Journal of Universal Computer Science, vol. 25, no. 7 (2019), 816-839
submitted: 30/8/18, accepted: 9/5/19, appeared: 28/7/19 © J.UCS

Improving Ontology Matching Using Application Requirements for Segmenting Ontologies

Diego Pessoa, Ana Carolina Salgado, Bernadette Farias Lóscio
(Center of Informatics, Federal University of Pernambuco (UFPE), Recife, Brazil
{derp,acs,bfl}@cin.ufpe.br)

Abstract: Ontology matching is concerned with finding relations between elements of different ontologies. In large-scale settings, some significant challenges arise, such as how to achieve a reduction in the time it takes to perform matching and how to improve the quality of results. Current techniques involve the use of ontology segmentation to overcome having such a large number of elements to compare. However, current methods usually select the most relevant ontology elements based on the number of relationships, which may dismiss some elements should they have fewer or no relationships. Therefore, we propose an algorithm for ontology segmentation based on application requirements, in such a way that the users can specify the concepts that are the most relevant in their application context to generate the segments which will be used as an input for the matching. In the experiments, we found a general reduction in the execution time and some significant quality improvements, depending on what matcher is applied. In order to assess the proposed algorithm, we considered some well-known evaluation measures, such as precision, recall, and F-Measure.

Key Words: Segmented-based Ontology Matching; Ontology Segmentation Algorithm; Goal-Oriented Ontology Matching; Semantic Web.
Category: H3.3, M.1, M.5

1 Introduction

Ontology Matching has been widely studied in the last few years, and consequently many Ontology Matching Systems (matchers) have been proposed in the literature ([Shvaiko and Euzenat, 2013, Otero-Cerdeira et al., 2014]). The results of Ontology Alignment Evaluation Initiative (OAEI) campaigns have demonstrated that matchers can achieve different performances depending on the ontologies considered [Achichi et al., 2017]. It therefore becomes challenging for a user to select the most suitable matcher for a particular matching task. This, in practice, increases the need for an automatic approach to select, combine and tune matchers.

On the other hand, it is also important to note that ontologies may be very heterogeneous. For example, it is common to encounter different terms and structures being used to describe the same real-world concepts. These differences are known as semantic heterogeneities, which make it necessary to match ontologies with a view to discovering relationships between the elements [Euzenat and Shvaiko, 2013]. An example of a relationship is equivalence, which stands for two different ontology elements representing the same real-world

concept (e.g., the classes Author and Creator or the properties dateCreation and created).

Furthermore, the advent of the Semantic Web has resulted in the demand for large-scale matchings, i.e., when one needs to match several ontologies which may contain tens (and even hundreds) of thousands of classes and properties [Rahm, 2011]. One example is the recent effort at mining the web in order to extract entities, relationships, and ontologies to build general-purpose knowledge bases such as DBpedia [Lehmann et al., 2015] and Google Knowledge Graph [Dong et al., 2014]. The integration of such entities shows promise for improving applications such as web-search and web-scale data analysis.

In such cases, due to the high number of ontologies to align, it is necessary to make a vast computational effort to find correspondences among several items. Moreover, an enormous human effort may be required to validate the suggested correspondences between these elements. As there are several ontology matchers available, their configuration and execution may be a complicated and time-consuming task for the user, so much so that the correct parameters and their respective effect on the quality of matching can only be asserted by using a trial-and-error process [Peukert et al., 2011].

One possible strategy to reduce the matching search space is to split ontologies into smaller segments [Rahm, 2011]. Two kinds of approaches to perform this process are (i) ontology summarization [Pouriyeh et al., 2018] and (ii) ontology segmentation [Seidenberg and Rector, 2006]. The former consists of splitting the ontology based on the semantic relationship of its elements. This strategy breaks ontologies into smaller parts with the aim of facilitating their use and maintenance or even to build a summary containing their central concepts. The latter handles the reduction by taking only elements related to some information provided by a specific application (e.g., [Noy and Musen,]). This approach is useful to identify the most relevant parts of an ontology based on a user's needs. In this paper, we propose an ontology segmentation algorithm which uses Application Requirements [See Section 4.1] as a way of acquiring the relevant information that is applied to generating the segments.

As to the problem of ontology segmentation for matching large-scale ontologies, we highlight the following main challenges:

(i) How to formalize application requirements in such a way that the definition of the most relevant elements will be considered when generating ontology segments?

(ii) How to reduce the search scope of ontologies so as to compare only the most relevant subset of ontology elements?

(iii) How to reduce the resulting correspondences so as to present only those related to concepts that are of interest to the user?

Regarding issue (i), we separate application requirements into two categories: Data Requirements (DR) and Operational Requirements (OR). DR are represented as a set of keywords which indicate the most relevant concepts that will be considered when generating ontology segments. OR are represented as a set of parameters that affect how the segments are generated. Regarding issues (ii) and (iii), we propose a novel ontology segment algorithm that will be used to define the set of requirements and, which will generate as an output, reduced versions of ontologies containing only the relevant elements. By doing so, we seek to reduce the scope of the matching search and consequently to generate correspondences more efficiently than traditional approaches do.

The main contributions of this paper are: (1) The specification of application requirements that lets the user conduct a search to generate customized ontology segments; (2) A novel algorithm for ontology segmentation that uses a keyword-search approach for generating segments based on application requirements.

The rest of this paper is organized as follows: Section 2 introduces ontologies and the ontology matching process. Section 3 briefly summarizes state-of-the-art approaches, and compares them with the one set out in this paper. Section 4 explains our algorithm for ontology segmentation based on application requirements. Section 5 discusses the evaluation approach and the results obtained. Finally, Section 6 summarizes this paper, presents some conclusions and suggests lines of future research that may lead to improving the approach proposed.

2 Ontologies and the Ontology Matching process

Ontologies can be viewed as a set of assertions that are meant to model some particular domain. Ontologies may represent several data and conceptual models such as database schema, UML models, and XML schema. Despite several definitions of ontology having been put forward over the years related to the aim of this paper, we consider the following definition [Acampora et al., 2012]:

Definition 1. An ontology is a triple $O = \langle C, P, I \rangle$, where:

- C is the set of *classes*, i.e., the set of concepts that populate the domain of interest;
- P is the set of *properties*, i.e., the set of relations existing between the concepts of the domain.
- I is the set of *individuals*, i.e., the set of objects of the real world, representing the instances of a concept.

Classes, properties, and individuals are generally referred to as entities. Properties that represent relations between classes are called Object Properties.

Properties that represent relations between a class and some literal values (e.g., a string, a number) are called Data Properties.

The ontology matching problem arises from there being high heterogeneity existing in ontologies that describe the same domain. Unfortunately, people or artificial agents may use different terms for the same meaning or use the same term to describe different things. The goal of matching ontologies is to reduce this heterogeneity, which may be present on many levels (e.g., syntactic, terminological, conceptual) [Euzenat and Shvaiko, 2013].

Definition 2. The Ontology Matching Process can be defined as a function f that takes pairs of ontologies o_s (source ontology) and o_t (target ontology) to be matched, an optional reference alignment A, a set of parameters p and a set of oracles and resources r so as to return an alignment A' between these ontologies:

$$A' = f(o_s, o_t, A, p, r) \tag{1}$$

Definition 3. A Correspondence (or Semantic Match) is denoted as a tuple $\langle e1, e2, k\rangle$, in which $e_1 \in O_1$, $e_2 \in O_2$ and $e_1 \equiv e_2$, being k a similarity measure concerning the elements referred. A correspondence set can be acquired from the execution of a matcher, as well as based on an expert user indication.

Definition 4. The Similarity Measure between the two elements that comprise a correspondence is expressed by a value in the range of $[0, 1]$ to indicate to what extent the given entities are similar. In this case, 0 stands for complete disparity and 1 for complete equality. There are several similarity measures proposed in the literature (such as linguistic, semantic, and structural) and it is commonplace to aggregate different similarity measures [Elshwimy et al., 2014].

Definition 5. An Alignment consists of a set of correspondences found between elements from different ontologies. In this sense, we can define $A = C_{\langle O_1, O_2\rangle}$, in which $C_{\langle O_1, O_2\rangle}$ represents the set containing the correspondences found between elements from the ontologies O_1 and O_2.

Definition 6. An Ontology Matcher or simply Matcher is a function or algorithm designed to lead the ontology matching process.

In a nutshell, the output alignment A is a set of correspondences generated by a matcher. Each correspondence is used to link an element belonging to the source ontology with a similar one belonging to the target ontology. Though there are other types, the scope of this work is limited to relationships of equivalence ($=$). This is because rather than propose a new matcher, existing ones are used to evaluate the proposed segmentation algorithm. Just as equivalence is the only relationship common to most matchers, so it is common to encounter only the equivalence relationship in reference alignments.

3 Related Work

In the case of large-scale ontology matching tasks [Rahm, 2011], reducing the scope of the search is an important aspect that needs to be considered so as to improve the quality of the potential alignment and to reduce the number of validations required by the user. However, in spite of the large body of work on ontology matching techniques, only a few are segmentation based. In the following, we describe some of these techniques and compare them to our approach.

In order to reduce the number of comparisons during a matching, Do and Rahm propose COMA++ [Do and Rahm, 2007], a divide-and-conquer approach based on breaking an ontology down into smaller parts, which are defined as ontology fragments. In practice, it only matches fragment pairs with a high similarity. However, to generate segments, COMA++ uses relatively simple heuristic rules. More specifically, it does not consider the computation of structural closeness between the concepts, which often results in there being very few or a great many segments, depending on the ontologies considered. Another issue is how to determine what segments are similar, as COMA++ only considers the root node of the segment, which may reduce the matching quality.

An extension of COMA [Massmann et al., 2011] (COMA 3.0) copes with these limitations by applying a structure-based clustering algorithm to segment the input ontologies into a set of disjoint sub-graphs [Algergawy et al., 2011]. Thus, elements that are structurally similar are grouped in the same cluster, which may increase the matcher's efficiency and reduce the execution time. However, if ontologies based on a fixed heuristic are segmented, this may result in a loss of semantic information, especially for elements near segment boundaries. In the case of obtaining a high number of segments, more information would be lost, thus reducing the quality of the matching results.

With the aim of reducing the semantic loss, Anchor-Flood proposes a matcher that avoids the ontology segmentation *a priori* [Seddiqui and Aono, 2009]. It starts by defining a pair of similar concepts across ontologies as an anchor point. Then, it proceeds toward the neighboring nodes, considering the locality of reference, which results in segments located around the anchors. Eventually, it may align parts of large ontologies and so output a relatively small segmented alignment, depending on the continued success of finding matching partners for the anchor considered.

Noy and Musen present traversal views as a way of defining a segment [Noy and Musen,], defined as an ontology view. The idea is to allow the user to specify an ontology subset comprising concepts of interest, the respective relationships, and the maximum depth. It enables the user to build segments incrementally, by extending the current view over many iterations. However, it assumes that the user (e.g., an Ontology Engineer) has a deep understanding of the ontology elements. Another issue is the strong need for the user's involvement

in selecting the relationships that should be traversed and associating each one of them to a level of recursion, which defines when the algorithm should stop running.

Queiroz-Sousa et al. introduced a method for summarizing ontologies by using user-defined parameters [Queiroz-Sousa et al., 2013]. An ontology summary is an excerpt containing the most important concepts of the ontology or the most relevant concepts for a user, while respecting the original relationship and properties of the input ontology. The authors define two relevance measures: the Centrality Measure and the Closeness Measure. The Centrality Measure considers the number of relationships between ontology concepts and the types of relationships between them. The Closeness Measure indicates the extent to which a concept is relevant, i.e., the more related concepts there are, the higher the closeness measure is. By introducing the Broaden Relevant Paths (BRP) Algorithm, they extract a global ontology view containing only the most relevant elements. Lambrix and Kaliyaperumal proposed a session-based ontology alignment framework to allow partial computation for generating mapping suggestions [Lambrix and Kaliyaperumal, 2016]. The concept of session-based stands for breaking the matching down into smaller tasks to enable the user to interrupt and resume the process if necessary. Regarding ontologies in the scope of a session, the framework proposed generates pairs of segments using a string-based approach to detect concepts in different ontologies with similar names. The segment pairs are retained based on a predefined number of maximum elements. However, as the segmentation strategy concerns a fixed number of elements, the user has no control over the concepts present in each segment, which makes it necessary to iterate every session to assure that a particular concept is present in the set of correspondences.

The aforementioned studies present approaches that make use of some element references (e.g., anchors) or the analysis of the number of relationships to extract an ontology excerpt containing only the most relevant elements. In the first case, by only defining anchors as a reference, the user needs to have a broad knowledge about the ontologies to perform complex assertions, for instance, knowing what element relationships are needed to define an anchor point. On the other hand, note that considering only the number of ontology relationships will not cover the cases when the user has an interest in concepts that have fewer or no references. In this paper, we introduce the definition of application requirements in combination with a keyword-search process, which provides the possibility of generating segments without requiring the user to know the structure of the ontologies. Furthermore, unlike other approaches, we consider that whenever an element satisfies the application requirements, it is taken as relevant, even if it does not have a high number of relationships in the ontology structure.

4 SOMA: An Algorithm for Segmentation of Ontologies for Matching Applications

In this paper, we propose SOMA (Segmentation of Ontologies for Matching Applications), an algorithm for generating ontology segments so as to improve the ontology matching process. As a segmentation strategy, we consider a set of application requirements to extract the most relevant elements in the context of application-specific ontology matching context.

The intention of SOMA is not to create a new matcher, but to generate ontology segments based on application requirements and if needed to allow the segment alignments to be generated by using any existing matcher. By considering segments rather than the whole ontologies in the matching, the run time can be reduced and efficiency can be improved, since fewer elements are compared and the elements of the segment are related to the user's interests.

Figure 1 presents the workflow of the SOMA algorithm. In the initial step, we receive as an input a pair of ontologies (source and target) that are imported into the Ontology Catalog, converting them into a standard representation. Then, we generate the source ontology segment by extracting only the elements related to the application requirements. Finally, we generate the segment for the target ontology by extracting the elements related to the source ontology segment.

In the following sections, we define application requirements, present a motivating example and describe each one of the steps comprising the SOMA algorithm.

4.1 Application Requirements

The aspect which distinguishes this work from previous approaches is the adoption of application requirements as a means for specifying which concepts are the most relevant to the user. According to [Sommerville, 2010], application (or system) requirements consist of an abstract description of characteristics, attributes, abilities or qualities that a system should provide to be useful to its users. Requirements may range from a high-level statement of a service or a system constraint to a detailed mathematical functional specification. Functional requirements are those that describe functionalities or a system service. Non-functional requirements are those that define system properties or constraints (e.g., response time, reliability). This paper takes as a baseline a set of application requirements as a way of improving ontology matching tasks, by an on-demand generation of ontology segments. In this context, we formalize these requirements as follows.

Definition 7. Application Requirements are denoted as $AR = \{r_1, r_2, \ldots, r_n\}$, where each r_i describes an application-related need regarding the matching

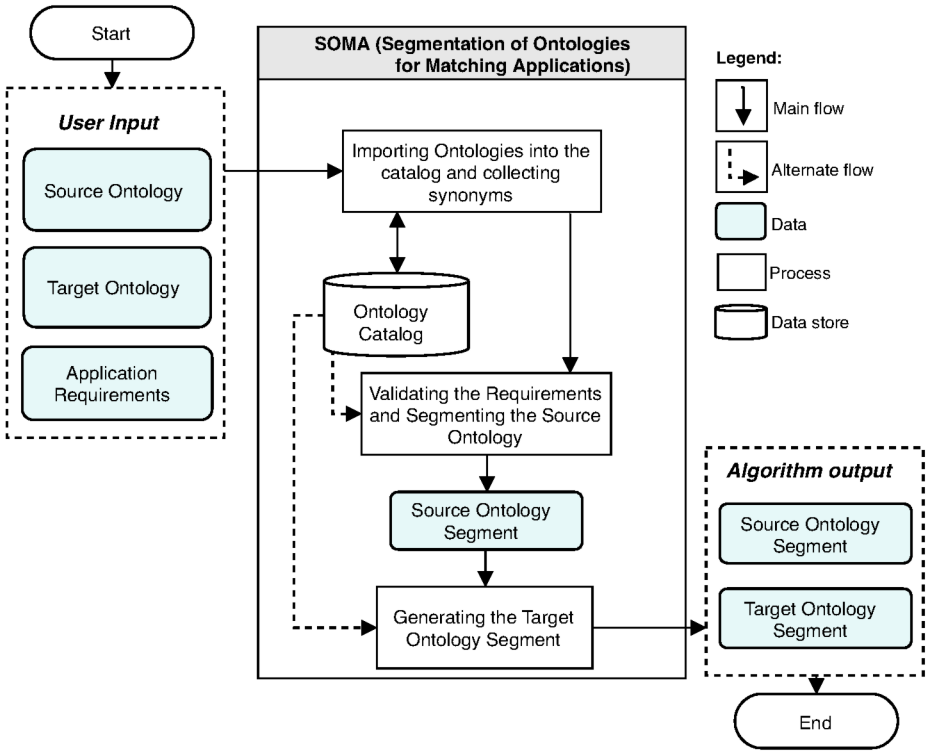


Figure 1: Workflow of the SOMA algorithm.

process. We divide AR into two categories: Data requirements, which stands for requirements regarding the concepts that should be included in the matching and operational requirements, which describe the configuration parameters that will generate the segments. We detail these requirement types as follows.

Definition 8. Data Requirements (DR) are represented by a set of keywords $\{k_1, k_2, \ldots, k_n\}$ for specifying the n concepts that are important to the user in the matching process. For instance, in the conference domain, the user can define as DR the keywords {author, email, conference} or {paper, reviewer, title, abstract}. These keywords are taken by the SOMA algorithm and are used to identify the nearest elements to be included in the segments.

Definition 9. Operational Requirements (OR) are represented by a set of parameters describing some user preferences regarding the segment generation process. They allow the user to customize how the segments are generated. Table 1 describes the OR considered in the SOMA algorithm: i) the segment extension level and ii) the similarity threshold. When the extension level requirement is defined as simple, segments are more abbreviated, as they consist of only the

related elements of keywords and their first-level adjacent elements. Alternatively, when the extension level is expanded, segments tend to be more densely populated, as they also contain a second level of relationship between elements. The similarity threshold is the minimum similarity value to ensure an ontology element is related to a keyword described in the data requirements.

Operational Requirement	**Possible values**
Segment Extension Level	simple, expanded
Similarity Threshold	Decimal number in the range of $[0, 1]$

Table 1: Specification of Operational Requirements

4.2 Motivating Example

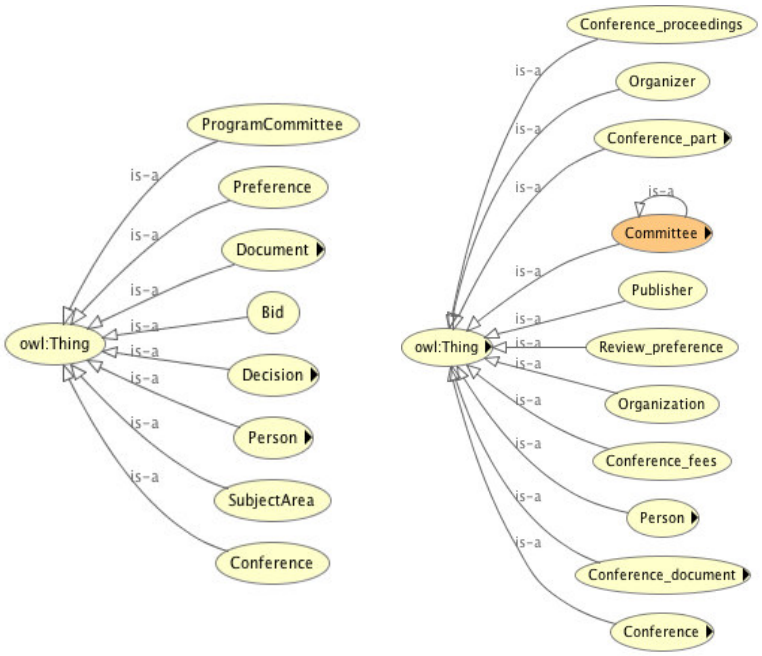
We demonstrate our approach more accurately by describing how it is applied to a practical example. Consider two ontologies O_1 and O_2. The ontologies describe data about conferences such as documents and publishers as shown in Figure 2.

Let us assume that a user is interested in obtaining the most popular topics covered by papers (documents) published over a given period of the years. Considering that each ontology uses different terms to describe the same concept (e.g., Document in O_1 and Conference_document in O_2), in order to allow the integration of data from different sources, correspondences between these elements must be generated. In this case, despite there being many elements in O_1 and O_2, only a few are relevant to the application. Thus, there is no need to traverse the whole ontologies when the alignment is being generated. To illustrate this assumption, we define the following data requirements:

R1 *The application shall provide information about submitted papers (e.g., paper, reviewer, title)*

R2 *The application shall provide information (e.g., name, university, e-mail) about some person who is registered for a conference*

In order to fulfill respectively $R1$ and $R2$, we can define the following set of keywords: $K_1 = \{\text{paper}, \text{reviewer}, \text{title}, \text{abstract}\}$ and $K_2 = \{\text{person}, \text{name}, \text{author}, \text{email}, \text{university}\}$. We consider these keywords in the next sections so as to demonstrate how the segment is generated.



(a) Cmt Ontology (O_1) (b) Conference Ontology (O_2)

Figure 2: An excerpt containing first-level elements from Ontologies O_1 (a) and O_2 (b).

4.3 Importing ontologies into the catalog and collecting synonyms

The first step of the algorithm involves importing, pre-processing and storing ontologies into the Ontology Catalog. To this end, we first convert the ontologies from OWL or RDF formats to a labeled graph. During this process, we enrich the graph by adding synonyms (obtained from Wordnet[1]) for each node.

Many algorithms transform ontologies into labeled graphs. We use this strategy to facilitate how to deal with the ontology elements when generating the segment. We create a graph in which the nodes represent classes or properties, and the edges represent relationships between them. Figure 3 illustrates an example of how to represent ontologies as a graph when following this notation. The black nodes represent classes, the grey nodes represent object properties, the solid white nodes represent data properties and the dashed white ones are synonyms nodes. Between these nodes, there are some arrows indicating relationships such

[1] `http://wordnet.tspell.smu.edu`

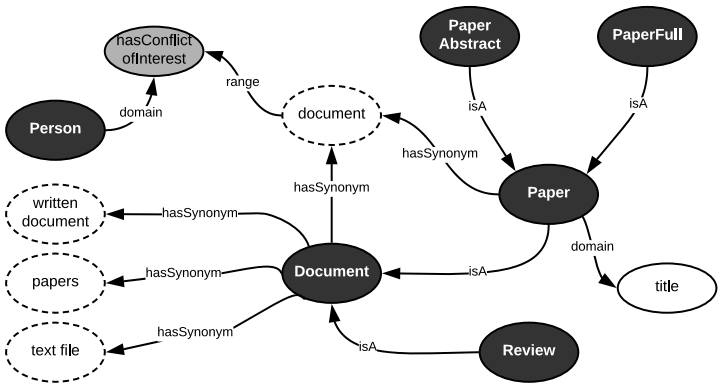


Figure 3: Example of the graph notation adopted in this work to represent ontologies.

as isA (specialization), domain and range.

In Figure 3, we have the triple (*Person, hasConflictOfInterest, Document*) to define an object property hasConflictOfInterest representing the relationship between *Person* (domain) and *Document* (range). On the other hand, we have the triple (*Paper, title, xsd:string*) to define a data property *title* representing the relationship between the class *Paper* (domain) and a literal value *string* (range). The notation of a graph can thus be used to represent the basic elements used in the matching process.

4.4 Validating the Requirements and Segmenting the Source Ontology

The next step concerns how to generate a segment from the source ontology, with a view to collecting only ontology elements in accordance with the Application Requirements.

The algorithm for segmenting the source ontology [See Algorithm 1] receives the source ontology SO and the set of application requirements (DR and OR) as input and returns a segmented version of SO containing only the elements (e.g., classes, properties) that are suitable for DR. The OR also has an impact on the process, as depending on some of the requirements chosen (e.g., extension level or keyword search threshold), a higher/lower number of elements may be the output.

To generate the segment, we parse each element (including the synonym nodes) of the source ontology and select the elements that match the keywords defined in DR. The $Match$ function outputs a value between $[0, 1]$ based on the linguistic similarity of the elements compared. The value of the similarity

Algorithm 1 Algorithm for segmenting the Source Ontology

```
1: function generateSourceOntologySegment(SO, DR, OR)
   ▷ SO corresponds to the source ontology graph;
   ▷ DR (Data Requirements) corresponds to the list of keywords representing
   the relevant concepts to be included in the segment;
   ▷ OR (Operational Requirements) corresponds to the list of configuration
   parameters that are used to generate the segment
   ▷ Match(e1, e2) is a function that calculates the linguistic similarity between
   e1 and e2, comparing element names and synonyms;
   ▷ Neighbors(n) is a function that returns the list of nodes that are adjacent
   to n.
   ▷ NodeAux is a temporary copy of an element to be included in the segment
2:     SO_seg ← {}     ▷ Initialize the subgraph SO_seg representing the segment
3:     for all e_i ∈ SO do ▷ Iterate ontology elements e_i (classes and properties)
4:         for all k_j ∈ DR do          ▷ Parse each keyword k_j defined in DR
5:             if Match(e_i, k_j) >= OR.similarityThreshold then
6:                 nodeAux ← {e_i}
7:                 if OR.extensionType == EXPANDED then
8:                     nodeAux ← nodeAux ∪ Neighbors(e_i)
9:                 end if
10:                SO_seg ← SO_seg ∪ nodeAux
11:            end if
12:        end for
13:    end for
14:    return SO_seg
15: end function
```

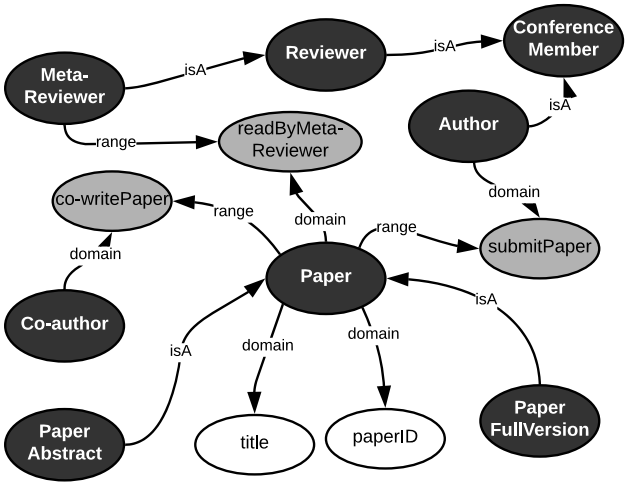
threshold is one of the available operational requirements [See Section 4.1]. The $Neighbors$ function receives a node and returns the set of adjacent ontology elements (classes and properties) in the graph. The resulting segment is the union of the set of elements (plus neighbors in the case of expanded extension type) that matches with the keywords defined in the DR.

Figure 4 shows two examples of source ontology segments regarding the ontology Cmt[2]. In the former, we illustrate how the segment is generated, based on the keywords K_1 (paper reviewer, title, abstract). In the latter, we show how the segment is generated for the keywords K_2 (person, name author, email, university).

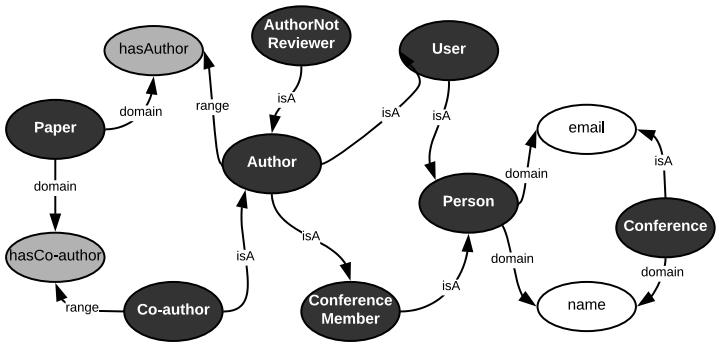
Note that for the first segment [See Figure 4(a)], SOMA includes the classes Person and Author and their neighbors in the segment, since the Cmt ontology

[2] http://oaei.ontologymatching.org/2017/conference/data/cmt.owl

contains elements related to these concepts. The same applies to the data properties email and name. However, as the Cmt ontology does not have any element related to university, this concept is absent in the segment. In the second segment [See Figure4(b)], SOMA extracts the classes Paper and Reviewer, and the data property title, as there are elements with the same name defined in keywords. The class Paper abstract is also considered, despite the keyword being different from the name of the element.



(a) Source Ontology Segment for R_1



(b) Source Ontology Segment for R_2

Figure 4: Excerpt of ontology segments for the ontology Cmt generated based on the data requirements DR_1 and DR_2

4.5 Generating the Target Ontology Segment

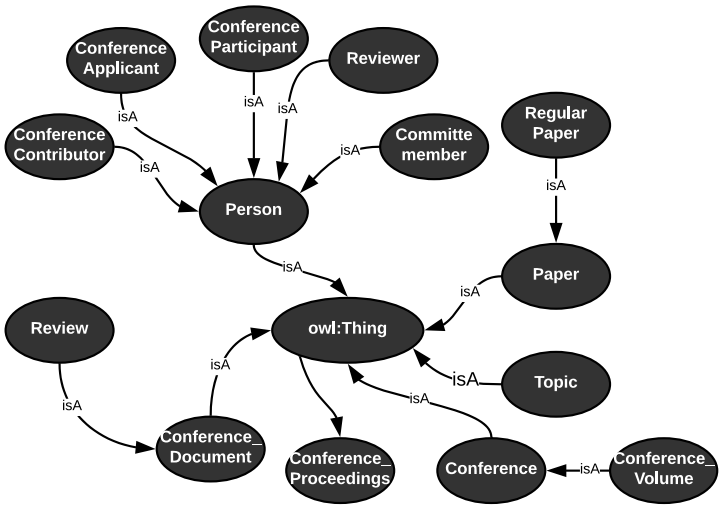
After obtaining the source ontology segment, we generate the segment for the target ontology (TO). For this process, we select the candidate elements based on the similarity value between the target ontology elements and those present in the source ontology segment SO_{seg}. [See Algorithm 2] for details how to generate the segment for the target ontology.

Algorithm 2 Algorithm for segmenting the Target Ontology

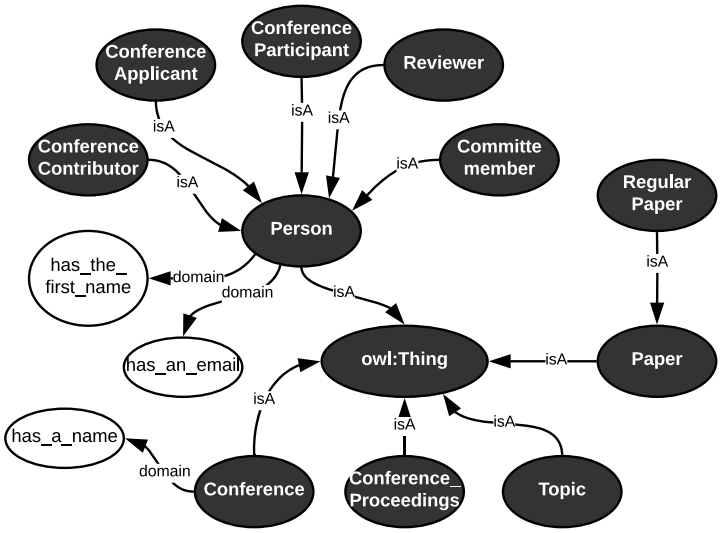
function GENERATETARGETONTOLOGYSEGMENT(TO,SO_{seg}, OR)
▷ TO corresponds to the target ontology graph;
▷ SO_{seg} corresponds to the source ontology segments previously obtained;
▷ OR (Operational Requirements) corresponds to the list of configuration parameters that are used to generate the segment;
▷ $Match(e_1, e_2)$ is a function that calculates the linguistic similarity between e_1 and e_2, by comparing the element names and synonyms;
▷ $Neighbors(n)$ is a function that returns the list of nodes that are adjacent to n.
▷ $NodeAux$ is a temporary copy of an element to be included in the segment
  $TO_{seg} \leftarrow \{\}$ ▷ Initialize empty ontology segment
  for all $e_i \in SO_{seg}$ **do** ▷ Parse each element e_i in SO_{seg}
    for all $e_j \in TO$ **do** ▷ Parse each element e_j in TO
      if $Match(e_i, e_j) \geq OR.similarityThreshold$ **then**
        $nodeAux \leftarrow \{e_j\}$
        if $OR.extensionType ==$ EXPANDED **then**
          $nodeAux \leftarrow nodeAux \cup Neighbors(e_j)$
        end if
        $TO_{seg} \leftarrow TO_{seg} \cup nodeAux$
      end if
    end for
  end for
  return TO_{seg}
end function

Figure 5 illustrates two examples of target ontology segments. In the first [See Figure 5(a)], we present the segment coming from the comparison of the source ontology segment generated for R_1 and the ontology Sofsem[3] as the target. The same applies to the second case [See Figure 5(b)], with the only difference being that we consider the source ontology segment generated for R_2 as the input.

[3] http://oaei.ontologymatching.org/2017/conference/data/Conference.owl



(a) Target Ontology Segment for R_1



(b) Target Ontology Segment for R_2

Figure 5: Excerpt of ontology segments for the ontology Sofsem, considering the source ontology segments illustrated in Section 4.4. as input

Note that the ontology Sofsem does not have a class Author. In this case, SOMA has included some classes in the segment that may correspond to an author, such as Conference Contributor, Conference Participant and Conference Applicant.

Once generated, the source and target ontology segments are both exported as ontology OWL files. These files may be the input for some ontology matching systems, which is what we did in our experiments, as shown in the following section. In this case, the assumption is that in the subsequent ontology matching process, a matcher may use more sophisticated techniques (e.g., structural-level matching) in order to identify more precisely the correspondences between the elements present in the segments.

5 Experiments and Analysis

For the purpose of evaluating our approach, we designed three application-based ontology matching scenarios using the conference benchmark provided by the OAEI 2017. In the first one, we consider how matching was conducted by each ontology in its entirety. In the other two, we focus on the matching between segments generated by SOMA. We present below the configuration of the experiment and compare the performance of OAEI matchers in these scenarios.

5.1 Experiment Configuration

In this section, we define the datasets, parameters, and hardware configurations used to perform the experiments in this work.

- **Datasets:** Table 2 details the datasets considered in the experiments and the respective number of classes, data and object properties. Although the OAEI conference track contains a total of 16 ontologies, we select only the ontologies that have available reference alignments[4] (gold standards).
- **Data Requirements:** To be a guideline on how to generate segments SOMA, we consider the data requirements DR1 and DR2 illustrated in the motivating example [See Section 4.2], in which each requirement represents the concepts that are of interest to an expert user.
- **Operational Requirements:** The set of operational requirements used in the experiments are defined in the following. The value expanded for the segment extension is taken because it generates larger segments, which improves the range for evaluating the accuracy of the generated results. The value considered to the keyword search threshold was defined based on preliminary experiments. For this purpose, we have tested several scenarios containing different values, and the value of 0.6 resulted, on average, in the inclusion of more suitable elements on the segments, regarding the targeted dataset.

[4] The reference alignments can be downloaded from http://oaei.ontologymatching.org/2017/conference/data/reference-alignment.zip

  - Segment Extension Level = EXPANDED
  - Keyword search threshold = 0.6
- **Hardware Configuration:** The following hardware configuration is used to perform the experiment execution:
  - Processor: Intel Core i5-3210M
  - CPU Speed: 2.50GHz x 2 core
  - RAM Capacity: 16GB

Ontology	Classes	Data Properties	Object Properties
Cmt	36	10	49
Sofsem	60	18	46
ConfTool	38	23	13
Edas	104	20	30
Ekaw	74	0	33
Iasted	140	3	38
Sigkdd	49	11	17

Table 2: Detailed information of ontologies from the OAEI Conference track used in the experiments

5.2 Implementation and Tools

We implemented the SOMA algorithm and the Ontology Catalog as Java Web applications. Furthermore, to support the conduct of the experiments, we also have implemented a Matchers Catalog, which supports the execution of multiple matchers. Some details on the implementation regarding each component are given below.

5.2.1 Ontology Catalog

The Ontology Catalog[5] stores ontologies by representing them in a single directed graph. The elements from different ontologies are distinguished by their URI prefix. Figure 6 depicts the metadata structure of the ontology catalog. We use the OWL API to traverse all ontology elements, reading classes such as

[5] The Ontology Catalog source code is available at `https://github.com/dass-cin/ontology-catalog`

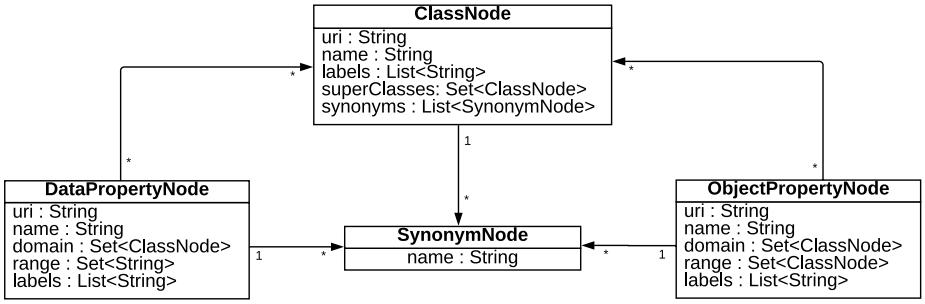


Figure 6: Metadata structure of the ontology catalog.

ClassNode, data properties such as DataPropertyNode and object properties such as ObjectPropertyNode. The SynonymNodes are obtained for each element by searching for synonyms in Wordnet. We also used the OWL API[6] to read/write ontologies and a graph database (Neo4J[7]) to store and search the ontology graph. The option for Neo4j is motivated by the ease of graph manipulation due to the Cypher[8] query language.

The importance of using the Ontology Catalog to support the SOMA algorithm is because the ontology can be dealt with while using less memory. Since we consider large-scale ontologies, it could be unfeasible to load all elements into the memory so as to generate the segment. Given the number of possibles relationships that there are in an ontology, a graph allows navigation at several levels. Furthermore, storing data as a graph opens up new possibilities in comparison to relational databases [Vicknair et al., 2010], because of the addition of custom element relationships (e.g., synonyms) and the possibility of using existing graph algorithms for navigation and searching within ontologies.

5.2.2 Matchers Catalog

As we wished to provide a general infrastructure for generating alignments, we also developed a Matchers Catalog[9]. The purpose of which was to abstract the matcher component so that any matcher can be incorporated to generate correspondences. The resulting alignments are structured by using the Alignment API, which facilitates the integration of results generated by different matchers.

Figure 7 depicts the metadata structure of the Matchers Catalog. To describe a matcher, in addition to name and version, we consider the Web Service endpoint

[6] The OWL API can be downloaded from `http://owlcs.github.io/owlapi/`
[7] Neo4J is available to download at `https://neo4j.com`
[8] The Cypher documentation is available at `https://neo4j.com/developer/cypher/`
[9] The Matchers Catalog source code is available at `https://github.com/dass-cin/matchers-calalog`

in which the matcher is run, a set of features and the default configuration parameters that should be considered. Given that matchers do not provide a default Web Service endpoint, we implemented a wrapper component to import the matcher library and provide the service that receives the source/target ontologies and provides the alignment in accordance with the Ontology API format. In this initial version of the Matchers Catalog, we have incorporated some matchers which were registered in the last OAEI campaigns, such as COMA, LogMap, and AML.

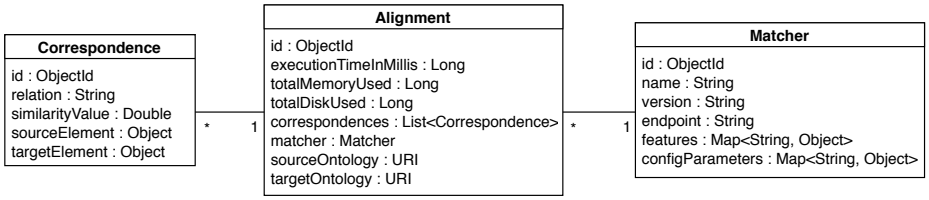


Figure 7: Metadata structure of the Matchers Catalog.

Once an alignment has been generated, it is stored in the matcher's catalog together with the respective correspondence set. We use these records both to evaluate the matcher's running performance (as long as we track the execution time and total memory used), and to assess the matcher's quality, by comparing the provided correspondence set provided to a gold standard.

5.2.3 SOMA Algorithm

Concerning the implementation regarding the SOMA Algorithm[10], we build a Java application by using the Spring Batch Framework[11]. We take as an input a CSV file, which makes it possible to determine several segment generations and the matching to run at once. The output summarizes the results of many executed matchings in a CSV containing the number of true positives, false positives, false negatives and some measures such as precision, recall, F-Measure and execution time.

About the algorithm operation, we use the lexical matcher S-Match-Wordnet[12] as similarity measure for the Match function. This matcher relies on semantic information taken from the linguistic resource WordNet to identify nodes that are semantically related. The option for this tool is because it is suitable for

[10] The SOMA algorithm source code is available at `https://github.com/dass-cin/soma`
[11] `https://spring.io/projects/spring-batch`
[12] S-Match-wordnet source-code available at: `https://github.com/s-match/s-match-wordnet`

lightweight matching and provides minimal results [Bella et al., 2017], which is especially useful during the segment generation, considering it requires a preliminary matching between a small number of ontology elements.

5.3 Experiments Performed and Results

In this section, we detail the experiments performed as well as the results obtained. The main objective of the experiments was to confirm that using the segments generated by SOMA in the ontology matching process results in reducing the scope of the search and improves the quality of the alignments. Thus, the first experiment is concerned with matching on the whole ontologies, while the remaining experiments address each one of the data requirements $DR1$ and $DR2$ so as to generate the segment generation, which is based on the matching between segments. Figure 8 shows the summary of results, concerning the measures of execution time, precision, recall and F-Measure.

The specification of experiments is described as follows:

- **Experiment 1 (exp1):** Matching based on the whole ontologies;
- **Experiment 2 (exp2):** Segment matching based on the keywords paper, reviewer, title, abstract;
- **Experiment 3 (exp3):** Segment matching based on the keywords person, name, author, email, university.

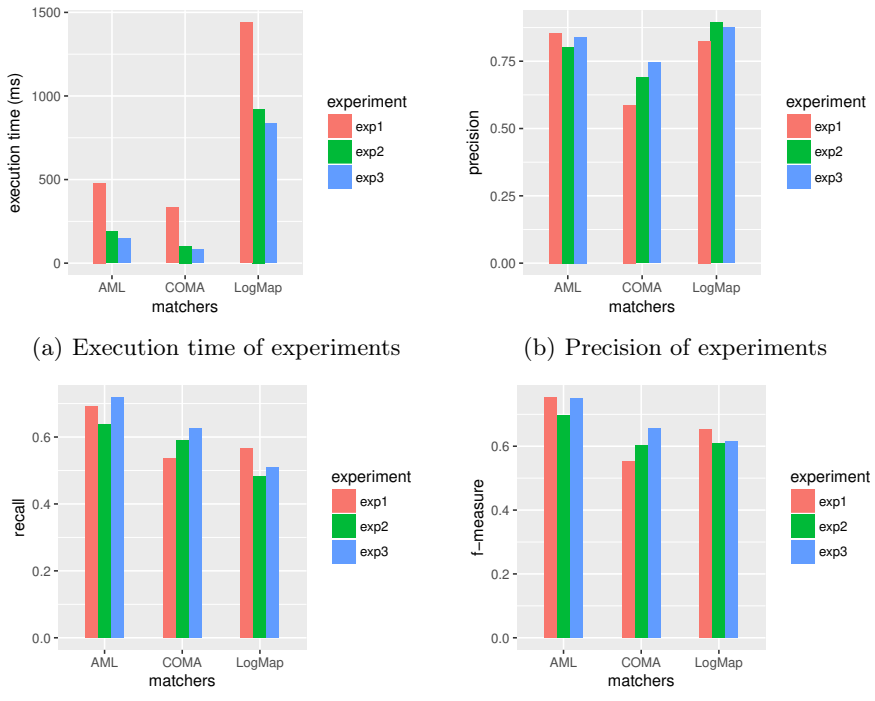
We evaluate the proposed algorithm using well-known evaluation measures such as precision, recall, and F-Measure. More precisely, precision (2) measures the ratio of the correctly found correspondences (true positives) over the total number of correspondences. Recall (3) measures the ratio of the true positives over the total number of expected correspondences (i.e., true positives and false negatives). We also consider the measure of the execution time that a matcher takes to complete the matching process between each pair of ontologies.

$$Precision = \frac{truePositives}{truePositives + falsePositives} \tag{2}$$

$$Recall = \frac{truePositives}{truePositives + falseNegatives} \tag{3}$$

$$F\text{-}Measure = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4}$$

As to the matching execution, we consider some active OAEI participants: COMA, AML, and LogMap. We delegate to each matcher the pairwise comparison between all the simple combinations of the seven ontologies (or respective



(a) Execution time of experiments

(b) Precision of experiments

(c) Recall of experiments

(d) F-measure of experiments

Figure 8: Results of the experiments considering precision, execution time, recall and F-Measure.

segments) described in [Section 5.1], which results in 21 comparisons. Since we consider three matchers, we have a total of 63 matchings per experiment.

In brief, our evaluation compares the results of matching segments generated by the SOMA algorithm against the comparison of the matching on the whole ontologies. The purpose is to confirm that by choosing to match the segments generated, we obtain a faster execution time and improve the quality of the alignments. We then performed two experiments with different sets of requirements (exp2) and (exp3) to illustrate segments containing different parts of ontologies. Therefore, this enabled it to be observed that the matchers have different results depending on the segment considered. An example from Figure 8 is that LogMap obtained the highest F-Measure for exp1, although COMA provided better quality results in exp3.

More precisely, Figure 8(a) illustrates the difference in the execution time acquired for each referred experiment. Comparing the results obtained for generating alignments between the whole ontologies (exp1) and between the segments generated by SOMA (exp2, exp3), we notice a substantial reduction in the

execution time for the latter, regarding the matchers considered.

Figure 8(b) compares the precision obtained in each experiment. We notice a slight improvement in the majority of cases in which SOMA segments were considered (exp2, exp3). This illustrates that there has been no loss in precision due to reducing the search space in order to consider only the concepts related to the data requirements.

In Figure 8(c), we present the comparison of the results obtained based on the recall. In this case, taking exp1 as a baseline, note that with regard to the matcher AML there is a discrete gain in exp3, whereas there is a slight decrease in exp2. Regarding the matcher COMA, there is an improvement in both the experiments. On the other hand, LogMap presented a decrease in the experiments with segments. This occurred because of the higher number of false positives delivered by LogMap when it considered the segments. This variation influences the case that takes F-Measure as a baseline [Figure 8(d)], as it is a harmonization measure between precision and recall.

In short, according to the results of the experiments, we notice an improvement in most cases when the segments were considered, indicating that the segments generated by SOMA reduced the matching scope, and at least maintained the quality of results, thus demonstrating the effectiveness of the algorithm for the large-scale ontology matching problem.

The variation of performance observed in the experiments indicates that, depending on the dataset considered and the metric taken for the matcher evaluation, it is possible to achieve results of higher/lower quality. This fact has motivated the search for approaches that can enumerate the aspects that somehow have an impact on the acquisition of results of quality, with the aim being to recommend the most suitable matchers for a particular matching task. Exploring the possibilities of undertaking such approaches is a line that further research could usefully undertake.

6 Conclusions and Future Work

In large-scale settings, the ontology matching process is challenging due to the high number of elements available for comparison, and the possibility of obtaining an excessive number of correspondences as a result, which can make human validation difficult. In this context, this work proposes an algorithm for segmenting ontologies so as to reduce the scope of the matching search by considering only the elements that are most relevant to the user. Based on this, we introduce the use of application requirements, that are categorized into data requirements - a set of keywords which indicate the most important concepts to the user, and operational requirements - a set of parameters to use when generating the segment. In the experiments, we examined the matchers' performance when

matching on the whole ontologies and on the segments generated by SOMA. The results obtained indicate that the segment-based matching was performed significantly faster, and maintains the quality of the results.

In future studies, we intend to propose a framework to introduce the capability of selecting the most suitable matchers based on the features of ontologies (or segments) provided. We plan to use the framework together with SOMA to exploit the matching between different types of ontologies and how the characteristics of ontologies affect the quality of the matchers and execution time. Our assumption is that the use of ontology segments provides the possibilities of having a more reliable identification of the matching characteristics, thereby facilitating the identification of the most suitable matcher. Furthermore, we can expand this investigation to deal with the problem of tuning the matchers, by assessing how the matcher is run with regard to different values of configuration parameters. Other applications of this work would be to select an ontology by using keywords, as we can use the data requirements to search for ontologies that contain similar (or synonyms) elements in different sources.

References

[Acampora et al., 2012] Acampora, G., Loia, V., Salerno, S., and Vitiello, A. (2012). A hybrid evolutionary approach for solving the ontology alignment problem. *International Journal of Intelligent Systems*, 27(3):189–216.

[Achichi et al., 2017] Achichi, M., Cheatham, M., Dragisic, Z., Euzenat, J., Faria, D., Ferrara, A., Flouris, G., Fundulaki, I., Harrow, I., Ivanova, V., Jiménez-Ruiz, E., Kolthoff, K., Kuss, E., Lambrix, P., Leopold, H., Li, H., Meilicke, C., Mohammadi, M., Montanelli, S., Pesquita, C., Saveta, T., Shvaiko, P., Splendiani, A., Stuckenschmidt, H., Thiéblin, É., Todorov, K., Trojahn dos Santos, C., and Zamazal, O. (2017). Results of the Ontology Alignment Evaluation Initiative 2017. In *OM 2017 - 12th ISWC workshop on ontology matching*, pages 61–113, Wien, Austria. No commercial editor. achichi2017a.

[Algergawy et al., 2011] Algergawy, A., Massmann, S., and Rahm, E. (2011). A Clustering-based Approach For Large-scale Ontology Matching . *ADBIS*.

[Bella et al., 2017] Bella, G., Giunchiglia, F., and McNeill, F. (2017). Language and domain aware lightweight ontology matching. *J. Web Semant.*, 43:1–17.

[Do and Rahm, 2007] Do, H.-H. and Rahm, E. (2007). Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885.

[Dong et al., 2014] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. (2014). Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–610. Google LLC, Mountain View, United States.

[Elshwimy et al., 2014] Elshwimy, F. A., Algergawy, A., Sarhan, A., and Sallam, E. A. (2014). Aggregation of similarity measures in schema matching based on generalized mean. *2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW)*, pages 74–79.

[Euzenat and Shvaiko, 2013] Euzenat, J. and Shvaiko, P. (2013). *Ontology Matching*. Springer Publishing Company, Incorporated, 2nd edition.

[Lambrix and Kaliyaperumal, 2016] Lambrix, P. and Kaliyaperumal, R. (2016). A Session-based Ontology Alignment Approach enabling User Involvement. *Semantic Web Journal*.

[Lehmann et al., 2015] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., and Bizer, C. (2015). DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6(2):167–195.

[Massmann et al., 2011] Massmann, S., Raunich, S., Aumüller, D., Arnold, P., and Rahm, E. (2011). Evolution of the COMA match system. In *OM'11: Proceedings of the 6th International Conference on Ontology Matching - Volume 814*, pages 49–60. University of Leipzig, CEUR-WS.org.

[Noy and Musen,] Noy, N. F. and Musen, M. A. Traversing Ontologies to Extract Views.

[Otero-Cerdeira et al., 2014] Otero-Cerdeira, L., Rodríguez-Martínez, F. J., and Gómez-Rodríguez, A. (2014). Ontology matching: A literature review. *Expert Systems with Applications*.

[Peukert et al., 2011] Peukert, E., Eberius, J., and Rahm, E. (2011). Rule-based construction of matching processes. In *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11*, page 2421, New York, New York, USA. ACM Press.

[Pouriyeh et al., 2018] Pouriyeh, S. A., Allahyari, M., Kochut, K., and Arabnia, H. R. (2018). A Comprehensive Survey of Ontology Summarization - Measures and Methods. *CoRR*.

[Queiroz-Sousa et al., 2013] Queiroz-Sousa, P. O., Salgado, A. C., and Pires, C. E. S. (2013). A Method for Building Personalized Ontology Summaries. *JIDM*.

[Rahm, 2011] Rahm, E. (2011). Towards Large-Scale Schema and Ontology Matching. *Schema Matching and Mapping*, (Chapter 1):3–27.

[Seddiqui and Aono, 2009] Seddiqui, M. H. and Aono, M. (2009). An efficient and scalable algorithm for segmented alignment of ontologies of arbitrary size. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):344–356.

[Seidenberg and Rector, 2006] Seidenberg, J. and Rector, A. L. (2006). Web ontology segmentation - analysis, classification and use. *WWW*.

[Shvaiko and Euzenat, 2013] Shvaiko, P. and Euzenat, J. (2013). Ontology matching: state of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176.

[Sommerville, 2010] Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition.

[Vicknair et al., 2010] Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA. ACM.