley et al. [10]	2004	Utilizes manually constructed small ontology snippets and data ranges to discover schema matchings	n:m
Melnik et al. [18]	2003	RONDO: Uses an algorithm which propagates nodes similarity to neighbouring nodes within the iterative fix point computation	1:1
Doan et al. [8]	2003	GLUE: Provides a composite matcher for ontology taxonomy matching based on data instances exploitation using machine learning techniques	n:m
Hernandez et al. [13]	2002	CLIO: Utilizes data instances (Naive Bayes Learner) and element level schema matching algorithms and generates SQL based mappings	1:1

proposed framework holistically exploits the set of input schemas, thus exploring the contextual information beyond two schemas. The algorithms DCM and MGC makes it feasible to compute the n:m schema matching. The iMAP and DCM/MGC approaches exploit schemas with low depth for elements (schema considered as hierarchical structures) and show a very limited use of structural matchers.

Another interesting work regarding XML documents transformation in [15] presents comprehensively the 1:1, 1:n and n:1 schema matching approach between two schemas. The algorithms used in this work are massively dependent on structural information of schemas. The proposed method worked in two step. First, preliminary matching between leaf nodes of the two schemas are computed using an external domain ontology and leaf node similarity. In second step the contextual aspect of nodes is computed as path similarity, i.e., similarity between the ancestor nodes of the current node being matched. In the same application domain, work by Boukottaya and Vnoirbeek [4] further enhances the contextual aspect of the nodes. The approach takes into account three levels of context similarity; ancestor, child and leaf. The two research works help in exploiting the hierarchical structures with larger depth.

[21] presents a large scale schema integration framework, which uses a top down matching approach. The technique follows the idea that some level of ancestor similarity is must for an element down the hierarchy. Linguistic and tree mining algorithms are used to compute element level matchings. Like wise QOM [9] is a robust, time performance method for RDF based schema and ontology matching. Use of ontologies/ontologies for finding complex matchings have also been demonstrated in some works. [10] uses manually created mini-ontologies for domain specific concepts, to generate n:m correspondences between two schemas. We give a brief survey of schema matching with reference to local match cardinality in Table 1.

## 4. COMPLEX MATCH DISCOVERY - OUR APPROACH

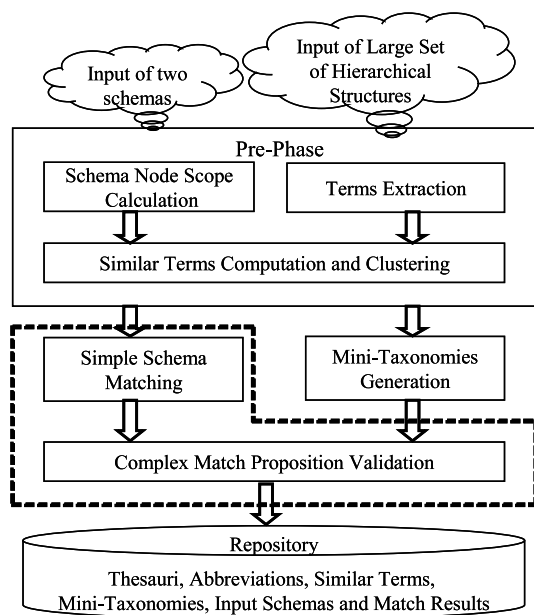
In this section we describe our approach toward complex match discovery. We base our work on some previous research presented in [21, 20]. Our work is based on the concepts defined as Schema Tree, Label Semantics, Node Semantics and Scope Properties (Ancestor and Right-hand-side node) as given in section 2.

### 4.1 Architecture

The architecture of our approach for complex match discovery is shown in Figure 2. The approach is composed of four modules: (i) *Pre-Phase*, (ii) *Mini-Taxonomies Generation*, (iii) *Simple Schema Matching*, and (iv) *Complex Match Proposition Validation*, supported by a repository which houses oracles and concepts' mini-taxonomies, schemas and match results for future reuse.

*Pre-Phase* module processes the input as trees, calculating the depth-first node number and scope (Figure 6), for each of the nodes in the input schema trees. At the same time, for each schema tree a listing of nodes is constructed, sorted in depth-first traversal order. As the trees are being processed, a sorted global list of labels over the whole set of schemas is created by the *Terms Extraction* sub-module.

In *Similar Terms Computation and Clustering* sub-module, label concepts are computed using linguistic techniques. We tokenise the labels and expand the abbreviated tokens using an abbreviation oracle. Currently, we utilise a domain specific user defined abbreviation table. Further, we make use of token similarity, supported by an abbreviation table and a manually defined domain specific synonym table. Label comparison is based on similar token sets or similar synonym token sets. The architecture is flexible enough to employ additional abbreviation or synonym oracles or arbitrary string matching algorithms (details in [21]). Similar labels are clustered together and each cluster is represented by a label from the respective cluster, which is the most



**Figure 2: Architecture for complex matching discovery and validation using automatically generated mini-taxonomies.**

frequent label with in the set of input schemas (details in [20]).

*Mini-Taxonomies Generation* module uses a tree mining approach to extract frequent sub-trees from the set of schema trees. We use frequent pattern growth mining algorithm based on research in [24]. During the process all labels with in a cluster of similar labels are logically replaced by the cluster representative label. This helps in computing the right count of frequency for a label concept [20].

*Simple Schema Matching* module, generates simple matchings, along with complex match propositions, based upon labels' linguistic similarity and node's contextual similarity using node scope values. The node scope criteria is used to evaluate the ancestor similarity factor with time performance. *Complex Match Proposition Validation* (CMPV) module forms the main core of this research work. We use a novel algorithm to validate the proposed complex matches with the help of automatically generated domain specific mini-taxonomies. These two modules form the core of our work, presented in this paper. Following sections provide a detail discussion about them.

The *Repository* is an indispensable part of the system. It houses oracles: thesauri and abbreviation lists. It also stores schemas and mappings, and provides persistent support to the mapping process.

## 4.2 Simple Match Discovery

In this section we describe the *Simple Schema Matching* module. Before we move on, first we give a brief description of Pre-Phase module.

## Pre-Phase

Algorithm *nodeScope* (Figure 3) is a recursive method, for calculating node scope of each node in the schema, generating a schema node list  $V$ . It takes as input the current node, its parent reference, and the node list. In its first activation, reference to the schema root is used as the current node: there is no parent, and  $V$  is empty. A new node list element is created (L3.2-6)<sup>4</sup> and added to  $V$ . Next, if the current node is a leaf, a recursive method *update* (Figure 4) is called. This adjusts the rightmost node reference for the current  $V$  element and then goes to adjust its ancestor entries in  $V$ . This method recurses up the tree till it reaches the root, and adjusts all elements on the path from the current node to the root. Next, in *nodeScope* (L3.11-12), the method is called for each child of the current node. Parameter  $d$  gives the depth level with in the tree during the depth-first traversal. We utilise the node depth value for adjusting the match confidence related to ancestor mapping. At the start of the recursive algorithm for node scope calculation, value of  $d$  is 0, depicting level 0 or root node level. It is incremented at L3.10, before the recursive call for next level (L3.12).

```

Algorithm : NodeScope

Data:  $p, c, V, d$ 
 $p$ : parent node list element,  $c$ : current node,
 $V$ : nodes list,  $d$ : node depth
Result:  $V$ 
1 begin
2    $x \leftarrow$  New nodesListElement( $c$ )
3    $x.number \leftarrow$  length( $V$ )
4    $x.parentNode \leftarrow p$ 
5    $x.rightMostNode \leftarrow \emptyset$ 
6    $x.depth \leftarrow d$ 
7   Add  $x$  to  $V$ 
8   if  $c$  has no children then
9     update( $x, x$ )
10   $d \leftarrow d + 1$ 
11  for each child of  $c$  do
12    nodeScope( $x, child, V, d$ )
13 end
  
```

**Figure 3: Algorithm for schema tree node scope calculation, along with node depth.**

```

Algorithm : update

Data:  $x_c, x_r$ 
 $x_c$ : Current nodes list element
 $x_r$ : Right most node element of
current nodes list element
1 begin
2    $x_c.rightMostNode \leftarrow x_r$ 
3   if  $x_c \neg$  Root Node then
4     update( $x_c.parentNode, x_r$ )
5 end
  
```

**Figure 4: Algorithm for updating scope entries.**

*Simple Schema Matching* module matches two schemas for possible mappings based upon the linguistic and contextual (ancestor) similarities. The method discovers mappings with 1:1, 1:n and n:1 cardinality and proposes n:m complex

<sup>4</sup>Lx.y refers to line y of algorithm in figure x

matches between the elements of the two schemas. The proposed n:m mappings are validated with the help of mini-taxonomies representing possible domain concepts.

During simple match process, a match for every node (L5.8) from source schema tree list  $V_1$  to target schema tree list  $V_2$  is calculated. For each input node  $x$ , a set  $V_t$  of possible mappable target nodes in the target schema is created, producing the target search space for  $x$ . The criterion for the creation of this set of nodes is node label equivalence or partial equivalence (L5.9,10).  $V_t$  can have one or several (L5.11) candidate target nodes. Next step is the ancestor map existence check. The method returns a value  $\alpha$  based on the ancestor node mapping and the proximity of the mapping node to the current node. If the target search space  $V_t$  has more than one node, method *ancestorMap* is executed

```

Algorithm : SimpleMatch

Data:  $V_1, V_2$  Lists of nodes of Schema 1 and 2
1  $L_x$  Source list  $V_1$  node label
2  $L_{xtl}$  Set of labels of candidate nodes in target list  $V_2$ 
3  $V_t$  Set of candidate nodes in target list  $V_2$ 
4  $xt$  Target node
5  $\alpha$  Similarity Confidence
6  $\beta$  Match Type
7 begin
8   for each node  $x \in V_1$  do
9      $L_x \leftarrow \text{lab}(x)$ 
10     $L_{xtl} \leftarrow \text{similarLabels}(L_x, V_2)$ 
11    if  $L_{xtl} \neq \emptyset$  then
12       $V_t \leftarrow \text{VT}(L_{xtl})$ 
13       $\alpha_{temp} \leftarrow 0.0$ 
14       $\alpha \leftarrow 0.0$ 
15      for each node  $xt_i \in V_t$  do
16         $\alpha_i \leftarrow \text{ancestorMap}(x, xt_i)$ 
17        if  $\alpha_i > \alpha_{temp}$  then
18           $\alpha_{temp} \leftarrow \alpha_i$ 
19           $\alpha \leftarrow \alpha_i$ 
20           $xt \leftarrow xt_i$ 
21      if ( $\text{Leaf}(x) \wedge \text{Leaf}(xt)$ ) then
22         $\beta \leftarrow 11$ 
23      else
24        if ( $\neg \text{Leaf}(x) \wedge \neg \text{Leaf}(xt)$ ) then
25           $\beta \leftarrow n$ 
26        else
27          if ( $\text{Leaf}(x) \wedge \neg \text{Leaf}(xt)$ ) then
28             $\beta \leftarrow 1n$ 
29          else
30            if ( $\neg \text{Leaf}(x) \wedge \neg \text{Leaf}(xt)$ ) then
31               $\beta \leftarrow n1$ 
32      map( $x, xt, \beta, \alpha$ )
33 end

```

Figure 5: Algorithm for simple matching between two schemas.

for each perspective matching node (L5.15-20). The candidate target node with the highest  $\alpha$  value is selected as the mappable target node.

In *ancestorMap* method two functions are executed in parallel. Firstly, it checks for map existence for some ancestor node of current source node and secondly the proximity of the ancestor node from the current node. Ancestor node check is performed by utilising the ancestor node scope prop-

erty, and if a map exists, then the proximity is calculated and returned as  $\alpha$ . To proceed further, another added check is confirmed that the target schema node to which the source schema ancestor node is mapped to, is also an ancestor of the candidate target node. This is a variation of upward co-copy distance discussed in research [16]. The ancestor map checking also helps us partially solving the inverted node mapping problem; example 6 describes this advantage. The following formula shows the proximity calculation.

$$\alpha = 1/(ddif_s + ddif_t)$$

where  $ddif_s$  is the depth difference between the current source node and the ancestor node for which a mapping exists and  $ddif_t$  is the depth difference between the candidate target node and the ancestor node in target schema to which the source ancestor node is mapped to. The maximum value for  $\alpha$  is 0.5, i.e., when the source schema ancestor node is the parent of current source node and the target schema node to which it is mapped to is the parent node of candidate target node.

**Example 6 (Inverted Node Similarity):** Consider the two schema trees  $S_1$  and  $S_2$  in Figure 6. Top-down depth first matching traversal guides us to match  $S_1[0,5]$ book  $\leftrightarrow S_2[0,5]$ book,  $S_1[1,5]$ author  $\leftrightarrow S_2[1,5]$ writer (*author* is synonym of *writer*). Next, we have  $S_1[2,2]$ name  $\leftrightarrow S_2[3,3]$ name with  $\alpha = 0.33$  and following is the  $S_1[3,5]$ address  $\leftrightarrow S_2[2,5]$ address. Thus overcoming the problem when *address* and *name* have siblings relationship in source schema tree and parent child relationship in target schema tree. •

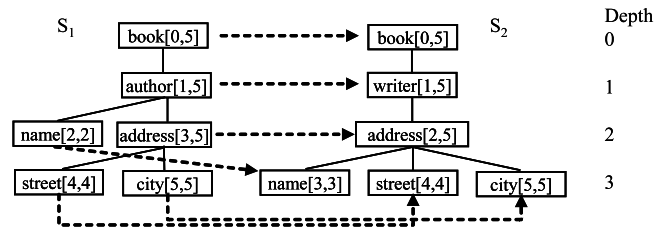


Figure 6: Input hierarchical structures with partial inverted node scenario.

After selecting the best match for the source node, the method proposes the type of mapping. By default it is a 1:1 mapping, since we are comparing one node from source schema to one node in target schema. Our method is more XML schema oriented where leaf nodes denote the real data values. Therefore we propose the type of mapping biased toward the leaf nodes as discussed in section 2.1. This aspect is handled in simple match algorithm with the help of parameter  $\beta$  (L5.21-31). Finally, the map function creates the possible mapping from source node to target node (L5.32). The n:m complex match propositions between non-leaf nodes are further exploited in the complex match proposition validation module. The related algorithm is given in Figure 7.

### 4.3 Mini-Taxonomy Snippets Aspect

Before we move to the complex match proposition validation part, we present another data structure, *Mini-Taxonomy*

*Snippets*, which helps in the validation process. In this section we present briefly our approach for detection of these *Mini-Taxonomy Snippets* or ontological concepts, as a hierarchical structure, from available domain specific schema tree structures. A domain concept is considered to be small tree structure which we call a Mini-Taxonomy. The idea is similar to research done in [10]. We follow the automatic approach [20] rather than the manual snippet designing in [10].

The technique has an iterative nature based on incrementally extracting frequent sub-trees from a given set of trees. The sub-tree frequency in the forest of trees is user defined parameter. The algorithm takes as input the list of terms, with similar terms linked together to form a cluster (each cluster can have one or more terms). First task performed by the algorithm is to compute the frequency of each term in the forest of trees. Next, with in each cluster, the term with the highest frequency in the forest of trees is taken as the symbol representing the cluster. The frequency of the cluster symbol is computed by adding frequencies of all the terms in the cluster. From here on the algorithm executes similar to frequent sub-tree mining algorithm given in [24], with similar label cluster symbols as the starting labels for the data structure showing frequent sub-tree patterns. After the process is complete, and we have a list of sets of mini-taxonomies. Each list representing set of mini-taxonomies of same size. Next we replicate this list of sets of mini-taxonomies, by replacing the cluster level similar labels in the list. Thus giving us all possible mini-taxonomies which can be considered as concept representation, frequently utilised by domain users.

#### 4.4 Complex Match Proposition Validation

The basic idea behind the Complex Match Proposition Validation (CMPV) is to some how validate the n:m match propositions created in simple matching module. Our approach tries to utilise small conceptual taxonomies already generated from large number of schemas with in a specific domain as enumerated above. For simplicity, we structure these mini-taxonomies as individual XML schema instances and generate a data structure *MiniTax* for them, similar to the input source and target schema, using the same *node-Scope* algorithm given in Figure 3.

Algorithm *ComplexMatchValidation* takes as input  $V_1, V_2$  (source and target schema nodes lists respectively) and *MiniTax* (list of lists of mini-taxonomies nodes). The algorithm traverses each node  $x$  of source schema,  $V_1$ , with a n:m map type (depicted as  $nm$ ) toward  $xt$  node of target schema,  $V_2$  (L7.6,9). The objective of the algorithm is to certify that the leaf nodes of the sub-tree rooted at such a node  $x$  in the source schema can form a n:m mapping with the leaf nodes of sub-tree rooted at node  $xt$  in the target schema. The algorithm extracts leaf nodes labels from sub-tree rooted at  $x$  into a temporary set data structure *sLeaf* and similarly populates a data structure *tLeaf* for  $xt$  (L7.11,12).

Next, the algorithm traverses the *MiniTax* data structure (L7.15) for mini-taxonomies with root node label similar to node labels of  $x$  and  $xt$  (L7.17,21). Further, the leaf nodes labels of these mini-taxonomies are extracted into temporary set data structures *mTsLeaf* (root label similar to  $x$  label) and *mTtLeaf* (root label similar to  $xt$  label) at L7.18,22.

Finally, the validation check is executed, i.e., if *sLeaf* is a subset of *mTsLeaf* and *tLeaf* is a subset of *mTtLeaf*. If such mini-taxonomies are found in *MiniTax*, it certifies that the concepts at  $x$  and  $xt$  are similar and their respective leaf nodes can form a n:m mapping (L7.27-29), since the mini-taxonomies have been frequently used in the domain. Thus utilising the collective intelligence over a specific domain.

#### Algorithm : ComplexMatchValidation

```

Data:  $V_1, V_2, MiniTax$ 
 $V_1, V_2$  : Lists of nodes of schema 1 (source) and 2(target)
 $MiniTax$  : List of lists of mini-taxonomies
Each mini-taxonomy list,  $mT, V_1$  and  $V_2$  sorted on
depth-first order
1 sLeaf : Set of labels of leaf nodes of non-leaf node of  $V_1$ 
2 tLeaf : Set of labels of leaf nodes of non-leaf node of  $V_2$ 
3 mTsLeaf, mTtLeaf : Sets of labels of leaf nodes of
4 mini-taxonomy
5  $F_s, F_t$  : Boolean flags
6 begin
7   for each node  $x \in V_1$  do
8      $F_s \leftarrow \text{false}$ 
9      $F_t \leftarrow \text{false}$ 
10    if ( $\neg \text{Leaf}(x) \wedge \text{exists}(\text{map}(x, xt, \beta, \alpha)) \wedge \beta = nm$ ) then
11      //  $xt$  is non-leaf node in  $V_2$ 
12      sLeaf  $\leftarrow$  leafNodesLabels( $x$ )
13      tLeaf  $\leftarrow$  leafNodesLabels( $xt$ )
14       $L_x \leftarrow \text{lab}(x)$ 
15       $L_{xt} \leftarrow \text{lab}(xt)$ 
16      for each  $mT \in MiniTax$  do
17         $L_{mTroot} \leftarrow \text{lab}(mTroot)$ 
18        //  $mTroot$  is root node of  $mT$ 
19        if ( $\neg F_s \wedge L_{mTroot} = L_x$ ) then
20          mTsLeaf  $\leftarrow$  leafNodesLabels( $L_{mTroot}$ )
21          if sLeaf  $\subset$  mTsLeaf then
22             $F_s \leftarrow \text{true}$ 
23          if ( $\neg F_t \wedge L_{mTroot} = L_{xt}$ ) then
24            mTtLeaf  $\leftarrow$  leafNodesLabels( $L_{mTroot}$ )
25            if tLeaf  $\subset$  mTtLeaf then
26               $F_t \leftarrow \text{true}$ 
27          if  $F_s \wedge F_t$  then
28            break
29        if  $F_s \wedge F_t$  then
30           $\beta \leftarrow nm$ 
31           $\text{map}(\text{leafNodes}(x), \text{leafNodes}(xt), \beta)$ 
32 end

```

Figure 7: Algorithm for validation of complex matching between two schemas.

## 5. COMPLEX MATCH VALIDATION EXAMPLE

Figure 8 shows two schema trees related to books domain. A list of correspondences is shown in Figure 9 after the execution of simple matching (Figure 5). Simple match algorithm discover's one to one match between two corresponding elements of the two schemas. In parallel, it also proposes any possible complex match, according to the leaf/non-leaf status of the elements participating in the mapping. The possible complex match propositions are shown in brackets.

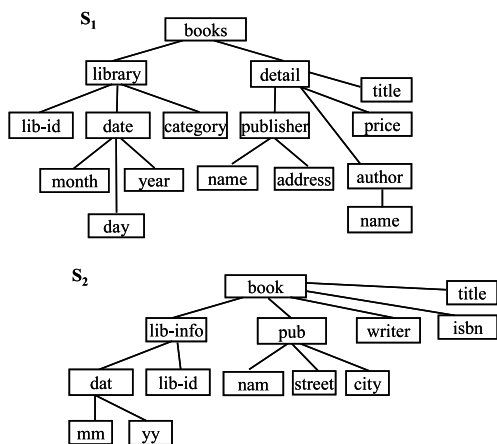


Figure 8: Two schema (trees)  $S_1$  and  $S_2$  for complex match discovery.

The scenario presents one n:1 and four n:m complex match situations. The n:1 match given by  $S_1$ .author[12,13]  $\leftrightarrow$   $S_2$ .writer[10,10] proposes that all leaf nodes of  $S_1$ .author[12,13] compositely correspond to  $S_2$ .writer[10,10]. Thus establishing the fact, the data against element  $S_1$ .name[13,13] matches to data represented by  $S_2$ .writer[10,10] element. The general analysis of the four complex matches ( $S_1$ .books[0,15]  $\leftrightarrow$   $S_2$ .book[0,12],  $S_1$ .library[1,7]  $\leftrightarrow$   $S_2$ .writer[1,5]) show that two propositions have relatively large sub-trees rooted at the non-leaf elements. Therefore presenting a collection of concepts rather than a single concept. This fact is enumerated by scanning the scope of the non-leaf node. The scrutiny of other two complex match propositions ( $S_1$ .date[3,6]  $\leftrightarrow$   $S_2$ .dat[2,4],  $S_1$ .publisher[9,11]  $\leftrightarrow$   $S_2$ .pub[6,9]) show that small size sub-trees reside at the elements of interest.

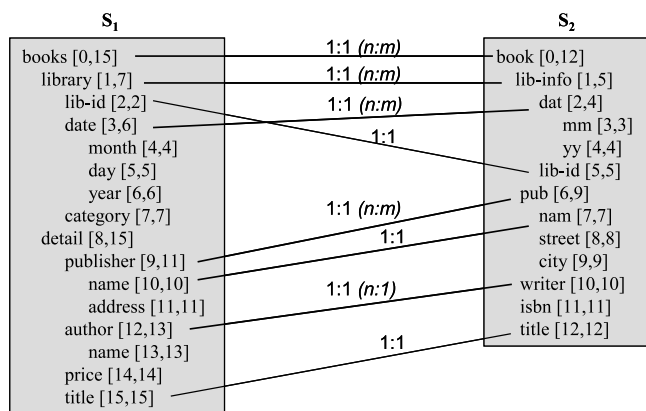


Figure 9: Element level mappings between schemas  $S_1$  and  $S_2$  after execution of simple match algorithm; complex match proposition shown in brackets.

The validation of these proposed complex matchings is done by the ComplexMatchValidation algorithm given in Figure 7. For this purpose, the set of already acquired mini-taxonomies (using ExSTax method [20]) are considered. There are two sets of mini-taxonomies as shown in Figure 10. Set (a) presents the mini-taxonomy representing

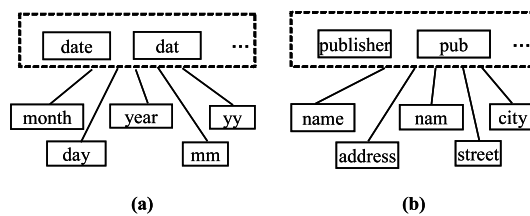


Figure 10: Mini-taxonomies extracted from large input of books domain schemas, using ExSTax method for complex match validation.

the *date* concept, which can be represented by *date*, *dat* or some other similar string, as the root node for the concept. The leaf nodes collection of *month*, *day*, *year*, *mm*, *yy* represent attributes describing the concept. Within one instance of the mini-taxonomy, the presence of synonymous leaf nodes is not possible e.g. if *month* and *mm* are synonymous then the two nodes will not exist together in a mini-taxonomy with root node *date* or *dat*.

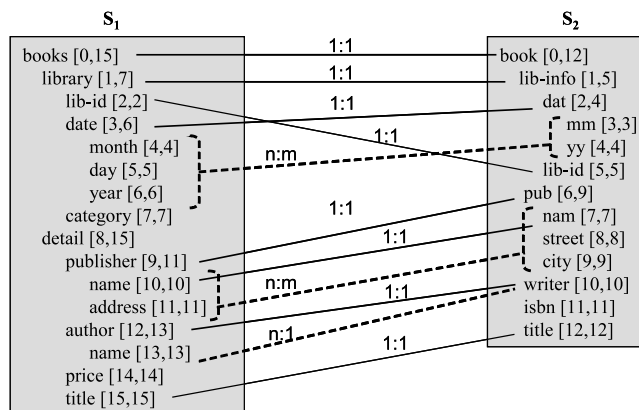


Figure 11: Mappings between schemas  $S_1$  and  $S_2$  after execution of complex match validation algorithm.

The execution of the validation algorithm occurs for each of the four n:m propositions. In case of  $S_1$ .books[0,15]  $\leftrightarrow$   $S_2$ .book[0,12] and  $S_1$ .library[1,7]  $\leftrightarrow$   $S_2$ .writer[1,5], no mini-taxonomy is found with root element similar to *books* or *book* and *library* or *lib-info*. As a result the two propositions are discarded and only 1:1 simple match is considered. Whereas in the other two cases the algorithm finds frequent mini-taxonomies in the form of date-month/day/year, dat-mm/yy, publisher-name/address and pub-nam/street/city<sup>5</sup>. The algorithm substantially authenticates the propositions and creates two more complex mappings as  $S_1$ .month[4,4], day[5,5], year[6,6]  $\leftrightarrow$   $S_2$ .mm[3,3], yy[4,4] and  $S_1$ .name[10,10], address[11,11]  $\leftrightarrow$   $S_2$ .nam[7,7], street[8,8], city[9,9]. The final mappings are shown in Figure 11.

## 6. EVALUATION

The prototype implementation uses Java 5.0. A PC with Intel Xeon, 2.33 GHz processor and 2 GB RAM, running Win-

<sup>5</sup> - and / delimiters denote downward and upward traversal within the tree, respectively

**Table 2: Characteristics of domain schema trees used in the mini-taxonomy computation experiments.**

Domain	BOOKS1	BOOKSEARCH	JOBS	AUTO	AIRTRAVEL	REALESTATE	COURSES
Type	Synthetic	Real	Real	Real	Real	Real	Real
Number of Schemas	176	19	20	14	20	14	42
Average nodes per schema	8	6	5	5	14	9	8
Largest schema size	14	12	8	10	21	20	17
Smallest schema size	5	3	4	3	6	4	2
Schema Tree Depth	3	2	2	2	2	2	4
n:m match propositions	2	2	1	1	3	2	2

dows XP was used. We selected several data sets, BOOKS1, BOOKSEARCH<sup>6</sup>, JOBS, AUTO, AIRTRAVEL, REALESTATE and COURSES, as the input hierarchical structures for our experiments. Characteristics of these sets of schemas in given in Tab. 2.

A manual analysis of the above mentioned real domain schemas showed that there existed very limited possible sets of elements (set of leaf nodes representing some concept), which may participate in a complex map. Since such sets are rare, finding them as frequent mini-taxonomies is even more scarce. Following is a possible list of concepts against which n:m mappings may exist in the above schemas.

- (i) date/depart/return : month,day,year <-> mm,yy <-> month,day,time
- (ii) address/location : name,address <-> nam,street,city <-> street1, street2,city <-> address1,address2 <-> AreaCode,Country <-> city,state
- (iii) telephone : tel\_res,tel\_off <-> morn\_tel, even\_tel,night\_tel <-> tel\_mobile,tel\_fix
- (iv) name : firstName,lastName <-> f\_name,mi,l\_name
- (v) passengers : adult,child,infant <-> adult,senior,child <-> adult\_12-65,adult\_65,child\_2-4, child\_5-12,infant
- (vi) return/depart : month,day <-> month,day,year,time
- (vii) schedule : days,time,room <-> DayTime,room <-> Times, Place <-> TimeBegin,TimeEnd,Room,Building <-> time,building
- (viii) car model : vfrom,vto <-> fyear,tyear

To verify our CMPV method, we induced some of these sets of elements to several real schemas, to extract them as mini-taxonomies. Our implementation showed the expected results, since the overall test scenarios have been synthetically generated.

### Discussion

The idea of using mini-taxonomies works well, if their leaf nodes can represent some real complex match with a contextual map at ancestor level. And the ancestor level mapping is highly dependent on label matching. Working with real world schemas showed (i) the possibility of complex match is very limited, and (ii) the label matching framework should be exceptionall good.

<sup>6</sup><http://metaquierer.cs.uiuc.edu/repository>

Considering the first issue of complex match possibility, manual scrutiny of above mentioned schemas showed very limited number of n:m matches. For example in 20 schemas of AIRTRAVEL domain (with average of nodes per schema equal to 14), there were three concepts, *passengers*, *return* and *depart*, which could be considered for complex match. But the second issue of good label matching deficiency abates the matching process. For example, there existed 13 different labels for passenger concept in the domain, i.e., How many travelers are going, Number of passengers, NumAdults, adultsChildren, persons, adultnoChildno, passengers, number of travelers, how many people are going, num-pax, travelers, who is going on this trip, passengers-adults-num. Computing a high frequency of *passengers* concept (mini-taxonomies) existence was unaccomplishable because of the multi-word, cryptic labels also including homonyms.

Our idea has some similarity to Embley et al. [10] work, which uses manually created concept ontology snippets for complex match discovery. Our approach tries to show that it can be done automatically in a scenario, where there is the availability of large number of domain schemas. Secondly using the ancestor level contextual map, the simple match results overcome the inverted parent-child match problem. Thus making the system more robust and dependable. Although the approach is similar to upward cotopy distance measure, but it is more flexible in execution. We do a top-down traversal during match process, i.e. parents are matched before children. Therefore, the nearest ancestor with an adequate match in the target schema is enough to verify the context.

### 7. CONCLUSION AND FUTURE WORK

In this paper, we have presented an automatic approach for discovering complex match. It is based on a tree mining technique supporting large size schema matching. It extends the tree mining data structure proposed in [24] and exploits ancestor scope properties (integer logical operations) on schema nodes to enable fast calculation of contextual (hierarchical) similarity between them. It originality lies in the use of mini-taxonomies in complex match discovery. Mini-taxonomies are frequent subtrees within this forest of schema trees. Furthermore, our approach is implemented as a prototype and validated by experiments using different real schemas.

Firstly, the method computes simple element level matchings between two schemas. The algorithm linguistically matches nodes labels and ancestor level match existence (in ontology matching called upward cotopy distance) for this purpose. At this point, the matches have 1:1 cardinality, but our method also proposes the possible complex matches intrinsically. Our approach is based on the leaf or non-leaf status of

the node, putting forward the match proposition that when a non-leaf node is matched to a non-leaf node, there is the probability of n:m match between the leaf nodes of the two non-leaf nodes. The algorithm CMPV certifies this proposition, indirectly utilising the collective intelligence of the domain users. This is achieved by using mini-taxonomies snippets, extracted from large number of input schemas developed and used over the specific domain by users.

The work presented in this paper has lots of prospective applications. Our approach can provide an automatic matching environment, supported by the collective intelligence of the domain users. In future, we plan to extend the label level matching techniques. We intend to utilize state of the art lexical matchers and linguistic dictionaries. Secondly, we will be extending the idea to evolving schemas matching in the large scale scenario. Finally, our aim is to fully incorporate our technique for web based interface forms (containing hierarchical nested fields) matching and collaboration among different metadata based virtual social environments.

## 8. REFERENCES

- [1] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *ACM SIGMOD*, 2005.
- [2] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
- [3] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-Strength Schema Matching. *SIGMOD Record*, 33(4):38–43, 2004.
- [4] A. Boukottaya and C. Vanoirbeek. Schema matching for transforming structured documents. In *ACM DocEng*, 2005.
- [5] Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok. Frequent subtree mining - an overview. *Fundamenta Informaticae*, 66(1-2):161–198, 2005.
- [6] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex semantic matches between database schemas. In *ACM SIGMOD*, pages 383–394, 2004.
- [7] H.-H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.
- [8] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Y. Halevy. Learning to match ontologies on the Semantic Web. *VLDB J.*, 12(4):303–319, 2003.
- [9] M. Ehrig and S. Staab. QOM – Quick Ontology Mapping. In *ISWC*, pages 683–697, 2004.
- [10] D. W. Embley, L. Xu, and Y. Ding. Automatic direct and indirect schema mapping: Experiences and lessons learned. *ACM SIGMOD Record*, 33(4):14–19, 2004.
- [11] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an Algorithm and an Implementation of Semantic Matching. In *ESWS*, pages 61–75, 2004.
- [12] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *KDD*, pages 148–157, 2004.
- [13] M. A. Hernandez, R. J. Miller, and L. M. Haas. Clío: A semi-automatic tool for schema mapping. In *ACM SIGMOD*, 2002.
- [14] A. Jhingran. Enterprise Information Mashups: Integrating Information, Simply – Keynote Address. In *VLDB*, pages 3–4, 2006.
- [15] J.-S. Lee and K.-H. Lee. Computing simple and complex matchings between xml schemas for transforming xml documents. *Information and Software Technology - Elsevier*, 48:937–946, 2006.
- [16] A. Maedche and S. Staab. Measuring similarity between ontologies. In *EKAW*, 2002.
- [17] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, 2002.
- [18] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *SIGMOD*, pages 193–204, 2003.
- [19] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
- [20] K. Saleem and Z. Bellahsene. Automatic extraction of structurally coherent mini-taxonomies. In *ER*, 2008.
- [21] K. Saleem, Z. Bellahsene, and E. Hunt. Porsche: Performance oriented schema mediation. *Information Systems - Elsevier*, 33:637–657, 2008.
- [22] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *J. Data Semantics IV*, pages 146–171, 2005.
- [23] G. Wang, V. Zavesov, R. Rifaieh, A. Rajasekar, J. Goguen, and M. Miller. Towards user centric schema mapping platform. In *VLDB Workshop Semantic Data and Semantic Integration*, 2007.
- [24] M. J. Zaki. Efficiently Mining Frequent Embedded Unordered Trees. *Fundamenta Informaticae*, 66(1-2):33–52, 2005.