Improving Clustering-Based Schema Matching Using Latent Semantic Indexing

Alsayed Algergawy[1,2(✉)], Seham Moawed[3], Amany Sarhan[2], Ali Eldosouky[3], and Gunter Saake[4]

[1] Institute of Computer Science, Friedrich Schiller University of Jena, Jena, Germany
alsayed.algergawy@uni-jena.de
[2] Department of Computer Engineering, Tanta University, Tanta, Egypt
[3] Department of Computer Engineering, Mansoura University, Mansoura, Egypt
[4] Department of Computer Science, University of Magdeburg, Magdeburg, Germany

Abstract. The increasing size and the widespread use of XML data and different types of ontologies result in the big challenge of how to integrate these data. A critical step towards building this integration is to identify and discover semantically corresponding elements across heterogeneous data sets. This identification process becomes more and more challenging when dealing with large schemas and ontologies. Clustering-based matching is a great step towards more significant reduction of the search space and thus improving the matching efficiency. However, current methods used to identify similar clusters depend on literally matching terms. To keep high matching quality along with high matching efficiency, hidden semantic relationships among clusters' elements should be discovered. To this end, in this paper, we propose a Latent Semantic Indexing-based approach that allows retrieving the conceptual meaning between clusters. The experimental evaluations reveal that the proposed approach permits encouraging and significant improvements towards building large-scale matching approaches.

Keywords: Schema matching · Large-scale matching · Latent semantic indexing · Partitioning-based matching · Hierarchical clustering method · Vector Space Model (VSM) · Document similarity

1 Introduction

Schema matching is the task of identifying and discovering correspondences between semantically similar elements of two schemas or ontologies [31,33]. The demand for schema matching is high in a diverse number of data application scenarios, such as data integration [10,16] and web service discovery [4,20]. Due to heterogeneities inherent in schemas, manual matching becomes expensive, extremely tedious, and error prone. Therefore, efforts are invested in the development of automated schema matching systems. Furthermore, the rapidly increasing size and use of XML schemas and ontologies adds additional dimensions of challenges to cope with the large matching problem [30].

A. Hameurlain et al. (Eds.): TLDKS XV, LNCS 8920, pp. 102–123, 2014.
DOI: 10.1007/978-3-662-45761-0_4

To deal with these challenges, several approaches have been designed to improve the performance of the matching process for large-scale schemas involving both matching aspects: *effectiveness* and *efficiency* [5,14,19,21,32,35]. These solutions include matching techniques that depend on the partition-based principle [2,14,21]. These partition-based matching techniques divide input schemas/ontologies into a set of partitions and execute a partition-wise matching between the two schemas. The partitioning is performed in such a way that each partition of the first schema is matched with only a small subset of the partitions of the second schema (ideally, only with one partition) [30]. The entities of the dissimilar partition pairs can be eliminated from further matching process thus reducing the search space to achieve better efficiency. Space complexity of the matching process is also reduced. Reducing the search space of the matching process indeed achieves better matching efficiency, however, it does not guarantee the matching quality. Determining and selecting similar clusters for further matching plays an important role to keep high matching quality along with high matching efficiency.

To partition input schemas/ontologies, COMA++ uses relatively simple heuristic rules to partition the input schemas, often resulting in too few or too many partitions [14]. Both MOM and Falcon have been applied only to certain ontology languages and cannot be applied to other data models [21,35]. Algergawy et al. use a bottom-up clustering scheme which utilizes the context-based structural node similarities [2]. To determine similar partitions, COMA++ only uses limited information about the partition (only the root node of the partition) to determine the similarity between partitions of the input schemas. On the other hand, solutions, such as Falcon [21], fully evaluate the input ontologies to assess the partition similarity. In Algergawy et al. [2], a light-weight similarity measure is applied that considers all elements of each cluster pair and represents each cluster as a cluster document. It uses the Vector Space Model and TF-IDF to determine the similarity between cluster documents.

Unfortunately, the Vector Space Model (VSM) depends upon literally matching document terms with those appearing in a query [8]. The inaccuracy of lexical matching methods is coming from the inability to determine concepts between documents and the query. So, the literal terms in a user's query may not match those of a relevant document (synonymy). In addition, most words have multiple meanings (polysemy), so terms in a user's query will literally match terms in irrelevant documents. Latent semantic indexing (LSI) is a more suitable approach that allows retrieving information on the basis of a conceptual topic or meaning of a document [12,22]. To this end, in this paper, we capture features introduced by the latent semantic indexing technique in large-scale schema matching problems. In particular, we first represent input schemas as rooted labelled trees, called *schema trees*. The use of a common data structure, *schema tree*, to model input schemas, enables matching among different schemas and ontologies. We then develop an agglomerative clustering algorithm to partition each schema tree into a set of disjoint groups. The clustering algorithms depends on the structural properties of the schema tree. To identify and determine similar clusters across

two cluster sets representing two schema trees, we develop an LSI-based technique which is able to discover hidden semantic relationships between similar clusters. Once having similar clusters, we finally apply a set of element matchers to get correspondences between their elements. To verify the performance of the proposed approach, we conducted a set of experiments in order to prove its superiority upon previous work.

To sum up, the main contributions of the paper can be stated as follows:

- addressing the problem of partitioning-based schema matching,
- developing and elaborating an XSOM-based parser to facilitate XML data representation,
- proposing an LSI-based approach to determine similar clusters in the context of schema matching, and
- conducting an intensive set of experiments to validate the proposed approach.

The rest of the paper is structured as follows. Related work is presented in Sect. 2. We describe latent semantic indexing in Sect. 3. We then introduce the proposed matching framework in Sect. 4, concentrating on similar clusters identification. We report experiments conducted and analysis results in Sect. 5. Section 6 concludes the paper.

2 Related Work

Semantic heterogeneity is a key problem in different data sharing systems, be it a federated database [6], a data integration system [15,16], a web service [20], or a peer data management system [18]. Involved data sources are typically designed independently, and hence use different schemas. To obtain meaningful interoperation, one needs a semantic mapping between the schemas, i.e. a set of expressions that specify how the data in one source corresponds to the data in the other. Hence, the specific problem of schema matching has to be addressed before mapping is constructed. To this aim, a set of correspondences among similar elements in different schemas has to be identified. Manually constructing a match is a very labor intensive task that requires complete knowledge of the semantics of the data in the schemas being matched. Solutions that try to provide some automatic support for schema matching have received steady attention over the years [7,31,33].

Unfortunately, most of these systems severely lack performance when dealing with large matching problems. Consequently, several approaches have been proposed to address the problem of matching two large schemas [2,14,19,21,30,32,35]. Promising areas for large-scale schema matching lie in four main directions: reduction of search space for matching, parallel matching, self-tuning match workflows and reuse of previous match results [30]. In this section, we pay great attention to the approaches that perform reduction of the search space. The standard approach of cross join evaluation for schema matching reduces match efficiency and quality. In order to reduce the search space for matching, two methods can be used: early pruning of dissimilar element pairs and partition-based matching.

Quick ontology matching (QOM) was one of the first approaches to implement the idea of early pruning of dissimilar element pairs [17]. It iteratively applies a sequence of matchers and can restrict the search space for every matcher. Peukert et al. introduce a set of filter operators within match workflows to prune dissimilar element pairs (whose similarity is below some minimal threshold) from intermediate match results [28]. They also propose a rule-based approach to rewrite match workflows for improving efficiency, in particular by placing filter operators within sequences of matchers [27].

COMA++ was one of the first systems to support partition-based schema matching [14]. It depends on fragment matching which has two phases. The first phase determines fragments of the two schemas and identifies the most similar ones. Detecting similar fragments is some kind of light-weight matching via the similarity of fragment roots. The second phase identifies corresponding elements between each pair of similar fragments. Finally, the fragment-based match results are merged to obtain the complete output mapping [14].

Another matching system that supports partition-based matching is Falcon-AO [21]. It initially partitions the ontologies into relatively small disjoint blocks by using structural clustering. Then, matching is applied to the most similar blocks from the two ontologies. To determine block similarity, the algorithm utilizes the so-called anchors. Anchors are highly similar element pairs that are determined before partitioning by a combined name/comment matcher. Dynamic partition-based matching is supported by AnchorFlood [32]. It avoids the a-priori partitioning of the ontologies by utilizing anchors (similar concept pairs). It takes them as a starting point to incrementally match elements in their structural neighborhood until no further matches are found or all elements are processed. Thus the partitions (segments) are located around the anchors.

Zhong et al. propose an unbalanced ontology matching approach, which concerns matching a lightweight ontology with a more heavyweight one [36]. They abstract the subontology (partition) from the heavyweight ontology that is most similar to the smaller one and consider this sub-ontology for matching. To determine this sub-ontology, the approach needs to carry out a nested loop to determine the similarity values between concepts from the two ontologies. To this end, name-based similarity measures such as Edit distance and WordNet have been used. Concepts from the larger ontology with similarity values higher than a predefined threshold are then selected. Finally, the subontology is determined by evaluating the subgraphs around the similar elements found in the first step. We observe that in order to determine a similar sub-ontology, whole concepts from two ontologies have to be compared using name-based similarity measures, which is not efficient for large matching problems.

Algergawy et al. uses a clustering-based matching approach that is based on an agglomerative bottom-up hierarchical fashion [2]. It is generic and can be applied to different data models including XML schemas. The clustering scheme is performed based on the context-based structural node similarities. Then, a light weight linguistic technique is used to find similar partitions to match.

This technique makes use of the Vector Space Model (VSM) for computing the similarity between clusters.

To sum up, partitioning-based matching techniques improve the matching efficiency, however, they do not guarantee a high matching quality. Identifying and selecting similar partitions for matching plays an important role in this aspect. To the best of our knowledge, most of current matching techniques ignore this role. Therefore and in order to address these challenges, we introduce a new LSI-based approach to correctly identify and select the similar clusters.

3 Latent Semantic Indexing

One typical scenario of human machine interaction in information retrieval is by natural language queries: the user formulates a request, e.g., by providing a number of keywords or some free-form text, and expects the system to return the relevant data in some amenable representation, e.g., in form of a ranked list of relevant documents. Many retrieval methods are based on simple word matching strategies to determine the rank of relevance of a document with respect to a query. It is well known that literal term matching has severe drawbacks, mainly due to the ambivalence of words and their unavoidable lack of precision as well as due to personal style and individual differences in word usage.

A popular approach that depends on literal term matching is the Vector Space Model (VSM) [8,12]. The vector space model procedure can be divided into three stages. The first stage is the document indexing where content bearing terms are extracted from the document text. The second stage is the weighting of the indexed terms to enhance retrieval of document relevant to the user. The last stage ranks the document with respect to the query according to a similarity measure. The VSM considers the terms in documents as being independent from each other, an assumption which is never satisfied by the human language. An idea can be expressed in many ways (synonymy) and, moreover, many words may have multiple meanings (polysemy).

Latent Semantic Indexing (LSI) [12,22] is a statistical technique which tries to surpass some limitations imposed by the traditional Vector Space Model (VSM). It exploits the dependencies between words by assuming that there is some underlying or "latent" structure in word usage across documents that is partially obscured by variability in word choice and this structure can be revealed statistically.

LSI projects queries and documents into a space with "latent" semantic dimensions. In the latent semantic space, a query and a document can have high cosine similarity even if they do not share any terms. We can look at LSI as a similarity metric that is an alternative to word overlap measures like *tf.idf* [25]. LSI usually takes the (high dimensional) vector space representation of documents based on term frequencies [14] as a starting point and applies a dimension reducing linear projection. The specific form of this mapping is determined by a given document collection and is based on a Singular Value Decomposition (SVD) of the corresponding term/document matrix. The general

claim is that similarities between documents or between documents and queries can be more reliably estimated in the reduced latent space representation than in the original representation. The rationale is that documents which share frequently co-occurring terms will have a similar representation in the latent space, even if they have no terms in common. LSI thus performs some sort of noise reduction and has the potential benefit to detect synonyms as well as words that refer to the same topic. In many applications this has proven to result in more robust word processing.

To make the paper self-contained, in the following, we present main steps of LSI [22]:

- **Constructing Term Document Matrix.** Each term is represented by a row and each document is represented by a column. Initially, each cell a_{ij} in the matrix A is represented by the number of times the associated term appears in the indicated document, tf_{ij}. Once the matrix is created, local and global weighting functions can be applied to each non-zero element in the matrix. The weighting functions transform each cell, a_{ij} of A, to be the product of a local term weight which describes the relative frequency of a term in a document, and a global weight, g_i, which describes the relative frequency of the term within the entire collection of documents. The local weighting function of $\log(tf_{if}+1)$ decreases the effect of large differences in frequencies. The global weighting function of Entropy, which is defined as $1+\sum_j \frac{P_{ij}\log(P_{ij})}{\log(n)}$ where $P_{ij} = \frac{tf_{ij}}{gf_i}$, is the total number of times the term appears in the entire collection of n documents, gives less weight to terms occurring frequently in a document collection. Therefore, each non-zero element in the term-document matrix is represented as:

$$a_{ij} = (1 + \sum_j \frac{P_{ij}\log(P_{ij})}{\log(n)}) \times \log(tf_{ij}+1). \tag{1}$$

- **Decomposing the Term Document Matrix.** LSI applies singular value decomposition (SVD) to the matrix A. In SVD, a rectangular matrix is factored into the product of other three matrices as in

$$A = USV^T \tag{2}$$

where U is an $m \times m$ orthogonal matrix, $U^TU = I_m$, V is an $n \times n$ orthogonal matrix, $V^TV = I_n$, and S is a diagonal matrix of decreasing singular values such that $s_{1,1} \geq s_{2,2} ... \geq s_{r,r} > 0$, and $s_{i,j} = 0$ where $i \neq j$. I_m and I_n are the identity matrices of orders m and n, respectively. The matrix U gives a vector for each term in LSI space, while the matrix V represents each document as a vector.
- **Dimensionality Reduction.** In LSI, it is not the intent to reproduce A. The main goal is to retain the largest singular values. In the literature, this is called dimensionality reduction. LSI computes a low rank approximation to A using a truncated SVD [22]. Let k be an integer and $k \ll min(m, n)$, U_k is

defined to be the first k columns of U, and V_k^T to be the first k rows of V^T. Let $S_k = diag[s_1, ..., s_k]$ contain the first k largest singular values as in the following equation:

$$A_k = U_k S_k V_k^T \tag{3}$$

This is a new pseudo term-document matrix with reduced dimension. The SVD operation, along with this reduction, has the effect of preserving the most important semantic information in the text while reducing noise and other undesirable artifacts of the original space of A.

– **Incorporating the Query and Ranking the Documents.** A query, similar to a document, is a set of words which must be represented as a vector in the k-dimensional space. It can be represented as:

$$q = q^T U_k S_k^{-1} \tag{4}$$

where q is the vector of words in the users query, multiplied by the appropriate term weights. The sum of these $k-$dimensional terms vectors is reflected by the term $= q^T U_k$ and the right multiplication by S_k^{-1} differentially weights the separate dimensions. Thus, the query vector is located at the weighted sum of its constituent term vectors. The query vector can then be compared to all existing document vectors, and the documents ranked by their similarity to the query. A common similarity measure can be used to reflect the relationship between the query vector and every document vector. Typically, the results are ranked and top-k documents *or* documents exceeding some cosine threshold are returned to the user.

4 The Matching Framework

In this section, we introduce the proposed schema matching framework. The framework consists of four main steps, as shown in Fig. 1. In the following, we describe each step focusing on the parsing and similar cluster determination steps.

4.1 Data Model and Schema Preparation

XML is a flexible modeling language with self-explanatory tags that allow the storage of information in semi-structured formats [1]. There are two types of XML data: XML schema and XML document. An XML schema allows describing the structure and the legal building blocks for an XML document, while an XML document (document instance) represents a snapshot of what the XML document contains. Several XML schema languages have been proposed [23]. Among them, XML document type definition (DTD) and XML Schema Definition (XSD) are commonly used. DTD has limited capabilities compared to other schema languages, such as XSD. Moreover, XML schema definition (XSD) aims to be more expressive than DTD and more usable by a wider variety of

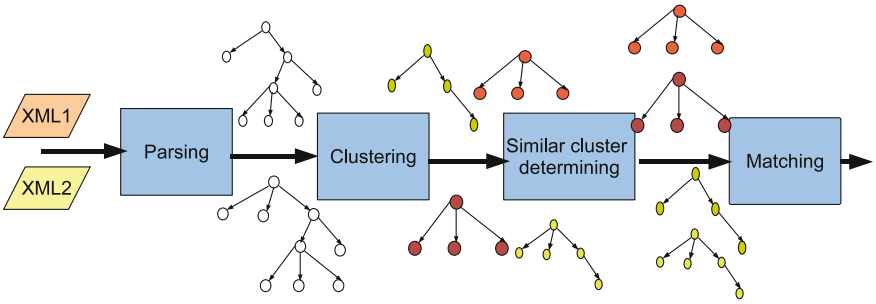

Fig. 1. Schema matching steps.

applications such as XQuery[1], SOAP, and web services[2]. Therefore, through the paper, we use the term "schema" to denote XML schema (XSD).

An XML schema consists of a set of components. The XML schema components can be broadly classified into three main groups as described below:

- **Primary components** may or must have names and include the following components: simple type definitions, complex type definitions, element declarations, and attribute declarations. The element and attribute declarations must have names, while the type definitions may have names.
- **Secondary components** must have names. Attribute group definitions, identify constraint definitions, model group definitions, notation declarations, type alternatives, and assertions are examples of such components.
- **Helper components** provide small parts of other components, they are not independent of their context and contain components such as annotations, model groups, particles, wildcards, and attribute use.

To make the proposed approach more generic, the input XML schemas should be internally represented using a common data model. The choice of which data model should be used is an important step towards building a reasonable schema matching system. The data model should be able to normalize schemas that are represented by different schema languages, thus eliminating syntax differences between schemas. Most current schema matching systems choose graph data structure as the internal representation [31,33]. The choice of graphs as an internal representation for the schemas to be matched has many motivations. First, graphs are well-known data structures and have their algorithms and implementations. Second, by using the graph as a common data model, the schema matching problem is transformed into another standard problem; graph matching. XML schemas can also be represented as trees by dealing with nesting and repetition problems using a set of predefined transformation rules [24].

In our implementation, we represent XML schemas as rooted, labeled trees, called *schema trees, ST* [5]. A schema tree consists of a finite set of nodes and

[1] http://www.w3.org/TR/xquery/.
[2] http://msdn.microsoft.com/en-us/library/ee265410(v=bts.10).aspx.

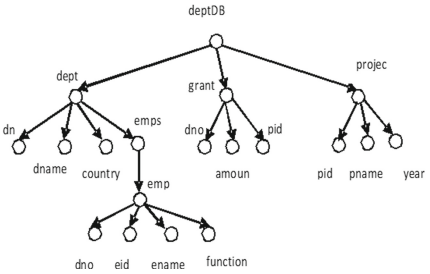

Fig. 2. Schema tree, *deptDB*.

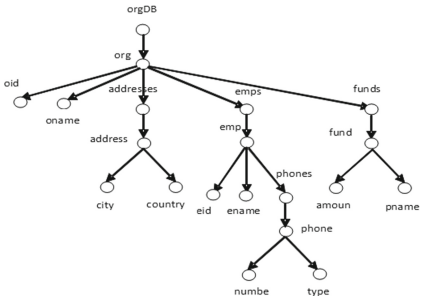

Fig. 3. Schema tree, *orgDB*.

a finite set of edges. Each node is uniquely identified by an object identifier and expresses the component' features, such as element name and element datatype, while each edge represents the relationship between every two nodes. Figures 2 and 3 present the schema tree representation of two XML schemas taken from [9]. Both *deptDB* and *orgDB* represent information about departments with their employees and grants, as well as the projects for which grants are awarded. The figures show the tree representation of the two schemas, wherein each node is associated with the name label, such as *grant* and *funds* from *deptDB* and *orgDB*, respectively.

We use the XML Schema Object Model (XSOM) parser[3] to parse input XML schemas. XSOM is a Java library that allows applications to easily parse XML schema documents and to inspect information in them. The library is a simple and effective implementation of "schema components" as defined in the XML schema. The parsing process starts by defining a new class using the XSOM. Through the constructor, we create an empty tree which will be filled with schema elements extracted through the parsing operation. As in [3,5], we classify schema tree nodes into two kinds: *atomic nodes* and *complex nodes*. Atomic nodes are the leaf nodes in the schema tree while complex nodes are the internal nodes inside the tree. We then instantiate an object of the XSOM parser through a defined class. This class enables us to get all constructs of the schema and related schemas and put them into memory for further processing, by using the defined object methods. Once the schema component is resolved, we iterate through global declarations inside the root element to iteratively build the corresponding schema tree.

4.2 Schema Clustering

Once a schema is parsed and internally represented as a schema tree, the next step is to divide it into a set of disjoint sub-trees. By this step, we aim to simplify the

[3] https://xsom.java.net.

matching processing, especially when dealing with large-scale schemas. To this end, we make use of our clustering algorithm presented in [2]. To make the paper self-contained, we briefly present the algorithm. Clustering is a useful technique for grouping nodes such that nodes within a single cluster are structurally similar, while nodes in different groups are dissimilar. First, we introduce the *node context*, which is defined as the node surroundings. This means that the context of a node, $C(v_i)$, is the combination of the node itself as well as all parents and children of the node. Based on the node context, we then compute the structure similarity between every pair of nodes in the schema tree.

The structure similarity between two nodes v_i and v_j which exist in the same ST is computed based on the number of common nodes between their contexts, $|C(v_i) \cap C(v_j)|$. Based on this structural similarity, we construct a link between each node pair, containing the two nodes and their structural similarity. The set of generated links constitutes a hash table called the *links hash table*. By using a threshold value greater than 0 we can dramatically reduce the number of entries in the links hash table. It should be noted that the similarity is assumed to be 0 if there is no pre-computed link. This table is used as an input for the clustering algorithm.

We develop an agglomerative clustering algorithm, which produces a tree representing the hierarchy of clusters in a bottom-up way. The algorithm mainly consists of the following four steps:

1. *Preparation.* The structural similarity is computed and the links hash table is then constructed.
2. *Cluster initialization.* In this step, the bottom level of the cluster hierarchy is developed by representing each node as a cluster.
3. *Cluster hierarchy construction.* This is the main step of the clustering algorithm. It is devoted to build the cluster hierarchy by merging elements from different clusters to form one cluster based on specified merging criteria.
4. *Best cluster selection.* It selects the cluster solution. More information can be found in [2].

Example 1. By applying the clustering algorithm to the two schema trees illustrated in Figs. 2 and 3, we get two cluster sets. $CSet_1 = \{C_{11}, C_{12}\}$ and $CSet_2 = \{C_{21}\}$ for *deptDB* and *orgDB* schemas, respectively, as shown in Fig. 4. The figure indicates that *deptDB* is partitioned into two semantically structured clusters. The first, C_{11}, represents projects and their funds, while the second, C_{12}, represents departments and employees working on these projects. Figure 4 also shows that the *orgDB* schema is not partitioned since the structural organization of the schema is not semantically clear like the *deptDB* schema. This example shows the ability of the clustering algorithm to correctly cluster schema trees into semantically structured partitions.

4.3 Similar Cluster Determination

The proposed approach focuses on 2-way or pairwise schema matching where two related input schemas are matched with each other. As mentioned before, the

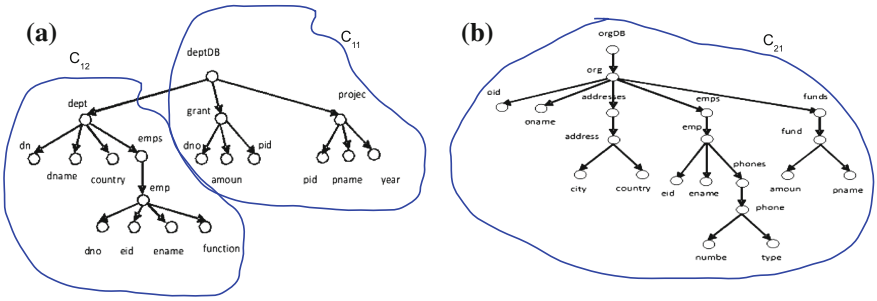

Fig. 4. Schema tree partitions

clustering algorithm divides each schema tree into a set of clusters. Each cluster contains a set of nodes that are structurally similar. The task is to determine which sets of clusters are similar. This information is used later as input for the matching algorithm.

Latent semantic indexing aims to detect semantically similar partitions (clusters) in the two schema trees. The motivation here is to reduce the match overhead by applying matching on similar partitions only and ignoring the irrelevant ones. Algorithm 1 is proposed to achieve this task. It accepts two sets of clusters as input and processes them to determine similar clusters across the two sets. The algorithm has the following main steps, as shown in Algorithm 1.

– *Preparation of term-document matrix.* The algorithm starts by initializing the similar cluster set, *Sim_Clust*, by setting it to the empty set, *line 1.* Then, to construct the document term matrix, all elements in the first cluster set ($CSet_1$) are extracted and analyzed. A set of normalization processes has been applied to the element names in order to obtain non-repeating terms in the cluster set. The normalization process has the following steps:
  - Tokenization. Each element name inside the cluster is parsed into a set of tokens using delimiters, such as punctuation, uppercase or special symbols, etc. For instance, *deptDB* $\rightarrow$ {*dept, DB*}.
  - Elimination. Tokens that are neither letters nor digits are eliminated.

  We have thus created the term vector by collecting the names of the nodes. As a next step, the term-document matrix, A, is initially created with each matrix cell representing the number of times the associated node name appears in the indicated cluster document, *line 2.* Finally, we apply the log-entropy weighting function, Eq. 1 on each entry in the document matrix, *line 3.* Thus, the term-document matrix is ready for the next stage.
  It should be noted that other normalization techniques are needed, such as the expansion method, especially when schema element names are too short. *dno* and *pid* are examples that are needed to be expanded.
– *Applying singular value decomposition.* To construct a semantic space wherein the names of the nodes in the $CSet_1$ and clusters that are closely associated are placed near one another, we apply the singular value decomposition technique.

Algorithm 1. Similar clustering determination

Require: Two sets of clusters, $CSet_1 = \{C_{11}, C_{12}, ..., C_{1n}\}$ and $CSet_2 = \{C_{21}, C_{22}, ..., C_{2m}\}$

Ensure: A set of similar clusters, $Sim\_Clust = \{(C_{1i}, C_{2j}) | C_{1i} \in CSet_1, C_{2j} \in CSet_2\}$

{// **Step 1: Preparation**}

1: $Sim\_Clust \Leftarrow \emptyset$;
2: $A \Leftarrow analysis(CSet_1)$;
3: Compute for each entry in A :a_{ij}
$a_{ij} \Leftarrow (1 + \sum_j \frac{P_{ij} \log(P_{ij})}{\log(n)}) \times \log(tf_{ij} + 1)$

{// **Step 2: Singular Value Decomposition & reduction**}

4: Apply SVD to $A : A = USV^T$
5: Dimensionality reduction: $A_k = U_k S_k V_k^T$

{// **Step 3: Query incorporating and folding**}

6: $Q \Leftarrow analysis(CSet_2)$;
7: $Q_k \Leftarrow Q^T U_k S_k^{-1}$;

{// **Step 4: Similarity calculating and ranking**}

8: **for** $column_j \in Q_k$ **do**
9: $q_j \Leftarrow Q_k(j)$;
10: **for** $column_i \in A_k$ **do**
11: $d_i \Leftarrow A_k(i)$;
12: $simMat[i][j] = sim(q_j, d_i)$;
13: **if** $simMat[i][j] \geq threshold$ **then**
14: $Sim\_Clust.put(C_{1i}, C_{2j})$
15: **end if**
16: **end for**
17: **end for**

The technique factorizes a term-document matrix into its left singular vectors, right singular vectors, and singular values, *line 4*. Each node name within the cluster set is now represented by a singular vector via matrix U. Additionally, each cluster is represented by a singular vector via matrix V.

- *Reduction.* To reduce the noise and redundancy, LSI uses a truncated SVD, *line 5*, which consists in retaining only the largest k singular values and deleting the remaining ones which are smaller and thus considered unimportant. The columns corresponding to the small singular values are also removed from U and V. So, SVD allows the arrangement of the space to reflect the major associative patterns in the data, and ignore the smaller, less important influences. As a result, terms that do not actually appear in a document may still up close to the document, if that is consistent with the major patterns of association in the data.
- *Folding.* The following step is to prepare a set of clusters in the second cluster set $CSet_2$. Each cluster is treated as a user query. First, we analyze the element names of each cluster and we apply the same normalization process applied before on elements of the first cluster set to the second cluster set elements.

The query is then treated as an ordinary document and hence it should be put with new coordinates in the reduced $k-dimensional$ space, *lines 6&7*.

– *Calculating similarities*. Now, the two cluster sets have been prepared for comparison: one as a set of vectors via the matrix V, the second as a set of vector via the query matrix Q. Each vector in the two matrices represents a cluster. The current task is to compute the similarity between two sets of clusters and select similar clusters. As shown in the algorithm, *lines 8 to 17*, a query vector is extracted and compared with all the other cluster set elements. A cosine measure is used to compute the similarity between two vectors. If the computed similarity exceeds a specified threshold, the two clusters constitute a similar cluster pair to be then added to the final result, Sim_Clust.

The computed similarities between cluster pairs of the two schemas are used to construct a so-called cluster similarity matrix, *line 12*. If the computed similarity between each two clusters exceeds a specific threshold, the two clusters are put in the similar cluster set, *lines 13&14*.

4.4 Walk-Through Example

We provide an example that elaborates the proposed algorithms and gives more details about how to determine similar clusters. In this example, we use two schema trees illustrated in Figs. 2 and 3. We formulate the problem in this example as follows: given two cluster sets $CSet_1 = \{C_{11}, C_{12}\}$ and $CSet_2 = \{C_{21}\}$ shown in Fig. 4, identify similar clusters across the two cluster sets.

– Step 1: We select the cluster set of larger number of clusters, $CSet_1$, to construct the term-document matrix. After applying the normalization process on element names, we get the matrix A, as shown below. Each element in the matrix shows the number of occurrences of each term in the associated cluster (document). After getting the matrix A, we apply the log-entropy weighting scheme to get $A_{entropy}$.

$$\begin{bmatrix} dept \\ DB \\ dno \\ dname \\ country \\ emps \\ emp \\ eid \\ ename \\ function \\ grant \\ amount \\ pid \\ project \\ pname \\ year \end{bmatrix} A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 2 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 2 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \Rightarrow A_{entropy} = \begin{bmatrix} 0 & 0 \\ 0 & 0.69 \\ 0.09 & 0.06 \\ 0.69 & 0 \\ 0.69 & 0 \\ 0.69 & 0 \\ 0.69 & 0 \\ 0.69 & 0 \\ 0.69 & 0 \\ 0.69 & 0 \\ 0 & 0.69 \\ 0 & 0.69 \\ 0 & 1.1 \\ 0 & 0.69 \\ 0 & 0.69 \\ 0 & 0.69 \end{bmatrix}$$

– Step 2: The log-entropy matrix is then decomposed into three matrices as given by Eq. 2, where U is an 16×16 orthogonal matrix, S is an 16×2 diagonal matrix, and V is an 2×2 orthogonal matrix. Since S has only two eigenvalues, the SVD method should reserve only two columns in U and neglect the rest, and S should be limited only to two rows, and its dimensions should be truncated. After applying the SVD scheme, we get the following V and S matrices (U is not presented to save space since it is an 16×2 matrix).

$$V = \begin{bmatrix} -0.007 & -1 \\ -1 & 0.007 \end{bmatrix} S = \begin{bmatrix} 2.02 & 0 \\ 0 & 1.84 \end{bmatrix}$$

Low rank approximation to A, called A_k, can be created through the truncated SVD, via Eq. 3. For truncation, we assume to truncate 98 % of the singular values. In this example, there is no truncation and the matrices remain the same.

– Step 3: Incorporating the query, we incorporate the clusters of the *orgDB* schema into the new dimensional space created by SVD and its reduced form processes. The schema is partitioned into one cluster according to the applied threshold. Hence, we have one query which is analyzed and presented as an $m \times 1$ matrix. This matrix is projected onto the reduced term-document space via Eq. 4. The new coordinates of this query are represented in vector q, where $q = [-0.131 - 0.262]$.

– Step 4: The final step is applying cosine similarity function and ranking the documents as follows.

$sim(C_{11}, C_{21}) = sim(d_1, q) = 0.897$ and $sim(C_{12}, C_{21}) = sim(d_2, q) = 0.442$

Solving the same example using VSM yields the following results:

$sim_{VSM}(C_{11}, C_{21}) = 0.373$ and $sim_{VSM}(C_{12}, C_{21}) = 0.224$. If we set a threshold value of 0.3, then we get $Sim\_Clust(C_{11}, C_{21}), (C_{12}, C_{21})$ using the LSI-based method, while the similar cluster set contains only one similar cluster using the VSM-based method. From this example, it has been shown that the computed similarities by LSI are higher than those computed by VSM due to the ability of LSI to correlate semantically related terms that are latent in the collection of documents. Furthermore, the documents as well as the query vectors are represented by the new dimensions with semantic correlation between them.

4.5 Match Similar Clusters

Once settling on the similar clusters of the two schemas, the next step is to fully match similar clusters to obtain the correspondences between their elements. Each pair of the similar clusters represents an individual match task that is independently solved. Match results of these individual tasks are then combined to a single mapping, which represents the final match result. Since the matching part is not the main focus on the paper, we employ both name and type similarity measures to quantify the similarity between two similar cluster elements [3]. We simply introduce the two similarity measures (interested readers can refer to [3] for more details).

– *Name similarity measure:* Element names are considered important information sources for schema matching. Each element name should be normalized into a set of tokens and a set of string similarity measures can be applied on these tokens. Based on results presented in [3], we employ three string-based measures, namely, *Levenstein distance*, *N-gram distance*, and *Jaro similarity* [11].
– *Type similarity measure:* Although the element name is considered a necessary source for determining the element similarity, the consideration for other features also plays a different role. The element data type is another schema information that makes a contribution in determining the element similarity. The type similarity measure aids to prune some of the false positive matches produced from the name similarity measure. XML schema data types are divided into 12 communal types[4]. Therefore, in this paper, we build a data type similarity table. We calculate the similarity value for each data type pair based on the constraining facets of XML schema[5]. For more details, refer to [34].

Once the similarity between elements from two similar clusters has been computed using the name and type matchers, a weighted sum function is used to aggregate these similarity values. Elements with similarity values higher than a predefined threshold are selected as partial matching results. Finally, partial results from all similar clusters are combined to produce the final matching result.

5 Experimental Evaluation

To evaluate the effectiveness of the proposed approach, we conducted a set of experiments utilizing real-world schemas and ontologies. We ran all our experiments on 2.67 GHz Intel (R) Core i5 processor with 4 GB RAM running Windows 7. The proposed approach has been developed and implemented in Java.

5.1 Data Set

We collected data sets from different domains with different characteristics, as shown in Table 1. The table illustrates that the collected schemas are from 8 different domains[6] with different sizes ranging from small to large schemas. Within each domain, we use two schemas in order to apply the proposed approach. We choose these data sets to demonstrate the applicability of our approach to different data sources having different characteristics. More details about data sets in Table 1 can be found in [14,29].

[4] http://www.w3.org/TR/xmlschema-2/.
[5] XML Schema - Data Types Quick Reference, http://www.xml.dvint.com/.
[6] http://queens.db.toronto.edu/project/clio/index.php#testschemas.

Table 1. Data set specification.

Domain	Tested sources	No. of elements
Spicy	deptDB/orgDB	19/20
University	Uni1/Uni2	11/11
Web	Yahoo/ebay	37/37
TPC_ H	TPC_H1/TPC_H2	43/17
Finance	finan1/finan2	14/14
GeneX	GeneX1/GeneX2	75/85
Mondial	Mondial1/Mondail2	117/108
PO(large)	OpenTran_Invoice/OpenTran_Order	1113/1162

5.2 Evaluation Criteria

In our implementation, we consider two levels of evaluations, which can help answering the following two questions:

- Which is the better technique to determine similar clusters; LSI-based or VSM-based?
- What is the effect of both LSI-based and VSM-based techniques on the overall matching quality?

In order to answer these questions, we use the same criteria used in literature in terms of *precision*, *recall*, and *F-measure* [8]. In general, precision P can be defined as the degree of correctness of the result. In answering the first question, the result means the similar clusters, while in the second question it means match results (i.e., correspondences). It measures the ratio of correctly identified results (true positives, t_P) over the total number of identified results (true positives plus false positives f_P). It can be computed as: $P = \frac{t_P}{t_P + f_P}$. Recall, R, assesses the degree of completeness of the system. It measures the ratio of correctly identified results (true positives, t_P) over the total number of correct results (true positives plus false negatives f_N). It can be computed as: $R = \frac{t_P}{t_P + f_N}$.

However, neither precision nor recall alone can accurately assess the matching quality [13]. Precision evaluates the post-match effort that is needed to remove false positives, while recall evaluates the post-match effort that is needed to add true negatives to the final match result. Hence, it is necessary to consider a trade-off between them. There are several methods to handle such a trade-off, one of them is to combine both measures. The mostly used combined measure is *F-measure*. *F-measure* is the weighted harmonic mean of precision and recall. The traditional F-measure can be defined as:

$$F\text{-}measure = 2 \times \frac{P \times R}{P + R} \tag{5}$$

5.3 Experimental Results

LSI-Based Approach Quality. We conducted two sets of experiments to validate the performance of the proposed approach and to answer the mentioned questions. The first set is devoted to answer the question "whether LSI-based or VSM-based technique is better in determining similar clusters in the context of partitioning-based schema matching". We validated the proposed approach using XML schemas illustrated in Table 1. Each XML schema is parsed and represented as a schema tree. The clustering-based approach, in [2], is applied to partition each schema tree into a set of clusters. To determine similar clusters among two sets of clusters, we applied both our LSI-based approach and the VSM-based approach [2]. The elements of cluster similarity matrix are ranked according to their similarity to each other and the similar clusters have been selected when their similarities exceed a predefined threshold. Results are summarized in Fig. 5.

Results represented in Fig. 5 can be classified into three main categories. The first one considering the University (Spicy and Finance which are not drawn) and TPC_H schemas, as shown in Fig. 5(a,b), illustrates that F-measure has its best values at low threshold and it decreases with increasing threshold values. The LSI-based method has an F-measure of nearly 1 over threshold values ranging between 0 and 0.5, and then the F-measure decreases to reach zero at threshold of 0.6 (for University) and 0.9 (for TPC). However, the VSM-based method has its best value at only two threshold values and it decreases to reach zero at a threshold value of 0.4 (for both schemas). This can be explained as both the

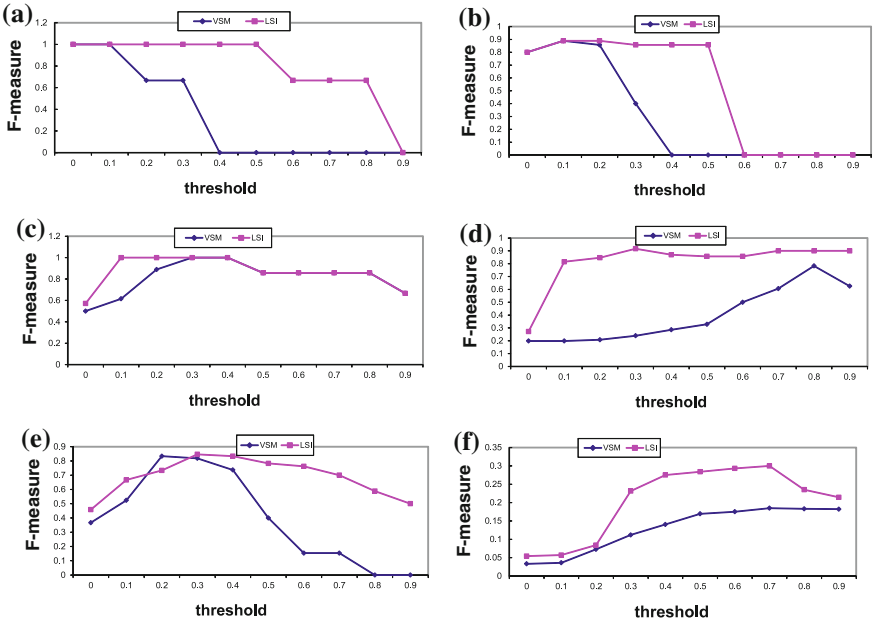

Fig. 5. Similar clusters quality

University (Spicy) and TPC_H have higher name heterogeneities, which make the VSM-based method fail to determine the correct similar clusters.

The second category considers the Web and Mondial schemas, as shown in Fig. 5(c,d). These schemas have lower name heterogeneities, which give the VSM-based method the chance to correctly identify similar clusters. The figures also show that as the threshold increases, the F-measure increases. However, the LSI-based method has higher F-measure values than the VSM-based method, especially for the Mondial schema.

The third category represents the PO schemas. It is the most difficult match task, since these schemas are highly heterogeneous. Therefore, the quality of similar cluster determination is lower compared to the other cases. However, the LSI-based method keeps higher F-measure values than the VSM-based method.

To sum up, Fig. 5 shows that the LSI-based technique outperforms the VSM-based technique across tested schemas. The figure also illustrates the capability of the LSI-based method to cover the hidden semantic relationships between cluster elements, which results in more accurate and quality similar clusters determination.

Furthermore, we compared the LSI-based and the VSM-based method with respect to the number of produced similar clusters. Based on results reported in Fig. 5, we found that the "best" similar clusters occur at different threshold values. Therefore, we decide to select a suitable threshold value to conduct this comparison. To this end, we select the similar clusters produced at a threshold value of 0.3. Results are reported in Table 2. The table represents the number of generated clusters after applying the clustering algorithm, the number of real similar clusters, the number of similar cluster (both correct and total numbers) generated by both techniques and its quality (F-measure). The table also verifies the results presented in Fig. 5. Table 2 illustrates that the LSI-based method outperforms the VSM-based method. This is can be explained due to the ability

Table 2. Comparison between LSI-based and VSM-based techniques at threshold of 0.3

Domain	No. of clusters	No. of real similar clusters	LSI-based		VSM-based	
			No. similar cluster correct/total	F-measure	No. similar cluster correct/total	F-measure
Spicy	2/1	2	2/2	1.0	1/1	0.67
University	1/2	2	2/2	1.0	1/1	0.67
Web	4/3	4	4/4	1.0	4/4	1.0
TPC_H	6/1	4	3/3	0.867	1/1	0.4
Finance	1/2	2	2/2	1.0	2/2	1.0
GeneX	10/8	12	11/14	0.85	9/10	0.8
Mondial	10/10	11	11/13	0.92	11/81	0.23
PO(large)	57/56	80/112	100/112			

of the LSI-based method to discover the hidden semantic relationships between schema element names.

Effect of LSI-Based on Matching Quality. The second set of experiments has been conducted to study the effect of both LSI-based and VSM-based methods on the matching quality. After selecting the "best" similar clusters, we first applied the name matcher on each matching task using data sets shown in Table 1. We then applied both the name and type similarity measures on the same matching task. Each task produces a subset of the match result. These subsets are then combined to generate the final match result. The final match result is evaluated using evaluation criteria, including precision, recall, and F-measure. Results for the matching quality are reported in Fig. 6.

The figures show, in general, that the LSI-based method has higher matching quality than the VSM-based method. This fact can be observed for the *University, Spicy, TPC_H*, and *PO* schemas. This can be clarified as these schemas have a high degree of semantic heterogeneity, and the LSI-based method has the ability to discover hidden semantic relationships between schema elements. However, the *Web, Finance* and *Genex* have less degree of heterogeneity, which results in nearly equal matching quality by both methods. In the case of the *Mondial* schema, which contains nearly no semantic heterogeneity, a large number of false positives are produced by the VSM-based method. Therefore, the LSI-based method produces higher matching quality than the VSM-based method w.r.t. the *Mondial* schema. It should be noted that the name matcher is more effective than the type similarity measure, and using the type measure makes a slight improvement in the matching quality.

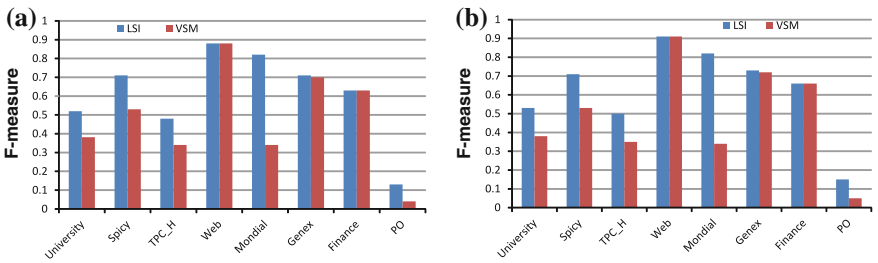

Fig. 6. Match quality comparison.

6 Conclusions

Partitioning-based techniques have become well-known approaches to match large schemas and ontologies. It has been proven that they improve the matching efficiency, however, they do not guarantee the matching quality. Identifying similar partitions of two schema trees is a crucial step before the matching process. To this end, in this paper, we introduced a new approach to cope with the

problem of similar cluster determinations in the context of matching large-scale schemas. The proposed approach captures the features introduced by the Latent semantic indexing scheme to discover hidden semantic relationships between two sets of clusters. We in particular developed a matching framework focusing on the similar cluster determination step. Input schemas are first parsed and represented internally as schema trees to make the matching framework more generic. We then applied a clustering algorithm to partition each schema tree into a set of clusters. Further, we introduced a LSI-based algorithm to identify and determine similar clusters. To validate the performance of the proposed approach, we conducted a set of experiments utilizing different data sets comparing it with the classical vector space model (VSM)-based approach. The results proved that the LSI-based method outperforms the VSM-based method in determining the most similar clusters. It has the ability to discover hidden semantic relationships between schemas' elements. Therefore, the LSI-based method produces better matching quality than the VSM-based method. In future work we plan to extend the framework to explore the effect of the LSI-based method on matching efficiency. We need to validate some optimization techniques to enhance the LSI-based method.

Acknowledgments. This paper is a revised and extended version of the paper presented in [26]. A. Algergawy partially worked on this paper while at Magdeburg University.

References

1. Abiteboul, S., Suciu, D., Buneman, P.: Data on the Web: From Relations to Semistructed Data and XML. Morgan Kaufmann, San Francisco (2000)
2. Algergawy, A., Massmann, S., Rahm, E.: A clustering-based approach for large-scale ontology matching. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 415–428. Springer, Heidelberg (2011)
3. Algergawy, A., Nayak, R., Saake, G.: Element similarity measures in XML schema matching. Inf. Sci. **180**(24), 4975–4998 (2010)
4. Algergawy, A., Nayak, R., Siegmund, N., Köppen, V., Saake, G.: Combining schema and level-based matching for web service discovery. In: Benatallah, B., Casati, F., Kappel, G., Rossi, G. (eds.) ICWE 2010. LNCS, vol. 6189, pp. 114–128. Springer, Heidelberg (2010)
5. Algergawy, A., Schallehn, E., Saake, G.: Improving XML schema matching using Prüfer sequences. DKE **68**(8), 728–747 (2009)
6. Aslan, G., McLeod, D.: Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution. VLDB J. **8**(2), 120–132 (1999)
7. Bellahsene, Z., Bonifati, A., Rahm. E.: Schema Matching and Mapping. Springer, Heidelberg (2011).
8. Berry, M.W., Drmac, Z., Jessup, E.R.: Matrices, vector spaces, and information retrieval. SIAM Rev. **41**(2), 335–362 (1999)
9. Bonifati, A., Mecca, G., Pappalardo, A., Raunich, S., Summa, G.: Schema mapping verification: the spicy way. In: EDBT 2008, France, pp. 85–96 (2008)

10. Chiticariu, L., Hernández, M.A., Kolaitis, P.G., Popa, L.: Semi-automatic schema integration in Clio. In: VLDB'07, pp. 1326–1329 (2007)
11. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: IIWeb, pp. 73–78 (2003)
12. Deerwester, S., Dumais, S.T., Harshman, R.: Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. **41**, 391–407 (1990)
13. Do, H.H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: The 2nd International Workshop on Web Databases (2002)
14. Do, H.H., Rahm, E.: Matching large schemas: approaches and evaluation. Inf. Syst. **32**(6), 857–885 (2007)
15. Doan, A., Halevy, A.: Semantic integration research in the database community: a brief survey. AAAI AI Mag. **25**(1), 83–94 (2005)
16. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann, San Francisco (2012)
17. Ehrig, M., Staab, S.: QOM – quick ontology mapping. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 683–697. Springer, Heidelberg (2004)
18. Halevy, A.Y., Ives, Z.G., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: 19th International Conference on Data Engineering, pp. 505–516 (2003)
19. Hamdi, F., Safar, B., Reynaud, C., Zargayouna, H.: Alignment-based partitioning of large-scale ontologies. In: Guillet, F., Ritschard, G., Zighed, D.A., Briand, H. (eds.) Advances in Knowledge Discovery and Management. SCI, vol. 292, pp. 251–269. Springer, Heidelberg (2010)
20. Hao, Y., Zhang, Y.: Web services discovery based on schema matching. In: ACSC 2007, pp. 107–113 (2007)
21. Hu, W., Qu, Y., Cheng, G.: Matching large ontologies: a divide-and-conquer approach. DKE **67**, 140–160 (2008)
22. Landauer, T.: Handbook of Latent Semantic Analysis. Lawrence Erlbaum, Mahwah (2007)
23. Lee, D., Chu, W.W.: Comparative analysis of six XML schema languages. SIGMOD Rec. **9**(3), 76–87 (2000)
24. Lee, M.L., Yang, L.H., Hsu, W., Yang, X.: Xclust: clustering XML schemas for effective integration. In: CIKM'02, pp. 63–74 (2002)
25. Manning, C.D., Raghavan, P., Schutze, H.: Introduction to Information Retrieval. Cambridge University Press, New York (2008)
26. Moawed, S., Algergawy, A., Sarhan, A., Eldosouky, A., Saake, G.: A latent semantic indexing-based approach to determine similar clusters in large-scale schema matching. In: Catania, B., et al. (eds.) New Trends in Databases and Information Systems. AISC, vol. 241, pp. 267–276. Springer, Heidelberg (2014)
27. Peukert, E., Berthold, H., Rahm, E.: Rewrite techniques for performance optimization of schema matching processes. In: EDBT, pp. 453–464 (2010)
28. Peukert, E., Eberius, J., Rahm, E.: A self-configuring schema matching system. In: 28th International Conference on Data Engineering (ICDE), 2012, pp. 306–317 (2012)
29. Peukert, E., Massmann, S., Konig, K.: Comparing similarity combination methods for schema matching. In: GI-Workshop, pp. 692–701 (2010)
30. Rahm, E.: Towards large-scale schema and ontology matching. In: Bellahsene, Z., Bonifati, A., Rahm, E. (eds.) Schema Matching and Mapping. Data-Centric Systems and Applications, pp. 3–27. Springer, Heidelberg (2011)

31. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
32. Seddiquia, M.H., Aono, M.: An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. Web Semant. **7**(4), 344–356 (2009)
33. Shvaiko, P., Euzenat, J.: Ontology matching: state of the art and future challenges. IEEE Trans. Knowl. Data Eng. **25**(1), 158–176 (2013)
34. Thuy, P.: Hybrid similarity measure for XML data integration and transformation. Ph.D. thesis, Seoul, Korea (2012)
35. Wang, Z., Wang, Y., Zhang, S.-S., Shen, G., Du, T.: Matching large scale ontology effectively. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 99–105. Springer, Heidelberg (2006)
36. Zhong, Q., Li, H., Li, J., Xie, G.T., Tang, J., Zhou, L., Pan, Y.: A Gauss function based approach for unbalanced ontology matching. In: ACM SIGMOD International Conference on Management of Data, (SIGMOD 2009), pp. 669–680 (2009)