SeeCOnt: A New Seeding-Based Clustering Approach for Ontology Matching

Alsayed Algergawy[1,2](✉), Samira Babalou[3], Mohammad J. Kargar[3], and S. Hashem Davarpanah[3]

[1] Institute of Computer Science, Friedrich Schiller University of Jena, Jena, Germany
alsayed.algergawy@uni-jena.de
[2] Department of Computer Engineering, Tanta University, Tanta, Egypt
[3] Department of Computer Engineering, University of Science and Culture, Tehran, Iran

Abstract. Ontology matching plays a crucial role to resolve semantic heterogeneities within knowledge-based systems. However, ontologies contain a massive number of concepts, resulting in performance impediments during the ontology matching process. With the increasing number of ontology concepts, there is a growing need to focus more on large-scale matching problems. To this end, in this paper, we come up with a new partitioning-based matching approach, where a new clustering method for partitioning concepts of ontologies is introduced. The proposed method, called *SeeCOnt*, is a seeding-based clustering technique aiming to reduce the complexity of comparison by only using clusters' seed. In particular, *SeeCOnt* first identifies and determines the seeds of clusters based on the highest ranked concepts using a distribution condition, then the remaining concepts are placed into the proper cluster by defining and utilizing a membership function. The *SeeCOnt* method can improve the memory consuming problem in the large-scale matching problem, as well as it increases the matching quality. The experimental evaluation shows that *SeeCOnt*, compared with the top ten participant systems in OAEI, demonstrates acceptable results.

Keywords: Ontology matching · Clustering techniques · Large-scale matching

1 Introduction

Ontology is the main backbone of the Semantic Web, which provides facilities for integration, searching, and sharing of information on the web through making those information understandable for machines [14]. Despite this crucial role and due to the engineering of ontologies by different people or methods even if they are created for the same domain, there exist different sorts of heterogeneities. Semantic heterogeneity is a common and key problem in different knowledge-based systems [5,6]. To obtain meaningful interoperation, one needs a semantic mapping among ontologies. To cope with the semantic heterogeneity problem, a

T. Morzy et al. (Eds.): ADBIS 2015, LNCS 9282, pp. 245–258, 2015.
DOI: 10.1007/978-3-319-23135-8_17

set of correspondences that identify similar concepts across different ontologies have to be achieved through ontology matching. The construction of manual match is an error-prone and labor intensive task that requires complete knowledge of the semantics of the data in ontologies being matched. Solutions that try to provide some automatic support for ontology matching have received steady attention over the years [3,8,19,21].

Nowadays, there is a natural evolution of data, and consequently, there exist many complex and large-scale ontologies in the real domains. For example, the Foundational Model of Anatomy (FMA), SNOMED CT, and the National Cancer Institute Thesaurus (NCI) ontologies are semantically rich and contain tens of thousands of concepts[1]. However, to process these large-scale ontologies, the existing ontology matching tools have some problems, such as shortage of consumed memory and/or long time consumption [21]. For example, the OAEI campaign started in 2006 including the anatomy matching task as an evaluation criterion for large-scale matching. The anatomy matching task is to match the Adult Mouse Anatomy (2744 concepts) and the NCI Thesaurus (3304 concepts). However, in 2011, only 6 of 16 systems could process those ontologies. With the increasing number of concepts typical ontologies have, the OAEI campaign included a new match track, called *Large Biomedical Ontologies*. In 2014, only three matching systems could complete the total matching tasks in this track [22].

In order to enable matching of large-scale ontologies, dividing the ontologies into a set of partitions is a way which has been proposed so far via the methods such as divide and conquer [15], clustering [1], and modularization [23]. We argue that partitioning input ontologies plays a central role towards building an effective and efficient matching system. To this end, in this paper, we introduce a seeding-based clustering approach for partitioning ontologies, called *SeeCOnt*. In particular, input ontologies are parsed and represented as concept graphs. We then develop a *Ranker* function to rank ontology concepts exploiting concept graph features. The highest ranked concepts are then selected to constitute the cluster seeds. To assign remaining concepts to their proper clusters, we introduce a membership function. We demonstrate that *SeeCOnt* reduces the complexity of the comparisons by comparing concepts with only seeds instead of all the other concepts. Finally, we adapt the Falcon-AO matching system to apply our new approach. To validate the proposed approach, we conducted a set of experiments utilizing different data sets from OAEI. The experimental results display that *SeeCOnt* achieved acceptable performance compared with the top-ten matching systems participating recently in OAEI.

The rest of the paper is structured as follows. Related work is presented in Sect. 2. We describe the proposed approach Sect. 3. We report experiments conducted and analysis results in Sect. 4. Section 5 concludes the paper.

[1] http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/2014/.

2 Related Work

Because its importance, several approaches have been proposed to deal with the problem of matching two large ontologies [1,4,13,15,18,20,23]. Promising areas for large-scale matching lie in four main directions: reduction of search space for matching, parallel matching, self-tuning match workflows and reuse of previous match results [18]. In this section, we pay attention to the approaches that perform reduction of the search space. The standard approach of cross join evaluation for ontology matching reduces the matching quality and matching efficiency. In order to reduce the search space for matching, two methods can be used; early pruning of dissimilar element pairs and partition-based matching.

In general, partitioning-based matching has three main stages: (i) partitioning each input ontology into a set of sub-ontologies and determining similar partitions between two sets to form a matching task, (ii) applying a matching method to each matching task to produce a set of partial match results, and (iii) combining partial results to get the final match result. In the following we present a set of matching systems that follow this architecture.

Quick ontology matching (QOM) was one of the first approaches to implement the idea of early pruning of dissimilar element pairs [7]. It iteratively applies a sequence of matchers and can restrict the search space for every matcher. COMA++ was one of the first systems to support partition-based schema matching [4]. It depends on fragment matching which determines fragments of the two schemas and identifies the most similar ones.

Another matching system that supports partition-based matching is Falcon-AO [15]. It initially partitions the ontologies into relatively small disjoint blocks by using structural clustering. Then, matching is applied to the most similar blocks from the two ontologies. Dynamic partition-based matching is supported by AnchorFlood [20]. It avoids the a-priori partitioning of the ontologies by utilizing anchors (similar concept pairs). It takes them as a starting point to incrementally match elements in their structural neighborhood until no further matches are found or all elements are processed. Thus the partitions (segments) are located around the anchors.

Zhong et al. propose an unbalanced ontology matching approach, which concerns with matching a lightweight ontology with a more heavyweight one [25]. Algergawy et al. uses a clustering-based matching approach that is based on an agglomerative bottom-up hierarchical fashion [1]. The clustering scheme is performed based on the context-based structural node similarities. Then, a light weight linguistic technique is used to find similar partitions to match.

3 SeeCOnt

To cope with matching large ontologies, we present a new seed-based clustering approach, called *SeeCOnt*. As shown in Fig. 1, *SeeCOnt* consists of three components: *preprocessing*, *ranking*, and *clustering*. First, input ontologies are parsed and represented internally as labelled directed graphs, called *concept* *graphs*. During the preprocessing step, the number of cluster heads ($\mathcal{CH}$) is to be determined. To quantify the importance of a concept in the concept graph, we introduce a new function called *Ranker* exploiting the concept graph features. Finally, remaining concepts are placed into their corresponding clusters according to a proposed membership function. The outline of the *SeeCOnt* approach is shown in Algorithm 1. In the following sections, we portray the description of each phase of the algorithm.

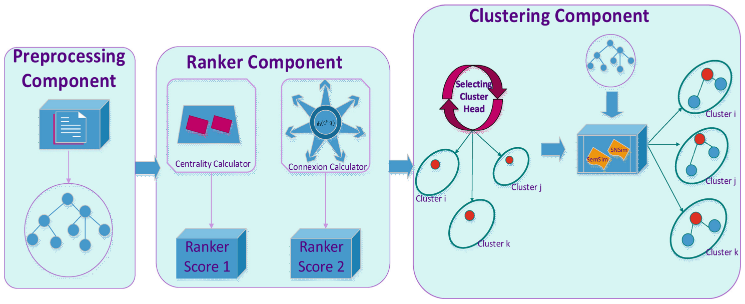

Fig. 1. Architecture of the *SeeCOnt* method.

3.1 Preprocessing

First input ontologies are parsed and inferred by Apache Jena[2] and then the concept graph is drawn by mapping the inferred result. We define *concept graph* $\mathcal{G} = (\mathcal{C}, \mathcal{R}, \mathcal{L})$ as a labeled directed graph. $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ is a finite set of nodes presenting the concepts of the ontology. $\mathcal{R} = \{r_1, r_2, ..., r_m\}$ stands for a finite set of directed edges showing various relationships between concepts in an ontology $\mathcal{O}$, such that $r_k \in \mathcal{R}$ represents a directed relation between two adjacent concepts $c_i, c_j \in \mathcal{C}$. $\mathcal{L}$ is a finite set of labels of graph nodes defining the properties of each concept, such as the names of concepts. $n(= |\mathcal{C}|)$ and m are the number of nodes (concept) and edges (relationship) in $\mathcal{G}$, respectively. Given the number of concepts in an ontology, the number of cluster heads (i.e. $\mathcal{K}$) can be computed according to the following equation.

$$\mathcal{K} = \frac{|\mathcal{C}|}{\epsilon} \tag{1}$$

where ϵ is the maximum size of each cluster ($\epsilon < |\mathcal{C}|$) and should be set by an expert depending on the number of concepts.

Once each ontology is represented as a concept graph, the next step is to partition concepts, $\mathcal{C}$, of each graph into a set of separate (disjoint) clusters $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k$ such that the cohesion of nodes in one cluster should be high, while the coupling of two clusters is low.

[2] https://jena.apache.org/.

Algorithm 1. Seeding-based clustering algorithm

```
Require: An ontology $\mathcal{O}$, a parameter $\epsilon$ limiting the maximum number of concepts
    in a cluster
Ensure: A set of clusters, $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k\}$
    {// Phase 1: Preprocessing}
 1: $\mathcal{T} \Leftarrow \emptyset$;
 2: $\mathcal{CG} \Leftarrow parse(\mathcal{O})$;
 3: $\mathcal{K} \Leftarrow \frac{|\mathcal{CG}|}{\epsilon}$;
    {// Phase 2: Concept Ranking}
 4: for $c_i \in \mathcal{C}$ do
 5:     $score_{c_i} \Leftarrow compute\_Ranker\_score(c_i)$;
 6: end for
 7: $\mathcal{CH} \Leftarrow select\_top\_rank(\mathcal{C})$;
    {//Phase 3: Clustering}
 8: initialize each cluster with each $\mathcal{CH}$ node;
 9: add each direct concept of $\mathcal{CH}$ to each cluster;
10: for $non - clustered\ c_i \in \mathcal{C}$ do
11:     $max\_sim \Leftarrow 0$;
12:     for $c_j \in \mathcal{CH}_k$ do
13:         $sim_{ij} \Leftarrow MemFun(c_i, c_j)$;
14:         if $sim_{ij} \geq max\_sim$ then
15:             $max\_sim \Leftarrow sim_{ij}$;
16:             $concept\_place \Leftarrow j$;
17:         end if
18:     end for
19:     $Clust.put(C_i, \mathcal{CH}_j)$
20: end for
```

3.2 Concepts Ranking

The seeding-based clustering algorithm starts by selecting a set of nodes distinguishing as *important nodes*. These nodes are then selected to be cluster heads, $\mathcal{CH}$. In order to identify a node as an important one, we should first quantify its role in the concept graph. To this end, we introduce a new function, called *Ranker*. This function should be as simple as possible but effective. I.e. the *Ranker* function should not consume much time, however, correctly rank concepts inside an ontology, given that we deal with the large-scale matching problem.

Ranker Function. The importance of a node in a concept graph is understandable through the node itself and its surroundings [11]. This matter leads us to use graph-theoretic measures based on graph connections in the *Ranker* function. In the following, we present two different implementation of the *Ranker* function. The first is based on the *centrality* measure of a concept, while the second depends on the context of the concept.

First Rank Function. The definition of "centrality" measure of a concept in a concept graph is derived from the social network analysis [9]. Each person is given a score based on his or her position at the network showing the importance of each individual. To consider the effect of the concept itself through its edges, we use a set of centrality measures, as given below.

- Degree Centrality ($C1$): This measure is the simplest measure that calculates the number of connections of a node. In a directed graph, there is an in-degree and out-degree centrality that calculates the number of input and output links, respectively. The relationships between nodes can be considered as a power source during concept ranking; nodes with high degree of centrality are certainly more prominent than the others, since they receive a great deal of power [16].
- Closeness Centrality ($C2$): This measure shows the importance of the close nodes to the others in the graph. In this measure, reaching cost of a node to the others is measured [16].
- Betweenness Centrality ($C3$): This measure is considered the most relevant in that context. It consists in computing on each node the fraction of shortest paths that pass through it [10].
- EcCentrality ($C4$): This measure calculates the maximum distance between pairs of nodes. The intuition is that one node is the central if no node is far from it [12].
- Stress Centrality ($C5$): This measure calculates the absolute number of the shortest paths through a node [17].

A summary of these centrality measures and their descriptions are shown in Table 1.

Table 1. Different Centrality Measures.

No.	*Name*	*Formula*	*Description*
1	Degree Centrality	$C1(c_i) = degreeCentrality(c_i)$	-
2	Closeness Centrality	$C2(c_i) = \frac{1}{\sum_{c_j \in C} distance(c_i,c_j)}$	distance (c_i,c_j) function is the shortest path between i and j nodes in the graph
3	Betweeness Centrality	$C3(c_i) = \sum_{s,t \in c_i} \frac{\sigma_{s,t}(c_i)}{\sigma_{s,t}}$	$\sigma_{s,t}(c_i)$ is the number of shortest paths from s to t through c_i, and $\sigma_{s,t}$ is the total number of shortest paths from s to t
4	EcCentrality	$C4(c_i) = \frac{1}{\max_{c_j \in C} distance(c_i,c_j)}$	-
5	Stress Centrality	$C5(c_i) = \sum_{s,t \in c_i} \sigma_{s,t}(c_i)$	$\sigma_{s,t}(c_i)$ represents the number of the shortest paths from s to t via c_i

The arising question now is which centrality measure(s) should be used to implement our *Ranker* function. During the selection process we need to optimize between two criteria: an accurate and fast measure. To this end, and based on our experimental results shown later, we select the combination of the *Degree, C1* and *closeness, C2* centrality measures. As a result we can formulate the score of the first *Ranker* function for a given concept, c_i, as below:

$$Ranker\_score_1(c_i) = C1(c_i) + C2(c_i) \tag{2}$$

Second Rank Function. During the employment of the first *Ranker* function, we observed that it is an effective measure but it requires a lot of time to rank concepts. This makes it unsuitable for matching large ontologies. Therefore, we propose another rank function that should be more applicable to large-scale matching. First, we introduce the definition of the concept *connexion set*, then show how to use this set to determine the importance of the concept.

Definition 1. *Given a concept graph $\mathcal{G} = (\mathcal{C}, \mathcal{R}, \mathcal{L})$, the connexion set of a concept $c_i \in \mathcal{C}$ is defined as: $\Psi(c_i, d) = \{SubClass(c_i, d) \cup SuperClass(c_i, d)\}$*

where $\Psi(c_i, d)$ is a set in which all the concepts with d levels that effect on c_i node. $SubClass(c_i, d)$ is the children of c_i with d hierarchical levels, and $SuperClass(c_i, d)$ is the parents of c_i with d hierarchical levels. It is evident that the importance of a concept increases as it has a larger number of surroundings. Based on this we propose the following score function that can be used to rank concepts of an ontology.

$$Ranker\_score_2(c_i) = |\Psi(c_i, d)| \tag{3}$$

Determining Cluster Heads. Once computing the importance of concepts of a concept graph, the next step is to select which concepts represent cluster heads, $\mathcal{CH}$. If simply the nodes with the highest score are selected as the cluster heads, distribution would be disregarded. To avoid this problem, the distance between two cluster heads is measured, and among the highest score nodes, those with a minimum distance of d from each other are selected as the cluster heads. For this purpose, we adopt the *Connexion* set with d levels defined before.

3.3 Finalizing Clustering

At first, the *SeeCOnt* algorithm creates one cluster for each cluster head. Then, it places direct children in the corresponding cluster and finally, for remaining nodes, a membership function is used to determine the cluster of each node. In general, clustering is done through the following three steps:

- *Seeding*: Creating a cluster for each cluster head, *Algorithm 1, line 8.*
- *Direct Spread*: Assigning direct children of each cluster head to the corresponding cluster, *Algorithm 1, line 9.*
- *In-Direct Spread*: Calling a membership function for the remaining nodes, *Algorithm 1, lines 10–18.*

The *direct Spread step* reduces the time complexity, since the number of comparisons will be reduced as well as applying the membership function for all nodes is time consuming. While by placing the nodes via the call of the membership function, the same results would still be achieved.

Membership Function. Once determining cluster heads, ($\mathcal{CH}$), and assigning direct children to their proper heads, the next step is to place remaining concepts into their fitting cluster. To this end, we develop a membership function, *MemFun*. First, each concept is associated with a flag, $\mathcal{F}$, such that if the $\mathcal{F}$ of c concept is false, it means c is not assigned to any cluster and thus, the membership function is called for the concept c. In addition, the $\mathcal{F}$ flag can only have one value, i.e. each node can be placed in only one cluster so that no overlap is observed in clusters. The membership function determines that each concept $c_i \in \mathcal{C}$ should be placed in which $\mathcal{T}_i, i < \mathcal{K}$ cluster. For this, the similarity of c_i with all $\mathcal{CH}_i$ is calculated and then c_i is placed in a cluster with the maximum similarity value. Using the proposed membership function, each concept is compared with Cluster Heads, instead of comparing with all concepts like whatever was done in [1,15], which reduces the complexity of comparison.

In order to measure the membership of a concept to a cluster head, a linear weighted combination of the following structural and semantic similarity measures is calculated as in the following equation:

$$\begin{aligned} MemFun(c_i, \mathcal{CH}_k) = {} & \alpha \times SNSim(c_i, \mathcal{CH}_k) \\ & + (1 - \alpha) \times SemSim(c_i, \mathcal{CH}_k) \end{aligned} \tag{4}$$

where α is constant between 0 and 1 to reflect the importance of each similarity measure, $ShareNeighbors(SNSim)$ and semantic similarity $SemSim$ are two similarity measures that quantify the structural properties of the concept c_i, respectively.

Shared Neighbors. This measure considers the number of shared neighbors between c_i and $\mathcal{CH}_k$. The shared neighbour measure plays an important role in structural similarity, because similar concepts have similar neighbors [2,24]. The neighbors of a concept are the concept's children, concept's parents, concept's siblings and the concept itself. In our implementation, we determine the neighbors of the concept c_i and the neighbors of the cluster head $\mathcal{CH}_k$, then determine how many concepts are common between these two sets.

$$SNSim(c_i, \mathcal{CH}_k) = \frac{|SN_{c_i} \cap SN_{\mathcal{CH}_k}|}{|SN_{c_i} \cup SN_{\mathcal{CH}_k}|} \tag{5}$$

where SN_{c_i} and $SN_{\mathcal{CH}_k}$ are the neighbor sets of the concept c_i and the cluster head $\mathcal{CH}_k$, respectively.

Hierarchical Semantic Similarity. It is evident that a higher semantic similarity implies a stronger semantic connection, so we first calculate the semantic similarities between the concept c_i and the cluster head $\mathcal{CH}_k$. The most classic semantic similarity calculation is based on the concept hierarchy. The hierarchy semantic similarity between c_i and $\mathcal{CH}_k$ can be defined as below:

$$SemSim(c_i, \mathcal{CH}_k) = \frac{2 \times N_3}{N_1 + N_2 + 2 \times N_3} \tag{6}$$

where N_1 and N_2 are the numbers of sub-concept relations from c_i and ,$\mathcal{CH}_k$ to their most specific common superconcept C, and N_3 is the number of sub-concept relations from C to the root of the concept hierarchy.

Matching. Once settling on the similar clusters of the two ontologies, the next step is to fully match similar clusters to obtain the correspondences between their elements. Each pair of the similar clusters represents an individual match task that is independently solved. Match results of these individual tasks are then combined in‘to a single mapping, which represents the final match result. We adopt the Falcon matching system [15] to perform this task.

4 Experimental Evaluation

To develop *SeeCOnt*, the open source Falcon-AO system[3] was used. It was implemented in Java with Apache 2.0 license. Falcon-AO has some components including PBM (Partition Block Match) [24]. PBM is used for large-scale ontology matching which was replaced by the *SeeCOnt*. All the experiments were carried out on Intel core i5 with 4 GB internal memory on Windows 7 with Java compiler 1.7. Ontologies were parsed using Jena Apache, and the mapping functions were implemented by Alignment API[4].

4.1 Data Set

We tested with two common data sets from the OAEI[5]: Conference and Anatomy. The conference data set containing 16 ontologies is much used in ontology matching evaluation. The Anatomy data set contains two ontologies of human and mouse anatomy with 3306 and 2746 concepts, respectively.

4.2 Evaluation Criteria

In our implementation, we attempt to get answers to the following questions:

- Which centrality measure should be used to implement the first *Ranker* function?
- Which ranker function should be used to implement the ranker component?

[3] http://ws.nju.edu.cn/falcon-ao.
[4] http://alignapi.gforge.inria.fr.
[5] http://oaei.ontologymatching.org.

– What is the relative performance of the *SeeCOnt* approach w.r.t. recent matching approaches?

In order to answer these questions, we carried out sets of experimental evaluations. In the following, we report on the answers of these questions.

4.3 Experimental Results

Centrality Measure Evaluation. We conducted the first set of experiments to decide which centrality measure(s) should be used to implement the first *Ranker* function. To this end, we performed an evaluation using three different ontologies: *Linkling*, *MICRO*, and *cmt* from the Conference dataset. In this set, all 32 combinations of the five centrality measures were assessed. We asked a number of experts to select the top ten important concepts while we did not say anything about our criteria to them. Due to differences between important concepts by experts, we selected the most common shared important concepts. The results of 32 combinations of these criteria on Linkling ontology are shown in Fig. 2, where each bar is dedicated to one combination of different criteria, C1 refers to Degree Centrality, C2 is Clossness Centrality, C3 is Betweenness Centrality, C4 is EcCentrality, and C5 is Stress Centrality. In each test we use one combination of C1–C5 criteria and select top ten important concepts, we also examine how many of these criteria are similar to expert judge. The test examines which combination was more similar to the experts view. Based on our criteria, different sets of combinations could achieve these criteria, such as the combinations C1+C2, C1+C4, C1+C2+C5, C1+C2+C3+C4 are more similar to whatever experts think. From these combinations, we selected C1+C2 because it outperforms the other combinations.

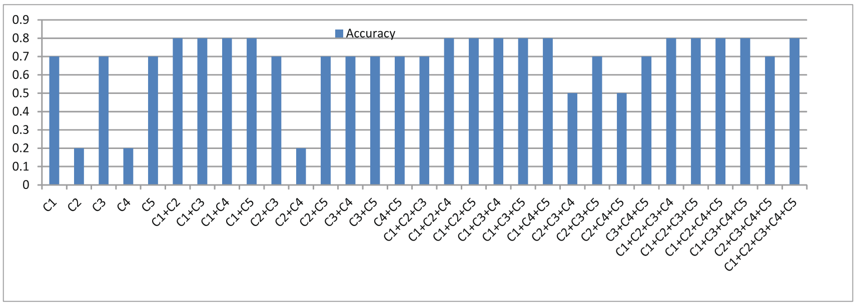

Fig. 2. Accuracy of combining centrality measures on *"Linkling"* ontology.

Selection of Ranker Function. In the previous experiments, we observed that the centrality measures effectively quantify the importance of ontology concepts. However, they consume much time, which makes using them for large-scale matching impractical. Therefore, we conducted another set of experiments to recognize which ranker function is suitable to the large matching problem. To this end, we compare the two ranker functions, implemented in Eqs. 2 and 3, respectively. We used the conference and anatomy ontologies for this test. Results are reported in Table 2. The table shows that the second ranker function outperforms the first one w.r.t. both matching quality and the time needed to complete the ranking process. This can be explained as the second ranker function exploits the concept connections which mostly reflects the importance of the concept without going into much details through computing the importance score as in the first ranker function. Therefore, we settle on the selection of the second ranker function to implement the ranker component.

Table 2. Comparing two ranker functions.

	ontology	First ranker	Second ranker
matching quality	conference	0.609	0.624
time (sec.)	anatomy/mouse	269.6	0.302
	anatomy/nci	355.8	0.57

***SeeCOnt* Quality.** This set of experiments has been conducted to validate the effectiveness and quality of the *SeeCOnt* approach. To this end, we use ontologies from the Conference and Anatomy data sets comparing *SeeCOnt* with Falcon-AO [15] and work done in [1]. Results are presented in Figs. 3 and 4. Figure 3 demonstrates that *SeeCOnt* produces higher precision, recall, and F-measure than the original Falcon-AO system. It could improves the quality of matching on the conference track by 7 % compared to the original Falcon system. However, as shown in Fig. 4, even if the *SeeCOnt* approach produces lower precision than the precision in both Algergawy' approach [1] and the original Falcon approach [15], however it achieves higher recall than the other two systems. This results in the F-measure produced by *SeeCOnt* is higher than the Algergawy' approach by 11 % and Falcon-AO by 14 %. These results demonstrate that our proposed

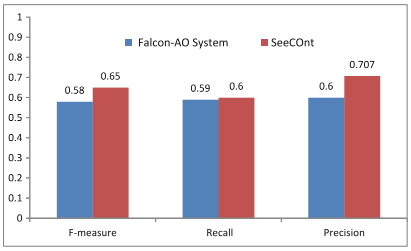

Fig. 3. Results for conference track.

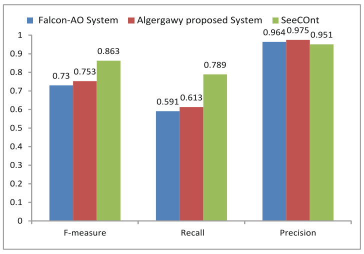

Fig. 4. Results for anatomy track.

seed-based clustering is capable of grouping similar concepts in one partition to be fully matched with another partition containing also similar concepts.

Figures 5 and 6 compare *SeeCOnt* with the top-ten matching systems, participating in OAEI competition held in 2011–2014, in the Conference test and in the Anatomy test, respectively. For simplicity of the chart, only F-Measure of each system is shown in Figs. 5 and 6. The horizontal axis shows the participating systems and the vertical axis shows F-measure. We see that *SeeCOnt* approach has comparable results with the others.

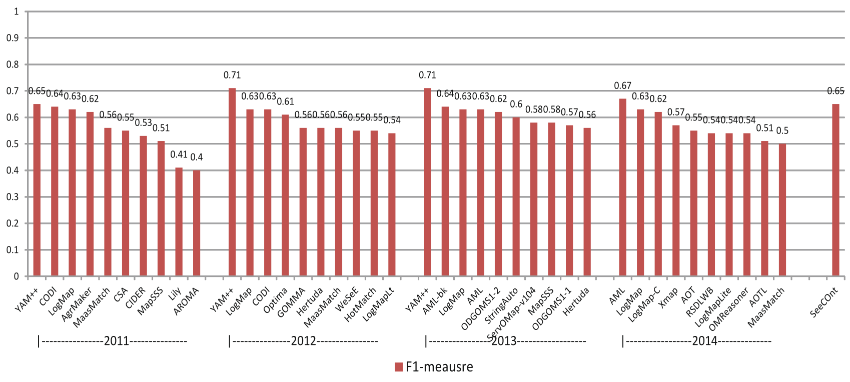

Fig. 5. Comparing *SeeCOnt* with top-ten systems Participating in OAEI Competitions in 2011–2014 in the Conference Test.

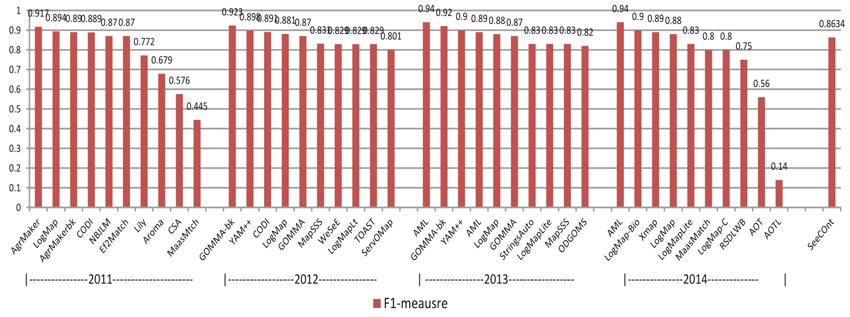

Fig. 6. Comparing *SeeCOnt* with top-ten Systems Participating in OAEI Competitions in 2011–2014 in the Anatomy Test.

5 Conclusions

In this paper, we introduced a new clustering approach, *SeeCOnt*, to be used within the context of matching large ontologies. *SeeCOnt* partitions a large-scale ontology to several disjoint sub-ontologies and the problem of large-scale ontology matching is converted into a set of small ontology matching tasks. Firstly, we represented input ontologies as concept graphs. We then introduced two different *Ranker* functions that can be used to quantify the importance of a concept in the graph. The highly important concepts are selected to be cluster heads. We further developed a new membership function that assign remaining concepts to their proper clusters. This membership function reduces the number of comparisons since it only compares each concepts to each cluster head. To validate *SeeCOnt*, we conducted an intensive set of experiments using the Conference and Anatomy data sets. We compared our proposed approach with recent matching systems participating in OAEI. Experimental results show that *SeeCOnt* presents acceptable performance. In the future, we plan to extend our work by looking for new strategies that determine and identify similar clusters and that match those similar partitions in parallel.

Acknowledgments. A. Algergawy'work is partly funded by DFG in the INFRA1 project of CRC AquaDiva.

References

1. Algergawy, A., Massmann, S., Rahm, E.: A clustering-based approach for large-scale ontology matching. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 415–428. Springer, Heidelberg (2011)
2. Algergawy, A., Nayak, R., Saake, G.: Element similarity measures in XML schema matching. Inf. Sci. **180**(24), 4975–4998 (2010)
3. Bellahsene, Z., Bonifati, A., Rahm, E.: Schema Matching and Mapping. Springer, Heidelberg (2011)
4. Do, H.H., Rahm, E.: Matching large schemas: approaches and evaluation. Inf. Syst. **32**(6), 857–885 (2007)
5. Doan, A., Halevy, A.: Semantic integration research in the database community: A brief survey. AAAI AI Mag. **25**(1), 83–94 (2005)
6. Doan, A., Halevy, A.Y., Ives, Z.G.: Principles of Data Integration. Morgan Kaufmann, USA (2012)
7. Ehrig, M., Staab, S.: QOM – quick ontology mapping. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 683–697. Springer, Heidelberg (2004)
8. Euzenat, J., Shvaiko, P.: Ontology Matching, 2nd edn. Springer, Heidelberg (2013)
9. Freeman, L.C.: Centrality in social networks conceptual clarification. Soc. Netw. **1**(3), 215–239 (1979)
10. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40**(1), 35–41 (1997)
11. Graves, A., Adali, S., Hendler, J.: A method to rank nodes in an RDF graph. In: 7th International Semantic Web Conference (Posters and Demos) (2008)
12. Hage, P., Harary, F.: Eccentricity and centrality in networks. Soc. Netw. **17**, 57–63 (1995)
13. Hamdi, F., Safar, B., Reynaud, C., Zargayouna, H.: Alignment-based partitioning of large-scale ontologies. In: Guillet, F., Ritschard, G., Zighed, D.A., Briand, H. (eds.) Advances in Knowledge Discovery and Management. SCI, vol. 292, pp. 251–269. Springer, Heidelberg (2010)

14. Hendler, J.: Agents and the semantic web. IEEE Intell. Syst. J. **16**, 30–37 (2001)
15. Hu, W., Qu, Y., Cheng, G.: Matching large ontologies: A divide-and-conquer approach. DKE **67**, 140–160 (2008)
16. Kermarrec, A.-M., Merrer, E.L., Sericola, B., Trdan, G.: Second order centrality: Distributed assessment of nodes criticity in complex networks. Comput. Commun. **34**, 619–628 (2011)
17. Koschützki, D., Lehmann, K.A., Peeters, L., Richter, S., Tenfelde-Podehl, D., Zlotowski, O.: Centrality indices. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418, pp. 16–61. Springer, Heidelberg (2005)
18. Rahm, E.: Towards large-scale schema and ontology matching. In: Bellahsene, Z., Bonifati, A., Rahm, E. (eds.) Data-Centric Systems and Applications, vol. 5258, pp. 3–27. Springer, Heidelberg (2011)
19. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
20. Seddiquia, M.H., Aono, M.: An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. Web Semant. **7**(4), 344–356 (2009)
21. Shvaiko, P., Euzenat, J.: Ontology matching: State of the art and future challenges. IEEE Trans. Knowl. Data Eng. **25**(1), 158–176 (2013)
22. Shvaiko, P., Euzenat, J., Mao, M., Jimnez-Ruiz, E., Li, J., Ngonga, A.: editors. 9th International Workshop on Ontology Matching collocated with the 13th International Semantic Web Conference (ISWC 2014) (2014)
23. Wang, Z., Wang, Y., Zhang, S.-S., Shen, G., Du, T.: Matching large scale ontology effectively. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 99–105. Springer, Heidelberg (2006)
24. Hu, W., Zhao, Y., Qu, Y.: Partition-based block matching of large class hierarchies. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 72–83. Springer, Heidelberg (2006)
25. Zhong, Q., Li, H., Li, J., Xie, G.T., Tang, J., Zhou, L., Pan, Y.: A Gauss function based approach for unbalanced ontology matching. In: the ACM SIGMOD International Conference on Management of Data, (SIGMOD 2009), pp. 669–680 (2009)