A Clustering-based Approach For Large-scale Ontology Matching

Alsayed Algergawy, Sabine Massmann and Erhard Rahm

Department of Computer Science, University of Leipzig
`{algergawy,massmann,rahm@informatik.uni-leipzig.de}`

Abstract. Schema and ontology matching have attracted a great deal of interest among researchers. Despite the advances achieved, the large matching problem still presents a real challenge, such as it is a time-consuming and memory-intensive process. We therefore propose a scalable, clustering-based matching approach that breaks up the large matching problem into smaller matching problems. In particular, we first introduce a structure-based clustering approach to partition each schema graph into a set of disjoint subgraphs (clusters). Then, we propose a new measure that efficiently determines similar clusters between every two sets of clusters to obtain a set of small matching tasks. Finally, we adopt the matching prototype COMA++ to solve individual matching tasks and combine their results. The experimental analysis reveals that the proposed method permits encouraging and significant improvements.

1 Introduction

There is a proliferation of schema- and ontology-based web data sources using models and languages, such as XML, RDF, and OWL [1]. Identifying semantic correspondences among such heterogeneous data sources and their metadata models (schemas and ontologies) is the biggest obstacle for making these data sources interoperable. The process of identifying these correspondences across different metadata models is called *schema matching* or *ontology matching*.

For its importance, a myriad of matching algorithms has been proposed and a large number of matching systems have been developed (see e.g., [17, 4, 2] for surveys). Unfortunately, most of these systems severely lack performance when dealing with large matching problems. The results of previous OAEI contests [1] show that more than half of the matching systems couldn't match large ontologies in less than one hour [12]. Consequently, several approaches have been proposed to address the problem of matching two large schemas, such as *MOM* [20], *COMA++* [6] and *Falcon* [11]. As we will further discuss in Section 2, the current approaches to partition-based matching have several limitations and the design space for such solutions has not yet sufficiently been explored.

In this paper we address two of these limitations. The first issue is *partition identification*. Some solutions, such as Falcon, are specific to certain ontology

[1] http://www.ontologymatching.org

languages and cannot be applied to other data models. Other solutions, such as COMA++, use relatively simple heuristic rules to partition the input schemas resulting often in too few or too many partitions. The second issue is *determination of similar partitions*. Some solutions, such as COMA++, only use limited information about the partition (only the root node of the partition) to determine the similarity between partitions of the input schemas, which results in less matching quality. Other solutions such as Falcon, fully evaluate the input ontologies to assess the partition similarity that leads to higher response time.

To cope with these challenges and limitations, we therefore propose and evaluate a new, more efficient, partition-based matching strategy. The proposed approach shares the general procedure to match large ontologies with existing matching systems. However, the approach introduces new methodologies to overcome the observed limitations. In particular, we make the following contributions:

– We propose a new clustering-based approach to cope with the large matching problem. The approach is generic. Similar to the current implementation of COMA++ [6], we first represent input schemas and ontologies as directed acyclic graphs, called schema graphs. We further apply a structure-based clustering algorithm to partition each input ontology into a set of disjoint sub-graphs. We thus achieve that elements that are structurally similar are in the same cluster, while elements in different clusters are dissimilar.
– Given the two cluster sets of the input ontologies, we apply a light-weight similarity measure to efficiently assess the similarity between cluster pairs. To this end, we represent each cluster as a *cluster document* and use of both the Vector Space Model and TF-IDF to determine the similarity between cluster documents. Having similar clusters, we adopt a standard match tool such as COMA++ to fully match elements inside similar clusters.
– We experimentally evaluate the efficiency and match quality of the proposed approach for different real-world schemas and ontologies. The resulting insights should be helpful for the development and evaluation of future match systems.

Section 2 discusses related work. We then introduce the basic definitions in Section 3. Sections 4 and 5 present the new approaches for structure-based clustering and identifying similar pairs of clusters. Section 6 presents experimental results. We conclude in Section 7.

2 Related work

To cope with matching two large ontologies, several techniques can be used, such as reduction of search space, parallel matching, and self-tuning [16]. Reducing the match search space aims to limit the number of element comparisons either by early pruning dissimilar element pairs [7, 14] or by partitioning the two ontologies [18, 6, 11, 19, 10]. *Quick ontology mapping* [7] was one of the first approaches that considers both matching quality and run-time complexity. It first determines match candidates based on element labels and evaluates structure

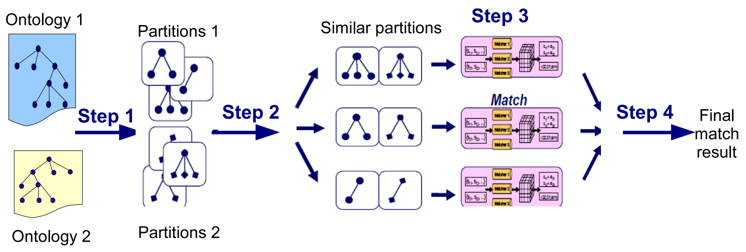


Fig. 1: Steps of matching two large ontologies.

properties only for the most similar pairs from the first step. The approach proposed in [14] uses a set of filters within the matching process to prune dissimilar element pairs from intermediate match results.

Similar to our approach, partition-based matching aims at partitioning input ontologies/schemas in such a way that each partition of the first ontology has to be matched only with a subset of the second ontology. As shown in Fig. 1, the skeleton of partition-based matching involves four main steps. Step 1, *partition identification*, partitions the input schemas into a set of disjoint clusters. The second step, *determination of similar partitions*, is devoted to identifying similar partitions. Once settling on similar partitions (called fragments in COMA++ [6] or blocks in Falcon [11]), in Step 3, normal matching algorithms can be used to determine local correspondences between similar partitions. Finally, from these local correspondences, Step 4 is to construct the final match result. COMA++ implements one of the first approaches for partition-based matching. Its approach called *fragment matching* [6] has the four general steps similar of Fig. 1. Fragment matching first partitions two input schemas into two set of fragments, which are then compared with each other to identify the most similar fragments in the two sets worth to be fully matched later. Both *Falcon* [11] and *Anchor-Flood* [19] focus on matching (OWL) ontologies but do not support matching of XML schemas or relational schemas. The Falcon system uses a specific structure-based clustering technique to partition entities of ontology into blocks. Matching is then applied to the most similar blocks from the two ontologies. The Anchor-Flood system follows a dynamic partitioning technique. It starts off with an anchor, a pair of look-alike concepts from each ontology, gradually exploring concepts by collecting neighboring concepts until no further matches are found or all concepts are processed. The partitions are located around the anchors and their size depends on the continued success of finding match partners of the considered concepts. Further details about different techniques of large-scale matching can be found in [16].

3 Preliminaries

We first present definitions and basic concepts used throughout the paper.

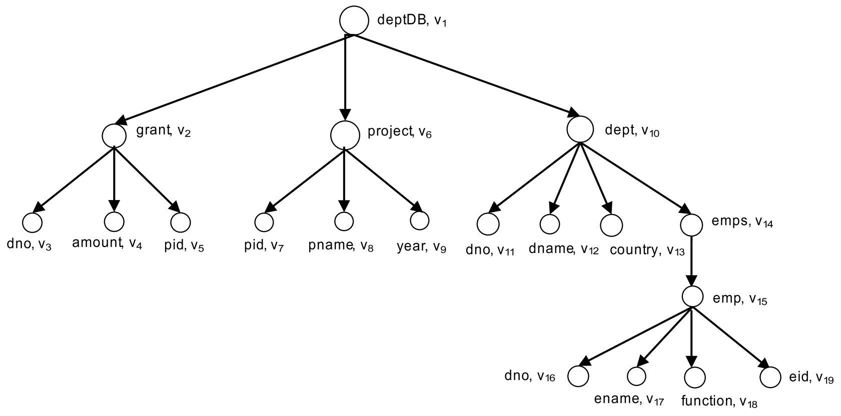


Fig. 2: Schema graph representation.

Schema graph. In order to make the proposed approach generic, we represent input schemas (e.g., XML schemas) and ontologies as labeled directed acyclic graphs, called *schema graphs* (SG).

Definition 1. *A schema graph is a rooted node-labeled directed acyclic graph. It is represented as a 3-tuple* (V, E, Lab_v)*, where:* $V = \{r, v_2, ..., v_n\}$ *is a finite set of nodes, each of them is uniquely identified by an object identifier (OID), where* r *is the schema graph root node.* $E = \{(v_i, v_j)|v_i, v_j \in V\}$ *is a finite set of edges.* Lab_v *is a finite set of node labels. These labels are strings for describing the properties of the element and attribute nodes, such as name and data type.*

Fig. 2 represents the schema graph representation of an XML schema taken from [3]. *DeptDB* represents information about departments with their employees and grants, as well as the projects for which grants are awarded. The figure shows that each node is associated with the node name and the node identifier. For example, the node v_1 has the name *deptDB*.

Node Context. The context of a node in a schema graph is represented by its descendants, ancestors, and siblings. The descendants of the node include both its immediate children and the leaves of the subgraphs rooted at the node. The immediate children reflect its basic structure, while the leaves reflect the node's content. Without loss of generality, to construct the context of a node, we consider descendants and ancestors of the node up to one level, i.e., the parents and the children elements, as well as the node itself. Formally, we introduce the definition of the node context ($\mathcal{C}$) as follows:

Definition 2. *Given a schema graph* $SG = (V, E, Lab_v)$*, the context of a node* $v \in V$ *is given by* $\mathcal{C}(v) = \{v_i | (v, v_i) \in E \cup (v_i, v) \in E \cup v\}$

For the schema graph in Fig. 2, $\mathcal{C}(v_6) = \{v_1, v_6, v_7, v_8, v_9\}$. We claim that the more contexts two nodes share, the higher their structural similarity is. We therefore define and use the following context-based similarity measure.

Definition 3. *Given two nodes v_i and $v_j \in SG$, the context similarity, σ, between them is computed using the node contexts as follows:*

$$\sigma(v_i, v_j) = \frac{|\mathcal{C}(v_i) \cap \mathcal{C}(v_j)|}{\sqrt{|\mathcal{C}(v_i)|.|\mathcal{C}(v_j)|}} \tag{1}$$

$|\mathcal{C}(v_i) \cap \mathcal{C}(v_j)|$ represents the number of common nodes between their contexts and $\sqrt{|\mathcal{C}(v_i)|.|\mathcal{C}(v_j)|}$ is the geometric mean of the two contexts'size used to normalize the value of the structure similarity. In fact, Eq.1 guarantees that the more common nodes the two nodes share, the higher context similarity they have. Furthermore, the equation shows that context similarity has several properties. Among them are: it is normalized, $0 \leq \sigma(v_i, v_j) \leq 1$, and symmetric, $\sigma(v_i, v_j) = \sigma(v_j, v_i)$.

Example 1. The node contexts of nodes v_2 , v_4 and v_6 are as follows: $\mathcal{C}(v_2) = \{v_1, v_2, v_3, v_4, v_5\}$, $\mathcal{C}(v_4) = \{v_2, v_4\}$ and $\mathcal{C}(v_6) = \{v_1, v_6, v_7, v_8, v_9\}$, respectively. The structure similarity between these nodes can be computed as follows: $\sigma(v_2, v_4) = 0.63$, $\sigma(v_2, v_6) = 0.2$, and $\sigma(v_4, v_6) = 0$.

4 Structure-based clustering

Our goal is to divide the schema graph into disjoint subgraphs in order to facilitate matching large ontologies represented as schema graphs. 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. In the following, we present a clustering algorithm based on the introduced context similarity so that structurally similar nodes are placed in the same cluster while the nodes of different clusters are structurally dissimilar. We first describe how to use the computed structure similarity to construct so-called *links.* After this we introduce the proposed clustering algorithm.

To avoid the repeated calculation of intra-ontology element similarities for clustering, we predetermine and store the structural similarity between selected node elements as so-called links. In particular, we are interested in the following set of element pairs for which the context similarity exceeds a predefined threshold, *th*:

$$links = \{L_i | L_i = (v_i, v_j, \sigma(v_i, v_j))\, s.t.\, \sigma(v_i, v_j) \geq th\,, v_i, v_j \in SG\} \tag{2}$$

Using this set of items (*links*) we construct a *links* hash table. Given a schema graph SG with n nodes, the worst case each node may be compared with $n-1$ other nodes resulting in quadratic number of comparisons. However, as shown in Example 1, $\sigma(v_4, v_6) = 0$ since the two nodes have no common nodes in their contexts. Therefore, we limit the comparison of a node with the set of neighboring nodes to achieve a linear number of comparisons. 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.

Algorithm 1 Clustering algorithm

```
Require: A schema graph, $SG = (V, E, Lab_v)$
Ensure: A dendrogram, a hierarchy of clusters
    {Stage 1: Preparation}
 1: $ClusterSet \Leftarrow \phi$, $Dendro \Leftarrow \phi$;
 2: $Nodes[] \Leftarrow SG.getNodes()$;
 3: $links[] \Leftarrow constructLink(SG)$;
    {Stage 2: Cluster initialization}
 4: for $n_i \in Nodes[]$ do
 5:    $Cluster\, C \Leftarrow new\, Cluster(n_i)$;
 6:    $ClusterSet.add(C)$;
 7: end for
 8: $Dendro.addLevel(ClusterSet)$;
    {Stage 3: Cluster hierarchy construction}
 9: $dist \Leftarrow 1$;
10: while $ClusterSet.size() > 1$ do
11:    $k \Leftarrow ClusterSet.size()$;
12:    $ClusterSet \Leftarrow mergeCluster(dist)$
13:    if $k > ClusterSet.size()$ then
14:       $Dendro.addLevel(ClusterSet)$;
15:       $computeIntraSim(ClusterSet)$;
16:       $k \Leftarrow ClusterSet.size()$;
17:    end if
18:    if $noMoreMerge()$ then
19:       $break$;
20:    end if
21:    $dist \Leftarrow dist + \delta$;
22: end while
    {Stage 4: Best cluster set selection}
23: return $Dendro.getBestCluster(BestLevel)$;
```

The clustering algorithm presented in this paper is an agglomerative hierarchical algorithm mainly extended from the SCAN approach [21], which is a very scalable algorithm in the area of network clustering. The algorithm produces a tree representing the hierarchy of clusters in a bottom-up fashion, called *dendrogram*. Initially, each node represents its own single-member cluster. The algorithm iteratively merges nodes of a schema graph in descending order of structure similarity to build the hierarchy. As shown in Algorithm 1, the proposed clustering algorithm proceeds in four stages as follows.

- *Preparation.* The algorithm accepts the schema graph, SG, to be clustered and prepares it for the next stages. The stage starts by initializing the output set of clusters (*ClusterSet*) and the cluster hierarchy (*Dendro*), *line* 1. Then, the algorithm proceeds to extract schema graph nodes, elements to be clustered, *line* 2, and constructs the *links* hash table.
- *Cluster initialization.* The initialization stage constructs the bottom level of the cluster hierarchy. Each node represents its own cluster resulting into n clusters in the cluster set (*ClusterSet*), *lines* 4 *to* 7. Once getting the initial cluster set, the bottom level of the hierarchy is added to the dendrogram, *line* 8.
- *Cluster hierarchy construction.* This is the main stage of the clustering algorithm and is dedicated to construct the cluster hierarchy. It first initializes

the distance between levels of hierarchy with 1, *line* 9. The algorithm iteratively merges clusters at a certain level until either the number of clusters reaches 1 or there is no possibility to merge more clusters. We keep the current size of the cluster set in variable k, *lines* 11& 16. If the number of clusters after merging is changed, *line* 13, the new cluster set is added to the cluster hierarchy at the specified level. Furthermore, as we will explain later, the intra-cluster similarity is computed and the k value is updated. After that the algorithm checks if there is a possibility to further merge clusters and finally updates the distance for the current hierarchy level.

– *Best cluster set selection.* The task of the final stage is to select the best cluster set. Each level in the dendrogram is associated with a value that represents the average value of intra-cluster similarities of clusters at that level. Therefore, the algorithm returns the cluster set at the level with the best value, *line* 23.

In the following we give more details considering the two main operations in the clustering algorithm: *cluster merging* and *intra-cluster similarity computation.*

Cluster merging. Once obtaining the first (bottom) level of the cluster hierarchy (*line* 8, Algorithm 1), we need to merge nodes into groups such that nodes in the same group are structurally similar while nodes in different groups are dissimilar. To this end, we call for a measure that quantifies relationship between individual clusters as well as a condition that should be satisfied to decide that nodes in two clusters have to be merged into one. To quantify the relationship between clusters, we rely on the pre-computed links. Having two clusters C_1 and C_2 containing k_1 and k_2 nodes (elements) respectively, the similarity between them can be expressed as the average context similarity of their elements. It can be represented as follows [9]:

$$Sim(C1, C2) = \frac{\sum_{i=1}^{k_1} \sum_{j=1}^{k_2} \sigma(v_{1i}, v_{2j})}{k_1 + k_2}. \tag{3}$$

where $\sigma(v_{1i}, v_{2j})$ is the context similarity between nodes $v_{1i} \in C_1$ and $v_{2j} \in C_2$ computed by Eq.1. Having this similarity between every cluster pair, a condition is required to decide if elements in the cluster pair should be merged. This condition has to reflect the level of the cluster hierarchy at which elements come together. Therefore, the introduced distance variable is used (*dist*, *line* 9). Elements of every cluster pair are combined when the similarity between the two clusters exceeds the predefined level similarity threshold ($1/dist$). The value of the level distance is then updated to reflect the nature of the next level (*line* 21, Algorithm 1).

It is worth noting that we add another condition in order to limit the merging process. Once two nodes in two different clusters have been merged into a new cluster, their links in the *links* hash table have been removed. The merging process stops when no more links are in the table, (*lines* 18& 19).

Intra-clustering similarity. The proposed clustering algorithm produces a cluster hierarchy (dendrogram) in a bottom-up fashion. The cluster solution does not give information regarding the cut-off level. Cutting off the hierarchical tree requires the selection of a suitable level. To select the best level, we compute intra-clustering similarity at each level (*line* 15, Algorithm 1).

The intra-clustering similarity measures the cohesion within a cluster, how similar nodes within a cluster are. This is computed by measuring the similarity between each pair of data within a cluster, and the intra-clustering similarity of a clustering solution is determined by averaging all computed similarities taking into account the number of nodes. Let at a certain level of the cluster hierarchy, L, be a number of clusters K of the n nodes of a schema graph. The intra-clustering similarity, $IntraSim$, at this level can be computed from the following formula:

$$IntraSim_L = \frac{\sum_{i=1}^{K} IntraSim(C_i)}{n}. \tag{4}$$

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

Example 2. Applying Algorithm 1 to the schema graph represented in Fig. 2, the cluster solution consists of two clusters $C_1 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ and $C_2 = \{v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, \}$ [2]. It should be noted that this cluster solution represents a reasonable solution in the sense that C_1 includes information about projects and grants for these projects, while C_2 represents information about departments and their employees.

Complexity analysis of clustering algorithm. Given a schema graph with n nodes, the algorithm contains four main stages. The worst case total cost of the preparation stage is $O(n(n-1)) = O(n^2)$ if every node in the schema graph has to be compared with all other $n-1$ nodes. However, on average, each node can only be compared with a set of nodes in the context of the node. With a typically constant average context size, this results in an average cost $O(n)$. The worst case total cost of the initialization stage is $O(n)$, and the time complexity of the final stage is $O(1)$. The cluster hierarchy construction stage initially iterates over n clusters, and then $n/2$ until either the number of clusters reduces to one or the merge condition, *line* 18, is satisfied. This results in an average time complexity of $O(n)$. Therefore, the time complexity of the clustering algorithm is $O(n)+O(n)+O(n)+O(1) = O(n)$. Results reported in the evaluation section verify and confirm this complexity.

[2] It should be noted that the cluster solution is based on the state of the schema graph. The state of schema graph represented in Fig. 2 is reduced. More information can be found in [6]

5 Determination of similar clusters

The goal of this step is to identify partitions (clusters) of the two schema graphs that are sufficiently similar to be worth matching in more detail. This aims at reducing the match overhead by not trying to find correspondences between unrelated partitions. The approach determines a *cluster document* per partition and makes use of the Vector Space Model (VSM) for computing the similarity between cluster documents.

To determine the similarity between clusters of different ontologies we can utilize different features of cluster elements, such as name, data type, cardinality constraints, etc. It has been verified that the node name is the most dominant feature [2]. Therefore, we construct a so-called cluster document based on the node name.

Definition 4. *Given a cluster J, the text document that contains the names of cluster elements is called a cluster document, CD_J.*

Adopting VSM provides the possibility to model document terms as elements of a vector space. Let $W_1, W_2, ..., W_t$ be the words (terms) in a cluster document. Let us suppose that there exists a unit length vector in the space corresponding to each word. We therefore can express each cluster document (CD_J) as a vector in terms of words as follows:

$$V_J = (W_{1J}, W_{2J}, ..., W_{tJ}) \tag{5}$$

where W_{iJ}s are real numbers reflecting the importance of word i in CD_J. Given a vector V_J representing the cluster document CD_J containing t words, the values of the vector elements can be computed using the $W_{iJ} = TF * IDF$ equation [5], where TF is the term frequency and IDF is the inverse document frequency.

To determine the cluster similarity, we propose the use of a light-weight similarity function based on the vector representation of the cluster document. Given two vectors V_{1I} and V_{2J} representing two cluster documents from two different ontologies, the cluster document similarity, $CDSim$, can be defined as the inner product (cosine) of the two vectors. It can be expressed as:

$$CDSim(CD_{1I}, CD_{2J}) = cos(V_{1I}, V_{2J}) = \frac{\sum_i^t W_{i1I}.W_{i2J}}{\sqrt{\sum_i^t (W_{i1I})^2 . \sum_i^t (W_{i2J})^2}} \tag{6}$$

where t is the size of the vectors. It should be noted that the equation yields a value of 0 when the elements of the two clusters do not have names in common, however, the similarity value becomes 1 when the elements of the two clusters have the same names. It is worth noting that representing cluster documents as vectors provides the possibility to utilize efficient similarity measures, such as the used one.

Now we are in a position to state the problem of similar clusters determination. Given two schema graphs SG_1 and SG_2 with n and m elements, and K and K' clusters, respectively. The problem is to identify the similar clusters

Table 1: Data set specification

Series	Tested schemas	No. matching tasks	min./max. No. of elements
1	PO (5 XDR)	10	27/74
	Spicy (4 XSD)	2	20/125
2	Webdirectory (4 OWL)	6	418/1132
3	Anatomy (2 OWL)	1	2746/3306

between the two sets. The computed similarities between cluster pairs of the two ontologies are used to construct a so-called cluster similarity matrix. Due to uncertainty inherent in ontology/schema matching, the best matching can actually be an unsuccessful choice [8]. To overcome this problem, the elements of the matrix are ranked according to their similarity to each other and the *top-k* [3] elements are selected from the ranked list.

Once settling on the similar clusters of the two ontologies, 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.

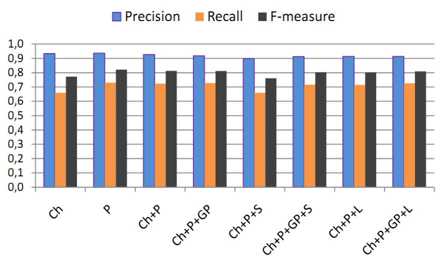
6 Experimental evaluation

In order to evaluate the performance of the clustering-based matching approach, we conducted a set of experiments utilizing real-world ontologies of different sizes. We aim to evaluate both the quality and the efficiency of the proposed approach. We ran all our experiments on a 2.66 GHz Intel(R) Xeon(R) processor with 4 GB RAM running Windows 7.

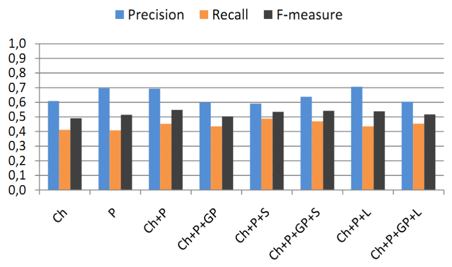
6.1 Data sets & evaluation criteria

Table 1 shows the characteristics of the test ontologies. In Series 1, we use five XML schemas for purchase orders (PO) taken from the COMA++ evaluation [6] and four XML schemas from [15]. In Series 2, we match four ontologies taken from the Web directory domain [13]. Series 3 contains a single match task taken from the OAEI initiative to match two large anatomy ontologies (AdultMouseAnatomy with 2,746 concepts vs. the anatomical part of the NCI Thesaurus with 3,306 concepts) [4]. We choose these data sets to demonstrate the applicability of our approach to different data sources represented in different models and having different characteristics. We performed the required matching tasks between schemas/ontologies within the similar domains with a total of 22 different matching tasks. More details about data sets in Table 1 can be found in [6, 13].

[3] k may equal 1, 2, or 3 based on the similarity value between clusters.
[4] http://www.ontologymatching.org/



(a) Series1: PO



(b) Series2: Web directory

Fig. 3: Matching quality.

To match elements within similar clusters, we used the *COMA++'s Allcontext* (a combination of Name, Path, Leaves, and Parents matchers) for match tasks of Series 1, the *Context* strategy (a Path matcher) for match tasks of Series 2, while the name matcher (without using synonyms) is used to perform the anatomy matching task. The threshold (*th*) used to construct *links* hash table is set to 0.15.

To measure the matching quality, we use the same criteria used in the literature, including *precision, recall* and *F-measure*. We call the execution time needed to perform the matching process including four steps of Fig. 1 the *response time*. We use it as a criterion of matching efficiency.

6.2 Experimental results

We present results for two sets of experiments. The first set is used to answer the following question: "Which node context shall be used in computing context similarity?". To this end, we made use of five different contexts, namely children (Ch), parents (P), grandparents (GP), siblings (S), and leaves (L) in eight different combinations. The experimental results on matching quality (precision, recall, and F-measure) are reported in Fig. 3.

Figs 3(a,b) give the matching quality for matching tasks of the PO and Web directory domains, respectively. For the PO domain, all the exploited contexts, except the child and Ch+P+S contexts, produce F-measure equal to or higher than 80%. It should be noted that both P and Ch+P contexts achieves the highest F-measure (82%), as shown in Fig. 3(a). Since schemas in the Web directory domain contain more heterogeneities and a simple matcher is used, the highest F-measure merely reaches 55% using also the Ch+P context, as shown in Fig. 3(b). This motivates and verifies our selection of the Ch+P context in computing context similarity. To verify this selection, we also investigated the generated number of partitions and the response time using of the ontologies in Series 2 (Web directory).

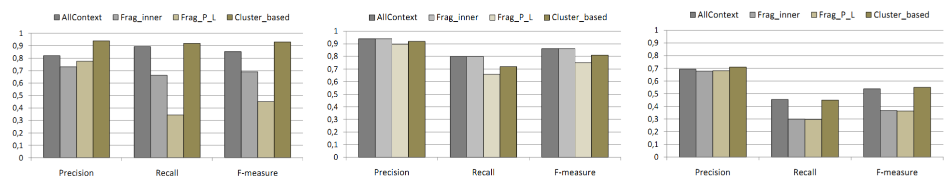
Table 2 illustrates the average number of partitions (clusters) generated using different node contexts. The Ch+P+S, P, and Ch contexts lead to mostly

Table 2: Average no. of partitions

Context	Ch	P	Ch+P	Ch+P+GP	Ch+P+S	Ch+P+S+GP	Ch+P+L	Ch+P+L+GP
Avg. partitions	38	62	22.8	19.8	112.7	28.9	25.6	19.1

higher number of partitions while the Ch+P context achieves a medium number of partitions that is largely in the same range for different match tasks. Furthermore, using this context performs on average faster than the other contexts. Hence, we conclude that the Ch+P context is most suitable for our clustering approach and we will use this choice in the next set of experiments.

The second set of experiments is used to compare the proposed approach with two current matching strategies in COMA++ and Falcon (for the anatomy match task). For COMA++ we consider the non-partitioned strategy (*AllContext*) and different *Fragment-based* strategies [6]. We choose different techniques to select fragments, such as *inner (Fragment_inner)* and *Parent_of_Leaves (Frag_P_L)*. The first selects inner nodes as roots of fragments, while parents of leaves are selected as roots of fragments in the second strategy. The experimental results are reported in Fig. 4.



(a) Series 1: Spicy (b) Series 1: PO (c) Series 2: Web directory

Fig. 4: Matching quality comparison.

Figs. 4(a, c) show that our proposed approach achieves, despite its reduced search space for matching, the best matching quality for the Spicy and Web directory schemas. The approach produces the highest F-measure compared with the other matching strategies. The clustering-based approach could correctly identify similar clusters which helps in achieving good recall; good precision is favored by the restricted search space reducing the risk of false positives. Fig. 4(b) also illustrates that the approach realizes a sufficient matching quality for the PO schemas.

We conducted another set of experiments to verify the matching efficiency. We measured the response time required to perform the specified match tasks of Series 2 illustrated in Table 1. We also compared the response time of the clustering-based approach to the mentioned strategies of COMA++. Results are reported in Fig. 5. The figure shows that the clustering-based approach outperforms the other strategies. The approach needs a total of 28 seconds to match the specified matching tasks. While, *AllContext*, *Fragment_inner* and *Fragement_parents_leaves* require 101, 72, and 66 seconds, respectively. We also analyzed the number of generated partitions (clusters or fragments) and we found that COMA++ generates more partitions than the new cluster approach. We also tested with *Frag_sub* [6], we found that only few partitions are determined so that no correspondences could be found for several matching tasks. We thus do not include the detailed results produced by the *Frag_sub* strategy.

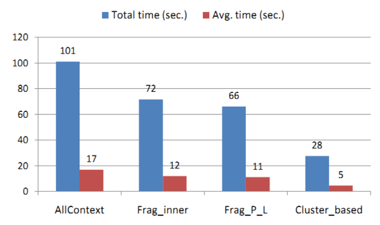


Fig. 5: Res. time comparison

	Clustering-based	Falcon
No. of partitions	84/80	139/119
Precision	0.975	0.964
Recall	0.613	0.591
F-measure	0.753	0.73
Res. time	58.8 sec	10 mins.

Table 3: Anatomy results

We finally evaluate our clustering approach on the anatomy match task and compare it with Falcon. For this purpose, we installed the publicly available Falcon system and run it on the same machine. Results are reported in Table 3. The table shows that our approach achieves a slight improvement in matching quality as Falcon, however, our system is about ten times faster (1 vs. 10 minutes).

In summary, the evaluation results show that the proposed approach achieves for different domains better matching response times compared to previously proposed partition-based strategies at a comparable or better match quality.

7 Conclusions

We proposed a new clustering-based matching approach for large-scale ontology matching. The proposed approach is generic and can be applied to different data models including XML schemas. It shares the same steps of other partition-based match strategies. However, it uses different techniques for partitioning and finding similar partitions. The partitioning process is based on a bottom-up clustering scheme utilizing context-based structural node similarities, while finding similar partitions to match is based on an effective and light-weight linguistic technique. To verify the performance of the proposed approach, we conducted several sets of experiments. The results show that the proposed approach presents significant and encouraging improvement, especially in runtime efficiency. In future work we want to further explore the design space of partition-based match strategies by taking further algorithms for the key steps and further application domains into account.

Acknowledgements: This work is supported by the Federal Ministry of Education and Research (BMBF), grant 03FO2152 (“Web Data Integration”).

References

1. S. Abiteboul, D. Suciu, and P. Buneman. *Data on the Web: From Relations to Semistructed Data and XML.* Morgan Kaumann, USA, 2000.
2. A. Algergawy, R. Nayak, and G. Saake. Element similarity measures in XML schema matching. *Information Sciences*, 180(24):4975–4998, 2010.
3. L. Chiticariu, M. A. Hernndez, P. G. Kolaitis, and L. Popa. Semi-automatic schema integration in Clio. In *VLDB'07*, pages 1326–1329, 2007.
4. N. Choi, I.-Y. Song, , and H. Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.
5. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.
6. H. H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.
7. M. Ehrig and S. Staab. QOM- quick ontology mapping. In *International Semantic Web Conference*, pages 683–697, 2004.
8. A. Gal. Managing uncertainty in schema matching with top-k schema mappings. *Journal on Data Semantics*, 6:90–114, 2006.
9. G. Guerrini, M. Mesiti, and I. Sanz. *An Overview of Similarity Measures for Clustering XML Documents. Emerging Techniques and Technologies.* 2007.
10. F. Hamdi, B. Safar, C. Reynaud, and H. Zargayouna. Alignment-based partitioning of large-scale ontologies. In *Advances in Knowledge Discovery and Management*, volume 292, pages 251–269. Springer, 2010.
11. W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *DKE*, 67:140–160, 2008.
12. O. A. E. Initiative, 2010. `http://20.ontologymatching.org/`.
13. S. Massmann and E. Rahm. Evaluating instance-based matching of web directories. In *11th Workshop on Web and Databases (WebDB)*, 2008.
14. E. Peukert, H. Berthold, and E. Rahm. Rewrite techniques for performance optimization of schema matching processes. In *EDBT*, pages 453–464, 2010.
15. E. Peukert, S. Massmann, and K. Konig. Comparing similarity combination methods for schema matching. In *GI-Workshop*, pages 692–701, 2010.
16. E. Rahm. Towards large-scale schema and ontology matching. In *Data-Centric Systems and Applications*, volume 5258, pages 1–25. Springer, 2010.
17. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
18. E. Rahm, H.-H. Do, and S. Massmann. Matching large XML schemas. *SIGMOD Record*, 33(4):26–31, 2004.
19. M. H. Seddiquia and M. Aono. An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *Web Semantics*, 7(4):344–356, 2009.
20. Z. Wang, Y. Wang, S. Zhang, G. Shen, and T. Du. Matching large scale ontology effectively. In *ASWC 2006, LNCS 4185*, pages 99–105, 2006.
21. N. Yuruk, M. Mete, X. Xu, and T. A. J. Schweiger. AHSCAN: Agglomerative hierarchical structural clustering algorithm for networks. In *International Conference on Advances in Social Network Analysis and Mining*, pages 72–77, 2009.