OntoPhil
Exploitation of Binding Points for Ontology Matching

Lorena Otero-Cerdeira, Francisco J. Rodríguez-Martínez, Tito Valencia-Requejo, and Alma Gómez-Rodríguez

Laboratorio de Informática Aplicada 2,University of Vigo, Campus As Lagoas, Ourense, Spain

Keywords: Ontology Matching, Ontology Alignment, Similarity Measure, Lexical Measure.

Abstract: This paper presents a new ontology matching algorithm, *OntoPhil*. The algorithm relies on the exploitation of some initial correspondences or *binding points* that connect the two ontologies used as input. First, it computes these *binding points* using a new lexical similarity measure which combines the information from a terminological matcher and an external one. Next, it discovers new binding points, by taking the initial ones as basis and by exploiting the specific features of the external structure of the ontologies matched. Finally, the *binding points* are automatically sifted out to obtain the final alignment. The proposed algorithm was tested on the benchmarks provided by the well known evaluation initiative OAEI, and compared to other matching algorithms. The experimental results show that *OntoPhil* is an effective approach and outperforms other algorithms that share the same principles.

1 INTRODUCTION

Guaranteeing the interoperability between new and legacy systems is a key point in systems' development. In this environment, the fields of *ontology* and *ontology matching* are fundamental lines of research to approach the issues related to systems interoperability.

An ontology provides a vocabulary to describe a domain of interest and a specification of the meaning of the terms in that context (Euzenat and Shvaiko, 2007). This definition typically includes a wide range of computer science objects, such as thesauri, XML schemas, directories, etc. These different objects identified as ontologies, belong to various fields of information systems such as web technologies (Antoniou and van Harmelen, 2004), database integration (Doan and Halevy, 2005), multi agent systems (van Aart et al., 2002) or natural language processing. Therefore, ontologies have been used anywhere as a way of reducing the heterogeneity among open systems. Nevertheless, the sole use of ontologies does not solve the heterogeneity problem, since different parties in a cooperative system may adopt different ontologies. Thus, in such case, there is the need to perform an *ontology matching* process to guarantee the interoperability.

Ontology matching is a solution to semantic heterogeneity since it finds correspondences between semantically related entities of the ontologies (Shvaiko and Euzenat, 2013). This set of correspondences is known as an *alignment*. A more formal definition of an alignment was provided in (Ehrig and Euzenat, 2005).

Ontologies are expressed in an ontology language. Although there is a large variety of ontology languages (Suárez-Figuero et al., 2011), in this paper we focus on OWL (W3C, 2013b), (W3C, 2013a), (Cuenca-Grau et al., 2008).

In this paper, we describe our combinational approach to ontology matching, the *OntoPhil* algorithm. This algorithm uses both terminological (string and language based) and structural methods to determine the correspondences among the entities (named classes, object properties and datatype properties) of the given ontologies. Initially, we determine lexical similarities among the entities in the ontologies using a new distance measure. By means of this distance we obtain an initial set of similarity values between the entities of the two ontologies. Then, using these initial matches as starting *binding points* between the two ontologies, we exploit their structural features to discover new *binding points*. This second step is applied iteratively until the algorithm converges. Finally, in the last step of the algorithm we apply some filters to dismiss the incorrect *binding points* and to

Otero-Cerdeira L., J. Rodríguez Martínez F., Valencia-Requejo T. and Gómez Rodríguez A..
OntoPhil - Exploitation of Binding Points for Ontology Matching.
DOI: 10.5220/0005011200050015
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2014), pages 5-15
ISBN: 978-989-758-049-9

turn the correct ones into the final output of the algorithm, i.e., the alignment between both ontologies.

The rest of the paper is organized as follows. Section 2 offers a vision of the state of the art in ontology matching. Then in section 3 we describe our algorithm in detail, and in section 4 we provide the evaluation and comparative results of the performance of our algorithm. Finally, section 5 includes the main conclusions and remarks.

2 RELATED WORK

The different available methods to compute the alignment between two ontologies can be classified into the following categories (Euzenat, 2004):

- *Terminological*: these methods are based on string comparison. The different string distance metrics used in these methods have been extensively tested and compared (Cohen et al., 2003).
- *Structural*: these methods are based on the structure of the entities (classes, individuals, relations) found in the ontology. This comparison can be internal, that is, regarding the internal structure of an entity but also attending to the comparison of the entity with other entities to which is related, being therefore external.
- *Extensional*: these methods compute the correspondences by analyzing the set of instances of the classes (extension).
- *Combinational*: these methods combine several of the previous methods.

This classification is made according to the *kind of input* which is used by the elementary matching techniques (Euzenat and Shvaiko, 2005). Nevertheless there are other classifications which consider different matching dimensions.

As seen in (Euzenat and Shvaiko, 2007), another classification can be also made according to the *granularity* of the matcher or to the interpretation of the *input information*.

Regarding *granularity*, matchers can be classified as:

- *Element-level*: these techniques compute correspondences analyzing just the entities without considering their relations with other entities.
- *Structure-level*: these techniques compute correspondences by analyzing how entities display in the structure of the ontology.

While according to the *input interpretation* matchers can be classified as:

- *Syntactic*: these techniques limit their input interpretation to the instructions stated in their corresponding algorithms.
- *External*: these methods exploit additional resources such as thesaurus or human knowledge to interpret the input.
- *Semantic*: these techniques use some formal semantics to interpret their input and justify their results.

These classifications for ontology matching techniques are mostly derived from classifications made for schema matching approaches such as those by Rahm & Bernstein (Rahm and Bernstein, 2001) or Do, Melnik & Rahm (Do et al., 2002), and from studies in the database field (Batini et al., 1986) (Spaccapietra and Parent, 1991). In fact, the ontology matching field has its origins in the database field from which it inherits several concepts and techniques, although it has evolved independently.

This independent evolution is reflected, on the one hand, in the amount of surveys and books that in the last decades specifically reviewed the ontology matching field such as (Rahm and Bernstein, 2001), (Choi et al., 2006), (Falconer et al., 2007), (Doan and Halevy, 2005), (Noy, 2004), (Euzenat and Shvaiko, 2005), (Gómez-Pérez and Corcho, 2002), (Parent and Spaccapietra, 2000). On the other hand, it is also stated by the number and variety of systems that address the ontology matching problem.

Some of the systems that have appeared in these years have participated in the Ontology Alignment Evaluation Initiative (OAEI) campaigns (Euzenat et al., 2011) which is an initiative that aims at evaluating the ontology matching tools.

In the OAEI 2012, 21 participants evaluated their algorithms in the different offered tracks. As revealed by a revision of these matching systems, the one that shares more similarities with ours is *LogMap*, although it is not the only one in literature that follows the same general outline. Other examples are: *Anchor-Flood* (Hanif and Aono, 2008), *Anchor-Prompt* (Noy and Musen, 2001), *ASCO* (Le et al., 2004), *Eff2Match* (Chua and Kim, 2010) and *SoBOM* (Xu et al., 2010). In all these systems, first some initial alignments are computed which are then used as a basis to continue obtaining new alignments between the input ontologies. Actually, the approach that we describe in this work is motivated by some of the ideas studied in these approaches, although there are fundamental differences in the way some structures are exploited and considered.

In the following we include a brief review of these algorithms.

Anchor-Flood initially takes as input an anchor that is obtained by a program module which is not considered part of the basic algorithm. Such module takes advantage of both lexical and statistical relational information to obtain some aligned pairs. Later, by exploiting the locality principle in the ontology graph, it assesses the neighboring information of these aligned pairs or anchors, obtaining for each one a pair of blocks that contain possible similar concepts between the ontologies. Then, these concepts are aligned exploiting their lexical, semantic and structural information with the purpose of discovering new pairs, which are later further processed. The main differences with *OntoPhil* are that it does not use external modules to compute the initial pairs, which are obtain using an internal lexical matcher, and that to exploit the initial pairs it uses just structural information.

Anchor-Prompt starts off a group of related terms (anchors), that can be both user-defined or automatically discovered by a lexical matcher. These initial pairs are seen as nodes in a sub-graph of the ontology. The algorithm continues with the assessment of which classes show a higher appearing frequency in the paths that connect these related terms, as they are considered as potentially new similar concepts. The similarity measure of these new pairs is computed by aggregating the measures obtained from all the paths where these pairs can be found. The only similarity with *OntoPhil* is the general outline to compute the initial pairs, done by means of a lexical matcher. The procedure followed to exploit these pairs is significantly different, as *OntoPhil* does not extract subgraphs and neither measures the frequency rate of the terms in the ontologies.

ASCO computes the alignment by applying TF/IDF techniques and relying in WordNet as external resource. In its initial linguistic phase, three different similarity measures are obtained (names, labels and descriptions). These measures are then combined to obtain the linguistic similarity value of the input entities. For names and labels, the similarity distance is computed with string metrics and WordNet while for the similarity of the description, TF/IDF techniques are used. Later, in a structural phase, the authors develop the idea that if the paths from the root of two classes (C_{O1} and C_{O2}) in the two hierarchies contain similar concepts, then C_{O1} and C_{O2} are likely to be similar too. The final output of the algorithm is obtained by combining the structural and linguistic values. The similarities with *ASCO* are limited to the general outline of the algorithm, which has two phases. However, the steps taken in each one of these phases are different. *OntoPhil* does not use TF/IDF techniques, and does not use different measures to compute the similarity values of the different input entities. Besides, the procedure to compute the structural similarity is different, as it will be explained in section 3.

Eff2Match computes the initial set of anchors with a exact string matcher that considers both names and labels of the entities. Then, it lists candidates for all entities in the source ontology that did not obtain a corresponding pair in the initial step. Namely, for each class, it obtains three vectors from the names, labels and comments (annotations) in the ancestors, descendants and the concept itself. Regarding properties, the vectors include the annotations for the property's domain and range classes as well as its own annotations. The similarity value between two concepts is obtained by means of an aggregation of the cosine similarity between the classes, ancestors and descendants vectors. After this, in the phase of anchor expansion, new pairs are detected using terminological methods to compare source and candidate entities. The procedure followed for *Eff2Match* is equivalent as the one in *OntoPhil*, however the way these initial pairs are exploit shows no resemblance. In *OntoPhil*, the initial pairs are exploit by means of a structural matcher, but the structural matcher itself does not output any similarity measure that has to be aggregated with the lexical one.

LogMap initially computes a set of correspondences known as anchor mappings which are almost exact lexical correspondences. To obtain these anchors, it obtains a lexical and a structural index of the ontologies. On these anchors, it iteratively applies, mapping repair and mapping discovery procedures which respectively refine and discover new mappings. From these anchors, new ones then are discovered by using the ontologies' extended class hierarchy. One of the main differences with *OntoPhil* is that we do not consider structural information to compute the initial pairs, besides, the procedure followed to discover new pairs is different.

SOBOM starts off a set of linguistically matched anchors. These anchors are obtained relying on textual (names, labels, etc.), structural (number of predecessor, descendants and constraints) and individual (number of existing individuals) information. Taking these anchors as starting point, the algorithm retrieves sub-ontologies and ranks them according to their depths. To obtain the concept alignments, it computes the similarity between the different subontologies obtained from the input ontologies according to their depths. Finally it uses the previously obtained concept alignments to retrieve the relationship alignments. The general outline is similar to *On-*

toPhil's, however the distances considered to retrieve these initial correspondences are different. Besides, *OntoPhil* does not retrieve sub-ontologies at any moment.

Another algorithm that we considered in our review is the one proposed by Akbari & Fathian in (Akbari and Fathian, 2010) which introduces a combinational approach to ontology matching. This algorithm sequentially applies lexical and structural matchers to compute the final similarity between the input ontologies. The lexical similarity is computed separately for the different kinds of entities (classes, object properties and data properties) and different similarity matrices are obtained. In the structural step, authors create a neighboring matrix for each node of ontology, then they compare the structure of these matrices to identify similar concepts. This algorithm also shares the same general outline with *OntoPhil*, as several of the previous ones, although the bigger differences are to be found in the actual procedure used to obtain the lexical similarity and the way the structural matcher is used.

Further details of the performance of *OntoPhil* are included in section 3, where it will become clearer the differences mentioned in this section.

3 A NEW ALGORITHM FOR ONTOLOGY MATCHING

This section presents our approach to ontology matching. It relies on the exploitation of some initial correspondences or *binding points* that connect both ontologies.

The process designed takes a couple of ontologies as input and consecutively applies, first some lexical matchers and later some structural matchers to obtain the final result, as represented in figure 1. Therefore the matching process followed by the algorithm is *sequential* as defined by Euzenat and Shvaiko in (Euzenat and Shvaiko, 2007).

We aim at discovering several *binding points* between the input ontologies by using some new lexical matchers. Thereafter, taking these *binding points* as pivots, an structural matcher is applied to the initial *binding points*. This matcher exploits particular features and properties of the ontologies to discover new *binding points*. Finally, by selecting and refining these *binding points*, we obtain the alignment between the input ontologies.

In the following sections we present a detailed view of each one of the steps and sub-steps of the algorithm. The order of the sections follows the sequence showed in figure 1.

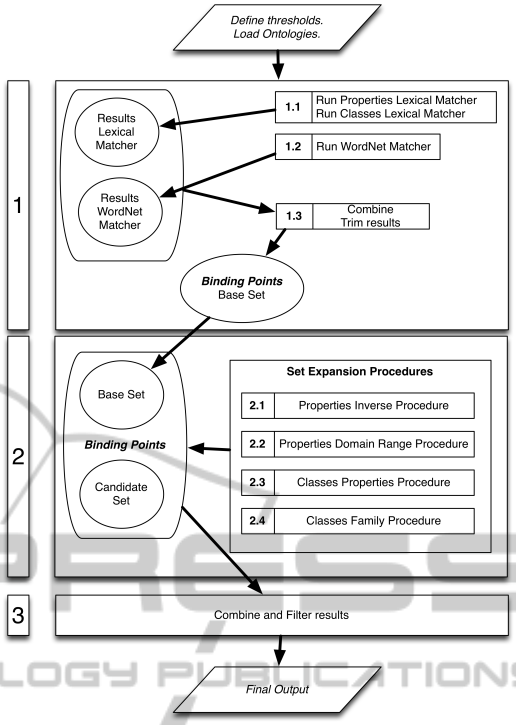

Figure 1: Schematic Diagram of Algorithm Steps.

3.1 *Step 1:* Obtaining Initial Binding Points

Obtaining the initial *binding points* is a crucial part of the matching process since these binding points are the basis on which the rest of the algorithm is built. To identify these initial *binding points*, we use two different lexical matchers. In particular, one uses the WordNet database as external resource and the other uses string and language-based distance measures. Using two matchers improves results because the second matcher helps in filtering the results provided by the first one. In this case, the amount of results obtained is sacrificed for their quality, since the better these initial *binding points*, the better the quality of the final output.

3.1.1 *Sub-step 1.1:* Lexical Matchers

This lexical matching phase will produce as result two distinct lexical matrices since classes and properties are separately aligned by means of a *ClassesLexicalMatcher (CLM)* and a *PropertiesLexicalMatcher (PLM)* respectively.

The $CLM = \{LexicalValue(c, c')\}$ denotes the matrix containing the set of similarity values between each $c \in C, c' \in C'$ and $PLM = \{LexicalValue(p, p')\}$ denotes the matrix containing the set of similarity val-

ues between each $p \in P, p' \in P'$ being, C and C' be the set of classes from ontologies o and o', and P and P' the set of properties from ontologies o and o' respectively.

Both these matching approaches use string and language-based methods to identify similar entities in the given ontologies. To do so a new distance measure is introduced.

This new measure is defined according to the following considerations. To compute the lexical similarity between two strings s_1 and s_2, the first step is to remove the figures that may appear on the strings. Numbers are omitted since they would only interfere in the final result and would not provide relevant matching information. After doing so, each string is tokenized by using as delimiters any non-alphabetical character, blank spaces and the uppercasing - lowercasing changes in the word, obtaining for each one of the original strings a bag of words, bs_1 and bs_2 respectively. Then, these bags of words are compared with the following procedure.

Step 1. First, the words shared by the two bags are removed, hence obtaining bs'_1 and bs'_2.

Step 2. If both bags are empty then the similarity measure between the input strings is 1.0. Otherwise, all the words left in the first bag bs'_1 are compared to the words left in the second bag bs'_2 by using the ***Jaro-Winkler Distance*** (Cohen et al., 2003) and considering the ***Levenshtein Distance*** (Cohen et al., 2003) as a complementary measure.

Step 3. If the Jaro-Winkler distance measure of two words, (a,b), returns a number greater than α, and for these words the Levenshtein distance measure returns a number lower or equal than β, then the combined similarity value of these words is set to 1.0. In previous works, these thresholds, α and β, have been empirically set to 0.90 and 1.0 respectively.

$$(JaroWinkler(a,b) > \alpha) \cap (Levenshtein(a,b) < \beta) \Rightarrow sim(a,b) = 1.0 \quad (1)$$

Step 4. In case Levenshtein distance measure indicates that the shortest word must be completely modified to be matched to the second one, this causes a proportional forfeit in the combined result. Otherwise, the result of the Jaro-Winkler distance measure is stored as result for that pair of words (a,b) in the bags.

$$Levenshtein(a,b) \geq minLength(a,b) \Rightarrow sim(a,b) = JaroWinkler(a,b) * (1 - \frac{Levenshtein(a,b)}{maxLength(a,b)}) \quad (2)$$

Step 5. Once every possible pair of words is assigned a value, the final result for the bags is computed by adding them up. In this sum an improvement factor ϕ is used to strengthen the similarity of the bags that share several words.

$$bagEvaluation(s_1,s_2) = \frac{\sum pairEvaluation(a,b) + repeatedWords * \phi}{1 + repeatedWords} \quad (3)$$

Step 6. The original strings, s_1 and s_2 are also compared using the Jaro-Winkler and Levenshtein distances. As final result, the best score between the original comparison and the bags comparison is returned.

$$LexicalValue(s_1,s_2) = max[bagEvaluation(s_1,s_2), stringEvaluation(s_1,s_2)] \quad (4)$$

This procedure is used in the following example. Let s_1 and s_2 be respectively: s_1 = *"ConferenceDinner"* and s_2 = *"Conference_Banquet"*.

[Step 1]
bs_1 = *[conference, dinner]*; bs_2 = *[conference, banquet]*
After creating the bags of words, the word *"conference"* is removed from both bags. So the bags remain as follows:
bs'_1 = *[dinner]*; bs'_2 = *[banquet]*

[Step 2]
JaroWinkler (dinner, banquet) = 0.54
Levenshtein(dinner,banquet) = 5.0

[Step 3]
JaroWinkler (dinner, banquet) $< \alpha \Rightarrow$
sim(dinner, banquet) $\neq 1.0$

[Step 4]
Levenshtein (dinner, banquet) < 6 *(no forfeit applied)*
sim(dinner, banquet) $= 0.54 * (1 - \frac{5}{7}) = 0.15$

[Step 5]
Since the word "conference" is common to both bags, the improvement factor ϕ is applied: *bagEvaluation(dinner, banquet)* = $\frac{0.15+1*\phi}{2}$ = *0.57*

[Step 6]
JaroWinkler(ConferenceDinner,Conference_Banquet)= 0.88
Levenshtein(ConferenceDinner,Conference_Banquet) = 6.0

LexicalValue(ConferenceDinner,Conference_Banquet)= max[0.57,0.88] = 0.88

Considering these results the final evaluation of the similarity for the strings *"ConferenceDinner"* and *"Conference_Banquet"* would be 0.88.

We compute separately the similarity among classes, object properties and data properties of the two input ontologies using this procedure and so, three lexical similarity matrices are obtained.

3.1.2 *Sub-step 1.2:* WordNet Matcher

As stated in figure 1, next step is to run the *WordNet matcher*. The WordNet matcher outputs a matrix that contains the set of similarity values between each class and property from the source ontology and the target ontology.

$$WNM = \{\{WordNet(c,c')\} \cup \{WordNet(p,p')\}\} \quad (5)$$

These similarity values are used to asses the accuracy of the values provided by the lexical matchers.

3.1.3 *Sub-step 1.3:* Combine and Select results

The matrices resulting from the previous steps are combined in a new structure, where all the candidate correspondences generated by the properties lexical matcher and the classes lexical matcher, are joined in a set.

This set is composed by tuples of the form: $(e_1, e_2, LexicalValue(e_1,e_2), WordNetValue(e_1,e_2))$, where e_1 and e_2 are the entities linked (classes or properties), $LexicalValue(e_1,e_2)$ is the value obtained from running the $ClassesLexicalMatcher(e_1,e_2)$ or the $PropertiesLexicalMatcher(e_1,e_2)$ and $WordNetValue(e_1,e_2)$ is the value obtained from running the $WordNetMatcher(e_1,e_2)$.

Next, all the initial *binding points* or candidate correspondences are sifted out. The sifting out is fundamental for the algorithm since it reduces the number of *binding points* that are used as the starting point for the next step. For this purpose, only the most accurate points are used, so a sifted set, (*Sifted_CS*), containing just the elements with a lexical value of 1.0 is constructed.

The better these initial binding points the higher the chances that the correspondences obtained in *Step 2* (see 3.2) are valid ones, therefore only those binding points with the highest lexical similarity (1.0) are chosen to be the start point of the *set expansion procedures*.

These expansion procedures aim at discovering new *binding points* between the two input ontologies by exploiting structural features of the ontologies.

3.2 *Step 2:* Discovering New Binding Points

As stated before, the sifted set (*Sifted_CS*), obtained in the previous step, becomes the *base set* on which the *expansion procedures* are built. The candidate *binding points* obtained in this step, can link either properties or classes and therefore there are *properties candidate correspondences* and *classes candidate correspondences*.

Depending on the structural feature that is exploited in each procedure, there is a different likelihood that the discovered candidate *binding points* are promising, therefore some of these procedures will directly update the *base set* by modifying or inserting new *binding points* while others will insert them in a *candidate set*.

Each one of the new discovered *binding points* is placed in a bag that identifies the procedure and subprocedure that led to its discovery. The procedures are sequentially applied, and each one exploits a different feature of the ontologies. In case a binding point is reached by several procedures, it is saved in all the corresponding bags. Initially those class and property candidate correspondences which are *binding points* in the *base set*, are put in the bags *CC_BASE_SET* and *CP_BASE_SET* respectively.

The different procedures applied are detailed in the following subsections. From equations (6) to (11), the notation used is represented in table 1.

Table 1: Notation used in equations.

Symbol	Meaning
q_1, p_1	properties in $Ontology_1$
q_2, p_2	properties in $Ontology_2$
c_1	class in $Ontology_1$
c_2	class in $Ontology_2$
$dom(p_1)$	domain set for property p_1
$dom(p_2)$	domain set for property p_2
$ran(p_1)$	range set for property p_1
$ran(p_2)$	range set for property p_2
$\#dom(p_1)$	domain cardinality for property p_1
$\#dom(p_2)$	domain cardinality for property p_2

3.2.1 *Sub-step 2.1:* Properties Inverse Procedure

The aim of this procedure is to discover new property candidate correspondences. For every properties correspondence in the *base set*, the procedure retrieves its *inverse properties correspondence*.This is only feasible if the properties from the original correspondence have defined an inverse property using the construct $owl:inverseOf$.

$$\forall(q_1,q_2,PLM(q_1,q_2),WNM(q_1,q_2)) \in BASE\_SET \iff \exists[((p_1,p_2,PLM(p_1,p_2),WNM(p_1,p_2)) \in BASE\_SET) \wedge(\exists q_1 \equiv \neg p_1) \wedge (\exists q_2 \equiv \neg p_2)] \quad (6)$$

If the inverse correspondence already exists in the *base set* the counter of its occurrences is increased, otherwise the new correspondence is inserted in the *CP_BASE_SET_INVERSE* bag.

3.2.2 *Sub-step 2.2:* Properties Domain Range Procedure

This procedure allows the identification of new class candidate correspondences. The first step is to retrieve all the property candidate correspondences in the *base set*. Thereafter the domain and range for each property in the correspondence is recovered from their corresponding ontologies.

Shall we consider first the domain sets. If both of them have only one class each, then these two classes constitute a new class candidate correspondence (see equation (7)). In case this new correspondence already exists in the *base set* or *candidate set* the number of its occurrences is increased, otherwise it is inserted in the *candidate set* in the bag *CC_DIRECT_DERIVED*.

$$\exists c_1 \in dom(p_1) \wedge \exists c_2 \in dom(p_2) \wedge \#dom(p_1) = \#dom(p_2) = 1 \Rightarrow (c_1,c_2,CLM(c_1,c_2),WNM(c_1,c_2)) \in BASE\_SET \quad (7)$$

If the cardinality of any of the domain sets is higher than 1, the *superclasses approach*, described next, is followed. The correspondences discovered with this procedure are inserted in the bag *CC_DIRECT_DERIVED_WITH_SET*.

For each of the classes in both domain sets, its superclasses are retrieved recursively until the higher level in the ontology hierarchy is reached. This way, for every class in every domain a temporary set containing all its superclasses is obtained. If there is a common superclass to several classes in the same domain set, then the intermediate classes are dismissed and only the initial class and the common superclass are considered. All the selected classes are integrated in the same set, so we will have only one final set associated to each domain. These sets are represented as $sup(dom(p_1))$ and $sup(dom(p_2))$ respectively, in equation (8). The classes from these final sets are combined to obtain new candidate correspondences.

$$sup(dom(p_1)) \times sup(dom(p_2)) = \{(c_1,c_2,CLM(c_1,c_2),WNM(c_1,c_2))\} \quad where \quad (c_1 \in sup(dom(p_1)) \wedge c_2 \in sup(dom(p_2))) \quad (8)$$

If any of these new correspondences already exists in the *base set* or *candidate set* their number of occurrences are properly modified. For the rest of the new correspondences, their lexical and WordNet values are retrieved from the matrices resulting from the lexical and WordNet matcher respectively, to create a new tuple. If the lexical value surpasses the threshold δ then the tuple is inserted in the *candidate set*, otherwise it is dismissed (see equation (9)). The δ threshold limits the amount of new correspondences that are inserted in the set by dismissing the less promising ones. We have experimentally determined, in previous works, that a value of 0.9 for δ threshold achieves the best results.

$$\forall t \in \{(c_1,c_2,CLM(c_1,c_2),WNM(c_1,c_2))\}, \quad t \in CANDIDATE\_SET \iff t.s^{lex} > \delta \quad (9)$$

This procedure that we have just described for the domain sets is also applied for the range sets.

3.2.3 *Sub-step 2.3:* Classes Properties Procedure

This procedure discovers not only class candidate correspondences but also property candidate correspondences. To do so, the first step is to retrieve all the existing class candidate correspondences from the *base set* and then, for every pair of classes, their *correlated properties set* is put together.

The set of correlated properties is the result of the cartesian product of the properties from the source ontology whose domain or range includes the first class of the explored correspondence, with the properties from the target ontology, whose domain or range includes the second class in that correspondence.

To choose a pair of properties for the correlated set, if the first class in the explored pair belongs to the domain of the first property in the candidate pair, then the second property in the candidate pair must also have the second class in the explored pair as part of its domain, otherwise the pair of properties is dismissed (see equation (10)).

$$Correlated\_Properties(c_1, c_2) = \{p_1\} \times \{p_2\} \quad where \quad (c_1 \in dom(p_1) \wedge c_2 \in dom(p_2)) \vee (c_1 \in ran(p_1) \wedge c_2 \in ran(p_2)) \tag{10}$$

Once all the correlated properties are identified, the procedure continues assessing their domains and ranges to determine whether there can be found new correspondences or not.

For each pair of properties belonging to the correlated properties set, their domains and ranges are retrieved. If a pair of properties is inserted in the correlated properties set because their respective domains has a pair of classes already aligned in the *base set*, then the domains receive the *aligned sets sub-procedure*, and the ranges receive the *non-aligned set sub-procedure*.

- ***Aligned Set Sub-procedure***

  The initial classes whose correlated properties are under assessment are removed from the aligned sets, since these classes are already part of the *base set*. In case that, after doing so, there is only one class left in each one of the aligned sets, then a new class candidate correspondence has been identified. If this new identified correspondence already exists in the *base set* or in the *candidate set* its number of occurrences is accordingly modified. Otherwise this new correspondence is added to the *candidate set*. This correspondence of classes is inserted as a new class candidate correspondence in the bag *CC_SOURCE_INC_WITHOUT_SET*.

  If the cardinality of the sets is bigger than 1 then the superclasses approach previously described in section 3.2.2 is followed, although the new correspondences are placed in the bag *CC_SOURCE_INC_WITH_SET*.

- ***Non-Aligned Set Sub-procedure***

  In the non-aligned sets, the first thing to check is the cardinality, if both sets have just one class then a new class candidate correspondence is created and put in the bag *CC_DIRECT_CLASSES*. This new correspondence is confronted separately with the *base set* and *candidate set*.

  If the class candidate correspondence already exists in the *base set*, then its number of occurrences is increased and the properties that link these classes become a new property candidate correspondence. This new correspondence is evaluated against the *base set* too in order to check for previous occurrences. Its inverse candidate correspondence is also assessed.

  These new property candidate correspondences are respectively set in bags *CP_DIRECT_CLASSES* and *CP_DIRECT_CLASSES_INVERSE*. If the class candidate correspondence is identified as new, then it is inserted in the *base set*.

  If the class candidate correspondence exists in the *candidate set* its number of occurrences is increased, and the corresponding properties are evaluated against the *candidate set* to become a new property candidate correspondence. Its inverse correspondence is also assessed. If the class candidate correspondence is identified as new it is placed in the bag *CC_PROPERTIES_WITHOUT_SET* and then it is inserted in the *candidate set* as well as the new property candidate correspondences identified in this step. The property candidate correspondence and its inverse, in case it exists, are placed in *CP_DIRECT_CLASSES* and *CP_DIRECT_CLASSES_INVERSE* respectively.

  In case the non-aligned sets cardinality is different from 1, then the superclasses approach is followed, as described in section 3.2.2. In this case the bag for the new class candidate correspondences is *CC_PROPERTIES_WITH_SET* and for the new property candidate correspondences *CP_DIRECT_CLASSES_WITH_SET* and *CP_DIRECT_CLASSES_WITH_SET_INVERSE*.

3.2.4 *Sub-step 2.4:* Classes Family Procedure

The Classes Family Procedure is the family approach to identifying new class candidate correspondences. Following this approach class candidate correspondences from the *base set* are retrieved and their familiar relations are exploited.

For each one of the classes in the class candidate correspondences, its superclasses, subclasses and sibling classes are recovered, identified in equation (11) as $sup(class)$, $sub(class)$ and $sib(class)$ respectively. Then, a cartesian product is applied between the two sets of classes recovered for each correspondences, dividing the results into the three identified levels. The new identified pairs are compared with the existing ones in both *base set* and *candidate set*.

$$FamiliarCorrespondences = \{(sup(c_1) \times sup(c_2)) \cup (sub(c_1) \times sub(c_2)) \cup (sib(c_1) \times sib(c_2))\} \tag{11}$$

As it has been done in previous steps, if a correspondence already exists in any set (*base set* or *candidate set*) its number of occurrences is increased, otherwise these new correspondence are inserted in the *candidate set* provided that the lexical valued corresponding to these new tuples surpass the δ threshold.

These new tuples are placed in the bag *CC_FAMILY*.

By applying the procedures described in sections 3.2.1 to 3.2.4 new *binding points* are discovered and classified in different types of bags.

3.3 Selecting Binding Points

After applying the procedures presented in section 3.2, a final version of the *base set* and *candidate set* is obtained. In case that any of these methods has caused the modification of the initial *base set* with new *binding points*, a new iteration is done. In this new iteration the *base set* is composed only by those *binding points* that were inserted in the *base set* in the previous iteration.

Iterations stop when no more modifications are done in the *base set* so the system converges. This convergence is always achieved as the number of possible binding points is finite.

After finishing all the iterations, results are combined and the selection process begins. This process is essential since it is a way of dismissing false correspondences and therefore defining the final output of the algorithm. For this process various restrictions have been defined which treat differently class candidate correspondences and property candidate correspondences. These restrictions are based in the idea that the different procedures exploit different features of the ontologies and therefore they outcome correspondences with different levels of accuracy. Hence, the location of the correspondences in different bags facilitates their selection, in order to choose for the final output the best possible ones.

At present, some of the simpler restriction rules taken into account for the algorithm are:

- *Retrieve those correspondences in the bag: CC_BASE_SET.*
- *Retrieve those correspondences in the bag: CC_DIRECT_DERIVED.*
- *For those property correspondences that share the same entity in the source ontology, retrieve those whose mean, between the WordNet value and lexical value, is higher.*
- *For property correspondences retrieve the Singletons.*

4 EVALUATION

The goal of any algorithm for ontology matching is to generate an alignment that discovers the correspondences, and only the correct ones, between two input ontologies. The correctness of these correspondences is evaluated against a reference alignment provided by the human interpretation of the meaning of the input ontologies.

To evaluate the accuracy of an ontology matching algorithm, the standard information retrieval metrics of *Precision, Recall and F-measure* are used.

The *precision* measures the ratio of correctly found correspondences over the total number of returned correspondences, which in logical terms reflects the degree of correctness of the algorithm.

The *recall* measures the ratio of correctly found correspondences over the total number of expected correspondences which in logical terms measures the degree of completeness of the alignment.

Even though precision and recall are widely used and accepted measures, in some occasions it may be preferable having a single measure to compare different systems or algorithms. Moreover, systems are often not comparable based solely on precision and recall. The one which has a higher recall may have a lower precision and vice versa (Euzenat and Shvaiko, 2007) and therefore the *f-measure* was introduced, as a ratio between precision and recall.

We have evaluated the accuracy of our algorithm using these measures and comparing our results with those provided by the algorithm of Akbari & Fathian (Akbari and Fathian, 2010) due to the similarities that our algorithm's structure shares with theirs because both algorithms use lexical and structural matchers applied sequentially to discover the correspondences between the ontologies. Akbari & Fathian's algorithm was tested using the benchmark test set provided in the Ontology Alignment Evaluation Initiative 2008 (OAEI-08) (Caracciolo et al., 2009).

This is a well known benchmark series of tests that has been used for several years. This allows the comparison with other systems since 2004. This benchmark is built around a seed ontology and variations of it (Aguirre et al., 2013) and its purpose is to provide a stable and detailed picture of the algorithms. These tests were organized into *simple tests (1xx)*, *systematic tests (2xx)* and *real-life ontologies (3xx)*. Recently the structure of the benchmark was changed and *real-life ontologies* were removed.

Table 2: Average performance of the algorithm proposed by Akbari & Fathian on the OAEI-08 benchmark test suite.

	1xx	**2xx**	**3xx**	**Average**
Precision	0.98	0.78	0.87	0.88
Recall	0.95	0.74	0.84	0.85
F-measure	0.96	0.75	0.85	0.86

From the results presented in tables 2 and 3, our algorithm shows better *precision* and *f-measure* values than the system proposed by Akbari & Fathian.

Table 3: Average performance of our algorithm on the OAEI-08 benchmark test suite.

	1xx	**2xx**	**3xx**	**Average**
Precision	0.97	0.96	0.92	0.95
Recall	1	0.71	0.82	0.85
F-measure	0.99	0.77	0.86	0.88

It is specially important for us the results obtained in the set of tests *3xx* (real-life ontologies), since we aim at integrating our system in a real-world application.

Besides, we have compared the results of our algorithm with those provided by the algorithms participating in the Ontology Alignment Evaluation Initiative 2012 (OAEI-12), as shown in table 4.

From these results we can outline that the proposed algorithm has a better average behavior than most of the competing systems, since even if our approach has not the highest value in any of the three measures *precision, recall and f-measure*, it certainly has the most balanced ones.

Table 4: Results obtained by the participants in the OAEI-12 compared with our approach.

	edna	AROMA	ASE
P	0.35	0.98	0.49
R	0.41	0.77	0.51
F	0.50	0.64	0.54
	AUTOMSV2	GOMMA	Hertuda
P	0.97	0.75	0.90
R	0.69	0.67	0.68
F	0.54	0.61	0.54
	HotMatch	LogMap	LogMapLt
P	0.96	0.73	0.71
R	0.66	0.56	0.59
F	0.50	0.46	0.50
	MaasMatch	MapSSS	MEDLEY
P	0.54	0.99	0.60
R	0.56	0.87	0.54
F	0.57	0.77	0.50
	Optima	ServOMap	ServOMapLt
P	0.89	0.88	1.0
R	0.63	0.58	0.33
F	0.49	0.43	0.20
	WeSeE	WikiMatch	YAM++
P	0.99	0.74	0.98
R	0.69	0.62	0.83
F	0.53	0.54	0.72
	OntoPhil		
P	0.95		
R	0.83		
F	0.79		

5 CONCLUSIONS

In this paper we have presented a novel ontology matching algorithm that finds correspondences among entities of input ontologies based on their lexical and structural information. The algorithm proposed relies on the exploitation of some initial correspondences or *binding points* that connect one ontology to the other. This is an adaptive way of matching two ontologies which allows the identification of new correspondences by exploiting the particular features of the matched ontologies.

Likewise, we have introduced a new lexical measure that determines the lexical similarity among entities by using the terminological information available for each pair of entities. For this lexical similarity measure a full and detailed example is provided to show how the similarity is computed.

The comparison of our algorithm to other similar ones, reflects that it has good and balanced results that encourage the work on this research line.

In spite of the promising start of this line work, there are still several steps that must be taken before it is fully functional. First, there is work to do to improve the algorithm. For instance, to improve the calculation of the initial binding points, to refine of the restriction rules so the amount of false positives retrieved in the alignment are kept to a minimum or to include other features such as multilingual support and instance matching. These improvements are being developed to be included in the algorithm as a way to enhance the results of *OntoPhil*.

There is also the need to validate the usefulness of the approach by testing it within a real environment. These real-environment tests will be very useful to check the robustness, scalability and flexibility of the algorithm, and its applicability in real-life problems. Our research group is currently working on applying this algorithm to a Smart City environment.

REFERENCES

Aguirre, Jose Luis; Eckert, Kai; Euzenat, Jérôme; Ferrara, Alfio; van Hage, Willem Robert; Hollink, Laura; Meilicke, Christian; Nikolov, Andriy; Ritze, Dominique; Scharffe, Francois; Shvaiko, Pavel; Svab-Zamazal, Ondrej; Trojahn, Cássia; Bernardo, Ernesto Jimenez-Ruiz; Grau, Cuenca, and Zapilko, Benjamin. Results of the Ontology Alignment Evaluation Initiative 2012. *The 7th International Workshop on Ontology Matching*, 2013.

Akbari, Ismail and Fathian, Mohammad. A Novel Algorithm for Ontology Matching. *Journal of Information Science*, 36(324):12, 2010.

Antoniou, Grigoris and van Harmelen, Frank. *Semantic Web Primer*. The MIT Press, 2004.

Batini, Carlo; Lenzerini, Maurizio, and Navathe, Shamkant Bhalchandra. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4), 1986.

Caracciolo, Caterina; Euzenat, Jérôme; Hollink, Laura; Ichise, Ryutaro; Isaac, Antoine; Malaisé, Véronique; Meilicke, Christian; Pane, Juan; Shvaiko, Pavel; Stuckenschmidt, Heiner; Šváb-Zamazal, Ondrej, and Svátek, Vojtech. Results of the Ontology Alignment Evaluation Initiative 2008. *Ontology Matching Workshop*, 2009.

Choi, Namyoun; Song, Il-Yeol, and Han, Hyoil. A Survey on Ontology Mapping. *SIGMOD Record*, 35(3), 2006.

Chua, Watson Wei Khong and Kim, Jung-Jae. Eff2Match results for OAEI 2010. *CEUR Workshop Proceedings*, 689, 2010.

Cohen, William W.; Ravikumar, Pradeep, and Fienberg, Stephen E. A Comparison of String Distance Metrics for Name-Matching Tasks. *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, 2003.

Cuenca-Grau, Bernardo; Horrocks, Ian; Motik, Boris; Parsia, Bijan; Patel-Schneider, Peter, and Sattler, Ulrike. OWL 2: The Next Step for OWL. *Journal Web Semantics: Science, Services and Agents on the World Wide Web*, 6:309–332, 2008.

Do, Hong-Hai; Melnik, Sergei, and Rahm, Erhard. Comparison of Schema Matching Evaluations. *NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, 2593:221–237, 2002.

Doan, Anhai and Halevy, Alon Y. Semantic Integration Research in the Database Community: A Brief Survey. *American Association for Artificial Intelligence*, 26: 83–94, 2005. URL `www.aaai.org`.

Ehrig, Marc and Euzenat, Jérôme. Relaxed Precision and Recall for Ontology Matching. *Integrating Ontologies*, 156:8, 2005.

Euzenat, Jérôme. State of the art on Ontology Alignment. *Knowledge Web*, 2, 2004.

Euzenat, Jérôme and Shvaiko, Pavel. A Survey of Schema-based Matching Approaches. *Journal on Data Semantics IV*, 5:21, 2005. URL www.ontologymatching.org.

Euzenat, Jérôme and Shvaiko, Pavel. *Ontology Matching*. Springer-Verlag, Berlin Heidelberg (DE), 2007.

Euzenat, Jérôme; Meilicke, Christian; Stuckenschmidt, Heiner; Shvaiko, Pavel, and dos Santos, Cássia Trojahn. Ontology Alignment Evaluation Initiative: Six Years of Experience. *Journal on Data Semantics - Springer*, 15:158 – 192, 2011.

Falconer, Sean M.; Noy, Natalya F., and Storey, Margaret-Anne. Ontology Mapping - A User Survey. *Proceedings of the Workshop on Ontology Matching (OM2007) at ISWC/ASWC2007*, pages 113–125, 2007.

Gómez-Pérez, Asunción and Corcho, Óscar. A survey on ontology tools. *OntoWeb*, 2002.

Hanif, Md. Seddiqui and Aono, Masaki. Anchor-Flood: Results for OAEI 2009. In Shvaiko, Pavel; Euzenat, Jérôme; Giunchiglia, Fausto; Stuckenschmidt, Heiner; Noy, Natalya Fridman, and Rosenthal, Arnon, editors, *OM*, volume 551 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

Le, Bach Thanh; Dieng-Kuntz, Rose, and Gandon, Fabien. On Ontology Matching Problems - for Building a Corporate Semantic Web in a Multi-Communities Organization. In *ICEIS (4)*, pages 236–243, 2004.

Noy, Natalya F. Semantic Integration: A Survey Of Ontology-Based Approaches. *SIGMOD*, 33(4), 2004.

Noy, Natalya F. and Musen, Mark A. Anchor-prompt: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 63–70, Seattle (USA), 2001.

Parent, Christine and Spaccapietra, Stefano. Database integration: The key to data interoperability. *Advances in Object-Oriented Data Modeling*, pages 221–253, 2000.

Rahm, Erhard and Bernstein, Philip A. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10:334 – 350, 2001. doi: 10.1007/s007780100057.

Shvaiko, Pavel and Euzenat, Jérôme. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Software Engineering*, 25(1), 2013.

Spaccapietra, Stefano and Parent, Christine. Conflicts and Correspondence Assertions in Interoperable Databases. *SIGMOD Record*, 20(4):49–51, 1991.

Suárez-Figuero, Carmen; García-Castro, Raúl; Villazón-Terrazas, Boris, and Gómez-Pérez, Asunción. Essentials In Ontology Engineering: Methodologies, Languages, And Tools. *Bioinformatics*, 2011.

van Aart, Chris; Pels, Ruurd; Caire, Giovanni, and Bergenti, Federico. Creating and Using Ontologies in Agent Communication. *Telecom Italia EXP magazine*, 2002.

W3C, . OWL: Web Ontology Language, 2013a. URL www.w3.org/2004/OWL/.

W3C, . OWL 2: Web Ontology Language, 2013b. URL http://www.w3.org/TR/owl2-overview/.

Xu, Peigang; Wang, Yadong; Cheng, Liang, and Zang, Tianyi. Alignment results of SOBOM for OAEI 2010. *CEUR Workshop Proceedings*, 689, 2010.