Learning Disjointness for Debugging Mappings between Lightweight Ontologies

Christian Meilicke[1], Johanna Völker[2], Heiner Stuckenschmidt[1]

[1] Computer Science Institute
Universität Mannheim, Germany
{christian,heiner}@informatik.uni-mannheim.de
[2] Institute AIFB
Universität Karlsruhe (TH), Germany
voelker@aifb.uni-karlsruhe.de

Abstract. Dealing with heterogeneous ontologies by means of semantic mappings has become an important area of research and a number of systems for discovering mappings between ontologies have been developed. Most of these systems rely on general heuristics for finding mappings, hence are bound to fail in many situations. Consequently, automatically generated mappings often contain logical inconsistencies that hinder a sensible use of these mappings. In previous work, we presented an approach for debugging mappings between expressive ontologies that eliminates inconsistencies by means of diagnostic reasoning. A shortcoming of this method was its need for expressive class definitions. More specifically, the applicability of this method critically relies on the existence of a high-quality disjointness axiomatization. This paper deals with the application of the debugging approach to mappings between lightweight ontologies that do not contain any or very few disjointness axioms, as it is the case for most of today's practical ontologies. After discussing different approaches to deal with the absence of disjointness axioms we propose the application of supervised machine learning for detecting disjointness in a fully automatic manner. We present a detailed evaluation of our approach to learning disjointness and its impact on mapping debugging. The results show that debugging automatically created mappings with the help of learned disjointness axioms significantly improves the overall quality of these mappings.

1 Motivation

A common way of integrating different ontologies describing the same or largely overlapping domains is to use formal representations of semantic correspondences between their classes and relations - also referred to as "ontology mappings". There are two major research challenges connected to ontology mappings. The first one is the development of a logical model for representing and reasoning about ontology mappings. The second one is the identification of semantic correspondences between elements in different ontologies as a basis for generating mappings between ontologies. Manual approaches for identifying semantic correspondences are often not feasible since real world ontologies, for example in the medical domain, often contain several thousand classes. As a response to this problem, a number of automatic and semi-automatic

tools for generating hypotheses about semantic correspondences have been developed (see [7] for an overview). The results of these tools, however, often contain a significant amount of errors caused by the use of general heuristics that are bound to fail in certain situations. Due to this fact, a revision of the mappings created by matching systems is often inevitable to guarantee the quality of the integration.

In this paper, we present a method for automatically revising mappings that can also be applied to mappings between lightweight ontologies. The method is an extension of previous work on debugging ontology mappings that relied on the existence of complex class descriptions and therefore was not applicable to a wide range of ontologies on the semantic web. The contribution of this paper compared to previous work is the following:

- We describe a variant of the original debugging method reported in [17] that does not require a specialized reasoning system but can be implemented on top of any description logic reasoner.
- We present a preprocessing step that follows the approach outlined in [27] in which missing disjointness axioms are added to lightweight ontologies as a basis for detecting malicious correspondences.
- We re-implemented the prototype described in [27] and developed an improved set of features, that enables us to build a more reliable classification model for learning disjointness axioms.
- We implemented a validation technique for debugging learned disjointness based on logical reasoning and performed a detailed evaluation of the automatically acquired axioms against a manually created Gold Standard.
- We evaluated the mapping debugging method using learned disjointness axioms and show that this approach significantly improves the result of existing matching systems.

In the following, we briefly introduce the approach for debugging ontologies and discuss different options for dealing with the lack of disjointness axioms in lightweight ontologies. In Section 3 we present an approach for learning disjointness axioms from examples that has been chosen as a basis for the preprocessing step. Section 4 provides details on the experiments we carried out and the results achieved. We conclude with a discussion of the approach and topics for future work.

2 Debugging Mappings

In previous work, we have proposed the use of logical reasoning about ontology mappings for improving the quality of automatically generated mappings both in terms of using reasoning inside matching systems to evaluate mapping hypotheses [16] and as a basis for debugging mappings generated by a matching systems a posteriori [17]. In the following, we provide relevant definitions of the ontology matching task and briefly present an approach for debugging mappings as well as different options for extending this approach to lightweight ontologies.

2.1 Ontology Mapping Basics

Given two ontologies, $\mathcal{O}_1$ and $\mathcal{O}_2$, describing the same or largely overlapping domains of interest. According to Euzenat and Shvaiko [7], correspondences between elements of these ontologies can be defined as follows.

Definition 1 (Correspondence). *Given ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, let Q be a function that defines sets of matchable elements $Q(\mathcal{O}_1)$ and $Q(\mathcal{O}_2)$ of ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. Then a* correspondence *is a 4-tuple $\langle e, e', r, n\rangle$ such that $e \in Q(\mathcal{O}_1)$ and $e' \in Q(\mathcal{O}_2)$, r is a semantic relation, and n is a confidence value from a suitable structure $\langle D, \leqslant\rangle$.*

The generic form of definition 1 allows to capture a wide class of correspondences by varying what is admissible as matchable element, semantic relation, and confidence value. In this work, we impose the following additional restrictions on correspondences: We only consider correspondences between classes. We also restrict r to be the equivalence relation. In the following we use the prefix notation $1\colon C$ to refer to class C from ontology $\mathcal{O}_1$. Given ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, the equivalence correspondence $\langle 1\colon A, 2\colon B, =, 1.0\rangle$ is correct if everything that we account to be an instance of $1\colon A$ also has to be accounted to be an instance of $2\colon B$ and vice versa. Note that most of today's matching systems only generate equivalence correspondences. Finally, we assume that the confidence value is represented numerically on $D = [0.0, 1.0]$. The confidence value n can be seen as a measure of trust in the fact that the correspondence holds. The higher the confidence degree with regard to the ordering $\leqslant$, the more likely relation r holds between matchable elements e and e'. In the following we define a mapping between two ontologies as a set of correspondences between these ontologies.

Definition 2 (Mapping). *Given ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$, let Q be a function that defines sets of matchable elements $Q(\mathcal{O}_1)$ and $Q(\mathcal{O}_2)$ of ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively. A set of correspondences $\mathcal{M}$ is a mapping between $\mathcal{O}_1$ and $\mathcal{O}_2$ iff for all $\langle e, e', r, n\rangle \in \mathcal{M}$ we have $e \in Q(\mathcal{O}_1)$ and $e' \in Q(\mathcal{O}_2)$.*

This notion of a mapping between ontologies has been used as the basis of a standardized API[3] for handling ontology mappings and has become the de facto standard for representing automatically generated mappings through its use in the Ontology Alignment Evaluation Initiative [6, 8].

2.2 A Debugging Method

To apply logical reasoning to ontology mappings, we first have to translate them into a logical representation. Several formalisms have been proposed for this purpose (compare e.g. [23]). In this paper, we use standard description logics as a basis for encoding mappings in order to make use of existing, highly optimized reasoning systems. In particular, based on the specification of a mapping, we generate a merged ontology containing the axioms from the two ontologies to be mapped and equivalence axioms between classes in these ontologies that represent the mapping.

[3] `http://alignapi.gforge.inria.fr`

Definition 3 (Merged ontology). *Given a mapping $\mathcal{M}$ between ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$. The merged ontology $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$ of $\mathcal{O}_1$ and $\mathcal{O}_2$ connected via $\mathcal{M}$ is defined as $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2 = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \{t(c) \mid c \in \mathcal{M}\}$ with t being defined by $t(\langle 1{:}\,C, 2{:}\,D, = , c\rangle) = 1{:}\,C \equiv 2{:}\,D$ converting equivalence correspondences into equivalence axioms of $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$.*

The merged ontology provides a basis for reasoning about the impact of a mapping on classes in the mapped ontologies. Of particular interest is the identification of incoherences that are introduced by the mapping. More specifically, we are interested in classes in the mapped ontologies that were satisfiable before the merge operation becoming unsatisfiable in $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$. Such an incoherence must have been introduced by the mapping. We call a mapping that introduces incoherence an inconsistent mapping.

Definition 4 (Inconsistency of a mapping). *Given a mapping $\mathcal{M}$ between ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$. $\mathcal{M}$ is consistent iff there exists no class $i{:}\,C$ with $i \in \{1, 2\}$ such that $\mathcal{O}_i \not\models i{:}\,C \sqsubseteq \bot$ and $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2 \models i{:}\,C \sqsubseteq \bot$. Otherwise $\mathcal{M}$ is inconsistent.*

Obviously, some of the correspondences of an inconsistent mapping $\mathcal{M}$ have to be incorrect, because we would not accept a mapping that imposes restrictions on $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$ making some of the classes in $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$ unsatisfiable. In order to identify those elements of a mapping causing a class to become unsatisfiable, we can use techniques known from the area of classical diagnosis [19] that have already successfully been applied in the context of diagnosing inconsistencies in description logic ontologies [21]. A central notion with this respect is the one of a minimal conflict set. In our setting, a minimal conflict set is an inconsistent subset of $\mathcal{M}$ which contains no real subset that is also inconsistent.

Definition 5 (Minimal conflict set). *Given a mapping $\mathcal{M}$ between ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$. A mapping $\mathcal{C} \subseteq \mathcal{M}$ is a minimal conflict set iff $\mathcal{C}$ is inconsistent and each $\mathcal{C}' \subset \mathcal{C}$ is consistent.*

Several approaches have been proposed to explain and resolve incoherences in ontologies [13, 22]. At first sight, these approaches can be applied in a straight forward way to the problem of mapping debugging by resolving the incoherence of $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$. Our definition of a minimal conflict set, in particular, is closely related to the notion of a minimal incoherence-preserving sub-TBox, a central notion in the context of ontology debugging. Nevertheless, there are two major differences between ontology and mapping debugging. On the one hand, we will never remove any of the axioms in $\mathcal{O}_1$ or $\mathcal{O}_2$. We are not even interested in which of these axioms is taking part in causing the incoherence. On the other hand, correspondences are labeled with confidence values that can be seen as a measure of trust in the correctness of the correspondence. Contrary to this, in standard approaches for resolving incoherences each axiom is implicitly equally weighted, since no further information about its correctness is given.

The approach presented in algorithm 1 takes into account the peculiarities of mapping debugging. It can be seen as naive strategy that transforms an inconsistent into a consistent mapping by a sequence of rational decisions. First, the correspondences of $\mathcal{M}$ are sorted descending due to their confidence value. An empty mapping $\mathcal{M}'$ is

Algorithm 1

RESOLVECONFLICTS($\mathcal{M}$)

```
1: M' ← ∅
2: SORTDESCENDING(M)
3: while M ≠ ∅ do
4:    c ← REMOVEFIRSTELEMENT(M)
5:    M' ← M' ∪ {c}
6:    if not ISCONSISTENT(M', O1, O2) then
7:       M' ← M' \ {c}
8:    end if
9: end while
10: return M'
```

instantiated that will, finally, contain a consistent subset of $\mathcal{M}$. In each iteration correspondence c with the highest confidence value is removed from $\mathcal{M}$ and added to $\mathcal{M}'$. Whenever $\mathcal{M}'$ becomes inconsistent, c has to be rejected and is removed from $\mathcal{M}'$. This decision is motivated by the fact, that there exists a minimal conflict set $\mathcal{C} \subseteq \mathcal{M}'$ such that $c \in \mathcal{C}$ and there exists no $c' \in \mathcal{C}$ with lower confidence than c. The algorithm terminates after $|\mathcal{M}|$ iterations. The resulting mapping $\mathcal{M}'$ is consistent and each superset of $\mathcal{M}'$ is an inconsistent mapping. By calling ISCONSISTENT($\mathcal{M}', \mathcal{O}_1, \mathcal{O}_2$) the merged ontology $\mathcal{O}_1 \cup_{\mathcal{M}'} \mathcal{O}_2$ is created and the coherence of this ontology has to be checked. Thus, the algorithm can be implemented on top of any reasoner capable of reasoning in the merged ontology $\mathcal{O}_1 \cup_{\mathcal{M}} \mathcal{O}_2$.[4]

2.3 Problems with Lightweight Ontologies

For many matching problems mapping inconsistencies will only occur if the ontologies to be matched contain disjoint classes. On the one hand, the disjointness of two classes can be explicitly stated by a disjointness axiom. Evaluations of existing ontologies have shown that, even in expressive ontologies, these axioms are often missing [28]. On the other hand, disjointness can be derived from expressive definitions, that will rarely occur in lightweight ontologies. This has already been identified as a problem in our previous work where missing expressivity often leads to a situation where only a small subset of all incorrect correspondences are detected [17]. Thus, the whole approach is useless in the context of lightweight ontologies, for instance ontologies specified in RDF Schema. This is a major drawback as according to Ding and Finin in mid 2006 [4] about half a million ontologies where defined using RDF schema while disjointness axioms where only found in less than 2,500 files on the web. As this questions the usefulness of our approach on a large scale, we have to find ways to deal with the lack of disjointness axioms in most real world ontologies. There are several possible solutions to this problem:

Manual Preprocessing A straightforward approach is to require a manual preprocessing step in which disjointness axioms are added to the ontologies to be mapped. This approach is not really feasible for large ontologies.

[4] We implemented the algorithm on top of the Pellet reasoner [24].

Alternative Consistency Criteria In previous work, we have also experimented with alternatives to the notion of mapping inconsistency as a basis for debugging mappings. In particular, we have used the notion of stability which requires that a mapping should not induce any changes in the hierarchies of the mapped ontologies. It turned out, however, that this is too restrictive in many practical cases as real ontologies often disagree on the taxonomic relation between classes.

Disjointness Assumptions In [20] Schlobach proposes a method called semantic clarification as a basis for debugging lightweight ontologies. The method is based on the assumption that sibling classes are always disjoint unless this introduces a conflict. We could use this assumption in our setting as well and add disjointness axioms for all sibling classes of a class thus enriching the ontologies to be mapped. Taking a closer look at this approach, however, reveals that the effect is almost the same as for the stability criterium and thus also too restrictive in many cases.

As all of the approaches mentioned above have problems, we follow a different approach in this paper. The idea is to learn which classes in a lightweight ontology should actually be regarded as being disjoint. We analyze expressive ontologies and derive features indicating the disjointness of classes. Based on these features, we decide which classes in a lightweight ontology are most likely disjoint and add the corresponding axiom to the merged ontology as a basis for debugging. Details of this approach are presented in the following section.

3 Learning Disjointness Axioms

Our approach to the automatic acquisition of disjointness axioms relies on a machine learning classifier that determines disjointness of any two classes. The classifier is trained based on a "Gold Standard" of manually created disjointness axioms (cf. Section 4.1), i.e. pairs of classes each of which is associated with a label – "disjoint" or "not disjoint" – and a vector of feature values. As in our earlier experiments [27], we used a variety of lexical and logical features, which we believe to provide a solid basis for learning disjointness. These features are used to build an overall classification model on whose basis the classifier can predict disjointness for previously unseen pairs of classes. We implemented all features and auxiliary methods for training and classification within the open-source tool LeDA[5] (*Learning Disjointness Axioms*), a complete redesign and re-implementation of our original prototype and publicly available under the LGPL license.

In the following, we give a brief overview of the features we used for the experiments that we report on in Section 4. The current feature set differs from the original one [27] in that it focuses more on lexical and ontology-based similarity, which turned out to work very well in previous experiments. At the same time, we omitted several "weak" features including, e.g., OntoClean meta-properties and enumerations. For details with respect to our selection of features and evaluation results, the interested reader is referred to [26].

[5] `http://ontoware.org/projects/leda/`

Taxonomic Overlap. In description logics, two classes are disjoint *iff* their "taxonomic overlap", i.e. the set of common individuals *must* be empty. Because of the open world assumption in OWL, the individuals of a class do not necessarily have to *exist* in the ontology. Hence, the taxonomic overlap of two classes is considered not empty as long as there *could* be common individuals within the domain that is modeled by the ontology. Following these considerations, we developed several methods to compute the actual or possible overlap of two classes. Both of the following formulas are based on the Jaccard similarity coefficient [12].

$$\mathrm{f}_{overlap_i}(c_1, c_2) = \frac{|\{i \in I | c_1(i) \wedge c_2(i)\}|}{|\{i \in I | c_1(i) \vee c_2(i)\}|} \quad \mathrm{f}_{overlap_c}(c_1, c_2) = \frac{|\{c \in C | c \sqsubseteq c_1 \sqcap c_2\}|}{|\{c \in C | c \sqsubseteq c_1 \sqcup c_2\}|}$$

These two features are complemented by f_{sub}, that represents a particular case of taxonomic overlap, while at the same time capturing negative information such as class complements or already existing disjointness contained in the ontology. The value of f_{sub} for any pair of classes c_1 and c_2 is 1 for $c_1 \sqsubseteq c_2 \vee c_2 \sqsubseteq c_1$, 0 for $c_1 \sqsubseteq \neg c_2$ and $undefined$ otherwise.

Semantic Distance. The semantic distance between two classes c_1 and c_2 is the minimum length of a path consisting of subsumption relationships between atomic classes that connects c_1 and c_2 (as defined in [27]).

Object Properties. This feature encodes the semantic relatedness of two classes, c_1 and c_2, based on the number of object properties they share. More precisely, we divided the number of properties p with $p(c_1, c_2)$ or $p(c_2, c_1)$ by the number of all properties whose domain subsumes c_1 whereas their range subsumes c_2 or vice-versa. This measure can be seen as a variant of the Jaccard similarity coefficient with object properties considered as undirected edges.

Label Similarity. The semantic similarity of two classes is in many cases reflected by their labels – especially, in case their labels share a common prefix or postfix. This is because the right-most constituent of an English noun phrase can be assumed to be the lexical head, that determines the syntactic category and usually indicates the semantic type of the noun phrase. A common prefix, on the other hand, often represents a nominal or attribute adjunct which describes some semantic characteristics of the noun phrase referent. In order to compute the lexical similarity of the two class labels, we therefore used three different similarity measures: *Levenshtein*, *QGrams* and *Jaro-Winkler*.

WordNet Similarity. In order to compute the lexical similarity of two classes (their labels, to be precise), we applied two variants of a WordNet-based similarity measure by Patwardhan and Pedersen [18].[6] This similarity measure computes the cosine similarity between vector-based representations of the glosses, that are associated with the two synsets.[7] We omitted any sort of word sense disambiguation at this point, assuming that every class label refers to the most frequently used synset it is contained in.

[6] `http://www.d.umn.edu/~tpederse/similarity.html`

[7] In WordNet, a synset is a set of (almost) synonymous words, roughly corresponding to a class or concept in an ontology. A gloss is a textual description of a synset's meaning, that most often also contains usage examples.

Features based on Learned Ontology. As an additional source of background knowledge about the classes in our input ontology we used an automatically acquired corpus of Wikipedia articles. By querying Wikipedia for each class label[8] we obtained an initial set of articles some of which were disambiguation pages. We followed all content links and applied a simple word sense disambiguation method in order to obtain the most relevant article for each class. For each class label we considered the article to be most relevant, which had, relative to its length, the highest "terminological overlap" with all of the labels used in the ontology. The resulting corpus of Wikipedia articles was finally fed into Text2Onto [3] to generate an additional background ontology for each of the original ontologies in our data set (cf. Section 4.1), consisting of classes, individuals, subsumption and class membership axioms.

Based on this newly acquired background knowledge, we defined four features: *subsumption*, *taxonomic overlap of subclasses* and *individuals* – all of these are defined as their counterparts described above – as well as document-based *lexical context similiarity*, which we computed by comparing the Wikipedia article associated with the two classes. This type of similarity is in line with Harris' distributional hypothesis [10] claiming that two words are semantically similar to the extent to which they share syntactic contexts.

4 Experimental Evaluation

In the following we describe the experimental evaluation of disjointness learning and its effects on mapping debugging. Both groups of experiments have been conducted on the OntoFarm data set, described in Section 4.1. In Section 4.2 we focus on the experimental setting for disjointness learning and present the most important results. Finally, in Section 4.3 we discuss how to measure the effects of using learned disjointness in the context of mapping debugging and describe as well as discuss the debugging experiments and their results.

4.1 Data Sets

We evaluated our approach using automatically created mappings between ontologies of the OntoFarm data set. It consists of a set of ontologies in the domain of conference organization that have been collected by researchers of the Knowledge Engineering Group at the University of Economics Prague [25].[9] The ontologies cover the structure of a conference, involved actors, as well as issues connected with the submission and review process.

Meanwhile the OntoFarm data set consists of 13 ontologies, six of which were used in our experiments. We omitted the other ontologies, since not all of the matching systems in our experiments were able to provide the corresponding mappings and because several ontologies were added to the data set after we had already setup the experiments.

[8] Labels that were written as one word, though consisting of nominal compounds or other types of complex noun phrases.

[9] The ontologies are available at `http://nb.vse.cz/~svabo/oaei2006/`.

Ontology	Expressiveness	Classes	Properties	Disjointness
CMT	$\mathcal{ALCIF}(D)$	30	59	27
CRS	DL-lite	14	17	12
CONFTOOL	$\mathcal{SIF}(D)$	39	36	43
EKAW	$\mathcal{SHIN}(D)$	77	33	83
PCS	$\mathcal{ELUIF}(D)$	24	38	0
SIGKDD	$\mathcal{ELI}(D)$	51	28	0

Table 1. Ontologies chosen from the OntoFarm collection.

Table 1 summarizes core characteristics of the ontologies we used, including their size and expressive power in terms of the underlying logic as well as the number of already existent disjointness axioms.[10]

Gold Standards. In order to obtain a reference set of disjointness axioms for training and evaluating LeDA as well as to get an upper bound for the evaluation of mapping debugging, we manually added a minimal and complete number of disjointness axioms to the ontologies described above.[11] For these sets of explicit disjointness axioms, we computed the transitive closure by "materializing" all implicit disjointness relationships (positive examples). All pairs of classes whose disjointness could not be inferred from the initial, minimal set of axioms were considered not disjoint, thus serving as negative examples in the Gold Standard. This way we obtained a logically "cleaner" and much bigger data set than in of our earlier experiments with learning disjointness.

To evaluate the results of mapping debugging we had to manually construct reference mappings consisting of equivalence correspondences for all pairs of ontologies. In order to create reference mappings of high quality, three people familiar with the domain of conferences individually constructed mappings for each pair of ontologies. In case of a disagreement the correctness of a correspondence was decided by a majority vote. It turned out that there was only little disagreement with respect to the correctness of correspondences.

4.2 Disjointness Learning

In this section, we present the evaluation of our approach to the automatic acquisition of disjointness axioms in which we built a classification model from the features described in Section 3. After introducing our evaluation setting in Section 4.2 we give an overview of the results that we obtained by evaluating our classification model against the Gold Standard of manually created axioms (cf. Section 4.2).

[10] For more information about the given logic classes we refer the reader to [1].

[11] A set of disjointness axioms D is minimal with respect to ontology $\mathcal{O}$ iff for all $d \in D$ we have $\mathcal{O} \cup D \setminus \{d\} \not\models d$.

Evaluation Setting

Training and Test Data. Unlike in our earlier experiments where a single ontology had to serve as a basis for both training and testing, the conference ontologies data set allows us to use $6 \times 5 = 30$ different combinations of ontologies for the evaluation of learning disjointness: for each of the 6 ontologies, we thus performed 5 experiments using each of the remaining ontologies as training data, and finally averaged over the individual results.

Note that we removed all previously existing disjointness axioms from the ontologies prior to training and classification, because we wanted to get comparable results for all ontologies, independently of their respective numbers of existing disjointness axioms (cf. Table 1).

When testing on any of the ontologies, we always classified (and evaluated against) all possible pairs of classes – not just those explicitly marked as disjoint by the user. This is because we hoped that the resulting redundancy would help to rule out incorrectly classified pairs of classes in a post-processing (debugging) step. As a classifier for all experiments, we used Weka's implementation of NaiveBayes with default parameters[12], which turned out to perform slightly better in our initial tests than Decision Trees and SVMs – especially on the smaller data sets.

Baseline and Evaluation Measures. We generated macro-average values for precision, recall and F-measure[13] by averaging over the respective results on the sets of positive and negative examples. As a reasonable baseline for our evaluation, we computed a majority baseline for accuracy (Acc_{base}), that is defined as the number of examples in the majority class (e.g. "not disjoint") divided by the overall number of examples. The majority baseline represents the performance level that would be achieved by a naïve classifier that labels all entities in the test set with the majority class, i.e. "disjoint" for all ontologies in our data set. This simple, yet efficient strategy is hard to beat, especially for data sets that are relatively unbalanced and biased towards one of the target classes.

Debugging of Learned Disjointness We implemented a disjointness debugging technique based on formal reasoning as final step of our approach. This technique is similar to the mapping debugging approach presented in algorithm 1. Given a set of learned disjointness axioms D making statements about the classes in ontology $\mathcal{O}$, we apply the following procedure: We sort the set of learned disjointness axioms D according to their associated confidence values (i.e. probabilities generated by our NaiveBayes classifier) and start with the highest ranked axioms. Each disjointness axiom $d \in D$ is temporarily added to $\mathcal{O}$. Afterwards, the class hierarchy of ontology $\mathcal{O}$ is recomputed. If $\mathcal{O}$ is still coherent d can be accepted and becomes a permanent part of $\mathcal{O}$. To reduce the amount of reasoning, we additionally check whether $\mathcal{O} \models d'$ for all $d' \in D$ that have not been accepted or rejected yet and accept d' in case of entailment. If $\mathcal{O}$ is no longer coherent d is rejected and removed from $\mathcal{O}$. This procedure is an efficient approach that both ensures coherence and increases the quality of learning disjointness.

[12] `http://www.cs.waikato.ac.nz/ml/weka/`

[13] In the following we use the term F-measure to refer to the F_1-measure, where recall and precision are evenly weighted.

Results The evaluation shows that our approach can reliable determine disjointness for any given pair of classes. As detailed by Table 2, the comparison of our results with the Gold Standard set of disjointness axioms yields a macro-average F-measure of up to 80.0%. The accuracy (i.e. the fraction of correctly classified pairs) is even higher ranging from 76.5% to 91.3% which is always above the majority baseline Acc_{base}.

Test on	P	R	F	Acc	Acc_{base}	Acc_{debug}
CMT	0.838	0.785	**0.800**	0.841	0.685	0.842
CONFTOOL	0.792	0.765	0.770	0.855	0.803	0.857
CRS	0.809	**0.801**	0.793	0.867	0.824	N/A
EKAW	**0.882**	0.760	0.797	**0.913**	**0.851**	**0.922**
PCS	0.828	0.749	0.767	0.812	0.663	N/A
SIGKDD	0.778	0.641	0.679	0.765	0.704	0.767

Table 2. Results of learning disjointness averaged over all training ontologies (NaiveBayes classifier; macro-avg. precision, recall and f-measure)

Debugging the sets of automatically acquired disjointness axioms as described in Section 4.2 further improves the average accuracy, e.g., from 91.3% to 92.2% Acc_{debug} for EKAW. Table 3 shows a more detailed evaluation of this post-processing step for individual pairs of training and test ontologies. The first column lists the ontologies that were enriched with automatically acquired disjointness axioms, whereas the second column indicates the respective training ontologies, i.e. the ontologies that were used to setup the classifier. For each training ontology, we obtained an incoherent variant of the original ontology and an accuracy value Acc that was computed by comparison with the respective Gold Standard. The last column of Table 3 lists the relative numbers of disjointness axioms that had to be removed in order to debug the ontology. For example, 2.35% of the learned disjointness axioms had to be removed from EKAW after training had been performed on the CRS ontology. In all cases, debugging further improved the overall quality of the learning results, which led to an increased accuracy (Acc_{debug}).

Test on	Training on	Acc	Acc_{base}	Acc_{debug}	Removed
CMT	CRS	0.830	0.685	0.839	1.14%
CONFTOOL	CRS	0.869	0.803	0.881	1.41%
EKAW	CMT	0.925	0.851	0.932	0.76%
	CRS	0.892		0.913	2.35%
	PCS	0.919		0.940	2.25%
	SIGKDD	0.904		0.911	0.89%
SIGKDD	CMT	0.779	0.704	0.780	0.09%
	CONFTOOL	0.776		0.776	0.09%
	CRS	0.773		0.781	0.90%
	PCS	0.744		0.744	0.11%

Table 3. Individual results of learning disjointness after debugging (NaiveBayes classifier)

The complete data sets of our learning disjointness experiments – including the Gold Standard as well as all the training data and classification results – are available online and can be downloaded from the LeDA homepage.[14]

4.3 Mapping Debugging

In the following we describe a group of experiments where we used the results of the previously presented evaluation of LeDA as basis in the mapping debugging process. Before reporting the most important results we first focus on the experimental setting.

Experimental Setting In order to adequately measure the effects of learned disjointness statements on mapping debugging, we compare three types of mappings, namely (1) the mappings automatically generated by a matching system, (2) the debugged mappings where the debugging process is based on the reference disjointness ontologies and finally (3) on the ontologies with learned disjointness statements. Comparisons between input mappings and mappings debugged with reference disjointness measure the quality of the debugging process in a best case scenario with respect to the completeness and correctness of disjointness statements. This comparison will thus be an upper bound for debugging based on learned disjointness. In the following we focus on the question whether the effects of debugging based on learned disjointness are still good enough to increase the quality of a mapping, in particular, we will compare the results to the upper bound based on the reference disjointness.

We have chosen four matching systems participating in the consensus track of the OAEI 2006 and 2007 campaign, in particular those systems providing mappings for all pairs of ontologies listed in Table 1 as well as generating meaningful confidence values. These system are Falcon-AO [11] (participating 2006 and 2007 with similar results), OLA (2007) [9], RiMOM (2006) [14] and HMatch (2006) [2]. For all of these systems we compared the types of mappings described above and measured the size of the mappings in number of correspondences, precision (P), recall (R) and F-measure (F). Computing these values with respect to the debugging based on learned disjointness, for one mapping between ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ we performed mapping debugging based on all $5 \times 5 = 25$ variants for $\mathcal{O}_1$ and $\mathcal{O}_2$ differing with respect to the ontology used as training data.

Results The results of these experiments are presented in Table 4. For each matching system we aggregated over all correspondences in the generated mappings. The first 4 data columns present the characteristics of the mappings generated by the matching systems. Obviously, there is a significant difference between the matching systems we used in our experiment. Falcon-AO generates the best mappings compared to the other systems, in particular with respect to mapping precision. This is partially caused by the fact that RiMOM and OLA on the one hand extract a one-to-one mapping using a low threshold or no threshold at all. HMatch, on the other hand, generates many-to-many mappings which results in a low precision with respect to the data set under discussion where all reference mappings are one-to-one mappings.

[14] `http://ontoware.org/projects/leda/`

Matching System	Automatically generated mappings				Debugged mappings based on reference disjointness				Debugged mappings based on learned disjointness			
	#	P	R	F	#	P	R	F	#	P	R	F
FalconAO-07	123	0.829	0.803	0.816	115	0.870	0.787	0.826	110.4	0.873	0.759	0.812
HMatch-06	203	0.404	0.646	0.497	113	0.708	0.630	0.666	99.2	0.736	0.575	0.646
RiMOM-06	332	0.301	0.787	0.435	232	0.427	0.780	0.552	211.8	0.460	0.767	0.575
OLA-07	389	0.257	0.787	0.387	233	0.425	0.780	0.550	204.4	0.436	0.702	0.537

Table 4. Number of correspondences (#), precision (P), recall (R) and F-measure (F) aggregated over input mappings generated by matching systems, debugged mappings based on the use of ontologies extended by reference disjointness, and debugged mappings based on the use of ontologies extended by learned disjointness.

The different characteristics of the matching systems are also reflected in the results of the debugging process. Based on the reference disjointness the debugging increases the F-measure for Falcon-AO by only 1%, while for the other systems we can measure a gain between 12% and 17%. This is based on the fact that for Falcon-AO only a small number of correspondences are removed (6.5%), while for the other systems between 29.5% and 44.3% of all correspondences are removed.

4.4 Discussion

One might argue that the results of these systems could also be optimized to a significant degree by e.g finding an appropriate threshold or extracting a one-to-one mapping. This objections points to an important advantage of our approach. The proposed debugging method adapts to the characteristics of the matching problem without any additional information about appropriate thresholds or any other additional information based on e.g. experiences with similar matching problems.

One might still criticize, that while our approach indeed increases the quality of a rather bad input mapping, it does not optimize a top matching system by a notable degree. This objection neglects an important aspect. A system that performs very well for a certain matching problem might produce poor results for a different problem. While OLA, for example, performed not very well with respect to our evaluation, it was among the top six matching systems of the OAEI 2007 benchmark testcase [5] with an overall performance similar to Falcon-AO, the best system in our evaluation. Therefore, given a realistic matching problem without reference mapping, we cannot eliminate choosing a system that generates substandard or average results. Notice also, that even with respect to the mappings generated by Falcon-AO, we measure a slight improvement. This improvement is based on a trade-off between precision and recall, which makes our approach in particular interesting for applications that require a highly precise mapping.

Comparing the debugging results based on reference disjointness to the results based on learned disjointness, there are only minor differences. As we argued, debugging based on the reference disjointness can be seen as an upper bound. Therefore, the results based on learned disjointness are surprisingly good. This can partially be explained by the good results of learning disjointness discussed in Section 4.2. In some

cases we even obtained results topping the upper bound, e.g, compare the F-measure for RiMOM. As a general pattern we observe that learned disjointness based debugging generates more precise mappings, but decreases recall.

This pattern is based on the fact that even incorrect disjointness statements may sometimes have positive effects in the debugging process. An example will illustrate this point. For the EKAW ontology LeDA learns that the classes *PC_Member* ("program committee member") and *Demo_Chair* are disjoint. Even though the strong disjointness assumption is incorrect, we have to accept that a person can be an instance of *Demo_Chair* without being an instance of *PC_Member* and vice versa. Thus, introducing a subsumption axiom between *PC_Member* and *Demo_Chair* has to be considered as an error. Our way to detect conflicts in mappings is exactly based on the idea to check wether or not the propagation of a subsumption relation from one ontology to the other ontology can be accepted from a formal point of view. This means that, even though the disjointness axiom $\mathit{PC\_Member} \sqsubseteq \neg\ \mathit{Demo\_Chair}$ is too strong, we nevertheless draw the correct conclusion in the debugging process that we are not allowed to model a subsumption between both classes. This pattern occurred several times within our experiments and explains that the results of debugging based on learned disjointness in some situations even outperforms debugging based on a reference disjointness.

5 Conclusion and Future Work

In this paper, we have addressed the problem of creating high quality mappings between lightweight ontologies which are the dominant form of ontologies currently available on the web. Our approach, that has already been successfully applied to expressive description logic ontologies, was to perform a posteriori debugging based on inconsistencies introduced by the mappings. In this paper we showed that an automatic enrichment of lightweight ontologies with disjointness axioms leads to equally good results. In particular, we could show that a machine learning technique produces disjointness information with an accuracy of 85 to 90 percent. Based on these automatically created disjointness information we could improve the overall quality of the mappings by up to 16 percentage points. These results are quite impressive considering that once the classifier has been trained our approach is completely automatic and does not require any human interaction.

One of the most important topics for future work is related to the interdependency between learned disjointness axioms and automatically generated correspondences. In our setting we first added learned disjointness axioms to the ontologies and used this additional information to guide the process of mapping debugging. Conversely, it would also be possible to use the information encoded in the mappings to optimize the process of disjointness debugging. Notice that in each of these alternatives we have to trust in a heuristic method to debug the results of another heuristic method. The most promising solution to cope with this dilemma is to combine disjointness and mapping debugging in a unique step. In such a setting we first have to find a way to make confidence values for disjointness axioms and correspondences comparable. Having once defined a complete ordering of disjointness axioms and correspondences based on a normalized confidence

value, an algorithm similar to algorithm 1 should be applicable to an ordered set containing both disjointness axioms and correspondences. We expect such an algorithm to have positive effects on the accuracy of both disjointness and mapping debugging.

The method described in this paper is applicable in many different settings. The most straightforward application is to improve the results of an automatic matching system by a one-shot application of the method. This corresponds to the experiments performed in this paper and we can expect to get a quality improvement comparable to the ones reported in this paper. Another option for using the method is to integrate the debugging functionality into existing matching systems in terms of a special extraction function that extracts the final mapping from a similarity matrix over concepts. This option has been investigated in [15]. Finally, it is clear that completely automatic approaches will always have their problems. A possible alternative is to use debugging functionalities to support a human expert in the evaluation of an automatically generated mapping. Such an interactive method will help us at least partially to avoid the dilemma mentioned above. In particular, we can ask the user to mark a subset of the generated mapping as correct or incorrect and use the method above to derive the implications of this partial judgement. Such an approach will probably significantly speed up manual mapping evaluation and produce a high quality mapping due to the human in the loop.

Acknowledgements: Research reported in this paper has been partially financed by the EU under the IST-2006-027595 project NeOn as well as by the German Science Foundation (DFG) in the Emmy Noether Programme under contract STU 266/3-1 and the Multipla project.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2002.
2. S. Castano, A. Ferrara, and G. Messa. Results of the HMatch ontology matchmaker in OAEI 2006. In *Proceedings of the ISWC 2006 Workshop on Ontology Matching*, Athens, GA, USA, November 2006.
3. P. Cimiano and J. Völker. Text2Onto - a framework for ontology learning and data-driven change discovery. In A. Montoyo, R. Munoz, and E. Metais, editors, *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3513 of *Lecture Notes in Computer Science*, pages 227–238, Alicante, Spain, June 2005. Springer.
4. L. Ding and T. Finin. Characterizing the semantic web on the web. In *Proceedings of the 5th International Semantic Web Conference (ISWC-06)*, Athens, GA, USA, November 2006.
5. J. Euzenat, A. Isaac, C. Meilicke, P. Shvaiko, H. Stuckenschmidt, O. Sváb, V. Svátek, W. R. van Hage, and M. Yatskevich. Results of the Ontology Alignment Evaluation Initiative 2007. In *Proc. of the ISWC 2007 Workshop on Ontology Matching*, Busan, Korea, 2007.
6. J. Euzenat, M. Mochol, P. Shvaiko, H. Stuckenschmidt, O. Sváb, V. Svátek, W. R. van Hage, and M. Yatskevich. Results of the Ontology Alignment Evaluation Initiative 2006. In *Proceedings of the ISWC 2006 Workshop on Ontology Matching*, Athens, GA, USA, 2006.
7. J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer Verlag, 2007.

8. J. Euzenat, H. Stuckenschmidt, and M. Yatskevich. Introduction to the Ontology Alignment Evaluation 2005. In *Proceedings of the K-CAP 2005 Workshop on Integrating Ontologies*, Banff, Canada, 2005.
9. J. Francois, D. Kengue, J. Euzenat, and P. Valtchev. OLA in the OAEI 2007 evaluation contest. In *Proceedings of the ISWC 2007 Workshop on Ontology Matching*, Busan, Korea, November 2007.
10. Z. Harris. Distributional structure. In J. Katz, editor, *The Philosophy of Linguistics*, pages 26–47, New York, 1985. Oxford University Press.
11. W. Hu, Y. Zhao, D. Li, G. Cheng, H. Wu, and Y. Qu. Falcon-AO: Results for OAEI 2007. In *Proceedings of the ISWC 2007 Workshop on Ontology Matching*, Busan, Korea, November 2007.
12. P. Jaccard. The distribution of flora in the alpine zone. *New Phytologist*, 11:37–50, 1912.
13. A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proceedings of the 6th International Semantic Web Conference (ISWC-07)*, Busan, Korea, 2007.
14. Y. Li, J. Li, D. Zhang, and J. Tang. Result of ontology alignment with RiMOM at OAEI'06. In *Proceedings of the ISWC 2006 Workshop on Ontology Matching*, Athens, GA, USA, November 2006.
15. C. Meilicke and H. Stuckenschmidt. Analyzing mapping extraction approaches. In *Proceedings of the ISWC 2007 Workshop on Ontology Matching*, 2007.
16. C. Meilicke and H. Stuckenschmidt. Applying logical constraints to ontology matching. In *Proceedings of the 30th Annual German Conference on Artificial Intelligence (KI-07)*, Osnabrück, Germany, 2007.
17. C. Meilicke, H. Stuckenschmidt, and A. Tamilin. Repairing ontology mappings. In *Proc. of the 22nd Conference on Artificial Intelligence (AAAI-07)*, Vancouver, Canada, 2007.
18. B. Patwardhan and Pedersen. Using measures of semantic relatedness for word sense disambiguation. In *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics*, pages 241–257, February 2003.
19. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.
20. S. Schlobach. Debugging and semantic clarification by pinpointing. In *Proceedings of the European Semantic Web Conference (ESWC)*, 2005.
21. S. Schlobach. Diagnosing terminologies. In *Proceedings of the 20th Conference on Artificial Intelligence (AAAI-05)*, Pittsburgh, Pennsylvania, USA, 2005.
22. S. Schlobach, Z. Huang, R. Cornet, and F. van Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39(3), 2007.
23. L. Serafini, H. Stuckenschmidt, and H. Wache. A formal investigation of mapping languages for terminological knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI05*, Edingurgh, UK, August 2005.
24. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2), 2007.
25. O. Sváb, V. Svátek, P. Berka, D. Rak, and P. Tomasek. Ontofarm: Towards an experimental collection of parallel ontologies. In *Poster Proceedings of the International Semantic Web Conference 2005*, 2005.
26. J. Völker and A. Kesseler. Learning Disjointness Axioms. Technical report, Institute AIFB, Universität Karlsruhe, December 2007.
27. J. Völker, D. Vrandecic, Y. Sure, and A. Hotho. Learning disjointness. In E. Franconi, M. Kifer, and W. May, editors, *Proc. of the 4th European Semantic Web Conference (ESWC'07)*, volume volume 4519 of *Lecture Notes in Computer Science*, pages 175–189. Springer, June 2007.
28. T. D. Wang. Gauging ontologies and schemas by numbers. In *Proceedings of the Workshop EON Evaluation of Ontologies for the Web*, 2006.