International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:7, 2009

Instance-Based Ontology Matching Using Different Kinds of Formalisms

Katrin Zaiß and Tim Schlüter and Stefan Conrad
Institute of Computer Science
Heinrich-Heine-Universität Düsseldorf
40225 Düsseldorf, Germany
{zaiss,schlueter,conrad}@cs.uni-duesseldorf.de

Abstract—Ontology Matching is a task needed in various applications, for example for comparison or merging purposes. In literature, many algorithms solving the matching problem can be found, but most of them do not consider instances at all. Mappings are determined by calculating the string-similarity of labels, by recognizing linguistic word relations (synonyms, subsumptions etc.) or by analyzing the (graph) structure. Due to the facts that instances are often modeled within the ontology and that the set of instances describes the meaning of the concepts better than their meta information, instances should definitely be incorporated into the matching process. In this paper several novel instance-based matching algorithms are presented which enhance the quality of matching results obtained with common concept-based methods. Different kinds of formalisms are use to classify concepts on account of their instances and finally to compare the concepts directly.

Keywords—Instances, Ontology Matching, Semantic Web

I. Introduction

THE *Semantic Web*, an idea of Tim Berners-Lee, creates a vision of bringing meaning to the content of web pages. The semantics of the offered information should be structured and meaningfully processable for machines and Semantic Web applications. Ontologies are often used as a model for knowledge representation, made processable automatically by the use of languages like RDF and OWL. Ideally, there is one standardized ontology for each specific domain, which is referenced by every other ontology of this domain. Actually there are many different ontologies describing (nearly) the same domain, but using different labels or different structures. This implies the need of a Matching or Integration Process. Whenever there is a need to exchange information, first correspondences have to be found between different ontologies to get an overview of the common concepts. This process can hardly be done by hand for large ontologies, hence semi-automatic matching algorithms have to be developed. Mainly, there are three types of approaches to perform the matching process using different types of input. Concept-based algorithms try to compare ontologies only using information like labels, attributes, data types or structure. Instance-based algorithms use the set of instances to train learners or to compare instances using string similarity functions for instance. But most of the instance-based approaches do not work well whenever there is a lack of instances or if the instances of both ontologies do not have a lot in common. Hence, a lot of systems are hybrid, i.e. they combine concept- and instance-based algorithms. Due to the fact that instances provide a broader information content, that could definitely increase the quality of the match result, there is the need to develop an efficient instance-based matching algorithm.

The rest of the paper is organized as follows. First, related work is presented in Section II, whereas the most challenging tasks and deficits of existing systems are pointed out. Section III shortly introduces the used terminology. The theoretical basis of the presented methods is provided in Section IV. Section V describes the approach in detail, and Section VI presents an evaluation. The paper concludes with future work in Section VII.

II. Related Work

There are already a few systems providing instance-based ontology matcher. To motivate this work, some of the used methods are presented and their limitations are pointed out.

A. GLUE

An approach that uses instances as input is GLUE [1]. It makes use of machine learning techniques to gather information about the content or the syntactical representation of concepts helping to classify instances. The classification of the instances is then used to calculate a joint probability distribution. Using an appropriate similarity function, e.g. the Jaccard similarity, the distribution is transformed into a similarity value and into a similarity matrix. In a next step this matrix and domain-specific constraints and heuristic knowledge are applied to increase the matching quality. Finally, relaxation labeling (a graph algorithm) is used to determine an alignment.

The machine learning techniques used by GLUE do not work very well if instances differ syntactically. Another disadvantage is that GLUE cannot determine a mapping if there is a lack of instances.

B. COMA++

COMA++ [2] is a system which flexibly combines many different matchers, e.g. the comparison of names or data types or structural matcher. In [3] two instance-based matchers are proposed as an extension of the existing matcher library. The new instance-based matchers can be divided into constraint-based and content-based matching methods. The constraint-based approach classifies instances using general, numerical or pattern constraints. Among the general constraints there are methods calculating the average length or determining the set of the used characters. Numerical constraints check whether an instance is a number and determine its type, and pattern constraints test whether all instances belonging to one concept follow a certain pattern. The content-based algorithm compares the instances pair-wise using string-similarity functions like the edit distance.

The instance-based methods of COMA++ seem to be promising, but unfortunately the authors do not tell something about the used similarity measures. But the application of an appropriate similarity function is very important, since the comparison of the constraints for example is very challenging. A lot of question have to be answered, for instance if similar average length influences the similarity of two attributes.

C. Automatch

Another system using instance-based matching methods is Automatch [4]. For each domain experts create attribute dictionaries containing all possible values and the belonging probabilities of occurrence. The instances of an ontology are matched against these dictionaries, and the resulting mapping provides the basis for calculating an individual match score for each attribute. The several match scores are summed up and finally a mapping is determined by applying a "minimum cost maximum flow" graph algorithm.

One big drawback of Automatch is the huge effort needed to build the dictionaries and especially to calculate the probabilities.

D. Dumas

Dumas [5] is another instance-based matching approach which compares the set of instances to detect duplicates. This duplicate information is used to find similar attribute pairs and hence similar concepts.

If there are duplicates, this algorithm should work very well. In real-life datasets there should be duplicates, but if there are none the algorithm does not work, just as if there are no instances available at all.

E. Other Approaches

There are several other instance-based approaches like [6], [7] or [8] which are not covered by this paper due to minor importance for our approach. An overview and a short description of several instance-based matching system can be found in [9].

III. Terminology

To facilitate the description of the algorithms in Section IV and V the most important concepts are described in this section.

A. Matching

The matching problem describes the process of finding correlating entities representing the same real-world thing, i.e. similar entities or subsumptions. A mapping (the result of a matching process) between two given ontologies can be defined as follows: To map an ontology O onto an ontology P implies that for each entity (concept or relation) of O a corresponding entity of P having the same meaning is searched. Formally, the matching function performing this process can be defined as follows:

$$match : (O, P) \mapsto \{(c_1, c_2), \\ \text{with } c_1 \in O, c_2 \in P \text{ and } sim(c_1, c_2) > t\},$$

whereas sim is an arbitrary similarity function and t a predefined similarity threshold.

B. Similarity Measure

In order to compare two given vectors, a similarity measure is needed that is appropriate for the specific application. Two common distance measures are the Euclidean and the Manhattan distances [9]. For the following let $v, w \in \mathbb{N}_0^d$ denote two d-dimensional vectors.

The Manhattan distance $d^{L1}(v, w)$ and the Euclidean distance $d^{L2}(v, w)$ between v and w is given by

$$d^{L1}(v, w) = \sum_{i=0}^{d-1} |v_i - w_i| \text{ and}$$
$$d^{L2}(v, w) = \sqrt{\sum_{i=0}^{d-1} (v_i - w_i)^2}.$$

Distance functions calculate values between 0 and ∞, but for our purpose similarity values $sim \in [0, 1]$ are needed, where $sim = 1$ defines the maximum similarity. An appropriate transformation can be done by using the following equation:

$$sim = \frac{1}{1+distance}$$

The two above mentioned (dis-)similarity measures consider the value of each component. In some cases an angle-based measure like the cosine similarity is more appropriate to express similarity. Let $v * w$ denote the scalar product of the two vectors v and w, and $|v| = \sqrt{v * v}$ the length of v. The similarity function cosine : $\mathbb{N}_0^d \times \mathbb{N}_0^d \rightarrow [0, 1]$ is defined by

$$cosine(v, w) = \frac{v*w}{|v| \cdot |w|}.$$

Of course, the cosine similarity is also defined for $\mathbb{R}^d$ or $\mathbb{C}^d$, but our approach only uses the vector space of the natural numbers.

C. Evaluation Measures

For evaluation purposes some measure have to be defined, which can determine the quality and the matching accuracy, respectively.

Precision defines the proportion of correct classified mappings to the number of all retrieved mappings. A less mathematical and more intuitive definition of Precision is the following (cf. [10]) :

$$Precision = \frac{\text{\#correctFoundMappings}}{\text{\#foundMappings}}$$

The **Recall** measure calculates the fraction of correct classified mappings to the number of all possible correct mappings. Again, Recall can be defined using a more informal definition (cf. [10]):

$$Recall = \frac{\text{\#correctFoundMappings}}{\text{\#existingMappings}}$$

The **F-Measure** combines Precision and Recall to gain an overall result. It is defined as:

$$F - Measure = \frac{\text{2*Precision*Recall}}{\text{Precision+Recall}}$$

D. Regular Expressions

Regular languages or regular expressions provide an easy way to identify strings of interest in certain texts, or in our case to distinguish certain instances of ontologies. They are part of nearly every basic course in theoretical computer science (an overview about formal languages can be found e.g. in [11]), and of course every programming language has its own way to deal with regular expressions. Thus some short examples about how regular expressions (RegEx for shorthand) are used will be given in the following and in Figure 1 (cf. [12] for more detail).

- The RegEx
  ```
  R.*
  ```
  matches any string that starts with “R”, followed by an arbitrary number of any character or number (the dot denotes any character and the asterisk that this symbol can appear arbitrarily often), it would match e.g. “RegEx”.
- RegEx can be composed in many different variations. The expression
  ```
  .*(((c|C)onference)|((w|W)orkshop)).*
  ```
  would match any string in which Conference, conference, Workshop or workshop appear.
- For convenience there are many abbreviations, for example “\d” for any digit and “\w” for any character (small or capital) or digit, and special symbols like “\s” for a whitespace. An expression that makes use of this is
  ```
  [\w-\.]+@([\w-]+\.)+[\w-]{2,4}
  ```
  for email addresses. Note, that “[...]$\{2,4\}$” denotes that the expression in the squared brackets has to occur twice at minimum and maximum 4 times.

IV. How to handle instance information

Ontologies often include a lot of instances per concept which probably better describe the semantics of the concept than its meta information like concept name etc. Thus, instance information should definitely be used in a matching process. But how to catch the specific aspects of the instances? Our first approach, which has been shortly described in [13], makes use of predefined regular expressions (created by domain experts) to classify instances and finally concepts. The creation of regular expressions may be difficult for domain experts if they do not have experience in creating regular expressions. Hence, the process can be simplified by just giving catchwords instead of regular expressions (second approach). Both methods return a set of vectors describing all concepts of one ontology. In the following, these concept vectors are referred to as RECW vectors because they are either created using regular expressions or catchword groups but the processing of the vectors is the same. These vectors are compared pairwise with the ones of a second ontology using a similarity function. This process will be described in Section V. In the following, the process from specifying regular expressions and catchwords to creating RECW vectors is described for each of the two strategies.

A. First Approach - Predefined Regular Expressions

In the last decades many effort has been made in the research of language inferring. Especially interesting for our purpose is the inference of regular languages or expressions from given positive examples. Unfortunately, Mark E. Gold has stated in his seminal work [14] that it is impossible to learn the whole class of regular expressions only from positive examples. Until today many algorithms for inferring regular expressions from a finite number of positive examples have been proposed and implemented, but most of them, especially the ones actually used, are only heuristic, i.e. there is no precise characterization of the (sub-)class of language that is covered by them [15]. In addition to that these algorithms are not really practicable for our needs because even for alphabets with strong limited size the runtime is too long. A recent and elaborate algorithm for example is [16], but even for inferring expressions with alphabet size $|\Sigma| = 15$ it takes about 9 hours on a current PC system on the average.

Thus, a domain expert creates a list of regular expressions that fits to (the instances of attributes of) the ontologies. In order to describe an attribute through a regular expression, a certain amount of instances is compared to a list containing regular expressions, and the first fitting regular expression is assigned to the instance, and the regular expression that is assigned to the majority of the regarded instances belonging to a certain attribute is assigned to this attribute. Certainly, to avoid misassignment at this point, the regular expressions have to be ordered from very specialized at the beginning to more common at the end of the list. The process of rearranging the list is described more in detail in Algorithm IV-B-1.

B. Arranging Lists and Creating Vectors

After creating regular expressions manually by domain experts the list containing them has to be ordered from very specialized at the beginning to more common at the end of the list, because instances are always assigned to the first fitting regular expression. Let u denote the unordered list containing d different regular expressions. After applying Algorithm IV-B-1 on u for sequencing, a list l is obtained that is ordered appropriate for our purpose. Now a bijective function δ can be defined from $\{1, ..., d\}$ to the subspace Θ of regular expressions, which are contained in l (and u), with

$$\delta(i) = r,$$

where r is the i-th regular expression of the ordered RegEx list l.

Algorithm IV-B-1:

```
Require: unordered list $u$ containing $d$ RegEx
Ensure: ordered RegEx list $l$
  int[] counter=zeros(1,d); {init. d-dim array with zeros}
  {count RegEx matches}
  for all ontology $o$, concept $c \in o$, attribute $a \in c$ do
    insert “certain amount” of attribute instances $i \in a$ in set $s_a$;
    for all $i \in s_a$ do
      for all $k = 1, ..., d$ do
        if $k$-th RegEx of list $u$ matches instance $i$ then
          counter[k]++;
        end if
      end for
    end for
  end forlist $l$ =emptyList;
  {sort list $l$ according to counter[]}
  for all $k = 1, ..., d$ do
    find $m$ with counter[$m$] = $\max!$, counter[m]≥ 0; $l$.appendAtBeginning($m$-th RegEx of list $u$); counter[m]=-1;
  end for
  return $l$
```

With the process of comparing instances with regular expressions described in Algorithm IV-A, a function reg can be defined from the attribute space to the space of regular expression, which holds

$$reg(a) = r,$$

where a is an attribute of a certain concept and r the regular expression that is assigned to a (i.e. to the majority of instances examined belonging to this attribute).

After defining the two functions $\delta(i)$ and $reg(a)$ the RECW vector can be defined more easily: the RECW vector $v \in \mathbb{N}^d$ representing a concept c is the vector whose components v_i ($i = 1, ..., d$) contain the number of assignments from $\delta(i)$ (i.e. the i-th regular expression in the RegEx list l) to an attribute in c. Hence the sum of all components of v equals the number of attributes in c.

A short informal algorithm for building a RECW vector v is given by Algorithm IV-B-2, and an illustration of the process is given in Figure 1.

Algorithm IV-B-2:

```
Require: concept $c$, ordered RegEx list $l$
Ensure: RECW vector $v$ calculate $\delta$ and $\delta^{-1}$;
  {calculate function $reg$}
  for all attribute $a \in$ concept $c$ do
    compare certain amount of instances with RegEx list $l$ to determine $reg(a)$;
  end for
  {initialize RECW vector $v$}
  for all component $v_i$, $1 \leq i \leq d$ do
    $v_i = 0$
  end for
  {count allocated RegEx in accordant component}
  for all attribute $a \in c$ do
    $inc(v_{\delta^{-1}(reg(a))}, 1)$;
  end for
  return $v$
```

Naturally, the lack of instances is a severe drawback of all instance-based matching methods. Our approach creates zero-vectors for concepts without instances, and the similarity between such concepts, from which at least one is represented by a zero-vector, is defined as 0.

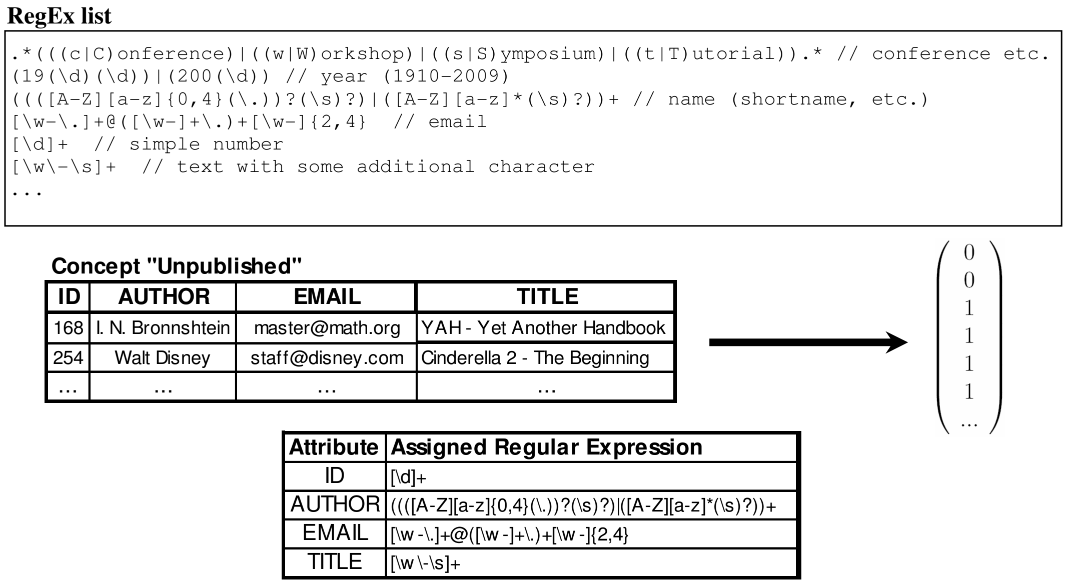
C. Second Approach - Transforming Catchwords

In most cases domain experts may not be able to formulate complex regular expressions, for example if the matching system shall be used in areas like biology or banking industry. Additionally, the creation of appropriate regular expressions may even be difficult or time-consuming for computer scientist for all non-trivial attributes. Hence, the process of creating regular expressions is facilitated, such that domain experts just have to specify catchwords. In this case, catchwords are a group of strings $(s_1, ..., s_n)$ which are often included in instances belonging to one attribute, i.e. they could be characterized as most probably common substrings. The domain expert just thinks about attributes that are likely for his domain and specifies frequently used substrings.

An concrete example for this is the field of bibliographic references, where an expert has to define catchwords belonging to the domain of paper classification. Thinking of the attribute “title of event”, which will surely appear when having ontologies about bibliography, he could specify catchwords like “conference”, “workshop” or “symposium”, because these words often appear when naming the title of the presentation event. An attribute like “email” could be described using catchwords like “@”, “.de”, “.org” and “.com”.

After specifying all catchword groups a list u is obtained with d entries, where each entry consists of a catchword group describing one possible attribute.

But in some cases it is difficult to specify catchwords, for example attributes like “author name” do not have frequently common substrings. In this case, the domain expert could give a set of examples, such as “John”, “Jim” or “Thomas” that are treated as catchwords. The same mechanism is necessary for numerical attributes, because usually they do not have common substrings. Alternatively, example values could be extracted automatically from one ontology within the match process, for example by taking the m most frequently appearing instances per attribute. Attributes like “month” or “day” normally have a limited space of possible instances, which cannot be described very well with user-defined catchwords. Due to this observation, the catchword lists always include predefined catchword groups consisting of all possible instance values for standard types like month and day. To describe instances by comparing them with catchwords or example values, the following algorithm is used: a certain amount of instances of



RegEx list

```
.*(((c|C)onference)|((w|W)orkshop)|((s|S)ymposium)|((t|T)utorial)).* // conference etc.
(19(\d)(\d))|(200(\d)) // year (1910-2009)
((([A-Z][a-z]{0,4}(\.))?(\s)?)|([A-Z][a-z]*(\s)?))+ // name (shortname, etc.)
[\w-\.]+@([\w-]+\.)+[\w-]{2,4}  // email
[\d]+  // simple number
[\w\-\s]+  // text with some additional character
...
```

Concept "Unpublished"

ID	AUTHOR	EMAIL	TITLE
168	I. N. Bronnshtein	master@math.org	YAH - Yet Another Handbook
254	Walt Disney	staff@disney.com	Cinderella 2 - The Beginning
...	...	...	...

$$\longrightarrow \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ \dots \end{pmatrix}$$

Attribute	Assigned Regular Expression
ID	[\d]+
AUTHOR	((([A-Z][a-z]{0,4}(\.))?(\s)?)|([A-Z][a-z]*(\s)?))+
EMAIL	[\w -\.]+@([\w -]+\.)+[\w -]{2,4}
TITLE	[\w \-\s]+

Fig. 1. Illustration and example of vector creation. An RegEx list with 6 specified regular expressions is given above (see Algorithm III-D for more details about RegEx). The concept “Unpublished” is displayed as a relation, containing some instances for every attribute. After comparing the instances with the RegEx list a regular expression is assigned to every attribute as shown in the tabular at the bottom. Finally, a RegEx vector for concept “Unpublished” is created as described in Algorithm IV-B-2.

each attribute is compared to the list of catchwords/examples and the one that fits to the majority of the instances is assigned to the attribute. A catchword or example group fits to an instance, whenever the instance contains one of the catchwords or example values. The process of vector creation remains the same (see Algorithm IV-B-2, just replace “RegEx” by “catchword/example” list).

V. The Matching Process

First, it is important to state, that the proposed instance-based algorithms should not work alone as the only method to compare the ontologies and to determine a mapping. Our method has to be integrated into a system containing different matchers and several features like relevance feedback to increase the match quality and to enable matching of concepts without instances. A possible system architecture is shown in Figure 2 (more details can be found in [17]).

Concept-based matching methods form the beginning 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. The exact specification of the used concept-based methods for evaluation purposes is given in the corresponding Section VI. Concepts that found a mapping partner do not participate in the next mapping step, i.e. the presented instance-based approach.

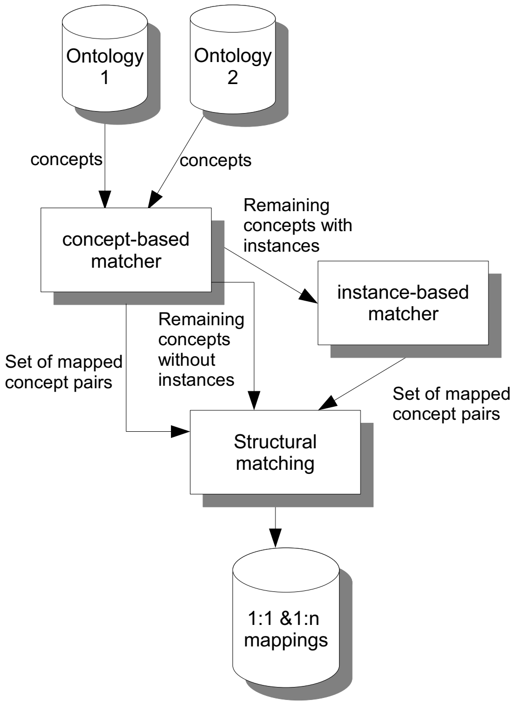

Ontology 1
Ontology 2
concepts
concepts
concept-based matcher
Remaining concepts with instances
instance-based matcher
Remaining concepts without instances
Set of mapped concept pairs
Set of mapped concept pairs
Structural matching
1:1 &1:n mappings

Fig. 2. Possible Matching Framework

A. General Process

The flow of our matching process is shown in Figure 3. First, a certain amount of instances is extracted for each attribute of each concept that provides instances. For each instance set one of the strategies described in IV is executed (but the choice of the strategy is fixed for the complete matching process). As a result of Algorithm IV-B-2 each concept is described by a vector containing numbers representing the quantity of each regular expression/catchword group that is assigned to an attribute of this concept. Finally there are two sets of RECW vectors (one for each ontology), which are compared pairwise using a similarity measure as described in III-B. The resulting similarities are used to determine 1:1 mappings by selecting those concept pairs whose similarity values are above a certain threshold and which do not participate in another mapping yet. Additionally, 1:n mappings can be found by comparing the sum of two (or more) vectors with another one (more details in Section V-B).

B. Creation of Candidate Mapping

The process of creating 1:1 and 1:n mappings is described in Algorithm V-B.

Algorithm V-B:

```
Require: similarity matrix $M$
  with $i$ identifier for a concept vector of $O_1$ and $j$ of $O_2$
  $[i, j]$ cosine similarity of the vectors
  threshold $t$
Ensure: set $Map$ of 1:n mappings
  for all $i \in [0, M.length() - 1]$ do
    define set $CM$ containing mapping candidates
    for all $j \in [0, M.length() - 1]$ do

      if $[i, j] \geq t$ then
        $Map \leftarrow (i, j)$
      else if $0 < [i, j] < t$ then
        $C \leftarrow j$
      end if
    end forconstruct power set $\mathcal{P}(CM)$
    for all $cm \in \mathcal{P}(CM)$ do
      sum up vectors for all concepts, whose identifier is $\in cm$
      result: vector $cmv$
      calculate sim $sc$ of concept vector $i$ and $cmv$
      if $sc > t$ then
        $Map \leftarrow (i, cm)$
      end if
    end for
  end for
  return $Map$
```

The whole similarity matrix is regarded and concept pairs having a similarity values above the threshold are directly added to set Map of Mappings. After that, a set CM is created for each concept $c \in O_1$ containing all concepts $v_i in O_2$ with $s < sim(c, v_i) < t$, whereas O_1 and O_2 are ontologies, $s \in [0, t[$ and t is a defined threshold. Then, the power set of CM is calculated. For each element cm of CM (which is a set of concepts) c and cm are compared by summing up the RECA vectors of the concepts contained in cm and calculating the similarity between the concept vector of c and this sum. If the similarity is above the threshold the belonging concepts are added to Map. Of course, the whole process has to be repeated for each concept $j \in O_2$.

The proposed method provides good results, as one can see in Section VI, but of course there are more sophisticated methods which can be tested in future developments. Especially the application of algorithms that try to find a mapping configuration that maximizes Precision and Recall in general (without just picking out the highest similarity values but doing more computation to have an appropriate overall result) have to be examined.

VI. Evaluation

A. Evaluation Framework

As mentioned in section V, 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 six similarity computation steps, which are described more in detail in the following:

- Label Comparison
- Trigram Calculation
- Prefix/Suffix Comparison
- Attribute Set Comparison
- Attribute Matching

The *Label Comparison* step calculates a string similarity between the concept names based on the edit distance.

The *Trigram Calculation* uses an ngram-like measure to express the similarity between two concept names (e.g. used by [18]). If two words A and B are compared using the trigram measure, the trigrams for each word are constructed and the accordances are counted. The number of accordances and the number of trigrams are combined to calculate a similarity by using the dice coefficient (cf. [19]) defined by:

$$\frac{2 * \text{accordances}}{\#\text{trigrams(A)} + \#\text{trigrams(B)}}$$

The Trigram measures is useful to find similar concept names containing typing mistakes for example.

The *Prefix/Suffix Comparison* is a simple comparison of the beginning and the end of the concept names. If one concept name is the prefix/suffix of the other one or vice-versa, the similarity value is 1, otherwise the similarity is 0. This measure is important to recognize if one concept is an abbreviation of another, e.g. ”phone” and ”phonenumber”.

In the *Attribute Set Comparison* step each attribute is compared to all the other ones using a SimSet similarity measure. First, the string similarity values between all attributes are computed since these establish the basis for the SimSet calculation. Each attribute can now be represented by a vector containing his similarity values. The vectors of all attributes belonging to the same concept are summed up and divided by the number of attributes. Finally, the scalar product of the two resulting vectors is computed representing the SimSet value ([10]).

The *Attribute Matching* measure examines the attribute sets separately. Again, for each attribute the string similarity values are calculated, but this time only to the attributes of the other concepts. For each concept the number of similarity values

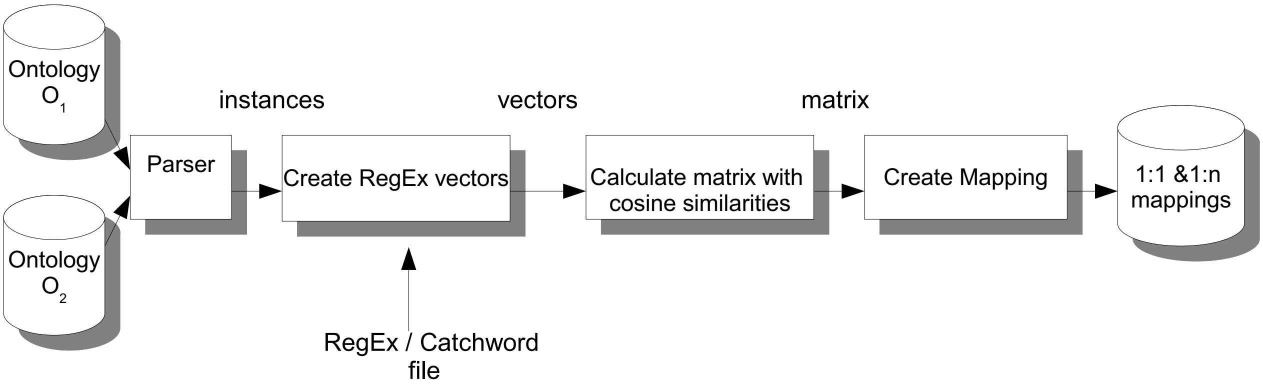


Fig. 3. Instance-Based Matching Process

above a certain threshold is counted (since the string similarity is bijective, it is the same number) and the similarity value is computed by calculating

$$\frac{\#\ \text{simValues} \geq \text{threshold}}{\#\ \text{attributes of concept A}} + \frac{\#\ \text{simValues} \geq \text{threshold}}{\#\ \text{attributes of concept B}},$$

i.e. the fraction of attributes of concept A having a mapping partner plus the fraction of attributes of concept B that are very similar to another one.
Finally, the value is normalized by dividing it by 2.

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, first the test are executed only using concept-based methods and then extended with the presented algorithms. For this purposes, some of the ontologies offered for the benchmark test series at the “Ontology Alignment Evaluation Initiative 2007” ([20]) 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 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”([21]).

B. Evaluation Results

Table I shows an overview of the composition of evaluated ontologies.

In detail, the following three test scenarios have been examined:

1) concept-based similarity calculation (only for concepts containing instances)
2) concept-based methods combined with the first approach
3) concept-based methods combined with the second approach

TABLE I
OVERVIEW OF THE ONTOLOGIES USED FOR EVALUATION

number	concepts	attributes	concepts with instances
101	36	78	28
102	74	101	12
202	36	78	28
205	36	78	28
230	28	73	25

Tests number 2 and 3 are executed using Manhattan, Cosine and Euclidean distance. The final similarity values are computed by taking the average of the calculated concept- and instance-based similarity. In the following, the different tests and their results are described in detail and several advantages and limitations are pointed out. Additionally, the Precision and Recall values are presented and explained for chosen tests, whereas the results for concept-based methods and the two presented approaches are subsumed. The different scenarios are referred by their numbers described above. Figure 4 summarize the results. The test 101 vs. 102 is omitted due to the fact that Precision and Recall are zero for all tests. The results for test 2 and 3 are represented by an average value calculated by the detailed values shown in Figure 5, because several similarity measures are used.

101 vs. 101: The matching of one ontology to itself is a trivial task, but for completeness this test was executed as well. Logically Precision and Recall are 1, and the similarity between equal concepts is 1, too. The values are the same for all tested similarity measures and all approaches.

101 vs. 102: 102 is an ontology irrelevant for the domain of bibliography, in this case it is a wine ontology. The used RegEx-files are created by domain experts and adapted to possible bibliographic data. Hence, the regular expressions fitting to the instances of 102 are very common (placed at the end of our RegEx file). The algorithms detects some matches, but the number of these false positives is very low which is a good result. Nevertheless, Precision and Recall are 0 for all

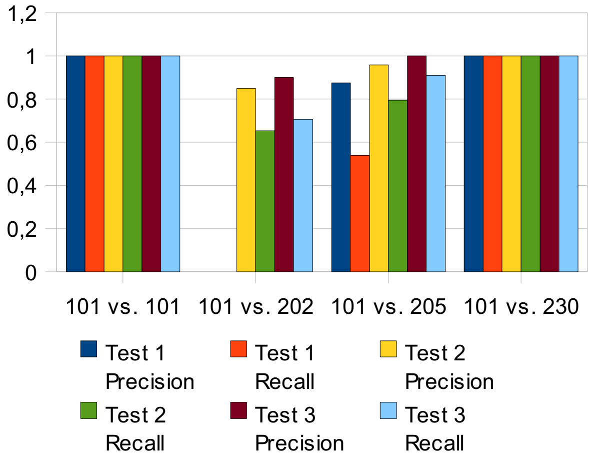


Fig. 4. Evaluation results

measures and approaches.

101 vs. 202: Ontology 202 has the same structure as 101 but all concept and attributes names are replaced by random strings. This results in an unacceptable matching results for the concept-based methods (test 1). No correct mapping can be found. In contrast, tests 2 and 3 provide good results which is reasonable since the instances are not affected directly by the changes.

101 vs. 205: In ontology 205 concept and attribute names are mostly replaced by synonyms. This matching task is quite difficult for concept-based matching systems, because synonyms have to be detected by using external resources. For our (instance-based) algorithm, this test equals the test 101 vs. 101, hence Precision and Recall are quite high.

101 vs. 230: Ontology 230 is still quite similar to 101 (same labels) but some concepts are flatted, e.g. the "Date" concept is rejected and information about a date is stored directly in the concept. The most difficult part of this test is the detection of the 1:2 mappings. If the flattened concepts provide instances only for a part of their attributes, they are more similar to the according concept of these attributes in the other ontology. An example: The concepts book (b_1) and date of 101 have to be mapped to book (b_2) of 230. Most of the attributes of b_2 are equal to those of date, such that b_2 and date are more similar than b_2 and b_1 (with a similarity above the threshold). Thus, our algorithm could be expanded, such that 1:1 mappings are taken into account when 1:n mappings are determined. Since the label of the concepts are very similar to those of 101 concept-based methods provide good results. Tests 2 and 3 confirm the results.

Our evaluation shows that the proposed method works quite well. It is an advantage, that concept label are not taken into account, such that mapping synonyms is easy. More problems occur, if the number of attributes containing instances is small. Since this algorithm should be embedded in a more complex system with different matchers, it can be stated that the proposed instance-based matching algorithm is very useful to improve matching quality.

VII. Conclusion

Ontology matching is a task used in several applications and performed by many systems using different algorithms. As shown the number of instance-based methods is limited and gives room for improvements. Our first proposed instance-based method makes uses of regular expressions to classify attributes by scanning instances. These regular expressions are used to compare and finally to match concepts. The evaluation states that the algorithm is helpful, especially for ontologies which differ considerable in labels. As described above this matcher has to be integrated into a more complex framework offering different types of matcher.

Before performing the matching process, domain experts have to build the regular expressions file adapted to a special domain. This results in an additional effort, but assuming that matching tasks are executed several times for the same domain these files can be reused. One can also think of a collection of different files (collected by-and-by or created by different domain experts), where the RegEx file has to be determined by the user or chosen dynamically by an automatic mechanism.

Our second approach uses catchwords and positive examples to classify instances. This avoids the problem of creating appropriate regular expressions, but the accuracy probably gets lost. Another problem is that example instances might not be representative for the domain because they contain typing mistakes or are randomly not included.

References

[1] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy, "Ontology Matching: A Machine Learning Approach," in *Handbook on Ontologies*. Springer, 2004, pp. 385–404.

[2] H. H. Do and E. Rahm, "COMA - A System for Flexible Combination of Schema Matching Approaches," in *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*. Morgan Kaufmann, 2002, pp. 610–621.

[3] D. Engmann and S. Maßmann, "Instance Matching with COMA++," in *Datenbanksysteme in Business, Technologie und Web (BTW 2007), Workshop Proceedings, 5.-6. März 2007, Aachen, Germany*, 2007.

[4] J. Berlin and A. Motro, "Database Schema Matching Using Machine Learning with Feature Selection," in *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002, Toronto, Canada, May 27-31, 2002, Proceedings*, 2002, pp. 452–466.

[5] A. Bilke and F. Naumann, "Schema Matching Using Duplicates," in *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, 2005, pp. 69–80.

[6] G. Stumme and A. Maedche, "FCA-MERGE: Bottom-Up Merging of Ontologies," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 2001, pp. 225–234.

[7] A. Doan, P. Domingos, and A. Y. Levy, "Learning Source Description for Data Integration," in *WebDB (Informal Proceedings)*, 2000.

[8] M. S. Lacher and G. Groh, "Facilitating the Exchange of Explicit Knowledge through Ontology Mappings," in *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference, May 21-23, 2001, Key West, Florida, USA*, 2001.

[9] J. Euzenat and P. Shvaiko, *Ontology Matching*. Heidelberg (DE): Springer-Verlag, 2007.

[10] M. Ehrig and S. Staab, "QOM - Quick Ontology Mapping." in *INFORMATIK 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Ulm, 20.-24. September 2004*. GI, 2004, pp. 356–361.

[11] J. Rothe, *Complexity Theory and Cryptology*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

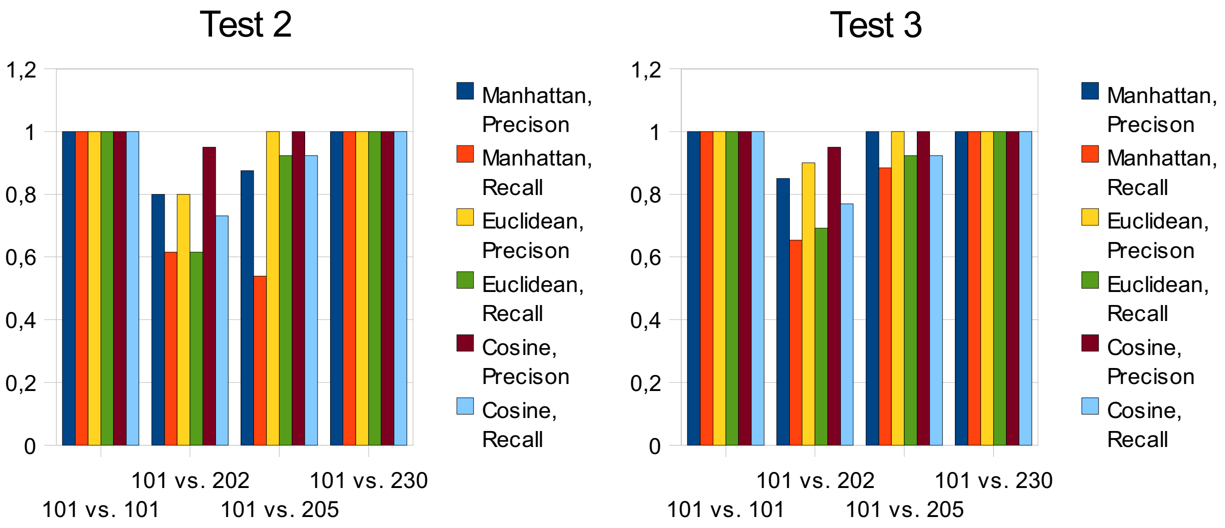


Fig. 5. Detailed results for Tests 2 and 3

[12] M. Habibi, *Real World Regular Expressions with Java 1.4*. APress, 2004.

[13] K. Zaiß, T. Schlueter, and S. Conrad, "Instance-based ontology matching using regular expressions," in *OTM Workshops*, ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 5333. Springer, 2008, pp. 40–41.

[14] E. M. Gold, "Language Identification in the Limit," *Information and Control*, vol. 10, no. 5, pp. 447–474, 1967. [Online]. Available: http://www.isrl.uiuc.edu/∼amag/langev/paper/gold67limit.html

[15] H. Fernau, "Algorithms for Learning Regular Expressions," in *ALT*, ser. Lecture Notes in Computer Science, S. Jain, H.-U. Simon, and E. Tomita, Eds., vol. 3734. Springer, 2005, pp. 297–311.

[16] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren, "Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data," in *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM, 2008, pp. 825–834.

[17] K. Zaiß, "Entwicklung eines Frameworks fuer instanzbasiertes Ontologie-Matching (Developing a Framework for instance-based Ontology Matching) ," in *Tagungsband zum 20. GI-Workshop über Grundlagen von Datenbanken (20th GI-Workshop on the Foundations of Databases), Apolda, Thringen, 13.-16. Mai 2008*, 2008.

[18] H. H. Do and E. Rahm, "Coma - a system for flexible combination of schema matching approaches." in *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, 2002, pp. 610–621.

[19] M. Kay and M. Röscheisen, "Text-translation alignment." in *Computational Linguistics*, vol. 19, no. 1, 1993, pp. 121–142.

[20] "Ontology Aligment Evaluation Initiative - 2007 Campaign," http://oaei.ontologymatching.org/2007/, 2007.

[21] "The DBLP Computer Science Bibliography," http://www.informatik.uni-trier.de/ ley/db/, June 2008.