Journal of Scientific & Industrial Research
Vol. 80, March 2021, pp. 221-229

# MPP-MLO: Multilevel Parallel Partitioning for Efficiently Matching Large Ontologies

Usha Yadav[1,2]* and Neelam Duhan[2]

[1]National Institute of Fashion Technology, Jodhpur 342 037, India

[2]J C Bose University of Science & Technology, YMCA, Faridabad 121 006, India

*Received 01 August 2020; revised 30 December 2020; accepted 12 February 2021*

The growing usage of Semantic Web has resulted in an increasing number, size and heterogeneity of ontologies on the web. Therefore, the necessity of ontology matching techniques, which could solve these issues, is highly required. Due to high computational requirements, scalability is always a major concern in ontology matching system. In this work, a partition-based ontology matching system is proposed, which deals with parallel partitioning of the ontologies at multilevel. At first level, the root based ontology partitioning is proposed. Match able sub-ontology pair is generated using an efficient linguistic matcher (IEI-Sub) to uncover anchors and then based on maximum similarity values, pairs are generated. However, a distributed and parallel approach of Map Reduce-based IEI-sub process has been proposed to efficiently handle the anchor discovery process which is highly time-consuming. In second level partitioning, an efficient approach is proposed to form non-overlapping clusters. Extensive experimental evaluation is done by comparing existing approaches with the proposed approach, and the results shows that MPP-MLO turns out to be an efficient and scalable ontology matching system with 58.7% reduction in overall execution time.

**Keywords:** Big Data, Large scale Ontology, MapReduce, Ontology Matching

## Introduction

In this data age, ontologies are gaining lot of consideration in Computer Science, especially in the field of Semantic Web technologies. Sharing of information and integration among various applications through ontologies has contributed tremendously in developing and deploying semantic web. The amount of ontologies is increasing day by day and gaining lot of popularity in Semantic query search[1], intelligent advisory systems, Semantic information integration etc. Handling semantic heterogeneity problem is very challenging and ontology matching system has been proposed to overcome this challenge by establishing interoperability between various applications that uses different yet related ontologies. One of the most challenging issues outlined in researches related to ontology matching systems is matching large scale ontologies. Terminological and conceptual level of large scale ontologies is very heterogeneous in nature and considered as the main reason for such issues. Furthermore, the resource requirement is another major challenge, exploring large scale ontologies requires large search space to uncover correspondences. Also, at each computational stage, there is a high requirement for main memory to store and process temporary results. Therefore, effectiveness and efficiency of any large scale ontology matching system will strongly get impacted from the mentioned factors. Researchers[2] proposed various data mining approaches to identify the appropriate features to match ontologies. The sensor[3] ontology matching system address the limitation of local optimal solution and Evolutionary Algorithm for matching large number of concepts. Facilitating the semantic interoperability in any domain requires integration of entities in upper ontology. Integration is very tedious task and requires human interventions despite many automated methods[4,5] are emerging. Machine learning models[6] are also trained using the knowledge base and external sources to match input ontologies. Aspect based semi automated ontology builder, (SOBA)[7] is implemented for semantic analysis. The limitation of existing approaches[8] in dealing large scale ontology are low accuracy, performance inefficiency and high computational complexity. The motivation behind the proposed work is the widely adoption of ontologies as a means for knowledge sharing and reuse. In large scale ontology matching system, each entity of input ontology should be matched with each of the entities

*Author for Correspondence
E-mail: usha.yadav.912@gmail.com

from the second input ontologies, resulting in high computational complexity. Also, various matching algorithms are needed to determine structural, linguistic and other similarity, which further increases the space and time complexity. Ontology partitioning requires only the comparison between the entities of match able cluster pair instead of the entire entities in input ontologies, which leads to reduction in space and computational complexity. The anchor identification process consumes high computation which requires efficient solution to process large data in distributed and parallel manner. Map Reduce framework could easily store and process large data sets. The key contribution of this work is mentioned below:

i) *Multilevel Partitioning*: Ontology partitioning at multilevel. The first level partition the ontology from root and second level partitioning generated the non-overlapping clusters.
ii) *Parallelism*: The input ontologies partition process is computed in parallel, which leads to reduction in computational complexity.
iii) *IEI-Sub Matcher*: An efficient linguistic matcher is proposed for finding the anchors between the all the entities of the both the input ontologies. This process requires high computation.
iv) *Map Reduce Based IEI-Sub Matcher*: To overcome the computational bottleneck of anchor discovery phase, Mapper and Reducer based algorithm is proposed to implement IEI-Sub Matcher to find the anchors efficiently.

## Methodology

The proposed framework **MPP-MLO** is described in Fig. 1, which composed of four components namely First Level Partitioning, Partitioned Ontology Candidate Mapping, Second Level Partitioning, and Final Alignments.

#### First Level Partitioning

In this module, pre-processing and entity document creation is performed after which root level partitioning is done as described in detail below.

***Preprocessing***

In this step, tokenization, Stop Word removal and stemming are done on label, comments and name of an entity. After this, the document for each entity of the ontology is formed based on following:

(1) Structural connection *SC(e,d)* of an entity $e \in E$, *(E* denotes entity set) is defined as in Eq.1

$$SC(e,d) = \{sub\ Class(e,d) \cup super\ Class(e,d)\} \quad \ldots (1)$$

where *sub Class(ei,d)* denotes the subclass or children of *e* within *d* levels. The *super Class(e,d)* denotes the super Class or the parents of *e* within *d* hierarchical level in the ontology.

(2) Linguistic description contains all the human readable information such as comments or labels provided to that entity.
(3) Closeness centrality is the measure which reflects the significance of nodes that are near to all other ones present in the graph. In Eq. 2 the cost of reaching all the nodes from one node is calculated. The *distance (i,j)* denotes the function to calculate the shortest path between the node *i* and *j* in the graph, where $j \epsilon V$, *V* is set of vertices in graph.

$$cc(i) = 1/\sum_j distance(i,j) \quad \ldots (2)$$

***Root Level Partitioning***

In this level of partitioning, it has been proposed that the input ontologies should be partitioned from root level. In all the previous work done[9,10] for

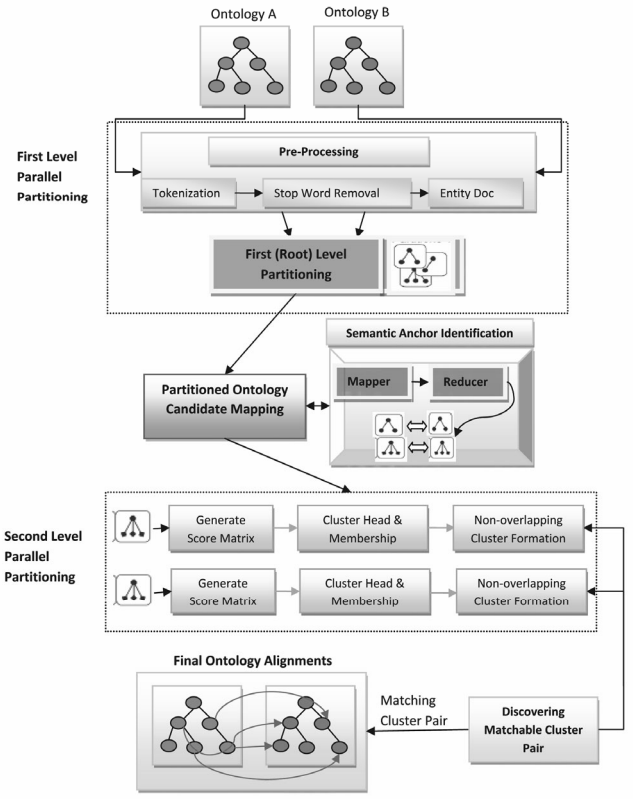


Fig. 1—Framework for Multilevel Parallel Partitioning for Efficiently Matching Large Ontologies

ontology matching using partitioning technique, main concern is the memory and the computation time consumed. Every entity is compared to all other entities in the same ontology either to calculate their intra similarity measures or for partitioning process. In intra similarity measures, similarity between entities present in input ontologies are calculated, each entity present in ontology is assumed to be a cluster and then merged. Suppose ontology contains 1500 entities, it means each entity will be compared to other 1499 entities for intra-similarity measure. Also, for partitioning process, each 1500 entities assumed to be individual cluster and finally merged to form final partitions. After following the same procedure, both the input ontologies are divided into partitioned ontologies known as clusters, and then mapping between the partitioned ontologies from both the input ontologies is done to find the *candidate mapping*, which means the two most similar cluster pair from both the input ontologies should be matched further to get alignments. It requires lots of computation time and space. Now suppose, in the same ontology containing 1500 entities, there are only 30 entities which are directly connected with root concept. So, without using much computation for finding intra similarity and further applying complex partitioning technique, the input ontologies can be easily partitioned at first level. So, instead of computing large $n \times m$ where $n$ and $m$ entities of input ontology 1 and input ontology 2 respectively, the next level of partitioning would be on $k \times l$, where $k,l$ are the entities of sub-ontolgies generated after first level partition and $k<n$, $l<m$.

**Partitioned Ontology Candidate Mapping**

This module is divided into two parts, one is to uncover anchor using Map Reduce based IEI-Sub (Improved EI-Sub[9]) matcher and other is to find matchable sub-ontology pair.

***Uncovering Anchor using IEI-Sub***

Entity pairs showing high linguistic similarity are known as *anchors*. Each sub-ontologies of input ontology *O* is matched with each sub-ontology of other input ontology *O'* for uncovering anchors. Entities in the sub-ontologies showing maximum anchor similarity becomes matchable sub-ontology pair for further alignment discovery. Without partitioning, cartesian product of all entities of the two large input ontologies needs to be processed for finding alignment, but using partitioning only a section of sub-ontology pair needs to be processed for the same, this helps in saving major computational time and space. Although, in this module of anchor uncovering, the computation time for $n$ and $m$ entities of input ontologies *O & O'* would be high, as it will require n×m comparisons.

In this system, efficient linguistic matcher IEI-Sub derived from EI-Sub and using wrinkle[11] is presented. The proposed function in Eqs 3 & 4, finds the commonality between string and used correction coefficient $p$ for the improvement of results. It recursively finds the common substring and then removes the common part and again starts with the leftover part.

$$Sim_{comm} = \frac{2 \times \sum_i len(commonstring)}{len(e_1)+len(e_2)} \quad \ldots (3)$$

$$IEI\text{-}Sub(e1,e2) = Sim_{comm} + (m+n) \times p \times (1 - Sim_{comm}) \quad \ldots (4)$$

where $e_1$ and $e_2$ are the entities in the sub-ontology *SO* and *SO'* respectively and $m$ and $n$ are the length of common prefix (start of the string) and the length of common suffix (end of the string) up to a maximum of four characters respectively and $p$ is constant scaling factor to improve results, whose value is 0.1. Similarity value is compared with threshold value, if similarity value is greater than threshold value, two entities will be called as anchor. It has been reported by OAEI 2007 that 50% of the total alignment can be generated using efficient linguistic matcher.

***Map Reduce based IEI-sub***

The proposed first level partitioning has decreased the computational time required for anchor discovery significantly, as the anchor discovery is only required within the sub-ontologies generated after first level partitioning. So, instead of comparing complete n×m entities present in two input ontologies, system needs to compare only $a \times b$ entities of sub-ontologies candidate pair, where $N,M$ are the entities in sub-ontology 1 and sub-ontology 2 and $N \times M < n \times m$. Even though the IEI-Sub matcher is computationally efficient, the anchor identification process still requires high computation time. With the advancement of Big Data technology which provides efficient solution to deal with high computation time and storage problems, it can prove to be very beneficial for finding anchor at this stage. So, in this module *hadoop* 2.7.3 platform is used which works on MapReduce framework. IEI-Sub matcher is modulated as per the mapper and reducer function for

computation in distributed and parallel environment for finding anchors. The proposed method based on Map Reduce is described in algorithm(1–3) for the Key Generation, Mapper and Reducer.

**Algorithm 1:GenerateKey(Sub-Ontology1 as A, Sub-Ontology 2 as B)**

1. declare empty datafile C
2. for each row in datafile A
2.1 assign incremental numeric key as Id
2.2 append ID to data row
2.3 append field datasource with value "A" to each row
3. for I in {1,N}, where N = number of rows in datafile A:
3.1 assign ID column with value i
3.2 for each row in datafile B
3.2.1 append Id to data row
3.2.2 append field datasource with value "B" to each row
3.2.3 append generated row to datafile C

**Algorithm 2: Mapper(Id, Value):**

1. for each Id:
1.1 get columns from datafile A
1.2 get colums from datafile B
1.3 create a commom structure to include values from both A and B
1.4 Emit (Id, [name, label, comments, datasource})

**Algorithm 3: Reducer(Id,Values):**

1. Initiate variable PREV_ID to null
2. Declare array ARR
3. Def calculate(ARR):
3.1 split ARR to variables, NAME, LABEL, COMMENTS, DATASOURCE
3.2 separate rows from datasource A and B
3.3 filter B for all relevant sub classes for A
3.4 for each item in B:
3.4.1 calculate similarity with A and assign to variable RESULT
3.4.2 Emit (Id, RESULT)
4. For each row in streaming input:
4.1 get ID from row
4.2 if PREV_ID is null or PREV_ID=ID:
4.2.1.1 append Values to ARR

else:

4.2.1.2 call function calculate(ARR)
4.2.1.3 empty ARR
4.2.1.4 set PREV_ID = ID

The linguistic similarity of label, comment and name between the entity pairs are calculated at the reducer level. The similarity value assigned to the entity pair is the highest linguistic similarity value calculated between the entity pair using their label, comment and local name. Only the entity pair having similarity value higher than the given threshold value qualifies as the anchor, whereas the remaining entities are simply ignored. Therefore, the output generated from the framework is fewer than n×m records. The set of anchors are the output produced from this framework, where it corresponds to <Key-Value> Pair. The key represents the distinctive entity pair identity and the value represents the similarity among them.

***Partitioned Ontology Candidate Mapping***

The anchor discovered in previous section was used to find the matchable sub-ontology pairs. If the two sub-ontologies share maximum number of anchors, then these two sub-ontologies are identified as matchable sub-ontology pair. Linguistic similarity implies the possibility of discovering more number of alignments. Therefore, if two sub-ontologies share high linguistic similarity, it means there is high probability of finding more alignments between them and presented in algorithm 4. Let $O_1$ and $O_2$ represent two input ontologies, $SO_1$ represents the set of sub-ontologies generated after first level partitioning of input ontology $O_1$, $ns_1$ is the number of sub-ontologies in $SO_1$. Similarly, $SO_2$ represents the set of sub-ontologies generated after first level partitioning of input ontology $O_2$ and $ns_2$ is the number of sub-ontologies in $SO_2$. The calculation based on which matchable sub-ontology pair is identified is shown in Eq. 5. It is the ratio between the anchors shared between two sub-ontologies to the total number of anchors present in them.

$$So\_sim(sc1_i,sc2_j)= \frac{\sum_{i=1}^{ns1}\sum_{j=1}^{ns2} 2\cdot ancor(\text{sc1i,sc2j})}{\sum_{sc1_k \epsilon SO_1} ancor(sc1_k,sc2_j)+\sum_{sc2_k \epsilon SO_2} ancor(sc1_i,sc2_k)} \quad \ldots (5)$$

**Algorithm 4: Matchable SubOntology Pair**

**Input:** Set of two SubOntologies$SO_1$ and $SO_1$, $ns_1$ is the number of sub-ontologies in $SO_1$, $ns_2$ is the number of sub-ontologies in $SO_2$

**Output**: Set of Matchable subontology pair, MS

$\gamma$=0.75, i=1, j=1

**//calculating the share anchor between two sub-ontology pair** (sc1i, sc2j)

for each i in range(0, $ns_1$)
for each j in range(0, $ns_2$)
shared_anchor+=$anchor$(sc1i, sc2j)
end
end

**//total number of anchor between two sub-ontology pair**

for each $sc1_k \epsilon SO_1$
tot1+=$anchor(sc1_k, sc2_j)$
end
for each $anchor(sc1_k, sc2_j)$
tot2+=$anchor(sc1_i, sc2_k)$
end

**//Calculating similarity between two sub ontology pair**

For each subontology in $SO_1$
For each sub-ontology in $SO_2$
So_sim($sc1_i$, $sc2_j$) = $\frac{shared\_anchor}{tot1+tot2}$
If((So_sim($sc1_i$, $sc2_j$)>γ)
MS= MS U ($sc1_i$, $sc2_j$)
end
end

The function *anchor (sc1$_i$, sc2$_j$)* computes the total number of anchors between the sub-ontology *sc1$_i$* and *sc2$_j$* where *sc1$_i$*$\epsilon$ *SO$_1$* and *sc2$_j$*$\epsilon$ *SO$_2$*. Threshold value, α[0,1] is also set to describe the criteria for minimum similarity. If the sub-ontology pair value is greater than the threshold value, they are identified as matchable sub-ontology pair. Due to the discovery of matchable sub-ontology pair, further alignment computation would decrease greatly.

**Second Level Ontology Partitioning**

Once the matchable sub-ontology pair is formed after candidate mapping process, these pairs are provided as an input for second level ontology partitioning to generate *non-overlapping clusters*. In this module, the second level partitioning of each sub-ontology of matchable sub-ontology pairs is done in parallel, thus reducing the computation time of cluster formation. Also, all the further computation is applied only on each pair of matchable sub-ontology rather than comparing each sub-ontology pair to other *(n-1)* sub-ontology pair, where *n* is the sub-ontology pairs generated after mapping process. So, to further partition the matchable sub-ontology pair sc1$_i$ and *sc2$_j$* where *sc1$_i$*$\epsilon$ *SO$_1$* and *sc2i* $\epsilon$ *SO$_2$* respectively in parallel to generate groups of disjoint clusters $x_1, x_2, x_3 \ldots\ldots x_k$, for each sub-ontology such that the cohesion among the entities in the clusters should be high and the coupling between the entities of clusters should be low. Based on this goal, second level partitioning is shown in algorithm 5 and its process is described in detail in the following sections.

***Finding Number of Partition***

Typically, if the objective criterion is not defined then determining the number of partition of a given ontology is done using trial and error technique to find out the optimal number of partitions.

***Entity Score Function***

All the entities in the ontology are ranked based on the entity itself and its neighbors. More is the score of the entity, significant is the entity and hence is chosen as *cluster head*. Score function should be computationally efficient and also effective. So, the *entity score function* to calculate the score of each entity is based on two parameters namely *structural connection* and *closeness centrality*. Each entity's document contains information about these parameters as described previously. If the entity has more surrounding nodes, it means it has more structural connection, thus having more score. Similarly, high is the closeness centrality, high is the score of the entity. Entity score function then calculates the score of each entity based on the given two parameters as shown in Eq. 6.

$$\text{Entity}_{\text{RankScore}} = \text{SC}(e, d) + \text{CC}(e) \quad \ldots (6)$$

where *SC(e,d)* is the structural connection as in Eq.(1) and *CC(e)* is the closeness centrality as in Eq. 2.

***Determining Cluster Head (CH)***

Once the score of each entity is computed by entity score functions, next task is to select the cluster head. If node with the highest score is chosen randomly as cluster head, there would be problem of distribution of the cluster head in the given graph. So to overcome this problem, a minimum of *d* distance is kept between two chosen entities as *cluster head*.

***Non-overlapping Cluster Creation***

In this module, each cluster is assigned under one cluster head, and all its direct sub classes or child nodes are placed in the cluster. The leftover entities

are placed using a membership function, which computes the leftover entity's membership to each cluster head and thereby assign those entities to the right cluster. Due to directly placing the child nodes into the cluster in this module is computationally efficient, as it saves lot of time in comparison and computing membership.

**Algorithm 5: Second Level Partitioning**

**Input:** Set of entities $E$ in sub-ontology after first level partitioning, $n$ number of optimal partitions

**Output**: Set of Cluster X

**STEP 1:** Calculating each entity score using ranking function

$$\text{Entity}_{\text{RankScore}} = \text{SC(e, d)} + \text{CC (e)}$$

**STEP 2:** Determining *'n'* cluster head:Entities having highest RankScore is determined as Cluster head with'd' distance apart.

**STEP 3:** All each cluster is assigned under one cluster head, and all its direct sub classes or children are placed in the cluster.

**STEP 4:** Each leftover entity is compared with Cluster head based on membership function and entity sharing maximum similarity value with the cluster head, is assigned to that particular cluster.

$$Member\ ship\ Function\ (e_i, CH_i) = \alpha \times SC\ (e_i, CH_i) + \beta\ NS\ (e_i, CH_i) + \gamma \times SS\ (e_i, CH_i)$$

**STEP 5:** Set of cluster generated is given as output.

***Membership Function***

At this stage, the direct child nodes of the cluster heads have already been placed in clusters. Next task is to build some membership function which can correctly categorize entity $e_i \epsilon E$ to the cluster $X_i$, $i <= k$. First of all, the remaining entities are assigned a variable *assign* whose default value is *false*, once the membership function is applied to the entity and it is be placed in some cluster, the value of *assign* variable is changed to *true*. Each entity is placed only in one of the cluster which results in non-overlapping clusters. Instead of comparing the leftover entities to all the other entities in the cluster, it is only compared with cluster heads. The similarity value between entities and each cluster heads is calculated using membership function, and entity sharing maximum similarity value with the cluster head, is assigned to that particular cluster. The membership function in Eq. 7 considers combination of three parameters to calculate overall similarity, such as *structural connection, naming similarity and semantic similarity* between the entity $e_i$ and $CH_i$ as follows:

$$MembershipFunction(e_i, CH_i) = \alpha \times SC(e_i, CH_i) + \beta \times NS(e_i, CH_i) + \gamma \times SS(e_i, CH_i) \quad \ldots (7)$$

where $\alpha,\beta,\gamma$ are constants and denote the importance given to each parameter and $\alpha+\beta+\gamma=1$, $SC(e_i,CH_i)$, $NS(e_i,CH_i)$, $SS(e_i,CH_i)$ are the structural connection, naming similarity and semantic similarity between entity and cluster head respectively. The structural connection measures the neighborhood similarity between entity and cluster head. More the number common neighbors both will share, more is the similarity value between entity and cluster head. The naming similarity measures the label or name of entity and corresponding cluster head. Researchers[12] showed that the name of nodes is the most dominant feature. For this purpose, the Levenshtein distance is used, also called string edit distance. The semantic similarity measures the semantic relation shared between entity and the cluster head such as hypernym, hyponym etc.

***Matchable Cluster Pair and Alignment Discovery***

Once the clusters are generated after applying second level partitioning on sub-ontology pairs, next step is to find anchors among them and then at last discovering matchable cluster pair for finding alignment using the same process as proposed in partitioned ontology candidate mapping. However, the anchor discovery is already done using IEI-sub method and the MapReduce framework. Therefore, only matchable cluster pairs are discovered in this section. All the matchable cluster pair generated as output will pass on to the powerful linguistic matcher VDoc[13] and then to GMO matcher[14] for final alignment discovery as done in other researches.[9,10]

**Experimental Results**

To prove the efficiency and the scalability of the proposed multilevel ontology matching system, different sizes of ontologies datasets are taken into account. The small ontology pair such as *Toursim AB* and *Russia 12*, used for partitioning, can be retrieved from http://ws.nju.edu.cn/falcon-ao/. Other large scale ontology taken into account such as FMA–NCI, NCI–SNOMED (40%), FMA–SNOMED (40%) can be downloaded from OAEI (Ontology Alignment

Evaluation Initiative). The enormous computation is required to match these three large pair of ontologies depending upon the number of matching used, more the number of matcher required in matching the ontologies pair, more would be the computation required. The experimental results are divided into number of phases such as F-measure and execution time using partitioning and without partitioning, anchor identification, IEI-Sub matcher and the experiment on precision, recall and F-measure.

**Experiment on Partitioning and without Partitioning**

This experiment demonstrated the requirement for partitioning of large ontology in ontology matching systems. The proposed system is transformed to find the alignment between the input ontology without partitioning. As shown in Fig. 2, MPP-MLO proved to be more efficient with partitioning as compared to without partitioning, although the accuracy of the system is slightly compromised due to the fact that the cartesian product of all the entities are not computed in case of MPP-MLO with partitioning. The computation required in MPP-MLO without partitioning is very high as compared to MPP-MLO with partitioning.

**Experiment on Anchor Identification**

The execution time required for discovering the anchors among the entities pair based on IEI-Sub, EI-Sub, I-Sub and SI-Sub are shown in Table 1. Although the execution time taken by IEI-Sub is almost same as of EI-Sub, and on an average, it is reduced by 13.4% as compared to the execution time taken by I-Sub. It can be clearly inferred that the SI-Sub execution time is less than ISI-Sub as the latter used very naïve similarity but in the following experiment, it is observed that this method is less effective as compared to others. The matchable cluster pair and the matchable sub-ontology pair are identified only on the basis of anchors discovered which further helps in finding final alignment set. Therefore, choosing the right matchable cluster pair is the crucial task for the overall ontology matching system.

Table 1— Execution time comparison for anchor identification of Falcon, LOMPT, PSOM2 and MPP-MLO

| | 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 |
| PSOM2 (EI-Sub) | 39, 919 | 125, 192 | 90, 118 |
| MPP-MLO (IEI-Sub) | 39, 515 | 124, 784 | 89, 798 |

**Experiment on IEI-sub Matcher using MapReduce**

In this experiment, anchor discovery using IEI-Sub over different number of nodes in Hadoop environment and using three large ontologies pair are compared and shown in Fig. 3. It can be observed that

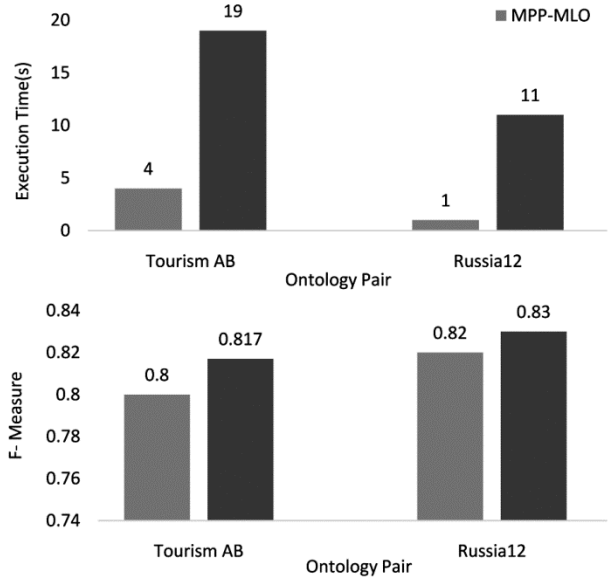


Fig. 2—Execution & F-measure of MPP-MLO and MPP-MLO without partitioning

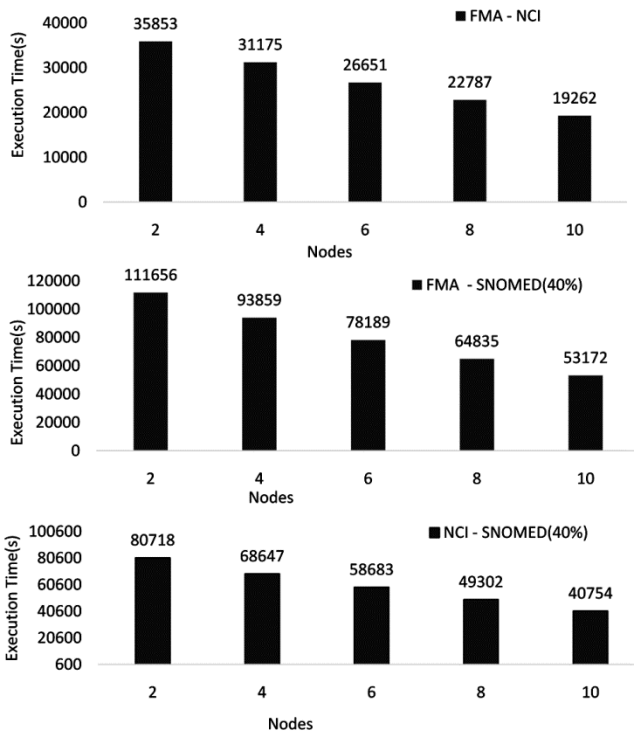


Fig. 3— Execution time by MapReduce based IEI-Sub for anchor identification

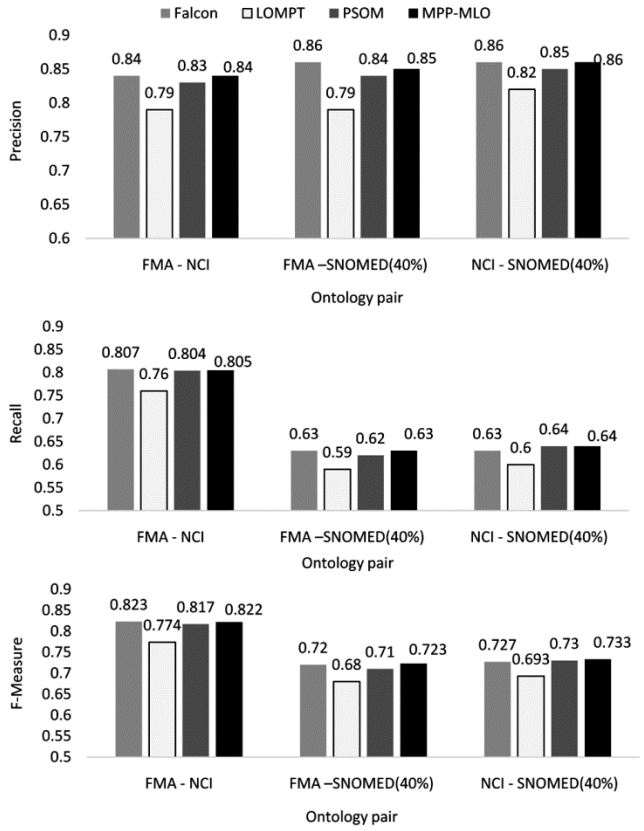


Fig. 4— Precision, Recall and F-Measure of Falcon, LOMPT, PSOM and MPP-MLO

there is almost 51.5% reduction in execution time for FMA–NCI and in case of NCI– SNOMED (40%) and FMA– SNOMED (40%) there is a reduction of 57.5% and 54.8% in execution time respectively. This proves that MPP-MLO achieves reduction in execution time and hence more scalable as compared to other existing ontology matching systems.

**Experiments on Performance Measure**

The main motive of proposing this ontology matching system, MPP-MLO is to achieve high accuracy, better efficiency and the scalability. FMA–SNOMED (40%), FMA–NCI and NCI–SNOMED (40%) are the pair of ontologies used for the evaluation. In this experiment, precision, recall and F-measure are evaluated as shown in Fig. 4. Based on the experiment and the results shown, it can be infer that the MPP-MLO achieve better accuracy as compared to PSOM and the LOMPT, due to multilevel partitioning and the IEI-Sub. Although, MPP-MLO precision is less than Falcon with a small margin, this is due to the fact that Falcon uses robust and high computational linguistic matcher to indentify anchor which in turn contributes in overall findings of alignments. The F-measure also shows that the MPP-MLO is effective than the other ontology matching systems.

Table 2—Comparison of execution time of Falcon, LOMPT, PSOM and MPP-MLO

| | FMA–NCI | FMA–SNOMED (40%) | NCI–SNOMED (40%) |
|---|---|---|---|
| Falcon | 47,745 | 165,480 | 135,694 |
| LOMPT | 38,896 | 133,376 | 116,701 |
| PSOM2 | 24,200 | 67,400 | 52,000 |
| MPP-MLO | 23,987 | 67,174 | 51,875 |

**Experiments on Total Execution Time**

Finally, the total execution times used by all the ontology matching system for finding out the final alignments are compared in Table 2. It is observed that almost 58.9% reduction in the execution time when compared with Falcon and specifically for NCI–SNOMED (40%), the reduction in execution time is around 61.7%. In case of LOMPT, the reduction in execution time is almost 50.3% and specifically for NCI–SNOMED, the reduction in execution time is 55.5%. In case of PSOM, the execution time is almost same, but the accuracy of the proposed system is more. As is can be proved seeing the results that MPP-MLO has better efficiency as compared to others.

## Conclusions

In this work, a novel multilevel parallel partitioning based ontology matching technique is proposed, which targets efficiency and effectiveness over the state of the art ontology matching technique. There is 58.7% reduction in execution time of the proposed system as compared to other existing approaches. In large scale ontology, as the size of cluster should be less for better computation time and space, so this is well achieved using partitioning at two levels without incurring extra overheads. The efficiency and the computational time are increased by 78.9% using concept of partitioning. The proposed system used MapReduce framework to handle the most time consuming process of finding anchor in order to achieve better scalability in parallel and distributed manner. On an average, 54.6% reduction in execution time using MapReduce framework. To discover the anchors set, a light weight and efficient linguistic matcher called IEI-Sub is proposed. The execution time of IEI-Sub is reduced by 13.5% as compared to I-Sub. Also for second level partitioning, non overlapping clusters are formed using score, ranking function and membership function, which increases the quality of the clusters formed.

## References


1 Selvi M S, Deepa K, Sangari M S & Mohankumar B, Improved Structured Robustness (I-SR): A Novel Approach to Predict Hard Keyword Queries, *J Sci Ind Res* **76** (2017) 38–43.

2 Belhadi H, Akli-Astouati K, Djenouri Y & Lin J C W, Data mining-based approach for ontology matching problem, *Appl Intell*, **50(4)** (2020) 1204–1221, doi:10.1007/s10489-019-01593-3.

3 Xue X & Chen J, Using Compact Evolutionary Tabu Search algorithm for matching sensor ontologies, *Swarm Evol Comput*, **48** (2019) 25–30, doi:10.1016/j.swevo.2019.03.007.

4 Stevens R, Lord P, Malone J & Matentzoglu N, Measuring expert performance at manually classifying domain entities under upper ontology classes, *J Web Semant*, **57** (2019) 100469, doi:10.1016/j.websem.2018.08.004.

5 Li Y, Jianhui Z, Liu J & Hou Y, Matching large scale ontologies based on filter and verification, *Math Probl Eng*, **2020** (2020), doi:10.1155/2020/8107968.

6 Laadhar A, Ravat F, Ghozzi F, Teste O, Megdiche I & Gargouri F, Partitioning and local matching learning of large biomedical ontologies, *Proc ACM Symp Appl Comput*, F**1477** (2019) 2285–2292, doi:10.1145/3297280.3297507.

7 Zhuang L, Schouten K & Frasincar F, SOBA: Semi-automated Ontology Builder for Aspect-based sentiment analysis, *J Web Semant*, **60** (2020) 100544, doi:10.1016/j.websem.2019.100544.

8 Mountasser I, Ouhbi B & Frikh B, Hybrid large-scale ontology matching strategy on big data environment, *ACM Int Conf Proceeding Ser*, (2016) 282–287. doi:10.1145/3011141.3011185

9 Hu W, Qu Y & Cheng G, Matching large ontologies: A divide-and-conquer approach, *Data Knowl Eng*, **67(1)** (2008) 140–160, doi:10.1016/j.datak.2008.06.003

10 Sathiya B, Geetha T V & Saruladha K, PSOM2—partitioning-based scalable ontology matching using MapReduce, *Sadhana - Acad Proc Eng Sci*, **42(12)** (2017) 2009–2024, doi:10.1007/s12046-017-0742-5

11 Winkler W E, The state of record linkage and current research problems, *Stat Res Div US Census Bur* (1999) 1–15, doi:10.1.1.39.4336

12 Lin F & Sandkuhl K, A survey of exploiting WordNet in ontology matching, in *IFIP International Federation for Information Processing*, Vol **276** (Springer, Boston, MA) (2008) 341–350. doi:10.1007/978-0-387-09695-7_33

13 Zhang H, Hu W, Qu Y. Constructing virtual documents for ontology matching using mapreduce, in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol **7185** LNCS (Springer, Berlin, Heidelberg) (2012) 48–63, doi:10.1007/978-3-642-29923-0_4

14 Hu W, Jian N, Qu Y & Wang Y, GMO: A graph matching for ontologies, in *CEUR Workshop Proceedings*, Vol **156**, (2005) 41–48.