Partial Ontology Matching Using Instance Features

Katrin Zaiß and Stefan Conrad

Institute of Computer Science
Heinrich-Heine-Universität Düsseldorf
D-40225 Düsseldorf, Germany
{zaiss,conrad}@cs.uni-duesseldorf.de

Abstract. Ontologies are a useful model to express semantics in a machine-readable way. A matching of heterogeneous ontologies is often required for many different applications like query answering or ontology integration. Many systems coping with the matching problem have been developed in the past, most of them using meta information like concept names as a basis for their calculations. This approach works well as long as the pieces of meta information are similar. In case of very differently structured ontologies or if a lot of possible synonyms, homonyms or meaningless meta information are used, the recognition of mappings gets difficult. In these cases instance-based matching methods are a useful extension to find additional correct mappings resulting in an improved matching quality, because instances provide a lot of information about a concept. This paper presents a novel instance-based matching algorithm which calculates different features using instances. These features characterize the concepts and are compared using different similarity functions. Finally, the similarity values are used to determine 1:1 mapping proposals.

1 Introduction

The matching of different kinds of information sources is an issue widely discussed in literature. Especially for applications like the Semantic Web ontologies become more and more important, and in many cases, e.g. in query systems, we have to find correspondences between two or more differently structured ontologies. To cope with this matching problem many systems have been developed. Most of the systems use meta- and structure information, which is a logical and promising approach. But such algorithms do not always provide good results, for example if ontologies are expressed in different languages or if the granularity differs significantly. In these cases it makes sense to consider the instances of an ontology. Instances can mostly be found on the lowest level of an ontology, and the according superior structure is not important for the instance-based matching process.

There are already a few systems providing instance-based ontology matchers (see [ES07]). Some of them use machine learning techniques to classify instances and use this classification to determine concept mappings (e.g. [DMDH04]). Another general approach is to extract the content of instances such that they become comparable in a more compact way. This can be done by calculating several features (e.g. [EM07]). Additionally, instance sets can be compared to detect duplicates which indicate concept similarity(e.g. [BN05]).

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part II, LNCS 5871, pp. 1201–1208, 2009.

The remainder of the paper is organized as follows: The next section introduces a few similarity functions needed for the matching process. Section 3 describes the general flow of the presented algorithm. The following sections 4 and 5 described the different parts of the presented algorithm, feature and similarity calculation, in-depth and the determination of candidate mappings is shortly commented. After the presentation of an evaluation in section 6, the paper concludes with future work in section 7.

2 Similarity Measures

One of the most important part of a matching process is the similarity function, because it "decides" how similar two concepts are. For our approach we need several simple similarity functions which will be defined in the following.

If two values d_1 and $d_2 \in \mathbb{R}$ have to be compared, the *numS* measure is used:

$$numS(d_1, d_2) = \frac{1}{1 + |(d_1 - d_2)|} \tag{1}$$

Within our matching algorithm we need to compare sets of strings. For this purpose the following function is used: Let S_1 and S_2 be two sets containing strings. The strings of the two sets are compared pairwise using the edit distance. Since it is usual to work with similarity values $\in [0, 1]$ the value calculated by the edit distance is normalized and transformed into a similarity value $simEdit$. All string pairs that have a similarity $simEdit$ above a certain threshold t_{edit} are taken into account when calculating the similarity between the two sets. The similarity is calculated using the following equation, V is the set containing the attribute pairs with $simdEdit \geq t_{edit}$.

$$setS(S_1, S_2) = \frac{|V|}{|S_1| * |S_2|}. \tag{2}$$

This similarity function is a quite naive measure that we used for evaluation purposes here. We work on a more precise equation that better represents the similarity of two sets. Two equal sets should have a similarity value of 1 (which is not the case when using 2), but we want to allow more than one mapping partner per string.

Equally, sets of numeric values have to be compared. For this purpose the number of similar values is counted and set into relation to the cardinality of the sets. Two numbers $d_1, d_2 \in \mathbb{R}$ are similar if they hold $|d_1 - d_2| \leq \epsilon$, where $epsilon$ is a dynamically defined parameter, e.g. the standard derivation obtained for all instances of the attribute this numeric set belongs to. Let N_1 and N_2 be two sets containing numeric values, W is the set containing the similar value pairs. The similarity between N_1 and N_2 is calculated using the following equation:

$$setN(N_1, N_2) = \frac{|W|}{|N_1| * |N_2|} \tag{3}$$

Equally to the definition of $setS$ this equation has to be improved to better express our comprehension of set similarity.

3 General Process

Instances provide a big amount of information which should be used to increase the matching quality. Especially in case of different levels of specification or the use of synonyms/homonyms, instances allow to find a mapping without considering the unsimilar meta information. Anyhow, instance-based matching algorithms can not work alone because not all ontologies and/or not all concepts contain instances. Hence, the algorithm presented in this paper has to be integrated into a more complex system (see [ZC09] for an exemplary framework).

Concept-based matching methods form the foundation of the whole process. Simple mapping assertions, including very similar concept names or URIs detected by calculating the edit distance e.g. can easily be determined and provide a basis for the instance-based process. Concepts for which a mapping partner have been found do not participate in the next mapping step, i.e. the presented instance-based approach. For all remaining concepts that provide instances our novel instance-based method is executed.

The starting point of the presented algorithm is the set of instances. For each concept that provides instances a sample of k instances is extracted for each attribute, where k is a parameter depending on the total number of instances. Certainly, the accuracy of the algorithm may be higher, if k equals nearly the total number of available instances, but the time complexity increases as well. Further attention to this point is given in section 6. The samples of all attributes of one concept are collected and provide a basis for the feature calculation step, in which several features like average length or frequent values are calculated depending on the type of the instances (String or Number).The feature vectors of all attributes belonging to the same concept are subsumed in a concept feature vector (i.e. a vector of feature vectors). The whole process will be described in-depth in section 4. Afterward the computed concept feature vectors are compared in a similarity calculation step (see section 5 for details). Different similarity functions are computed and the values are aggregated to a single one. Since the features are computed attribute wise, their similarities have to be propagated upwards to obtain a concept similarity. Finally, the pairwise concept similarities are used to determine a mapping.

4 Feature Calculation

Instances provide a lot of information about the semantics of a concept, in some cases more than meta information like a concept name. The difficulty is to extract the information given by the instances in a way that they can be compared among different concepts. In our approach several features are calculated which express the semantics of the concepts on the basis of their instances. In values of instances can have different data types (see [Pau07]). In this paper we only focuses on the data types “String” and the different numerical data types like “Decimal” or “Integer”. All other data types are more special and are an evidence for mapping partners themselves. The calculated features are subsumed in a feature vector.

The attributes of each concept normally have different data types, i.e. the feature calculation is done for each attribute, and the different feature vectors are collected in a concept feature vector. In the following the different features will be described in more detail.

4.1 String Features

String is a frequently used data type assigned to varying attributes, i.e. semantics can not be derived directly. But by calculating different features the meaning of a concept can be limited. We currently use the following features to characterize the attributes of each concept based on the string values (additionally the type of the feature is specified):

1. average length - Integer
2. frequent values - Set of values
3. common substrings - Set of Strings
4. special chars - Set of chars

The *average length* value depends on the semantics of the concept and its attributes respectively. The values of attributes like "name" normally have a lower average length than attributes like "comment" or "description". Sure, a similar average length does not directly indicate equality but together with the other features it is a useful hint.

The detection of *frequent values* is an important hint for similarity. Frequent values are instance values that are very often assigned to an attribute. The definition of "frequent" can be chosen dynamically by specifying a threshold. By default, values that appear in more than 20% of the given instance are considered as frequent. Depending on the number of different values and the choice of the threshold the result set may contain more than one value.

The use of patterns may characterize attributes as well, but the extraction of a pattern (e.g. specified as a regular expression) is a problem which is not solvable in an adequate time space. Hence, this approach searches for *common substrings*, i.e. substrings that appear frequently. This feature is very similar to feature 2, but concentrates on substrings instead of whole instances. The length of the substrings should be as long as possible, whereas the length is always greater than or equal to two, otherwise single letters could be treated as common substrings. Depending on the frequency threshold a set of different substrings of different length can be found.

Special chars can also be a hint for similar attributes and their concepts respectively. In fact, special chars are common substrings of length 1, but they are not detected while computing feature 3. In general, chars like "@", ".", "/" or "," are only used within special types of attributes, hence it is useful to extract the set of all appearing special chars for each attribute.

4.2 Numeric Features

The semantics of numeric values is more difficult to capture, because the number of possible values is more limited. We use the following features to characterize the concepts based on the numeric values:

1. average value - number
2. frequent values - Set of values
3. minimum value - number
4. maximum value - number
5. decimal places - number

The *average value* gives an impression of all given instances. The average of earning values is higher than the average of page numbers for example, hence this feature is one characteristic of numeric values. Since the calculation of an average value has a relatively low time complexity and the value gets more exact if more values are used, all instances are taken into account when calculating this feature.

The computation of *frequent values* equals the second feature of the string-based features described above. Values that appear very often, i.e. with a relative frequency above a predefined threshold, are determined. Similar frequent values can indicate similar concepts, since some attribute (e.g. "mark") are limited to a set of different values.

The *minimum* and *maximum* values define the range of the numeric values which can be a hint for similarity. Similar to the previous feature the minimum and maximum values of attributes like "'earnings"' will be higher than those of page numbers or marks.

The number of *decimal places* is another feature that can be taken into account when comparing numeric attributes and their corresponding concepts respectively. This value gives a hint on the precision of the attributes.

5 Comparing Concepts

The previous section demonstrates how to capture information about a concept on the basis of its instances. In this section we will show how to compare the features in order to determine a concept similarity and finally a mapping. For each attribute of a concept several features are computed and are subsumed in a feature vector. Each concept feature vector consists of several attribute feature vectors which either belong to a string-based or to a numeric attribute. To compare two concepts the attribute feature vectors are compared pairwise. Hence, we need to choose the right similarity functions for each case. If the two attributes have a different type, no similarity is calculated. The whole process is repeated for every possible concept pair combination between two ontologies.

In the following the measures for comparing the features vectors are defined formally.

For attributes whose instances consists of strings several string features are calculated. To define the similarity function which calculates a similarity between two string-based attributes, we need to define an order to the features. For this purposes the enumeration used in Subsect. 4.1 is taken. Hence, a string feature vector f consists of the four elements (in the named order) average length (double value), a set of frequent values, a set of common substrings and a set of special chars. The similarity function to compare two sets of string features is defined as follows: Let $f_a = (a_1, a_2, a_3, a_4)$ and $f_b = (b_1, b_2, b_3, b_4))$ be two attribute feature vectors of c_i and d_j respectively.

$$StringSim(f_a, f_b) = \frac{1}{4} * (numS(a_1, b_1) + setS(a_2, b_2) + setS(a_3, b_3) + setS(a_4, b_4)) \quad (4)$$

For further experiments the assignment of weights could be a useful extension (see section 7). In case of comparing numeric features we have to compare average value

(double), frequent values(set of values), minimum and maximum values and the number of decimal places. Let $f_e = (e_1, e_2, e_3, e_4, e_5)$ and $f_f = (f_1, f_2, f_3, f_4, f_5))$ be two attribute feature vectors of c_i and d_j respectively.

$$NumericSim(f_e, f_f) = \frac{1}{5} * (numS(e_1, f_1) + setN(e_2, f_2) + numS(e_3, f_3) + numS(e_4, f_4) + numS(e_5, f_5)) \quad (5)$$

Similar to the string-based similarity calculation the assignment of weights could be helpful to enhance the quality.

Each attribute of a concept c_i is compared to each attribute of another concept d_j, i.e. the corresponding feature vectors are compared as described before. The result is a similarity matrix. Since the mapping should consist of pairs of matched concepts the attribute similarities have to be propagated up to the concepts. For this purpose we use common propagation strategies as they can be found in literature (e.g. [EM07]). Finally, the resulting concept similarities provide the basis for determining a mapping.

6 Evaluation

After describing our instance-based matching approach in-depth, this section subsumes the experiments we have made.

To do an evaluation we need to define some parameters as noted in the associated sections, i.e. we have to determine values for the number of instances belong to a sample (k value) and several thresholds. We have to do more studies on this issue, but for evaluation purposes we calculate k dynamically as 20% of the available instance number, which provides appropriate results, both for matching quality and computational complexity. All thresholds are set to 0.5, i.e. all similarity values above 0.5 indicate similarity.

As mentioned in section 3, the proposed methods shall be used within a more complex matching system to enhance the quality of the determined mapping. For evaluation purposes a system was created containing concept-based and the presented instance-based matching algorithms. The concept-based similarity method consists of five similarity computation steps: Label Comparison (using the edit distance), Trigram Calculation, Prefix/Suffix Comparison and two methods that compare the attribute sets.

To evaluate our matching process, several tests have been executed using different ontologies and different amounts of instances. To show the improvement reached with our instance-based methods, the tests are executed only using concept-based methods first and then extended with the presented algorithm.

For this purposes, some of the ontologies offered for the benchmark test series at the “Ontology Alignment Evaluation Initiative 2007” ([OAE07]) are taken as the basis for the calculations. The benchmark ontologies all describe the domain of bibliography but differ in hierarchy, labeling, existence of instances, suppressing of comments etc. A big advantage of this benchmark is the additional specification of the expected alignment. Ontologies without instances are not useful for this evaluation, as well as simple changes on the concept level. Finally, the ontologies with numbers 101 (reference ontology), 102 (irrelevant ontology), 202 (random names), 205 (synonyms) and

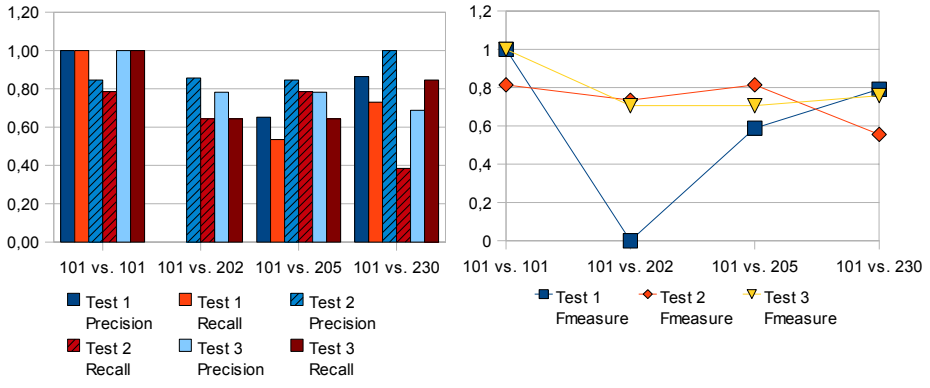

Fig. 1. Evaluation results

230 (flattened classes) have been chosen. Unfortunately, these ontologies do not provide many instances, hence additional ones have been inserted basing on real bibliographic data obtained from the "DBLP Computer Science Bibliography"([DBL08]).

The following test scenarios have been examined: only using concept-based methods (test 1), only using the presented instance-based matcher (test2) and concept-based methods combined with instance-based matcher (test 3), whereas the similarity values of both strategies are averaged (unweighted).
In the following the different tests and their results are shortly described and several advantages and limitations are pointed out. The different scenarios are referred by their numbers described above. An overview of the complete results is shown in Fig. 1.

101 vs. 101: The matching of one ontology to itself is a trivial task, but for completeness this test was executed as well. Some concepts do not have any instances, hence Recall cannot equal 1 for test scenario 2.

101 vs. 102: 102 is an ontology irrelevant for the domain of bibliography, in this case it is an ontology about wine. In all test scenarios some matches are detected, but the number of these false positives is very low which is a good result.

101 vs. 202: Ontology 202 has the same structure as 101 but names are replaced by random strings. In this case concept-based methods fail completely. In test scenario 2 the concept meta information are not considered, hence the result is nearly the same as in the comparison of 101 and 101 (values slightly differ because the dynamically chosen sample set is another one).

101 vs. 205: In ontology 205 concept and attribute names are mostly replaced by synonyms, which is a common problem for matching tasks. This matching task is quite difficult for concept-based matching systems as long as no external resources are used. For our (instance-based) algorithm, this test equals the test 101 vs. 101.

101 vs. 230: Ontology 230 is still quite similar to 101 (same labels) but some concepts are flatted. Concept-based methods work quite well, whereas in test 2 the Recall value is very low. This is a direct implication of the flattening process, because a lot of concepts share similar instance values.

To sum up we can state that the proposed algorithm works quite well and increases the matching quality in contrast to the exclusive use of concept-based methods. This observation is surely valid for nearly every instance-based method, but an evaluation only using instance-based methods (to evaluate them solely) is not reasonable because most matching systems use additional concept-based matchers. Especially in case of synonymous meta data or if random strings are used as concept names our approach enhances the matching quality significantly.

7 Conclusion and Future Work

In this paper we presented an instance-based matcher to improve the matching quality of common concept-based matchers. For this purpose several features are calculated (on the attribute level) using the instances . These features can be compared by applying several similarity measures and the similarity is propagated from the attribute to the concept level. The concept similarities provide a basis to determine a mapping. The evaluation shows that the matching quality generally increases if concept-based and instance-based matchers are combined in contrast to their exclusive application. But there is still room for improvements.

In future work we mainly want to focus on the specification of the similarity functions. Equally, the equations used to combine the feature similarities described in section 5 should be extended by the inclusion of weights.

Additionally the number of features is relatively low. In future work we plan to enlarge the set of calculated features, e.g. by the distribution of values or the standard derivation. We also have to spend more effort on evaluations.

References

[BN05] Bilke, A., Naumann, F.: Schema matching using duplicates. In: Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, Tokyo, Japan, April 5-8, 2005, pp. 69–80 (2005)

[DBL08] The DBLP Computer Science Bibliography (June 2008), http://www.informatik.uni-trier.de/~ley/db/

[DMDH04] Doan, A., Madhavan, J., Domingos, P., Halevy, A.Y.: Ontology Matching: A Machine Learning Approach. In: Handbook on Ontologies, pp. 385–404. Springer, Heidelberg (2004)

[EM07] Engmann, D., Maßmann, S.: Instance Matching with COMA++. In: Workshop Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW 2007), Aachen, Germany, March 5–6 (2007)

[ES07] Euzenat, J., Shvaiko, P.: Ontology matching. Springer, Heidelberg (2007)

[OAE07] Ontology Aligment Evaluation Initiative - 2007 Campaign (2007), http://oaei.ontologymatching.org/2007/

[Pau07] Biron, P.V., Malhotra, A.: XML Schema Part 2: Datatypes, 2nd edn (2007), http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

[ZC09] Zaiß, K., Conrad, S.: Instance-Based Ontology Matching Using Different Kinds of Formalisms. In: Proceedings of the International Conference on Semantic Web Engineering, Oslo, Norway, July 29-31, vol. 55, pp. 164–172 (2009)