Sādhanā Vol. 42, No. 12, December 2017, pp. 2009–2024
https://doi.org/10.1007/s12046-017-0742-5

PSOM^2—partitioning-based scalable ontology matching using MapReduce

B SATHIYA[1,*], T V GEETHA[1] and K SARULADHA[2]

[1]Department of Computer Science and Engineering, College of Engineering, Guindy, Anna University, Chennai 600025, India
[2]Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry 605014, India
e-mail: sathiyabalu89@gmail.com; tv_g@hotmail.com; charusanthaprasad@yahoo.com

MS received 20 April 2015; revised 15 October 2015; accepted 9 May 2017; published online 16 November 2017

Abstract. The growth and use of semantic web has led to a drastic increase in the size, heterogeneity and number of ontologies that are available on the web. Correspondingly, scalable ontology matching algorithms that will eliminate the heterogeneity among large ontologies have become a necessity. Ontology matching algorithms generally do not scale well due to the massive number of complex computations required to achieve matching. One of the methods used to address this problem is the use of partition-based systems to reduce the matching space. In this paper, we propose a new partitioning-based scalable ontology matching system called PSOM^2. We have designed a new neighbour-based intra-similarity measure to increase the quality of the cluster set formation for the partition-based ontology matching process. These sets of clusters or sub-ontologies are matched across the input ontologies to identify matchable cluster pairs, based on anchors that are efficiently discovered through a new light-weight linguistic matcher (EI-sub). However, in order to further increase the efficiency of the time-consuming anchor discovery process we have designed a MapReduce-based EI-sub process where anchors are discovered in distributed and parallel fashion. Experiments on benchmark OAEI (Ontology Alignment Evaluation Initiative) large scale ontologies demonstrate that the new PSOM^2 system achieves, on an average, 31% decrease in entropy of the clusters and 54.5% reduction in overall run time. Based on the experimental results, it is evident that the new PSOM^2 achieves better quality clusters and a major reduction in execution time, leading to an effective and scalable ontology matching system.

Keywords. Ontology matching; data integration; semantic heterogeneity; MapReduce; ontology partitioning.

1. Introduction

The semantic web is developed by the International standard body World Wide Web Consortium (W3C). In accordance with W3C, semantic web aims at providing a common framework that allows data sharing and reusing across various applications. The semantic web's aim is achieved by the ontology that is the new form of knowledge representation. According to the ontology search engine, SWOOGLE [1] the semantic web search engine has retrieved more than 10,000 ontologies from the web. Growth of the semantic web has led to an enormous growth in number and size of these ontologies. Due to the distributed nature of the web, different ontologies for same or similar domains are constructed, which leads to heterogeneity among these ontologies. The semantic technology called ontology matching is an effective way to enable interoperability among these heterogeneous ontologies.

As of today, there are a number of applications that require matching large ontologies, such as medicine [2], biology domains, large life-science ontologies [3], E-business [4], web directory [5] and web data [6]. Therefore, these emerging demands on matching large ontologies bring new challenges for ontology matching techniques. The list of challenges presented by Shvaiko and Euzenat [7] are large scale evaluation, efficiency of ontology matching, matching with background knowledge, matcher selection and self-configuration, user involvement, explanations of ontology matching results, collaborative and social ontology matching and alignment infrastructure. The open issue focused in this paper is the efficiency of the ontology matching systems. Based on the OAEI 2012 [8] report many ontology matching systems spend hours or weeks to match large ontologies or fail due to memory resource constraint.

Generally, in ontology matching systems each entity (class, property or instance) of one ontology should be matched with all entities of the other input ontology,

*For correspondence

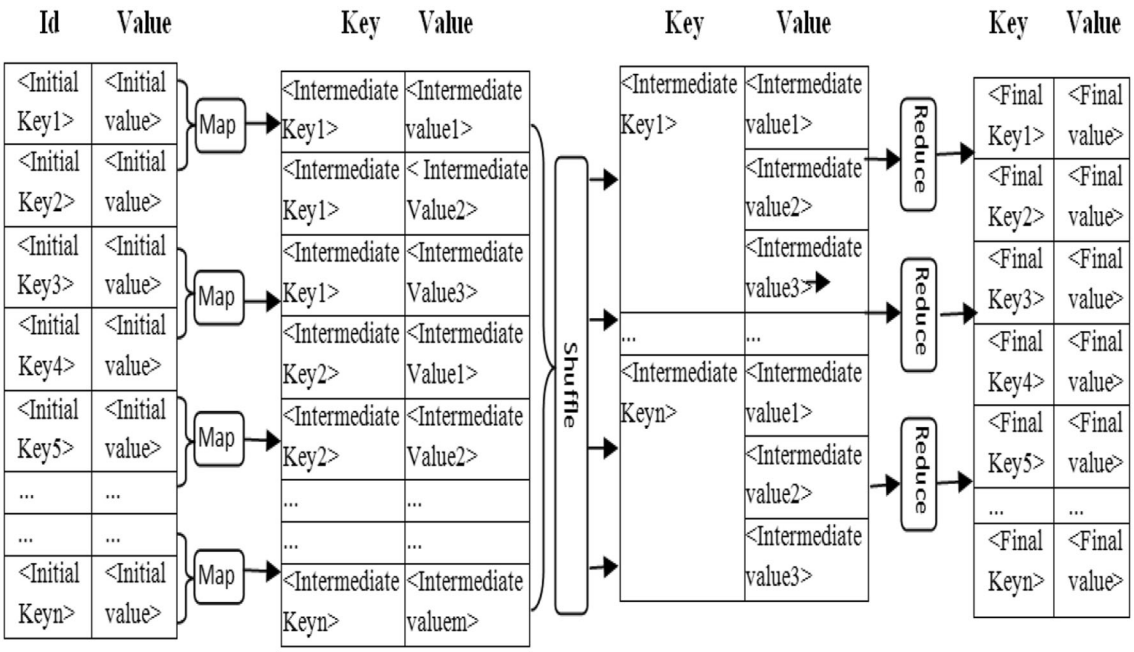

Figure 1. A sample MapReduce dataflow.

leading to *nm* number of computations where *n* and *m* are the number of entities in each of the input ontologies. A set of matchers exploring different information aspects of ontology such as linguistic, structural and instance matchers are used to identify the similar entity pairs. If *k* such matchers are used to process these entities, then totally *knm* number of computations are needed. As the size of input ontologies increases the number of computations increases, leading to large memory requirement and computation time.

According to Bellahsene *et al* [9], the scalability techniques available to handle large scale ontologies are ontology partitioning [10–13], early pruning of the dissimilar entities [14, 15], parallel matching [16], self-tuning of the match workflows [17] and reuse of previous match results [10]. However, these systems fail to utilize the advancements in the parallel and distributed computations such as MapReduce framework [18].

MapReduce is a programming model where large data records are processed by parallel and distributed algorithm using a cluster of nodes. The general dataflow of the MapReduce is depicted in figure 1. The large input data is spilt into several subsets of records and each record is represented as <key, value> pair. Each subset of records is processed by a function called mapper in parallel to produce <intermediate key, intermediate value> pair. These pairs are shuffled such that all values belonging to the same key are shuffled to the same subset of records. The reducer reads these records with the same key and produces the final output.

In this paper, a three-phase-partition based monolingual (input ontologies should belong to the same language) ontology matching system using MapReduce framework called PSOM^2 that aims at scalability with good accuracy is proposed. The three phases of the novel system are as follows: (i) the large input ontologies are decomposed into sets of small clusters using the partition algorithm, (ii) then the matchable cluster pairs across the input ontologies are identified based on the anchors distribution where anchors are linguistically similar entity pairs and (iii) finally, these matchable cluster pairs are processed by a set of available matchers to produce the output. The output is the set of similar entity pairs called alignments with the similarity value ranging between 0 and 1. Anchors are also considered as alignments since they represent linguistically similar entity pairs.

The partitioning will lead to huge reduction in match space, since only the matchable cluster pairs are considered for matching rather than the entire two input ontologies. Even though partition-based ontology matching systems achieve good scalability, the time required to find anchors is high. Hence the anchors are identified in parallel using the MapReduce framework, which leads to considerable reduction in execution time. Scalability is achieved by the proposed system through the following contributions.

- An efficient and effective neighbour-based intra-ontology similarity measure that is used by the ontology partition algorithm to decompose the input ontologies.
- Each of the input ontologies is partitioned in parallel using the single-node-thread-based parallelism, leading to reduction in computation time required for cluster sets formation.
- A new light-weight linguistic similarity matcher called EI-Sub that aims at discovering anchors has been

proposed. The anchors identification is the bottleneck process that consumes large percent of the total computation time. Therefore, the proposed EI-Sub matcher is designed for efficient matching.
- A new MapReduce-based EI-Sub matcher is also proposed that further reduces the computation time of the anchor discovery to a larger scale. To the best of our knowledge, PSOM^2 is the first system to apply MapReduce framework for anchor discovery.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 outlines the new ontology matching system: PSOM^2. Section 4 discusses in detail, the proposed intra-ontology similarity measure and the partition algorithm used. In section 5, the new light-weight linguistic matcher EI-Sub and MapReduce-based EI-Sub are discussed in detail. Further, the methodology used to identify the matchable cluster pairs is also presented. Section 6, outlines the two matchers V-Doc and GMO that are used to find the alignments. Section 7 demonstrates the various experimental results and section 8 concludes the paper with future work.

2. Related work

Even though several ontology matching systems exist, there are only very few matching systems that can handle large scale ontologies efficiently. In this section, ontology matching systems that can efficiently handle large scale ontologies are discussed. Readers are referred to Shvaiko and Euzenat [7], Bellahsene *et al* [9], Euzenat *et al* [19], Granitzer *et al* [20], Euzenat and Shvaiko [21] and Saruladha *et al* [22] for the complete literature survey on small and large scale ontology matching systems.

2.1 *Partition-based ontology matching systems*

The partitioned-based large scale ontology matching systems available are Anchorflood [12], COMA++ [10], Falcon [13], TaxoMap [16, 11], and LOMPT [23]. These systems decompose the large ontology into sets of small clusters, but each system uses different partitioning techniques.

Anchorflood uses a dynamic partitioning algorithm to create clusters using the pre-calculated anchors (similar entity pairs). For each anchor, a set of neighbour entities are collected based on which the ontology is partitioned into clusters. The anchors are discovered using exact string match between descriptions of the entity pairs, which may lead to lesser recall since the quality of the partition is based on the quality of the anchor input.

In COMA++, the ontology is viewed as a graph, where a cluster can be an entire graph or a subgraph. The cluster can either be manually selected by the user or the COMA++ can select subgraphs as clusters. Here, cluster is the collection of entities from the subgraph root to leaf of the subgraph. The matchable cluster pairs are identified based only on the cluster pairs root entity's linguistic similarity. Similarly, COMA++ is basically designed for XML ontologies and hence it does not use the semantic enrichment of the OWL–RDF ontologies. In addition, the simple heuristic rule for partitioning will lead to too few or too many partitions.

Unlike COMA++ and Anchorflood, the Falcon systematically decomposes the ontology into sets of clusters using the modified ROCK partition algorithm. The partition algorithm used is an agglomerative hierarchical partitioning algorithm that is based on depth-based structural proximity measure. Then, the matchable cluster pairs are identified using the anchors distribution across the cluster pairs. These anchors are discovered using the linguistic matcher I-Sub.

TaxoMap first identifies the anchors and the clusters are partitioned around these anchors using the same modified ROCK partition algorithm as that of Falcon. However, the partition algorithm of Falcon and TaxoMap can be applied only to the RDF language ontology and it is more time-consuming since the time complexity of the partition algorithm is $O(n^2)$ where n is the average number of entities of the input ontologies. TaxoMap explores only the labels and hierarchical relation for finding alignments, which leads to less accuracy of the system.

Gross *et al* [16] proposed a matching system that uses both partitioning and distributed-thread-based parallelism technique to reduce computation time. Both inter- and intra-matcher parallelisms are incorporated in this proposed matching system. Inter-matcher parallelism is achieved by parallelizing the linguistic and structural matcher. However, there is no reduction in memory requirements since the entire ontologies need to be processed. In the intra-matcher parallelism, the input ontology is decomposed into sets of clusters based on a size-based partitioning algorithm. These clusters are processed in parallel to find the alignments. The partitioning algorithm is a naive heuristic strategy and it also depends on the correct choice of size parameter to produce proper clusters. In addition, the Cartesian product of the cluster pairs should be processed by the matching system, leading to large computation time.

In LOMPT, an entity's neighbour-based partitioning of the ontology was proposed. This system used a new partitioning algorithm, which was adopted from the network clustering algorithm, AHSCAN [24]. Then, the set of anchors were discovered using a simple linguistic matcher called SI-Sub. The matchable cluster pairs are discovered based on the anchor distributions. The proposed system PSOM^2 is substantially improved work in terms of effectiveness and scalability of the LOMPT ontology matching system.

The drawback of these systems can be summarized as follows. (i) The partitioning techniques used are either naive or dependent on a specific language such as RDF, except LOMPT.

(ii) All these systems, except COMA++, involve a - process that consumes a lot of time to identify the set of anchors, whereas COMA++ uses a very simple approach, leads to less accuracy. Due to these drawbacks, various matching systems proposed different strategies for achieving scalability, which are briefly outlined in the following subsection.

2.2 *Other large scale ontology matching systems*

To handle large scale ontologies, QOM (Quick Ontology Matching) [14, 15] was proposed, where the matching space is reduced by pruning dissimilar entity pairs in the initial stage by a matcher that needs less computational cost. RiMoM [17] is a scalable self-tuning system that selects the set of matchers dynamically based on the input ontology characteristics. For example, the input ontology has predominant linguistic characteristics and the numbers of structural property and instances available are very less. In the RiMoM, only the linguistic matchers are deployed to discover the alignments rather than a default set of matchers. COMA++ reduces the computation time by reusing the already available alignments, if any. Giorgos *et al* [25] used a light-weight linguistic matcher called I-Sub based on edit-distance measure [26] to calculate similarity between the large ontology with less computation time. LogMap [27] is a scalable ontology matching system with a built-in ontology reasoner. The input ontologies are linguistically and structurally indexed using a highly optimized data structure. These processed data structures are used to extract sets of anchors. LogMap involves an iterative process that uses these sets of anchors and changes between alignment discovery and repair. To identify and correct the misalignments, a highly scalable ontology reasoner and greedy diagnosis algorithm are used.

The drawbacks of these systems can be summarized as follows. (i) The system that aims at scalability, by reducing the match space or employing a light-weight matcher, will have a considerable reduction in the accuracy. (ii) Despite dynamic selection of matchers, the self-tuning system needs to process the entire ontology, leading to larger execution time. The decrease in large computation time without compromising on the accuracy can be achieved by the modern parallelism paradigm: MapReduce. The MapReduce-based ontology matching systems are discussed in the following subsection.

2.3 *MapReduce-based ontology matching systems*

Uthayasanker and Doshi [28] proposed a MapReduce framework for partition-based ontology matching. This system utilizes the anchor-based ontology partitioning from TaxoMap matching system. Clusters from different input ontologies are said to form alignment sub-problems if there exist a significant number of anchors between those clusters. These alignment sub-problems are matched by the MapReduce framework in parallel to produce the alignments for each alignment sub-problem. Finally, a post-processing method such as crisscross and redundancy check is carried out to eliminate duplicate and inconsistent alignments. The two drawbacks of this system are as follows. (i) It executes coarse-granularity parallelism since inputs to the MapReduce framework are coarse-grained data, i.e., clusters rather than fine-grained data, i.e., entities. (ii) The time required for identification of the anchor set is the - process that consumes maximum time in partition-based matching system. Instead, the MapReduce framework is applied for finding alignments between the alignment sub-problems, which will not achieve good scalability.

Zhang *et al* [29] proposed an ontology matching system called V-Doc+ using MapReduce framework. In this system, a virtual document is constructed for each class, property and instance of the ontology using four MapReduce processes. Initially, two MapReduce processes are used to construct virtual documents of the entities and the blank nodes. Later, two MapReduce processes are used to exchange descriptions with the neighbour entities. Finally, another MapReduce process deploys a linguistic-weight-based similarity measure in the corresponding reducer, which uses these constructed virtual documents to find the alignments. Zhu and Hu [30] used the afore-discussed MapReduce-based virtual document construction for matching food metadata ontologies. These systems achieve remarkable reduction in execution time but there is considerable loss in the accuracy of the system.

In this paper, by the proposed system PSOM^2, the drawbacks of the existing partition-based and MapReduce-based matching systems are handled. A new intra-ontology similarity measure is proposed and a hierarchical clustering algorithm that is independent of language is deployed, unlike other matching system such as Falcon and TaxoMap. Also, a new light-weight linguistic matcher called EI-Sub is designed to scale down the time consumed for anchor identification.

Unlike the existing partitioned-based systems, which do not incorporate any advanced parallelism technique, and the existing MapReduce-based system, which does not either deploy fine-grain parallelism or compromise on effectiveness of the system, a novel MapReduce-based EI-Sub has been proposed. It works at fine-grained level, i.e., at each entity pair to further reduce the computation time of anchor identification without compromising on the effectiveness of the proposed system, leading to larger improvement in the efficiency.

3. Overview of the PSOM^2

The design of the proposed ontology matching system PSOM^2 is illustrated in figure 2. Inputs to the algorithm are the two large ontologies to be matched. The approach first

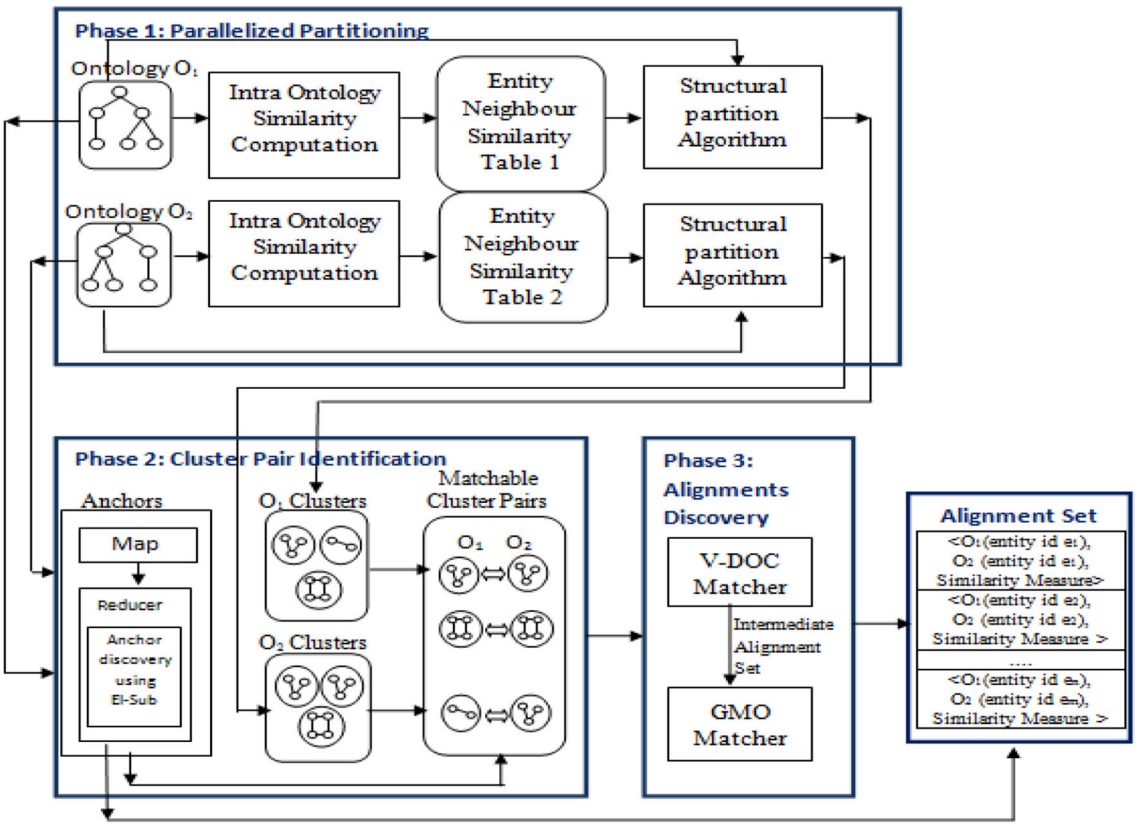

Figure 2. Design of the novel ontology matching system: PSOM^2.

pre-processes the OWL (*W*eb *O*ntology *L*anguage)/RDF (*R*esource *D*escription *F*ramework) ontology using the JENA library and converts it into RDF graph ontology. The intra-ontology similarity between the entities (i.e., classes and properties) of the RDF graph is calculated using the new neighbour-based intra-ontology similarity measure. The intra-ontology similarity values greater than the threshold (γ) are stored in the hash table. The partition algorithm decomposes the input ontologies into sets of clusters in parallel using the single-node-thread-based parallelization technique. Similar clusters across the input ontologies, called matchable cluster pairs, are discovered based on the anchor distribution. These anchors are identified based on the proposed parallelized light-weight string matcher EI-Sub using the MapReduce framework. These cluster pairs are to be further matched by two matchers: V-DOC [31] and GMO [32], for identifying the alignments. In general, the detailed design of the large scale ontology matching system can be divided into three major phases as follows: (i) neighbour-based intra-ontology similarity computation and parallelized partitioning of the input ontologies, (ii) identifying matchable cluster pairs using anchor distribution and (iii) entity level matching of matchable cluster pairs to identify alignments. Each phase is discussed in detail in the following sections.

4. Partitioning ontologies

In this section, the proposed neighbour-based intra-ontology similarity measure is described. Then, the partitioning algorithm based on the proposed neighbour-based intra-ontology similarity measure to identify clusters is discussed. The two input ontologies are partitioned in parallel using single-node-thread-based parallelism. The MapReduce framework is not used here, since the overhead time of MapReduce will be more than the time required to partition the ontology.

4.1 *Neighbour-based intra-ontology similarity computation*

Neighbour-based intra-ontology similarity between entities of the ontology is defined based on how closely the entities are placed in the hierarchies of the ontology. In general, the neighbour of an entity in an ontology graph is represented by its descendants, ancestors, siblings, etc. In LOMPT, immediate parent, immediate children and siblings are considered as neighbours. However, in horizontal ontologies, i.e., the ontologies with a large number of children, the neighbour of entities considering siblings are outsized,

leading to more computations. Hence in this system, the considered neighbours of an entity are immediate descendants (children), immediate ancestors (parents) and the entity itself. Let e be an entity in the graph and E be the set of entities in graph such that e ∈ E; then, the neighbour of e, denoted by N (e), is defined as follows:

$$\mathrm{N(e)} = \{\mathrm{w} \in \mathrm{E} | (\mathrm{w} \in \text{parents}\,(\mathrm{e})) \text{ or } (\mathrm{w} \in \text{children}\,(\mathrm{e}))\} \cup \{\mathrm{e}\}. \tag{1}$$

The proposed intra-ontology similarity measure based on the afore-defined neighbour is derived from the Dice's coefficient [33], which is used to find similarity between sets. Let $e_i, e_j \in E$; the intra-ontology similarity, denoted by $\delta(e_i, e_j)$, is defined as follows:

$$\delta(\mathrm{e}_i, \mathrm{e}_j) = \frac{|\mathrm{N}(\mathrm{e}_i) \cap \mathrm{N}(\mathrm{e}_j)|}{(\,|\mathrm{N}(\mathrm{e}_i)| + |\mathrm{N}(\mathrm{e}_j)|)/2} \tag{2}$$

where N(e) is defined based on Eq. (1). The proposed measure is a normalized and symmetry similarity measure, i.e., $0 \leq \delta(e_i, e_j) \leq 1$, and the following properties hold in accordance with Euzenat and Shvaiko [34].

Positiveness property: $\forall e_i, e_j \in O, \delta(e_i, e_j) \geq 0$.
Maximality property: $\forall e_i \in O, \forall e_j, z \in O, \sigma(e_i, e_i) \geq \sigma(e_j, e_k)$
Symmetry property: $\forall e_i, e_j \in O, \delta(e_i, e_j) = \delta(e_j, e_i)$.

When an entity of a partition shares a similar structure with one of its neighbours, their computed intra-ontology similarity will be large. In general, $(n^2-n)/2$ entity pairs are computed for the intra-ontology similarity values. However, in the proposed system the intra-ontology similarity value is computed only for the entity pairs that have common neighbours and hence reduction in computation time.

4.2 *Partitioning algorithm*

The partitioning algorithm presented in this section is adopted from the LOMPT system. The partition algorithm is used to divide the ontology O into sets of small disjoint clusters $o_1, o_2, \ldots, o_n$ (sub-ontologies) based on the intra-ontology similarity value. The clusters should be formed such that the set of entities in a cluster should have more structural closeness (cohesion) and the set of entities across clusters should have less structural closeness (coupling). In the following, the agglomerative hierarchical partitioning algorithm, which is based on the proposed neighbour-based intra-ontology similarity measure for better cluster quality and efficiency, is presented. The terminologies defined in the partitioning algorithm are discussed in the following subsection.

4.2a *Preliminaries*

Definition 1: Bond An entity pair whose intra-ontology similarity value is greater than the threshold γ is defined as a bond. The threshold γ is experimentally determined and the bond is denoted by B_m, which is defined as follows:

$$\mathrm{B_m} = \{(\mathrm{e}_i, \mathrm{e}_j, \delta(\mathrm{e}_i, \mathrm{e}_j)\} \text{ such that } \delta(\mathrm{e}_i, \mathrm{e}_j) > \gamma \text{ and } \mathrm{e}_i, \mathrm{e}_j \mathrm{E} \tag{3}$$

where $\delta(e_i, e_j)$ is the intra-ontology similarity between entities $e_i, e_j \in E$ computed using Eq. (2). These bonds are stored in entity neighbour similarity hash table called BONDSET. The entity set and the BONDSET are the inputs for the partitioning algorithm. The other similarity values less than γ are not considered for partitioning.

Definition 2: Coupling Let us consider two clusters: c_1 containing n_1 entities and c_2 containing n_2 entities. The coupling between these clusters is the average of the intra-ontology similarity between the entities of the clusters c_1 and c_2. It is defined as follows:

$$\text{coupling}\,(\mathrm{c}_1, \mathrm{c}_2) = \frac{\sum_{i=1}^{n1} \sum_{j=1}^{n2} \delta(\mathrm{e}_{1\mathrm{i}}, \mathrm{e}_{2\mathrm{j}})}{|n_1| + |n_2|}, \quad \mathrm{e}_{1i} \in \mathrm{e}_1\ , \mathrm{e}_{2j} \in \mathrm{e}_2 \tag{4}$$

where $\delta(e_{1i}, e_{2j})$ is the intra-ontology similarity between entities computed using Eq. (2), e_1 and e_2 are the sets of entities of clusters c_1 and c_2, and e_{1i} and e_{2j} are entities belonging to e_1 and e_2. This measure is used to decide whether two clusters should be merged or not in the partitioning process based on the threshold CutOff.

Definition 3: Cluster_Cohesion Cluster_Cohesion measures the cohesion of a cluster, i.e., how the entities in a cluster are structurally close. This measure is computed by the average of intra-ontology similarity value of all the bonds within a cluster. Let c be a cluster, e_i and e_j be the entities in cluster c and n be the number of entities in cluster c. The Cluster_Cohesion is formally defined as follows:

$$\textit{Cluster\_Cohesion}\,(\mathrm{c}) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} \delta(\mathrm{e_i}, \mathrm{e_j})}{n}. \tag{5}$$

Definition 4: ClusterSet_Cohesion The ClusterSet_Cohesion measures the cohesion of the cluster set produced by the partitioning algorithm. It is used as the one of the terminating conditions for the partitioning algorithm. If the cohesion of the cluster set is less than the threshold β, the partitioning algorithm should terminate. Otherwise, less-quality cluster sets are produced, i.e., less cohesion and more coupling between the clusters. This measure is computed by the average of all the Cluster_Cohesion within the cluster set. Let C be a set of clusters with K number of clusters, c_i be the i^{th} cluster in the cluster set C and N be the number of entities in the cluster set C. The ClusterSet_Cohesion is formally defined as follows:

$$ClusterSet\_Cohesion = \frac{\sum_{i=1}^{k} Cluster\_Cohesion\,(c_i)}{N}. \quad (6)$$

4.2b *Partitioning algorithm*: The partitioning algorithm presented is an agglomerative (i.e., bottom up) hierarchical algorithm derived from the AHSCAN (Agglomerative Hierarchical Structure Clustering Algorithm for Networks) [24] approach, which is a very scalable algorithm in the area of network partitioning. The main differences between the AHSCAN and the proposed partitioning algorithm are as follows. (i) In AHSCAN the intra-ontology similarity is scaled by the geometric mean, whereas, in the proposed system, arithmetic mean is used for two reasons. First, geometric mean is used when the values are dependent on each other. However, the neighbours of the entity pairs are independent of each other and hence arithmetic mean is the suitable choice. Secondly, arithmetic mean is chosen to reduce the computation time since the geometric mean involves more costlier computational operations than the arithmetic mean. (ii) The cohesion measure of the AHSCAN is calculated based on the random connection between the nodes of the network. However, ontology does not have random connections and hence a new cohesion measure is proposed.

As shown in figure 3, the proposed partitioning algorithm proceeds as follows. Initially each entity of the ontology forms a cluster. These initial clusters are stored in set CS and Temp_CS. Each cluster pair is calculated for the coupling value. If the coupling value is greater than the threshold CutOff, the cluster pair is merged to form a single cluster. This process is repeated for all possible cluster pairs. Upon merging, the bond whose entity pairs are in the same cluster should be removed from the hash table. The new cluster set is stored in Temp_CS and checked for the ClusterSet_Cohesion. If the ClusterSet_Cohesion is less than the threshold β the algorithm should terminate. Else the algorithm should repeat these steps with Temp_CS as the initial cluster set. The other three terminating conditions are the following: (i) all the entities are merged into one single cluster, (ii) there is no change in the cluster set between two iterations and (iii) there is no sufficient coupling between the clusters to do merging, i.e., no bond is available in the hash table, which is checked by the NoMoreMerge() function. Since the coupling between the clusters decreases as the size of clusters increases, the CutOff should be decreased accordingly.

Each of the two input ontologies are partitioned into sets of clusters, i.e., sub-ontologies using the aforediscussed partitioning algorithm. Next, each cluster pairs across the input ontologies are checked for similarity based on anchor distribution using the proposed MapReduce-based EI-Sub, which is discussed in the following section in detail.

5. Cluster matching

In the following section, the proposed light-weight linguistic matcher: EI-Sub and the MapReduce-based EI-Sub are discussed in detail. Then, the methodology used to discover matchable cluster pairs using anchors is presented.

5.1 *Anchor discovery using EI-Sub*

The entity pairs with high linguistic similarity are termed as anchors, which are primarily discovered for finding matchable cluster pairs. Only the entities between these matchable cluster pairs are matched rather than matching the Cartesian product of all entities of the two large input ontologies [9], leading to tremendous reduction in match space. However the time required for finding the set of anchors is huge, since comparison of nm number of entity pairs is needed, where n and m are the number of entities in the input ontologies. Consequently, there is a strong necessity for an efficient linguistic matcher to find the anchors. Hence in the proposed system, an efficient light-weight linguistic similarity matcher: EI-Sub (Efficient I-Sub) has been proposed to identify the anchor that is derived from I-Sub matcher. The existing linguistic matcher in the partitioned-based ontology matching system such as I-Sub [25] and SI-Sub [23] considers both commonality and difference between strings to compute the similarity value. However, the proposed EI-Sub considers only the sub-string commonality for efficiency. The differences between the strings are not considered for similarity computation as discussed by Giorgos *et al* [25], who stated that "difference should play a less important role on the computation of the overall similarity".

The linguistic descriptions such as entity's local name, label and comment are used for finding the similarity. The proposed measure is defined as the common substring length normalized to the input strings length. The process repeatedly finds and removes the longest common sub-string in the two strings compared, up to a minimum length. The minimum substring length is set to 2. The sum of the lengths of these substrings is then scaled to the length of the strings. Let p and q be two strings; the similarity between p and q is defined as follows:

$$\text{EI - Sub(p, q)} = \frac{2\sum_i \text{length}\,(\text{maxComSubString}_i)}{\text{length}\,(\text{p}) + \text{length}\,(\text{q})} \quad (7)$$

For example, the two strings 'numberofpages' and 'numpages' have the longest common sub-string 'pages'. After it is removed, the two new strings are 'numberof' and 'num'. In the second iteration the sub-string 'num' is removed, leaving 'berof' and ' '. The total length of the common sub-strings is now 8. Hence the EI-Sub value is 0.76 (16/21). If the similarity between two entities descriptions is greater than the threshold μ (experimentally determined), then

```
Algorithm: Partitioning
Input: E is the set of entities of the input ontology. H is the hash table which contains the
 bonds (intra ontology similarity values)
Output: A set of clusters C

for each entity ei in E
   Cluster ci = Create_cluster(ei)
   CS = CS U {ci}
end for
Temp_CS = CS
While (CS.size>1) // Termination Condition 1
   for each cluster ci in CS
      for each cluster cj in CS such that j != i
         if( coupling(ci,cj) > CutOff )
               temp_c = merge( ci, cj )
               Temp_CS = {Temp_CS - {ci,cj}} U {temp_c}
         end if
      end for
   end for
   if (CS.size != Temp_CS.size)
        Temp_CS_ClusterSet_Cohesion = ClusterSet_Cohesion(Temp_CS)
         if (Temp_CS_ClusterSet_Cohesion <β) then
                  break // Termination Condition 2
         else
                  CS = Temp_CS
         end if
   else break // Termination Condition 3
   end if
   If  NoMoreMerge() then
         Break // Termination Condition 4
   end if
   Update CutOff
end while
Return CS
```

Figure 3. Pseudo-code for partitioning algorithm.

these two entities are termed as anchors. The anchors are also considered as parts of the alignments.

Even though the time required for anchor identification across the entire two large ontologies is the maximum among all processes in the matching system, it is justified with the following claims. (i) As only matchable cluster pairs are processed for alignment identification, a considerable number of alignments are missed. These missed alignments can be compensated by the anchors since the anchors are identified from the entire input ontologies. (ii) According to the OAEI 2007 report, the light-weight linguistic matchers are capable of producing 50% of the total alignments. Hence, the anchors contribute in identifying a significant number of alignments.

5.2 *Anchor discovery using MapReduce-based EI-Sub*

Even though the proposed EI-Sub has reduced the computational time of anchor discovery considerably, it is the process that consumes the maximum time. To overcome this, a new MapReduce-based EI-Sub matcher where mapper and reducer functions of the MapReduce framework are adopted to deploy the anchor discovery process in distributed and parallel fashions has been proposed. Figure 4 depicts an example of dataflow for anchor finding in MapReduce. The input data to the MapReduce framework are represented as files, where each line (record) consists of a key–value pair. The number of such pairs is *nm* where *n* and *m* are, respectively, the number of entities in the first and second input ontologies. The key is represented as <uri1i, uri2j> where uri1i is the URI (Unique Resource Identifier) of the i^{th} entity in the first input ontology and uri2j is the URI of the j^{th} entity in the second input ontology. This key is used as the unique identifier for the entity pair. The value is represented as <Desc1i, Desc2j> where Desc1i is the linguistic description of the i^{th} entity in the first input ontology and Desc2j is the linguistic description of the j^{th} entity in the second input ontology. The linguistic description consists of three different elements such as local name, label and comment of an entity.

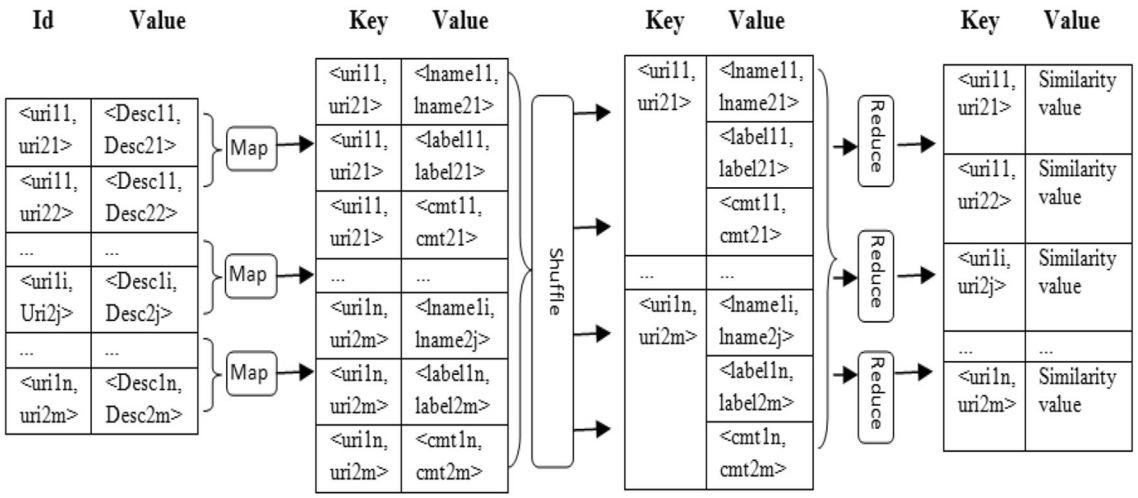

Figure 4. Dataflow example for anchor finding.

The logic behind concatenating three different linguistic information into the same key–value pair is as follows. Creating the input data file with *nm* records requires *nm* File Write operations. However, if we assume each three linguistic elements of an entity pair to be in separate key–value pairs, then 3 *nm* File Write operations are needed to create the input data file, which will lead to an extra preprocessing time. Henceforth, three linguistic elements are concatenated into a single key–value pair, which can be split by a mapper function in parallel fashion. Hence, for each record {<uri1i, uri2j>–<Desc1i, Desc2j>} the mapper produces three key–value pairs as follows: {<uri1i, uri2j>–<lname1i, lname2j>,<uri1i,uri2j>–<label1i, label2j>,<uri1i, uri2j>–<comment1i, comment2j>}. All the mappers together process *nm* input records and produce 3 *nm* output records. After mapping, the shuffler partitions all these records with the same key to the same reducer. Then, the reducer function where the EI-Sub matcher is deployed calculates the linguistic similarity between the entity pairs' local name, label and comment. The highest similarity among the entity pair's local name, label and comment is assigned as the similarity value of this entity pair. Reducer function qualifies the entity pair with similarity value greater than the threshold μ as the output record (anchor) while the other entity pairs are ignored. Hence the number of output records from the MapReduce framework is less than *nm* records. The output records are represented as key–value pairs where the key is the unique entity pair id (<uri1i, uri2j>) and value is the similarity value of the entity pair. These output records from the MapReduce framework are the sets of anchors.

5.3 *Finding matchable cluster pairs*

The matchable cluster pairs are identified using the anchors discovered. Two clusters are said to be a matchable cluster pair if more anchors are found between them. The basic idea is the following: more alignments can be identified between the two clusters if the linguistic similarity between the two clusters is high. Let the input ontologies be represented by O_1 and O_2; CS_1 is the set of clusters of O_1, nc_1 is the number of clusters in CS_1, CS_2 is the set of clusters of O_2 and nc_2 is the number of clusters in CS_2. The measure to identify the matchable cluster pairs is defined [13] as follows:

$$pm\_sim(c_{1i}, c_{2j}) = \frac{\sum_{i=1}^{nc_1} \sum_{j=1}^{nc_2} 2 \times \text{anchor}(c_{1i}, c_{2j})}{\sum_{c_{1k} \in CS_1} \text{anchor}(c_{1k}, c_{2j}) + \sum_{c_{2k} \in CS_2} \text{anchor}(c_{1i}, c_{2k})}. \quad (8)$$

The function anchor (c_{1i},c_{2j}) returns the number of anchors between the cluster c_{1i} and c_{2j} where $c_{1i} \in CS_1$ and $c_{2j} \in CS_2$. This measure is defined as the ratio of the number of anchors shared between the two clusters to the total number of anchors of both the clusters. The cluster pairs whose value is greater than the threshold $\eta(\eta \in [0,1])$ are termed as matchable cluster pairs. Based on Falcon, the η is assigned a value of 0.075. The number of matchable cluster pairs will be less than the number of all possible cluster pairs ($|CS_1||CS_2|$) and hence the cost of computation for further alignment discovery would be largely reduced.

6. Alignment discovery

The set of matchable cluster pairs obtained from the afore-discussed phase can be matched strenuously by a set of available matchers to discover the alignments. The proposed system, similar to Falcon [13], uses a powerful linguistic matcher: V-Doc [31] and a matcher based on incremental structure: GMO [32] in a sequential workflow, i.e., the output alignment of the V-Doc is given as input for GMO matcher. Both of these matchers are computationally intensive or heavy-weight matchers, which are used to find the alignments across the matchable cluster pairs. A brief outline of these matchers is given here.

The V-Doc matcher constructs the virtual document for each entity of the cluster from the matchable cluster pairs. The virtual document consists of details like local name, label and comments of the entity. The linguistic information of the neighbourhoods that is obtained from the RDF triple structure is also included in the virtual document to emphasize the correct meaning of the entity. These collected details are converted into weighted words. The similarities between the virtual documents of the entity pairs are identified using the vector space technique: TF/IDF [35, 36].

The GMO is an incremental matcher, which discovers new alignment based on the alignment form the V-Doc matcher. In GMO, the similarity between the entity pairs is computed by recursively broadcasting their structural similarity across the ontology, which is represented as an RDF Bipartite graph [37]. The set of alignments from the V-Doc and GMO matcher along with the anchor discovered should be heuristically aggregated to obtain the final alignment set. A detailed description on similarity aggregation strategy can be found for Falcon.

7. Experimental results

The proposed matching system is evaluated with varied size ontologies to prove the scalability. The ontology pairs used for the evaluation along with the number of classes are shown in table 1. The first two ontology pairs that are small in size are used for evaluating the partitioning algorithm. These two synthetic ontology pairs can be downloaded from the website: http://ws.nju.edu.cn/falcon-ao/. The next three very large ontology pairs are real world ontology pairs, which can be downloaded along with the reference alignments from the OAEI (Ontology Alignment Evaluation Initiative) website: http://oaei.ontologymatching.org/2013/. These ontology pairs: FMA–NCI, FMA–SNOMED (40%) and NCI–SNOMED (40%) are recognized as large ontology pairs by the OAEI. As depicted in table 1, (*knm*) number of computations are needed where, n and m are the number of entities in the two input ontologies and k is the number of matchers used in the ontology matching system. The last column of table 1 indicates the enormous number of computations required and this multiplies as k increases. Hence these ontology pairs are declared to be large due to the large number of concepts and very huge number of matching computations required for any basic ontology matching system.

Although numerous ontology pairs exist in the literature (Framenet, Yago knowledge base, UMLS, etc.), the metrics used for evaluating effectiveness such as precision, recall and F-Measure can be calculated only for the ontology pairs that have reference alignment. Since our proposed system aims at proving the efficiency with good effectiveness we have chosen these three large ontology pairs, which have reference alignments for evaluation. According to OAEI 2010 report [38], only 50% of the ontology matching systems are capable of matching the large ontologies within 1 h. The ontology matching systems also have a heavy memory requirement to match these large ontologies. Hence there is a need for better scalable ontology matching algorithm. The system is implemented in Java with the 64-bit, Java 1.7 compiler. All the experiments are conducted on an Intel core i7 processor 3.4 GHz desktop machine with 8 GB RAM memory and Windows 7 professional (SP1), 64-bit Operating System.

This section can be divided into the following subsections. (i) In subsection 7.1 the experiments carried out for the new intra-ontology similarity measure and the partitioning algorithm are discussed. (ii) In subsection 7.2 the experiments performed for the new linguistic matcher EI-Sub and the MapReduce-based EI-Sub are presented and (iii) experiments on the new PSOM^2 ontology matching system's F-Measure and execution time are discussed in subsection 7.3.

7.1 *Experiments on ontology partitioning*

The ontology pairs used for the evaluation of the cluster sets are Russia1–Russia2 and TourismA–TourismB. To

Table 1. List of ontology pairs and corresponding number of classes.

Ontology pair	Number of classes	Number of classes	Number of computations (assume $k = 1$)
Russia1–Russia2	Russia1 – 151	Russia2 – 162	24,462
TourismA–TourismB	TourismA – 340	TourismB – 474	161,160
FMA–NCI	FMA – 78,989	NCI – 66,724	5,270,462,036
FMA–SNOMED (40%)	FMA – 78,989	SNOMED (40%) – 122,464	9,673,308,896
NCI–SNOMED (40%)	NCI – 66, 724	SNOMED (40%) – 122,464	8,171,287,936

evaluate the cluster sets, reference clusters (correct clusters) are needed. Among the datasets mentioned in table 1, only the Russia1–Russia2 and TourismA–TourismB ontology pairs have reference clusters. Hence, even though the sizes of these ontologies are small, they are used for evaluating clusters. Moreover, in this section the accuracy of the new intra-ontology similarity measure and the partitioning algorithm is checked and not the scalability.

The evaluation metrics used in this section are execution time and entropy. The entropy is a standard cluster metric to measure the randomness of the clusters. Less the entropy of a cluster set, more the quality of the cluster set. A detailed description of this measure can be obtained for the Falcon [13]. As mentioned in section 4.2a, the entity pair whose intra-structural similarity value is greater than a threshold γ is defined as a bond. This threshold should be assigned the value for which the minimum entropy is achieved. As shown in figure 5, the ideal value of γ is 0.5 since it achieves the minimum entropy value.

In the first experiment, the efficiency achieved by the new neighbour-based intra-ontology similarity computation is depicted in figure 6 through the computation time. As shown in figure 6, the time required for the new measure is less compared with the existing measures for the following reasons. (i) The scaling factor of the similarity measure is the arithmetic mean rather than the geometric mean used in the AHSCAN's similarity measure method. (ii) The LOMPT's measures requires more time, since it also considers siblings for the neighbourhood.

In the second experiment, the entropy of the clusters formed by the partitioning algorithm using the new neighbour-based intra-ontology similarity is measured. The result of COMA is quoted from the experimental results for Falcon. The reasons to choose these systems are as follows. (i) Falcon is recognized as one of the best ontology matching systems in the OAEI 2007 report. (ii) COMA is a flexible matching system with a large number of inbuilt matchers. COMA was also considered in the OAEI workshop and yielded good results. (iii) Similar partitioning algorithms are used in LOMPT and PSOM². As shown in figure 7, the proposed system's partitioning algorithm yields less entropy. Specifically, it achieves, on an average, 31% decrease in entropy as compared with the Falcon, COMA and LOMPT. Even though the LOMPT and PSOM² deploy similar partitioning algorithms, the entropy of PSOM² is reduced by 16%. This reduction is achieved due to the effective evaluation of the intra-ontology similarity through the new measure.

The third experiment demonstrates the need for partition-based ontology matching system. The PSOM² is modified to match the input ontologies without partitioning. As shown in figure 8, the experimental results on execution

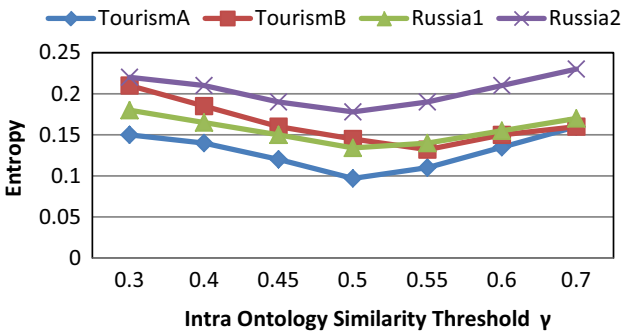

Figure 5. Variation of entropy of partitioning algorithm of PSOM² with the threshold γ.

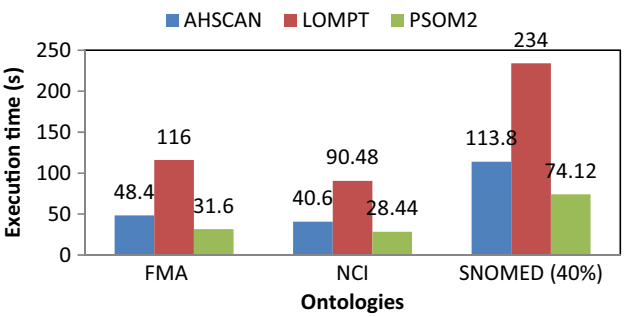

Figure 6. Execution time of new neighbour-based intra-ontology similarity measure.

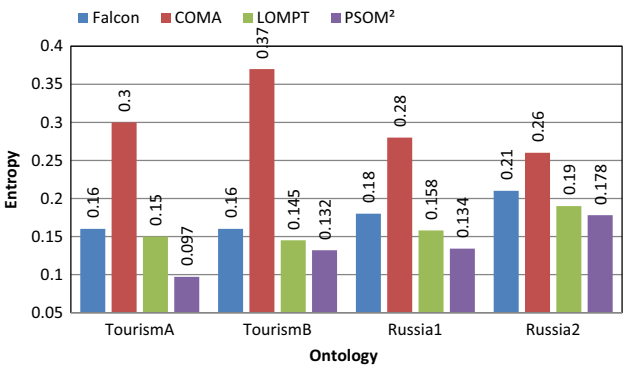

Figure 7. Entropy of partitioning algorithms of Falcon, COMA, LOMPT and PSOM².

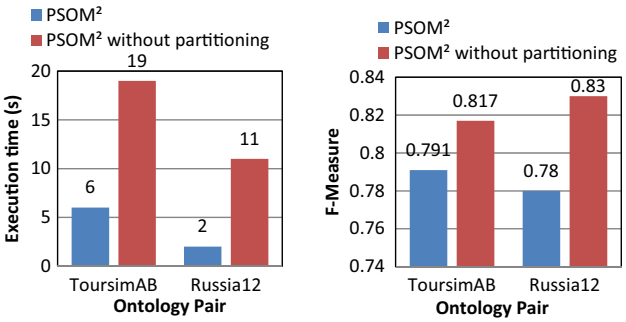

Figure 8. Execution time and *F*-Measure of PSOM² and PSOM² without partitioning.

time proved the efficiency achieved by PSOM^2 even though the accuracy of the PSOM^2 is marginally compromised since Cartesian product of the entity pairs is not evaluated for matching. The execution time required for PSOM^2 without partitioning is many folds higher than the PSOM^2 with partitioning. In addition, memory insufficiency problem occurred for the other large ontology pairs such as FMA–NCI, FMA–SNOMED (40%) and NCI–SNOMED (40%). Hence, the usage of partition-based ontology matching is twofold: (i) increased efficiency by reduced execution time and (ii) decrease in memory requirement.

Table 2. Execution time (s) of Falcon, LOMPT and PSOM^2 for anchor identification.

	FMA–NCI	FMA–SNOMED (40%)	NCI–SNOMED (40%)
(Falcon) I-Sub	44,214	148,392	106,448
(LOMPT) SI-Sub	35,623	117,267	84,413
(PSOM^2) EI-Sub	39,919	125,192	90,118

7.2 *Experiments on anchor identification*

The set of anchors discovered by the new light-weight linguistic matcher: EI-Sub is evaluated for efficiency and effectiveness. The ontology pairs used for evaluation are FMA–NCI, FMA–SNOMED (40%) and NCI–SNOMED (40%). The anchor threshold μ is experimentally determined by assigning the value that yields the maximum *F*-Measure (figure 9). *F*-Measure is considered for two reasons. (i) Anchors are also part of the final alignments. Hence, correct choice of μ yields better anchors and hence better alignments and (ii) the correct identification of the matchable cluster pairs are based on the anchors that result in good accuracy of the system. Based on the experiment, the threshold μ is assigned a value of 0.75. First, the time required to identify the anchors based on EI-Sub, I-Sub and SI-Sub is measured (table 2). On an average, the time reduction obtained by the EI-Sub over I-Sub is 13.5%. It is also noted that SI-Sub has less execution time than EI-Sub since it uses a very naive linguistic similarity measure that is less effective as evident by the following experiment.

The matchable cluster pairs are identified based on the anchors discovered. Only these matchable cluster pairs are processed to obtain the final alignment set. Hence, the correct choice of matchable cluster pairs is the important task. This is made possible only if the sets of both cluster and anchors obtained are of good quality. Hence, the time required to process the matchable cluster pairs is the metric to evaluate both the novel neighbour-based partitioning algorithm and the EI-Sub. Less time to match indicates better clustering and more accurate match between clusters, leading to significant reduction in execution time.

The time required by PSOM^2 is less compared with the Falcon and LOMPT systems (figure 10). The inference from the experiment is as follows. (i) Even though Falcon uses a more robust linguistic matcher to identify anchors, more time is required to process the matchable cluster pairs since the clusters formed have more randomness. When the clusters are random, identifying the correct choice of matchable cluster pairs is difficult. (ii) LOMPT and PSOM^2 have good entropy measure, i.e., both produce good quality clusters. However, the time required to process the matchable cluster pairs is more in LOMPT due to the less-effective naive SI-Sub matcher. The anchors discovered by SI-Sub fail to identify correct matchable cluster pairs, leading to more execution time. (iii) Since PSOM^2 produces good quality clusters and anchors set, the time required to process the matchable cluster pairs is the least.

In spite of the efficiency achieved in intra-ontology similarity computation and anchor finding in PSOM^2, the system still spends huge time in anchor finding. Hence to make the system more scalable, a new MapReduce-based EI-Sub matcher is proposed to reduce the time taken for anchor finding. To implement the MapReduce

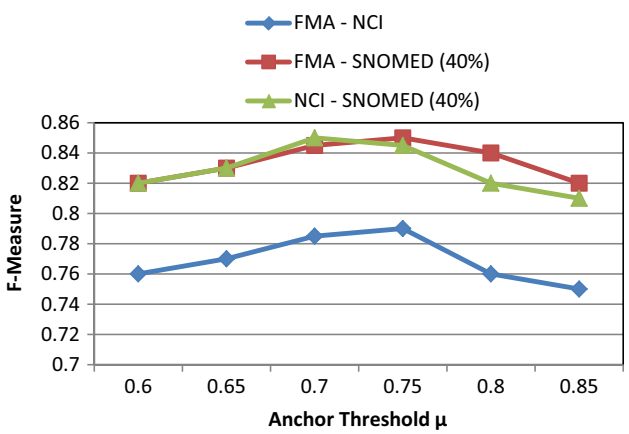

Figure 9. Variation of *F*-Measure based on the anchor threshold μ.

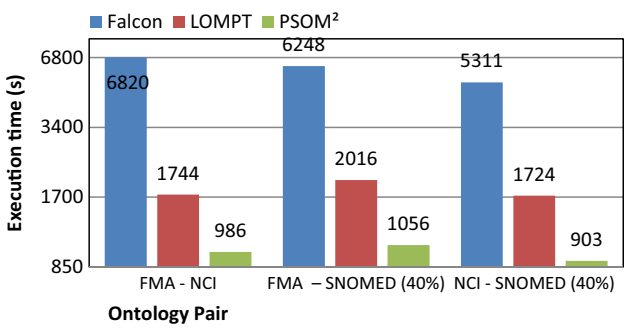

Figure 10. Time required to match the matchable cluster pairs.

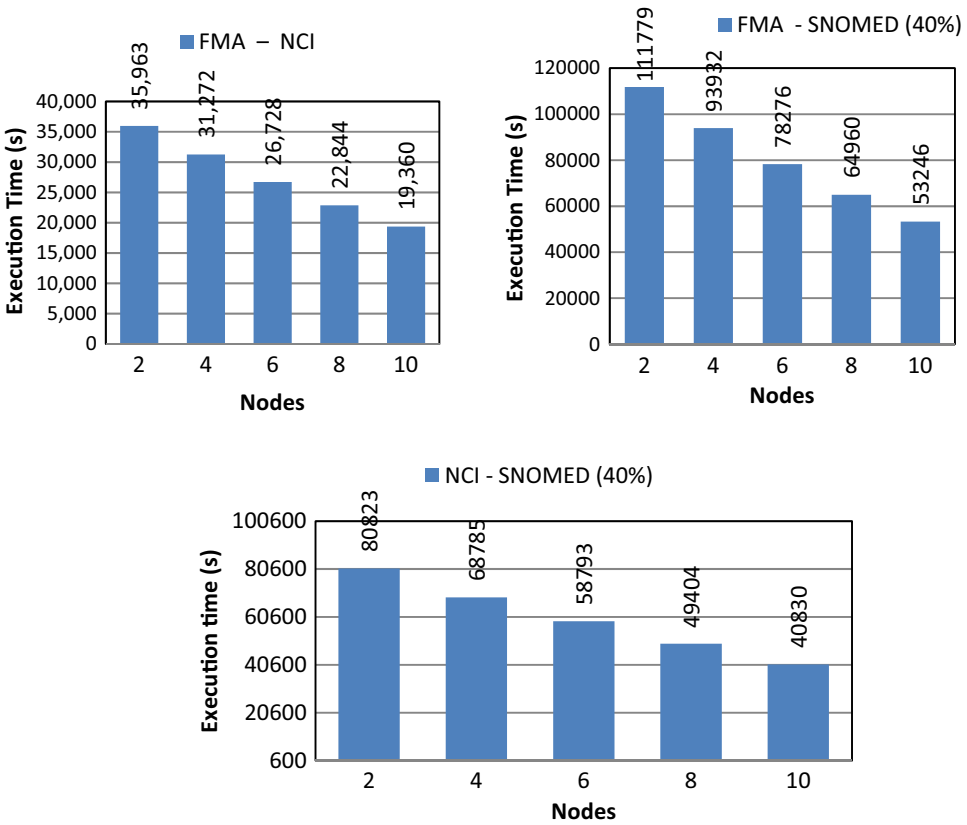

Figure 11. Execution time for anchor identification by EI-Sub using MapReduce.

environment, Hadoop clusters using Amazon EC2 clusters with 10 nodes are set up.

The percentage of reduced execution time achieved by MapReduce-based EI-Sub with 10 nodes over the EI-Sub (figure 11) for varying size ontology pairs are (i) 51.5% reduction for FMA–NCI, (ii) 57.46% reduction for FMA–SNOMED (40%) and (iii) 54.7% reduction for NCI–SNOMED (40%). The new PSOM^2 achieves large reduction in anchor finding time, leading to a more scalable system. It is also evident from the results that, as the size of the ontology decreases, the reduction in execution time also decrease. This is due to the fact that for smaller datasets the overhead time is more than the time reduced by parallelism.

To measure the efficiency achieved by the parallelism, the speedup metric [29] is used. Speedup is the ratio of the execution time without parallelism to execution time with parallelism. The speedup achieved by PSOM^2 is shown in figure 12. The achieved speedup increases as the number of nodes increases. Hence, the proposed system PSOM^2 can achieve better reduction in anchor discovery time as the number of nodes deployed increases.

7.3 *Experiments on precision, recall, F-Measure and execution time*

The main objective of the new ontology matching system: PSOM^2 is to achieve better scalability by reduced execution time with good accuracy. The ontology pairs used for evaluation are FMA–NCI, FMA–SNOMED (40%) and NCI–SNOMED (40%). PSOM^2 is evaluated with the metrics like precision, recall, *F*-Measure and execution time. The precision, recall and *F*-Measure are defined as follows:

$$\text{precision} = |\text{A} \cap \text{R}|/|\text{A}| \tag{9}$$

$$\text{Recall} = |\text{A} \cap \text{R}|/|\text{R}| \tag{10}$$

$$F - Measure = \frac{(2(precision \times recall))}{(precision + recall)} \tag{11}$$

where A is the set of alignments obtained by the PSOM^2 and R is the set of reference alignments. Inferences obtained for the experiments on precision (figure 13), recall (figure 14) and *F*-Measure (figure 15) of the matching systems are as follows. (i) PSOM^2 achieves better correctness of the alignments compared with the LOMPT system. Even though both PSOM^2 and LOMPT use the same partitioning algorithm, more precision is achieved in PSOM^2 due to the improved effectiveness of the new intra-ontology similarity measure and EI-Sub. (ii) However, PSOM^2 makes a small marginal compromise on the precision compared with the Falcon. Falcon uses the more robust, comparatively time-consuming linguistic matcher I-Sub to identify anchors, which also contributes to the final alignment. Experiments on the effectiveness of the

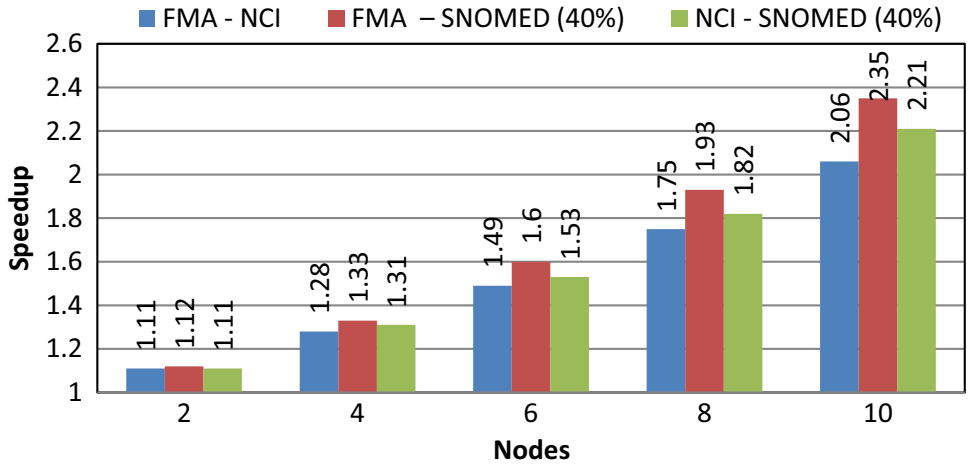

Figure 12. Speedup of EI-Sub using MapReduce.

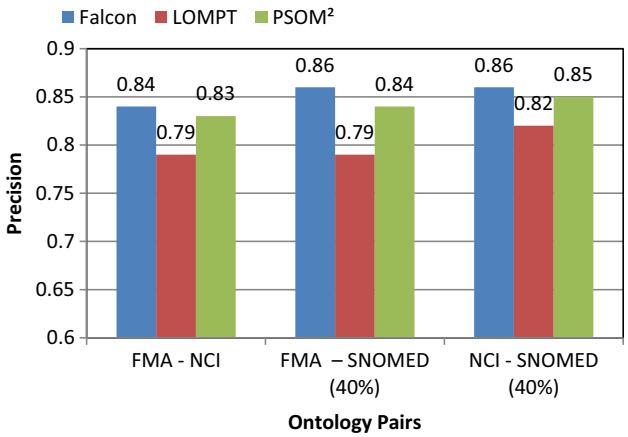

Figure 13. Precision of Falcon, LOMPT and PSOM^2.

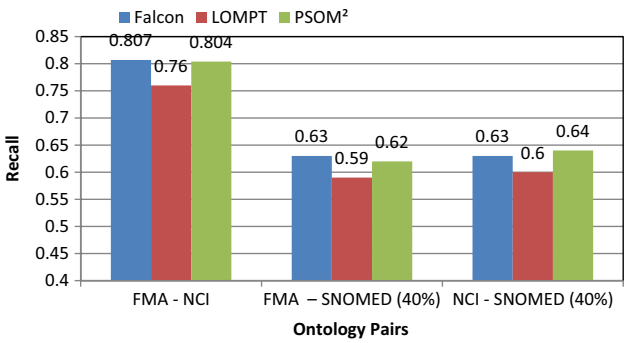

Figure 14. Recall of Falcon, LOMPT and PSOM^2.

matching systems depicts that the PSOM^2 achieves better effectiveness (*F*-Measure) than LOMPT and almost equal effectiveness as that from Falcon.

Finally, the execution times of the matching systems for producing the final alignments are measured (table 3). The

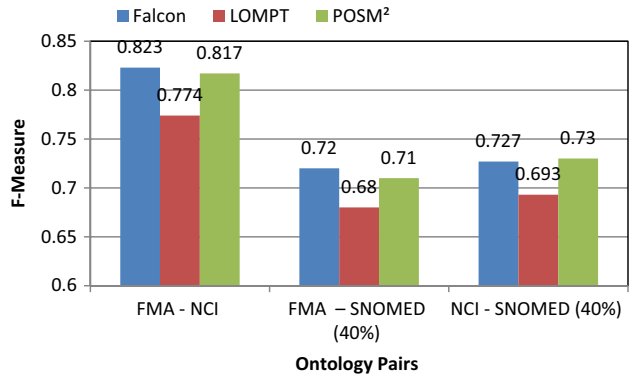

Figure 15. *F*-Measure of Falcon, LOMPT and PSOM^2.

Table 3. Overall execution time (s) of Falcon, LOMPT and PSOM^2.

	FMA–NCI	FMA–SNOMED (40%)	NCI–SNOMED (40%)
Falcon	47,745	165,480	135,694
LOMPT	38,896	133,376	116,701
PSOM^2	24,200	67,400	52,000

results obtained are as follows. (i) On an average, 58.8% of the execution time is reduced as compared with the Falcon. Specifically, 61.7% of the execution time is reduced as compared with the Falcon for NCI–SNOMED (40%) ontology pair. (ii) Compared with LOMPT, the execution time is reduced by 50.3%. Particularly, for NCI–SNOMED (40%) ontology pair, 55.4% reduction in execution time is achieved. It is evident from the results that PSOM^2 achieves better efficiency then the existing ontology matching systems: Falcon and LOMPT. It is also noted that the efficiency of PSOM^2 will increase as the size of the input ontology increases.

8. Conclusion

In this paper, a new scalable partition-based ontology matching algorithm called PSOM^2 is proposed, which aims at better efficiency and effectiveness over the existing matching algorithms. Specifically, a new neighbour-based intra-ontology similarity measure is proposed to effectively decompose the large ontologies into sets of small clusters in parallel. As the quality of the clusters increases, the chance of similar entity pairs getting matched increases, leading to increase in effectiveness. To achieve better scalability, the proposed system tackled the most time-consuming, anchor finding process. An efficient light-weight linguistic matcher called EI-Sub is introduced to identify the set of anchors in less time. Further, a new MapReduce-based EI-Sub matcher is proposed to discover the anchor in parallel and distributed ways.

In the future work, neighbour sets of the entities can be dynamically selected based on the ontology characteristics to increase the cluster quality. The execution time required to find anchors can be reduced by random selection of the entities rather than the Cartesian product of the entities. For very large ontologies, the processing time required to find alignments from the matchable cluster pairs can be further reduced by introducing parallelism through MapReduce framework.

Acknowledgement

This work was financially supported by Anna University, Chennai, India, through the Anna Centenary Research Fellowship.

References

[1] Ding L, Pan R, Finin T, Joshi A, Peng Y and Kolari P 2005 Finding and ranking knowledge on the semantic web. In: *Proceedings of the 4th ISWC 2005*, *Lecture Notes in Computer Science*, vol. 3729, pp. 156–170

[2] Zhang S, Mork P, Bodenreide O and Bernstein P A 2007 Comparing two approaches for aligning representations of anatomy. *Artif. Intell. Med.* 39(3): 227–236

[3] Kirsten T, Thor A and Rahm E 2007 Instance-based matching of large life science ontologies. In: *Proceedings of the Data Integration in the Life Sciences, DILS 2007, Lecture Notes in Computer Science*, vol. 4544, pp. 172–187

[4] Smith K, Morse M, Mork P, Li M, Rosenthal A, Allen D, Seligman L and Wolf C 2009 The role of schema matching in large enterprises. arXiv preprint arXiv:0909.1771

[5] Avesani P, Giunchiglia F and Yatskevich M 2005 A large scale taxonomy mapping evaluation. In: *Proceedings of ICSW 2005, Lecture Notes in Computer Science*, vol. 3729, pp. 67–81

[6] Su W, Wang J and Lochovsky F H 2006 Holistic schema matching for web query interfaces. In: *Proceedings of the International Conference on Extending Database Technology, EDBT 2006, Lecture Notes in Computer Science*, vol. 3896, pp. 77–94

[7] Shvaiko P and Euzenat J 2013 Ontology matching: state of the art and future challenges. *IEEE Trans. Knowl. Data Eng.* 25(1): 158–176

[8] Aguirre L J, Bernardo C G, Eckert K, Euzenat J, Ferrara A, Hague R W V, Hollink L, *et al* 2012 Results of the ontology alignment evaluation initiative 2012. In: *Proceedings of the 7th ISWC Workshop on Ontology Matching*, pp. 73–115

[9] Bellahsene Z, Bonifati A and Rahm E 2011 Towards large-scale schema and ontology matching. In: *Schema matching and mapping*, New York–Heidelberg: Springer, pp. 3–28

[10] Do H H and Rahm E 2007 Matching large schemas: approaches and evaluation. *J. Inf. Syst.* 3(6): 857–885

[11] Hamdi F, Safar B, Reynaud C and Zargayouna H 2009 Alignment-based partitioning of large-scale ontologies. In: *Advances in Knowledge Discovery and Management, Studies in Computational Intelligence*, vol. 292, pp. 251–269

[12] Hanif M S and Aono M 2009 An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *J. Web Semant. Sci. Serv. Agents WWW* 7(4): 344–356

[13] Hu W, Qu Y and Cheng G 2008 Matching large ontologies: a divide-and-conquer-approach. *J. Data Knowl. Eng.* 67(1): 140–160

[14] Ehrig M and Steffen S 2004 QOM—quick ontology mapping. In: *Proceedings of The Semantic Web—ISWC 2004*, pp. 683–697

[15] Peukert E, Berthold H and Rahm E 2010 Rewrite techniques for performance optimization of schema matching processes. In: *Proceedings of the 13th International Conference on EDBT 2010*, pp. 453–464

[16] Gross A, Hartung M, Kirsten T and Rahm E 2010 On matching large life science ontologies in parallel. In: *Proceedings of the 7th International Conference on Data Integration in the Life Sciences, DILS 2010, Lecture Notes in Computer Science*, vol. 6254, pp. 35–49

[17] Li J, Tang J, Li Y and Luo Q 2009 RiMOM: a dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.* 21(8): 1218–1232

[18] Dean J and Ghemawat S 2008 MapReduce: simplified data processing on large clusters. *Commun. ACM* 51(1): 107–113

[19] Euzenat J, Meilicke C, Stuckenschmidt H, Shvaiko P and Trojahn C 2011 Ontology alignment evaluation initiative: six years of experience. *J. Data Semant.* 15: 158–192

[20] Granitzer M, Sabol V, Onn K W, Lukose D and Tochtermann K 2010 Ontology alignment—a survey with focus on visually supported semi-automatic techniques. *Future Internet* 2(3): 238–258

[21] Euzenat J and Shvaiko P 2013 Overview of matching systems. In: *Ontology matching*, 2nd edn. Berlin–Heidelberg: Springer, pp. 153–193

[22] Saruladha K, Aghila G and Sathiya B 2011 A comparative analysis of ontology and schema matching systems. *Int. J. Comput. Appl.* 34(8): 14–21

[23] Saruladha K, Aghila G and Sathiya B 2012 LOMPT: an efficient and scalable ontology matching algorithm. *Proc. Eng.* 38: 2272–2287

[24] Yuruk N, Mete M, Xu X and Schweiger T A 2009 AHSCAN: agglomerative hierarchical structural clustering algorithm for networks. In: *Proceedings of the IEEE*

International Conference on Advances in Social Network Analysis and Mining, pp. 72–77

[25] Giorgos S, Stamou G and Kollias S 2005 A string metric for ontology alignment. In: *Proceedings of The Semantic Web—ISWC 2005*, pp. 624–637

[26] Levenshtein V I 1966 Binary codes capable of correcting deletions insertions and reversals. *Sov. Phys. Dokl.* 10(8): 707

[27] Jiménez-Ruiz E and Grau B C 2011 Logmap: logic-based and scalable ontology matching. In: *Proceedings of The Semantic Web—ISWC 2011*, pp. 273–288

[28] Uthayasanker T and Doshi P 2013 Speeding up batch alignment of large ontologies using MapReduce. In: *Proceedings of the Seventh IEEE International Conference on Semantic Computing (ICSC)*, pp. 110–113

[29] Zhang H, Hu W and Qu Y 2012 VDoc+: a virtual document based approach for matching large ontologies using MapReduce. *J. Zhejiang Univ. Sci.* 13(4): 257–267

[30] Zhu L and Hu W 2012 Towards matching food metadata in emergency decision-making using ontology and MapReduce. In: *Proceedings of the IEEE International Conference on Information Management, Innovation Management and Industrial Engineering*, vol. 2, pp. 498–501

[31] Qu Y, Hu W and Cheng G 2006 Constructing virtual documents for ontology matching. In: *Proceedings of the 15th International Conference on World Wide Web*, pp. 23–31

[32] Hu W, Ningsheng J, Yuzhong Q and Yanbing W 2005 GMO: a graph matching for ontologies. In: *Proceedings of the K-CAP Workshop on Integrating Ontologies*, pp. 41–48

[33] Dice L R 1945 Measures of the amount of ecologic association between species. *Ecology* 26(3): 297–302

[34] Euzenat J and Shvaiko P 2013 Basic techniques. In: *Ontology matching*, 2nd edn. Berlin–Heidelberg: Springer, pp. 73–117

[35] Raghavan V V and Wong S M 1986 A critical analysis of vector space model for information retrieval. *J. Am. Soc. Inf. Sci.* 37(5): 279–287

[36] Su X and Gulla J A 2006 An information retrieval approach to ontology mapping. *Data Knowl. Eng.* 58(1): 47–69

[37] Hayes J and Gutiérrez C 2004 Bipartite graphs as intermediate model for RDF. In: *Proceedings of the 3rd International Semantic Web Conference, Lecture Notes in Computer Science*, vol. 3298, pp. 47–61

[38] Euzenat J, Isaac A, Meilicke C, Shvaiko P, Stuckenschmidt H, Sváb O, Svátek V, Van Hage W R and Yatskevich M 2007 Results of the ontology alignment evaluation initiative 2007. In: *Proceedings of the ISWC+ASWC Workshop on Ontology Matching*, pp. 96–132