Exploiting Prolog and NLP Techniques for Matching Ontologies and for Repairing Correspondences*

Viviana Mascardi[1], Angela Locoro[2], and Fabrizio Larosa[1]

[1]DISI, Università degli Studi di Genova, Via Dodecaneso 35, 16146, Genova, Italy
viviana.mascardi@unige.it, 2003s140@educ.disi.unige.it
[2]DIBE, Università degli Studi di Genova, Via All'Opera Pia 11a, 16145 Genova, Italy,
angela.locoro@unige.it

Abstract. Providing efficient ontology matching algorithms is one of the means for pursuing semantic interoperability. In this paper we discuss an algorithm that exploits natural language processing techniques for matching ontologies and that post-processes the obtained alignment in order to find semantic inconsistencies. The algorithm has been entirely implemented in Prolog, whose usefulness was mainly evident in the post-processing phase. A careful analysis of the recent state-of-the art witnesses the originality of our matching algorithm which is based on the "Adapted Lesk Algorithm" for word sense disambiguation. The experiments we carried out, although in their early stages, are encouraging.

Keywords. Computational Logic, Semantic Interoperability, Natural Language Processing, Ontology Matching, Correspondence Repair

1 Introduction

Semantic interoperability, namely "*the ability of two or more computer systems to exchange information and have the meaning of that information automatically interpreted by the receiving system accurately enough to produce useful results*"[1], is one of the most relevant and lively research fields of the last fifteen years. The advent of ontologies in computer science in the early nineties [14], the settlement of Web Services in the beginning of the new millennium, the combination of both into Semantic Web Services, all witness the fervid activity of academia and industry for finding algorithms, languages, tools, and infrastructures aimed at realising the "semantic interoperability dream".

The reason for the interest in this topic is easy to explain. In a recent paper [26], Jan Walker et al. assess the value of information exchange and interoperability in the domain of health care, and state that a fully standardised system supporting information exchange and interoperability could yield a net value of 77.8 billion dollars per year once implemented. Other studies [10,5] confirm these impressive economic advantages in implementing solutions for semantic interoperability.

Since knowledge is more and more represented by means of ontologies, and different ontologies must be matched in order to make knowledge exchange possible, one

* Partially supported by the Italian project "Iniziativa Software CINI-FinMeccanica".

[1] Wikipedia, http://en.wikipedia.org/wiki/Semantic_interoperability, accessed March, 15, 2009.

of the means for pursuing semantic interoperability is that of providing efficient ontology matching algorithms. The first formalisation of the matching problem dates back to 2000 [2]. Today, due to the ever increasing availability of many ontologies in very different domains, ontology matching is becoming one of the most important activities in the Semantic Web area.

This paper deals with the problem of finding semantically correct mappings between couples of ontologies. It exploits an algorithm used for word sense disambiguation originally proposed by Michael Lesk [16] and adapted by Satanjeev Banerjee and Ted Pedersen [1]. The obtained alignment is then post-processed in order to further refine it by finding semantic inconsistencies.

The paper is organised in the following way: Section 2 provides some background knowledge on WordNet, the Lesk algorithm, ontology matching, and alignment repair. Section 3 discusses the algorithms that we have implemented, whereas Section 4 describes their Prolog implementation and discusses the preliminary results we have obtained. Section 5 concludes and highlights future directions of our work.

2 Background

In this section we provide a short background on the topics upon which our research roots: WordNet, the Lesk algorithm for word sense disambiguation, ontology matching, and alignment repair.

2.1 WordNet

WordNet [9] is a large lexical database of English, developed under the direction of George A. Miller. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. A synset or synonym set is defined as a set of one or more synonyms that are interchangeable in some context without changing the truth value of the proposition in which they are embedded.

Most synsets are connected to other synsets via a number of semantic relations that include:

Semantic relations between nouns. *Hypernyms*: Y is a hypernym of X if every X is a (kind of) Y; *hyponyms*: Y is a hyponym of X if every Y is a (kind of) X; *coordinate terms*: Y is a coordinate term of X if X and Y share a hypernym; *holonym*: Y is a holonym of X if X is a part of Y; *meronym*: Y is a meronym of X if Y is a part of X.

Semantic relations between verbs. *Hypernym*: the verb Y is a hypernym of the verb X if the activity X is a (kind of) Y; *troponym*: the verb Y is a troponym of the verb X if the activity Y is doing X in some manner; *entailment*: the verb Y is entailed by X if by doing X you must be doing Y; *coordinate terms*: those verbs sharing a common hypernym.

While semantic relations apply to all members of a synset because they share a meaning but are all mutually synonyms, words can also be connected to other words through lexical relations, including antonyms (opposites of each other) and derivationally related, as well.

2.2 Lesk Algorithm

The Lesk algorithm is an algorithm for word sense disambiguation introduced by Michael E. Lesk in 1986 [16]. It applies to short sentences and uses the number of words common to the definitions, or *glosses*, of each word in the sentence as a measure representing how much that gloss correctly defines the word. For example, if the sentence to disambiguate contains *bass* and *fishing*, and both words are defined by one gloss that contains *fish* (suppose gloss "a fish with a spiny dorsal fin, related to or resembling the perch"[2] for *bass*, and gloss "the sport, industry, or occupation of catching fish"[3] for *fishing*), then these two glosses have some chance to be the right ones for *bass* and *fishing* respectively. The original Lesk algorithm uses dictionaries like Oxford Advanced Learner's to retrieve glosses.

In 2002, Satanjeev Banerjee and Ted Pedersen adapted Lesk algorithm by using WordNet as the source of glosses [1]. They also adapted the algorithm in order to disambiguate each word in a sentence, whatever the sentence length. The algorithm operates in the following way:

Input: a sentence s; a target word tw to disambiguate within that sentence; the dimension $2n + 1$ of the window to consider as a context for tw
Output: a gloss that provides the right definition for tw
begin

1. create the list $context$ that contains the n words that belong to WordNet and that come before tw in s, the n words that belong to WordNet and that come after tw in s, and tw;
2. if tw is near the beginning or the end of the sentence, take additional WordNet words from the other direction;
3. for each word w in $context$
   (a) retrieve its synonyms, hypernyms, hyponyms, holonyms, meronyms, troponyms, and attributes from WordNet;
   (b) if the part of speech (noun, adjective, verb, ...) p that w plays in the sentence is known, than take into account only relations and synsets associated with w as a p, otherwise consider all possible relations and synsets for w;
   (c) retrieve glosses for each word in the list of synonyms and senses related to w obtained in the previous step: be it $g_1(w), g_2(w), ..., g_n(w)$;
   (d) the score of each gloss in the list associated with w, $score(g_i(w))$, is initially set to 0;
4. for each pair of words wa, wb in $context$
   (a) for each pair of glosses $g_j(wa)$ belonging to the list of synonyms of and sensed related to wa and $g_k(wb)$ belonging to the list of synonyms of and sensed related to wb
      i. define an overlap between $g_j(wa)$ and $g_k(wb)$ as the longest sequence of common consecutive words: this overlap results into a score stating how much $g_j(wa)$ is the right definition for wa, and $g_k(wb)$ is the right definition for wb;

[2] AskOxford.com, `http://www.askoxford.com/`.
[3] Msn Encarta Dictionary, `http://encarta.msn.com/encnet/features/dictionary/dictionaryhome.aspx`.

ii. update $score(g_j(wa))$ and $score(g_k(wb))$ by adding the scores obtained in the previous step;

5. to disambiguate the target word tw, select the gloss in $g_1(tw), g_2(tw), ..., g_m(tw)$ with highest score, and return it

end

2.3 Ontology Matching

This section is based on [8] and adopts definitions similar to those used there, eventually simplified for sake of clarity.

Definition 1: Correspondence. A correspondence between an entity e belonging to ontology o and an entity e' belonging to ontology o' is a 5-tuple $< id, e, e', R, conf >$ where:

- id is a unique identifier of the correspondence;
- e and e' are the entities (e.g. properties, classes, individuals) of o and o' respectively;
- R is a relation such as "equivalence", "more general", "disjointness", "overlapping", holding between the entities e and e'.
- $conf$ is a confidence measure (typically in the $[0, 1]$ range) holding for the correspondence between the entities e and e';

In our algorithm we only consider classes as entities and equivalence as relation.

Definition 2: Alignment. An alignment a of ontologies o and o' is a set of correspondences between entities of o and o'.

Definition 3: Matching Process. A matching process can be seen as a function f which takes two ontologies o and o', a set of parameters p and a set of oracles and resources r, and returns an alignment a between o and o' (adapted from [3]).

Matching techniques. An exhaustive survey on matching approaches can be found in [8]. Among the matching techniques, we just discuss those that fall under the *"Granularity / Input Interpretation"* classification, based on the granularity of the matcher and on the interpretation of the input information. In particular, we consider string-based methods and language-based ones.

1. **String-based methods.** These methods measure the similarity of two entities just looking at the strings (seen as mere sequences of characters) that label them. They include *Levenshtein Distance*, defined as the minimum number of insertions, deletions and substitutions of characters required to transform one string into another [17], and *SMOA Measure* which is a function of their commonalities (in terms of substrings) as well as of their differences [24].

2. **Language-based methods.** Language-based methods exploit natural language processing techniques to find the similarity between two strings seen as meaningful

pieces of text rather than sequences of characters. Some of them exploit external resources like WordNet, and exploit the semantic relations that it offers to compute the similarity. Language-based techniques include

- Tokenisation: names of entities are parsed into sequences of tokens. For example, *RedWine* becomes <*red, wine*> [12].
- Stemming: the strings underlying tokens are morphologically analysed in order to find all their possible basic forms. For example, *goes* becomes *go* [12].

These techniques are applied to names of entities before running string-based or lexicon-based techniques in order to improve their results [23].

Linguistic resources, such as common knowledge or domain specific thesauri are used in order to match words resulting from tokenisation and stemming of ontology entities based on linguistic relations between them (e.g., synonyms, hyponyms). For example, WordNet may be used to obtain meaning of words used in ontologies and relations between ontology entities can be computed in terms of bindings between WordNet senses. CtxMatch [4] represents the first instantiation of a language-based approach of this kind, discussed in [11].

Other matchers exploit the structural properties of thesauri, e.g., WordNet hierarchies. Hierarchy-based matchers measure the distance, for example, by counting the number of arcs traversed, between two concepts in a given hierarchy [13]. Several other distance measures for thesauri have been proposed in the literature [22,21].

The recent paper "A Survey of Exploiting WordNet in Ontology Matching" [18] discusses language- and WordNet-based matching techniques in detail. According to that paper and to many others that we have considered while analysing the state-of-the-art, no approaches based on the Adapted Lesk algorithm for ontology matching have been proposed so far. The algorithm we discuss in Section 3.1 is, to the best of our knowledge, an original contribution to the ontology matching research field.

2.4 Repairing Ontology Correspondences

As we will see in Section 4.2, where experimental results will be discussed, alignment methods may both produce wrong correspondences (the methods are not correct), and not produce right correspondences (they are not complete). For this reason, repairing wrong correspondences in an ontology alignment is a very pressing need. Very few attempts to solve this problem exist [15,19,6]. Among them, the approach that inspired our work and that we will briefly discuss here, is "Repairing Ontology Mappings" by C. Meilicke, H. Stuckenschmidt, and A. Tamilin [19].

The approach followed there is to interpret the problem of identifying wrong correspondences in an ontology alignment as a diagnosis task. Meilicke, Stuckenschmidt, and Tamilin formalise correspondences as "bridge rules" in distributed description logics and analyse the impact of each correspondence on the ontologies it connects. The basic assumption is that a correspondence that correctly states the semantic relations between ontologies should not cause inconsistencies in any of the ontologies. The encoding in distributed description logics allows the authors of [19] to detect these inconsistencies which are treated as symptoms caused by an incorrect correspondence. They

then compute sets of correspondences that jointly cause a symptom and repair each symptom by removing correspondences from these sets. The set of correspondences remaining after this process can be regarded as an approximation of the set of correct correspondences.

For example, consider two ontologies o and o' characterised respectively by axiom ax and axiom ax'

$$ax : \textit{author} \sqsubseteq \textit{person}$$
$$ax' : \textit{person} \sqsubseteq \neg\, \textit{authorization}$$

Suppose that the alignment to repair is the following:
$\{ < id_h, \textit{person}, \textit{person}, \equiv, \textit{1.00} >, < id_k, \textit{author}, \textit{authorization}, \sqsupseteq, \textit{0.46} > \}$

The alignment is inconsistent with respect to *authorization*. In fact, it is possible to derive by distributed reasoning, that *authorization* $\sqsubseteq$ *person* has to hold. At the same time, *authorization* and *person* are defined as disjoint concepts in ontology o'. In particular, this makes *authorization* unsatisfiable with respect to the global interpretation.

3 Matching Ontologies with Lesk and WordNet, and Repairing the Obtained Correspondences

In this section we describe our approach for generating ontology alignments by exploiting Lesk algorithm and WordNet and for repairing them in a supervised way, by detecting inconsistencies.

3.1 Exploiting "Lesk + WordNet" Algorithm to Match Ontologies

Given two ontologies o and o' to match, our language-based algorithm uses the adapted Lesk algorithm described in [1] (with some simplifications that have a very limited impact on the results it gives) to find the correct definition, or gloss, of any concept $c \in o$ and $c' \in o'$. The glosses found in this way are then compared to decide whether c corresponds to c' or not. Instead of taking the words before and after the target word to disambiguate in a given text, we take the neighboring concepts of c (those related by any kind of relation in the ontology) in the given ontology o, and we use these concepts as the context for disambiguating c.

The algorithm generates the $\mid o \mid \times \mid o' \mid$[4] couples of concepts $< c \in o, c' \in o' >$ and calls the function **is_correct_correspondence**(c, c') on each of them. We limit ourselves at considering the equivalence relation, thus we drop it from correspondences. Also, we drop correspondence identifiers for sake of readability. Thus, our correspondences are triples made by one concept in o, one concept in o', and a confidence in their equivalence. For the moment, the confidence is a binary value in $\{ 0, 1 \}$: 1 means that, according to the adapted Lesk algoritm, the concepts in the couple have exactly the same gloss; 0 means the gloss is not the same. Finer grained values may be given to the confidence, taking some similarity measure between glosses into account, as well as hyponymy, hypernymy, meronymy, and holonymy relations between the words they define.

The function **is_correct_correspondence** is defined by the following algorithm:

[4] $\mid o \mid$ is the number of concepts in o.

Input: an ontology o; an ontology o'; a concept $c \in o$; a concept $c' \in o'$; the WordNet Thesaurus
Output: an integer in { 0, 1 }
begin
1. if at least one between c and c', eventually after stemming, does not belong to WordNet, then return 0; otherwise
2. find the context of $c \in o$ by retrieving all the concepts $related(c) \in o$ that are related to c by any kind of relation apart from "disjoint";
3. stem c and any $related(c)$ found;
4. put the stemmed words (including c or its stem) that belong to WordNet into a list l, and discard those that do not belong to WordNet;
5. disambiguate the stem of c with respect to the context l using the "Lesk + WordNet" algorithm: be $disambiguated(c)$ the found gloss;
6. repeat steps from 2 to 5 for $c' \in o'$;
7. if $disambiguated(c)$ and $disambiguated(c')$ are the same gloss, then return 1; otherwise
8. return 0

end

The function **lesk_wordnet_matching** just calls **is_correct_correspondence** for each couple $< c, c' > \in o \times o'$; only those correspondences for which the function returns 1 are kept in the generated alignment. The function is defined by the following algorithm:

Input: an ontology o; an ontology o'; the WordNet Thesaurus
Output: an alignment a
begin
1. $a = \{\}$
2. for each couple $< c, c' > \in o \times o'$
   (a) if **is_correct_correspondence**(o, o', c, c', WordNet)
   (b) then $a = a \cup \{< c, c', 1 >\}$

end

3.2 Supervised Reasoning on Inheritance and Disjointness

The Lesk algorithm applied to the ontology matching process does not allow to detect semantic inconsistencies like the one described in Section 2.4. For this reason, we propose to involve the user in the process of repairing the alignment generated by the **lesk_wordnet_matching** function (or by any other matching method), in order to remove those correspondences that cause inconsistencies in the hierarchies of concepts induced by the inheritance relation. The inheritance relation is named *subClass* in OWL [25]; we translated into an *is_a* predicate, when we moved from the OWL representation of ontologies, to the Prolog one. Our repair algorithm identifies the couples of correspondences that raise an inconsistency, but cannot decide which one is to be removed. Hence, the user is asked to perform the choice of whether keeping or discarding a "suspicious" correspondence, after providing him/her with a careful explanation on the inconsistency raised.

Our idea is that the disjointness of concepts, that also propagates to sub-concepts, must be respected by the alignment.

Suppose that the alignment a to be repaired contains the correspondences $< c_i \in o, c'_i \in o', conf_i >$ and $< c_j = ancestor(c_i) \in o, c'_j \in o', conf_j >$. This means that $c_i \equiv c'_i$ and that $c_j \equiv c'_j$ to some extent given by $conf_i$ and $conf_j$.

Suppose that there is one ancestor of $c'_i \in o'$, let's name it $ancestor(c_i)'$, and that $ancestor(c_i)'$ is disjoint with c'_j. Since c'_i is a sub-concept of $ancestor(c_i)'$, c_i must be a sub-concept of $ancestor(c_i)'$ too (recall that c_i and c'_i are equivalent, according to the alignment). But c_i is also a sub-concept of $c_j = ancestor(c_i)$, and $ancestor(c_i)'$ is disjoint with $c_j = ancestor(c_i)$. In other words, c_i descends from two disjoint concepts $ancestor(c_i)'$, c_j, which is not consistent. This situation is depicted in Figure 1. The symmetric situation, where disjoint concepts are in o and not in o', must also be considered. The inconsistency might be due to either $< c_i \in o, c'_i \in o', conf_i >$ or $< c_j = ancestor(c_i) \in o, c'_j \in o', conf_j >$. Our algorithm cannot easily tell the wrong correspondence, thus, when a couple of inconsistent correspondences is detected, the user is prompted, and he/she can select the correspondence to remove, if any.

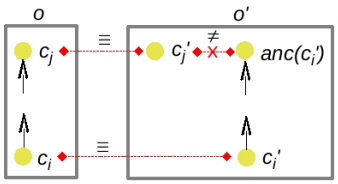

Fig. 1. Inconsistent correspondences.

The algorithm for reasoning about inconsistencies can be better described in Prolog than in an imperative language. The Prolog program accesses an atom table where the representations of the two ontologies o and o', and of the alignment a have been previously loaded.

Ontologies are defined by the *is_a(OntologyId, SubConc, SuperConc)* predicate defining inheritance relations, by the *rel(OntologyId, RelName, Concept, RelatedConcept)* predicate defining domain-dependent relations, and by the *disjointWith(OntologyId, Concept, DisjointConcept)* predicate that corresponds to the OWL *owl:disjointWith* element. An alignment consists of the correspondences that belong to it, represented by *map(ConceptInO1, ConceptInO2, Confidence)* atoms.

Ontologies o and o', and alignment a represented as above, are thus the input for the algorithm.

First of all, we define *disjoint* as a symmetric relation:

```
disjoint(O, C1, C2) :-
        disjointWith(O, C1, C2).
disjoint(O, C1, C2) :-
        disjointWith(O, C2, C1).
```

Then, we define the notion of ancestor as the transitive closure of the *is_a* predicate:

```
transitive_closure_is_a(_, C, C).
transitive_closure_is_a(Onto, C, SuperC):-
        is_a(Onto, C, Super),
        transitive_closure_is_a(Onto, Super, SuperC).
```

The **get_inconsistencies/5** predicate unifies the variable *Inconsistencies* with a term that contains the causes of the inconsistency raised by the mapping between *C1* in *Onto1* and *C2* in *Onto2* as shown in Table 1.

```
get_inconsistencies(Onto1, Onto2, C1, C2, Inconsistencies) :-
        findall(inconsistency(
                transitive_closure_is_a(Onto1, C1, SuperC1),
                transitive_closure_is_a(Onto2, C2, SuperC2),
                map(SuperC1, MapSuperC1, Con),
                disjoint(Onto2, MapSuperC1, SuperC2)),
            (map(C1, C2, Confidence),
            transitive_closure_is_a(Onto1, C1, SuperC1),
            transitive_closure_is_a(Onto2, C2, SuperC2),
            map(SuperC1, MapSuperC1, Con),
            disjoint(Onto2, MapSuperC1, SuperC2)),
            List1),
        findall(inconsistency(
                transitive_closure_is_a(Onto1, C1, SuperC1),
                transitive_closure_is_a(Onto2, C2, SuperC2),
                map(InverseMapSuperC2, SuperC2, Con),
                disjoint(Onto1, InverseMapSuperC2, SuperC1)),
            (map(C1, C2, Confidence),
            transitive_closure_is_a(Onto1, C1, SuperC1),
            transitive_closure_is_a(Onto2, C2, SuperC2),
            map(InverseMapSuperC2, SuperC2, Con),
            disjoint(Onto1, InverseMapSuperC2, SuperC1)),
            List2),
        append(List1, List2, Inconsistencies).
```

Table 1. The **get_inconsistencies/5** predicate.

The above Prolog predicate means that, given two ontology identifiers *Onto1* and *Onto2* and two concepts $C1 \in Onto1$ and $C2 \in Onto2$, the correspondence *map(C1, C2, Confidence)* raises an inconsistency if

- $\exists$ *SuperC1* $\in$ *Onto1*, and *SuperC1* is an ancestor of *C1*
- $\exists$ *SuperC2* $\in$ *Onto2*, and *SuperC2* is an ancestor of *C2*
- $\exists$ *MapSuperC1* $\in$ *Onto2* such that there is a correspondence *map(SuperC1, MapSuperC1, Con)* $\in a$
- *MapSuperC1* and *SuperC2* are disjoint in *Onto2*

or

- $\exists$ *SuperC1* $\in$ *Onto1*, and *SuperC1* is an ancestor of *C1*
- $\exists$ *SuperC2* $\in$ *Onto2*, and *SuperC2* is an ancestor of *C2*
- $\exists$ *InverseMapSuperC2* $\in$ *Onto1* such that there is a correspondence *map(InverseMapSuperC2, SuperC2, Con)* $\in a$
- *InverseMapSuperC2* and *SuperC1* are disjoint in *Onto1*

4 Prolog Implementation and Experimental Results

4.1 Implementation

Both the adapted Lesk algorithm and the supervised reasoning algorithm are implemented in Sicstus Prolog 3.11 and can be downloaded from `http://www.disi.unige.it/person/MascardiV/Software/LeskRepairMatching.html`.

Our adapted Lesk algorithm accesses the Prolog version of WordNet 3.0 available at `http://wordnet.princeton.edu/3.0/WNprolog-3.0.tar.gz`. It operates as described in Section 3.1. With respect to Banerjee and Pedersen's proposal [1], we have added a stemming stage implemented by the *recognize_words(ListOfWords, StemsOfWords)* predicate that translates nouns in plural form, and verbs in a form different from the infinite one, into the singular and infinite form, respectively. Also, as already explained, the context that we take into account depends on the ontology structure given by the *is_a* and *rel* predicates. The *get_concept_context(C, Onto, Context)* predicate is defined in the following way (*appendall* appends lists in a list, in the order they appear):

```
get_concept_context(C, O, Context):-
  findall(Conc, is_a(O, C, Conc), L1), findall(Conc, is_a(O, Conc, C), L2),
  findall(Conc, rel(O, _, C, Conc), L3), findall(Conc, rel(O, _, Conc, C), L4),
  appendall([L1, L2, L3, L4], Context).
```

In order to overcome a limitation of the 3.11 release of Sicstus Prolog, namely a very limited atom table that fills up just after loading two or three files of WordNet relations, we had to manually implement an "atom table cleaning" mechanism. Our attempts to load all WordNet files in the working memory and use them failed due to an "atom table full" error. This happened both using a Sicstus release for Windows and for Linux, and on different machines. For this reason we had to load each WordNet file at a time, extract all the atoms that we needed from it, according to the concepts we had to deal with in our ontologies, unload it, and explicitly call the atom garbage collector to make room for new atoms loaded from another WordNet file.

The goal to call in order to match ontologies *O1* and *O2*, whose Prolog representation is stored in file *OntoFile*, is *lesk_match(O1, O2, OntoFile, MappingFile, LogFile). MappingFile* is the file where the resulting alignment will be saved, whereas *LogFile* is the file where all the activities of the program will be traced.

As far as the supervised reasoning algorithm is concerned, its Prolog core has been already described in Section 3.2. Besides it, we have implemented a textual interface that allows the user to interact with the program.

The goal to call in order to repair the alignment stored in *MappingFile* is *repair(O1, O2, OntoFile, MappingFile, RepairedMappingFile, LogFile). O1* and *O2* are the ontology identifiers, and *OntoFile* is the file containing the Prolog representation of both of them. The alignment resulting after the repair process and a trace of the performed reasoning are written to two output files.

4.2 Results

In order to evaluate the feasibility of the adapted Lesk algorithm for matching ontologies, we have applied it to the ontologies represented in Figure 2. The ontologies have been created using Protégé, `http://protege.stanford.edu/`, have been exported into OWL, and have been manually translated into Prolog. The reference alignment between o and o', namely the correct and complete alignment computed by a domain expert, is

map(being, organism, 1). *map(nonliving, inanimate, 1).*
map('human being', human, 1).

We have run SMOA, Levenshtein, and WordNet-based matching algorithms described in Section 2.3, and implemented by the Alignment API, `http://alignapi.gforge.inria.fr/`, on o and o'.

Any ontology matching algorithm, be it string- or language-based, produces the correspondence $<$ *bank, bank, 1* $>$. However, *bank* in o has not the same meaning than *bank* in o': in fact, the first one refers to "*a building in which the business of banking transacted*" whereas the second one refers to a "*sloping land (especially the slope beside a body of water)*" (definitions are taken from WordNet 3.0). Since the Lesk-based disambiguation process of *bank* in the context [*building, banker*] provided by ontology o and *bank* in the context [*geological formation, river*] provided by ontology o' leads to two unrelated glosses, our algorithm classifies the correspondence as incorrect. The same happens with *bass* $\in o$ and *bass* $\in o'$, whose context in the respective ontologies allows Lesk algorithm to understand that the first one is the professional singer, whereas the second one is the fish.

All the ontology matching algorithms that we have run, rate the correspondence between *airplane* and *plane* with a very high value. However, the context of *plane* in o' is [*object, carpenter*]. This is enough for Lesk algorithm decide that it refers to "*a carpenter's hand tool with an adjustable blade for smoothing or shaping wood*", and not to "*an aircraft that has a fixed wing and is powered by propellers or jets*". In the end, the output of our adapted Lesk algorithm is

	
rel(o, performs_in, singer, theatre). rel(o, works_in, banker, bank). rel(o, fly_in, airplane, formation).	rel(o', has_habitat, bass, river). rel(o', performs_action, bird, sing). rel(o', lives_in, bass, 'North America'). rel(o', uses_tool, carpenter, plane). rel(o', is_near, bank, river).
disjointWith(o, being, nonliving).	disjointWith(o', animate, inanimate). disjointWith(o', dolphin, fish).

Fig. 2. Ontologies o and o'.

map(being, organism, 1). *map(nonliving, inanimate, 1).*
map('human being', human, 1). *map(formation, 'geological formation', 1).*

On the given input, the algorithm is complete, but not correct. The tricky correspondences that the SMOA, Levenshtein, and WordNet-based matching algorithms find, are correctly discarded by Lesk's one. Instead, both *formation* in the context [*nonliving, airplane*], and *geological formation* in the context [*inanimate, river, bank, sea, mountain*], are given the same sense "*(geology) the geological features of the earth*". The intended sense of *formation* was, instead, "*an arrangement of people or things acting as a unit*". In this case, the context was not informative enough for disambiguating the word properly.

Below we summarise the results obtained by our matching algorithm and by the SMOA, Levenshtein, and WordNet-based ones in terms of precision, recall and F-measure adapted for ontology alignment evaluation [7], obtained by exploiting the *GroupEval* method offered by the Align API.

	SMOA	**Levenstein**	**WordNet**	**Our algorithm**
Precision	0.10	0	0.11	0.75
Recall	0.33	0	0.33	1
F-measure	0.15	Not a Number	0.17	0.86

Precision is the number of correctly found correspondences with respect to the reference alignment (true positives), divided by the total number of found correspondences (true positives and false positives), and recall is the number of correctly found correspondences (true positives) divided by the total number of expected correspondences (true positives and false negatives). F-measure is the harmonic mean of precision and recall.

The table shows that on the given ontologies, our adapted Lesk algorithm gives the best results. In fact, SMOA and the WordNet-based algorithms only find the $<'$ *human being*$'$, *human*$>$ correct correspondence, whereas Levenshtein finds no correct correspondences. On the other hand, all the algorithms find many wrong correspondences.

To experiment our alignment repair algorithm, we have modified ontology o' by changing *sing* into *singer*. The adapted Lesk matching algorithm finds the correspondence $<$ *singer*, *singer*, 1 $>$. However, this would cause *singer* in o to descend from both *being* and *nonliving*, which are defined as disjoint concepts. The repair algorithm correctly detects this problem and prompts the user:

Checking the consistency of (singer in o, singer in o') with confidence 1
Concept singer in o descends from concept being; concept singer in o' descends from concept inanimate; concept nonliving in o is equivalent to concept inanimate in o' with confidence 1; concepts nonliving and being are disjoint in o; this raises an inconsistency!
What should I do with the (singer in o, singer in o') mapping?
What should I do with the (nonliving in o, inanimate in o') mapping?

5 Conclusions and Future Work

In this paper, we described our algorithms for ontology matching and supervised mapping repair and their implementation, and we discussed the preliminary results obtained with them.

These results are extremely encouraging but our work is open to many improvements: the experiment discussed in Section 4.2 has been carried out on two artificial ontologies, created ad hoc for containing concepts that can be found in WordNet and whose context might help Lesk algorithm to find their correct meaning. Our matching algorithm works well only on ontologies whose concepts belong to WordNet. This is a strong requirement, barely satisfied by most real ontologies.

Together with the students of the Artificial Intelligence course at DISI, however, we are currently working for overcoming this problem: the individual assignment for the academic year 2008/2009 requires to implement an algorithm in Prolog for matching ontology concepts to sets of WordNet words according to some well known distance

metric on strings to be chosen by the student. Students have to tokenise ontology concepts first. After tokenisation, the algorithm has to find correspondences between each concept token and one or more WordNet words. In the end, each ontology concept will be tagged with a set of WordNet words, based just on string metrics, and these tags will be used instead of the original ontology concepts by the Adapted Lesk algorithm, solving the main current limitation of our approach.

Another direction that we are exploring for partly overcoming this problem is to use the SUMO Upper Ontology [20] as a bridge between the ontology o whose concepts must be disambiguated, and WordNet. SUMO has been entirely mapped to WordNet 3.0 by hand: mapping o (or at least, those concepts of o that cannot be found in WordNet) to SUMO using some existing matching algorithm would give a mapping between o and WordNet for free, allowing us to run the adapted Lesk algorithm on the WordNet translation of o.

A significant improvement to our algorithm would be refining the confidence returned by our matching algorithm by setting it to some value < 1 if the concepts are not synonyms, but they are related by some semantic relation.

After these improvements will be implemented, we will start to test our algorithms on real ontologies. This experimentation will provide us with hints on how making our algorithms suitable for matching whatever ontologies, still keeping high precision and recall values. Optimisation issues will also be taken under consideration: we expect that performances will dramatically drop on large ontologies, and the tests' outcomes will help us in identifying some viable approach for balancing efficiency, precision and recall.

References

1. S. Banerjee and T. Pedersen. An adapted lesk algorithm for word sense disambiguation using WordNet. In A. F. Gelbukh, editor, *3rd International Conference Computational Linguistics and Intelligent Text Processing, CICLing 2002, Proceedings*, volume 2276 of *Lecture Notes in Computer Science*, pages 136–145. Springer, 2002.
2. Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.
3. P. Bouquet, J. Euzenat, E. Franconi, L. Serafini, G. Stamou, and S. Tessaris. Specification of a common framework for characterizing alignment. Technical Report D2.2.1, NoE Knowledge Web project, 2004.
4. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach and an application. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *2nd International Semantic Web Conference, ISWC 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2003.
5. S. B. Brunnermeier and S. A. Martin. Interoperability cost analysis of the u.s. automotive supply chain. Technical Report NIST 99-1, U.S. Department of Commerce, National Institute of Standards and Technology, 1999.
6. H. Stuckenschmidt C. Meilicke and A. Tamilin. Reasoning support for mapping revision. *Journal of Logic and Computation*, 2008.
7. H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In A. B. Chaudhri, M. Jeckle, E. Rahm, and R. Unland, editors, *Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops, 2002, Revised Papers*, volume 2593 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2002.

8. J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, 2007.
9. C. Fellbaum, editor. *WordNet – An Electronic Lexical Database*. The MIT Press, 1998.
10. M. P. Gallaher, A. C. O'Connor, J. L. Dettbarn Jr., and L. T. Gilday. Cost analysis of inadequate interoperability in the u.s. capital facilities industry. Technical Report NIST GCR 04-867, U.S. Department of Commerce, National Institute of Standards and Technology, 2004.
11. F. Giunchiglia and P. Shvaiko. Semantic matching. *Knowl. Eng. Rev.*, 18(3):265–280, 2003.
12. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic schema matching. In *International Conference on Cooperative Information Systems, CoopIS, Proceedings*, pages 347–365, 2005.
13. F. Giunchiglia and M. Yatskevich. Element level semantic matching. In *Meaning Coordination and Negotiation Workshop at the International Semantic Web Conference ISWC 2004, Proceedings*, 2004.
14. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.
15. S. H. Haeri, B. B. Hariri, and H. Abolhassani. Coincidence-based refinement of ontology matching. In *Joint 3rd International Conference on Soft Computing and Intelligent Systems and 7th International Symposium on advanced Intelligent Systems, SCIS+ISIS 2006, Proceedings*, 2006.
16. M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *5th Annual International Conference on Systems Documentation, SIGDOC '86, Proceedings*, pages 24–26. ACM, 1986.
17. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady akademii nauk SSSR*, 163(4):845–848, 1965. In Russian. English Translation in Soviet Physics Doklady 10(8), 707-710, 1966.
18. F. Lin and K. Sandkuhl. A survey of exploiting wordnet in ontology matching. In M. Bramer, editor, *Artificial Intelligence in Theory and Practice II, IFIP 20th World Computer Congress*, volume 276 of *IFIP*, pages 341–350. Springer, 2008.
19. C. Meilicke, H. Stuckenschmidt, and A. Tamilin. Repairing ontology mappings. In *22nd AAAI Conference on Artificial Intelligence, AAAI 2007, Proceedings*, pages 1408–1413. AAAI Press, 2007.
20. I. Niles and A. Pease. Towards a standard upper ontology. In C. Welty and B. Smith, editors, *FOIS 2001, 2nd International Conference on Formal Ontology in Information Systems, Proceedings*, pages 2–9. ACM Press, 2001.
21. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):17–30, 1989.
22. P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res. (JAIR)*, 11:95–130, 1999.
23. P. Shvaiko. Iterative schema-based semantic matching. Technical Report DIT-06-102, DIT - University of Trento, 2005.
24. G. Stoilos, G. B. Stamou, and S. D. Kollias. A string metric for ontology alignment. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *4th International Semantic Web Conference, ISWC 2005, Proceedings*, volume 3729 of *Lecture Notes in Computer Science*, pages 624–637. Springer, 2005.
25. W3C. OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004, 2004.
26. J. Walker, E. Pan, D. Johnston, J. Adler-Milstein, D. W. Bates, and B. Middleton. The value of healthcare information exchange and interoperability. *Health Affairs*, 2005. Web Exclusive, 19 January 2005.