Towards a Benchmark for Instance Matching *

Alfio Ferrara, Davide Lorusso, Stefano Montanelli, Gaia Varese

Università degli Studi di Milano,
DICo, 10235 Milano, Italy,
{ferrara, lorusso, montanelli, varese}@dico.unimi.it

Abstract. In the general field of knowledge interoperability and ontology matching, instance matching is a crucial task for several applications, from identity recognition to data integration. The aim of instance matching is to detect instances referred to the same real-world object despite the differences among their descriptions. Algorithms and techniques for instance matching have been proposed in literature, however the problem of their evaluation is still open. Furthermore, a widely recognized problem in the Semantic Web in general is the lack of evaluation data. While OAEI (Ontology Alignment Evaluation Initiative) has provided a reference benchmark for concept matching, evaluation data for instance matching are still few. In this paper, we provide a benchmark for instance matching, with the goal of taking into account the main requirements that instance matching algorithms should address.

1 Introduction

The increasing popularity of Semantic Web technologies makes the ontology matching process a crucial task. Ontology matching [1] aim is to (semi) automatically detect semantic correspondences between heterogeneous ontologies. It can be performed at two different levels: schema matching and instance matching. The objective of *schema matching* [2] is to find out a set of mappings between concepts and properties in different ontologies, while the aim of *instance matching* is to detect instances referred to the same real-world object. When comparing different knowledge representations, ontologies' schemas should be merged, in terms of concepts and properties describing the domain. Then, mappings between different descriptions (i.e., ontologies' instances) of the same object should be discovered, in order to achieve the goal of providing a data integration system over Semantic Web sources.
Instance matching is also crucial in projects like OKKAM[1] [3], where the main idea is that real-world objects' descriptions could be retrieved, univocally identified and shared over the Web.
Most research has been focused on schema level matching, while instance matching problem has been mainly studied in the database field, in which it is more

* This paper has been partially funded by the BOEMIE Project, FP6-027538, 6th EU Framework Programme.
[1] http://www.okkam.org/.

specifically called *record linkage* problem [4–6]. However, as shown in the paper, instance matching brings new problems in comparison to record linkage and requires specific technologies.

2 The Instance Matching Problem

The instance matching problem is defined as follows. Given two instances $i1$ and $i2$, belonging to the same ontology or to different ontologies, instance matching is defined as a function $Im(i1, i2) \rightarrow \{0; 1\}$, where 1 denotes the fact that $i1$ and $i2$ are referred to the same real-world object and 0 denotes the fact that $i1$ and $i2$ are referred to different objects.
In order to find out properly if two individuals are referred to the same real-world object, an instance matching algorithm should satisfy different kinds of requirements. As shown in Figure 1, those can be divided in three main categories.

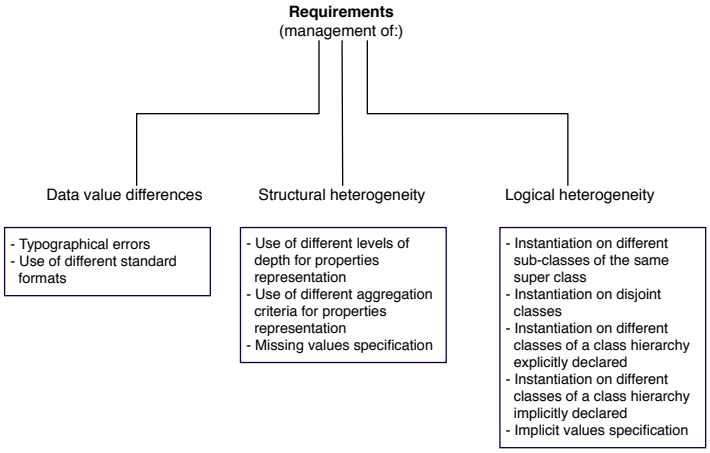

Fig. 1. Instance matching requirements

Data value differences. An instance matching algorithm is required to recognize, as better as possible, corresponding values, even if data contain errors or are represented using different standard formats. This issue has been addressed in the field of record linkage research, and the problem of comparing instances' property values is the same as comparing records' attribute values.

Structural heterogeneity. Instances belonging to different ontologies can not only differ within their properties values, but they can also have different structures. While in record linkage the structure of records is usually given and schema and record matching are different problems, in instance matching, schema and instances are more strictly related. Thus, besides the capability to evaluate the level of similarity between property values, instance matching techniques have to go beyond heterogeneous individual representations by identifying the pairs of matching properties between two considered instances.

Logical heterogeneity. A specific ontologies' matching problem, which is not taken into consideration in record linkage process, is the need to infer implicit knowledge, typically referred to concepts hierarchy within the ontologies.

3 Design of a Benchmark for Instance Matching

A widely recognized problem in the Semantic Web is the lack of evaluation data. While OAEI (Ontology Alignment Evaluation Initiative)[2] [7] has provided a reference benchmark for concept matching, evaluation data for instance matching are still few. Further works dealing with concept matching evaluation are those published in ESWC 2008 [8, 9]. In particular, they argue that ontology matching techniques cannot be evaluated in an application independent way, since the same matching technique can produce different quality results based on the end-to-end application that exploits the alignments.
In this paper, we provide a benchmark for instance matching. The aim of our benchmark is to take into account all the main requirements presented in the previous section and to provide a complete set of tests for instance matching algorithms evaluation. A contribution of our work is not only the definition of a specific benchmark, but also the definition of a semi-automatic procedure for the generation of several different benchmarks. In Figure 2, the overall process of benchmarks generation is shown. As an example of this general procedure, we describe in the following a specific instantiation of it, that is the creation of a specific benchmark for instance matching. That benchmark is available at *http://islab.dico.unimi.it/iimb/*.

3.1 Reference ABox Generation

First of all, we chose a domain of interest (i.e., the domain of movie data), and we created a reference ($\mathcal{ALCF}(D)$) TBox for it, based on our knowledge of the domain. The reference TBox is available at *http://islab.dico.unimi.it/ontologies/-benchmark/imdbT.owl*. This contains 15 named classes, 5 object properties and 13 datatype properties. The reference TBox is then populated by automatically creating a reference ABox. Data are extracted from IMDb [3] by executing a query

[2] http://oaei.ontologymatching.org/2007/benchmarks/.
[3] http://www.imdb.com/.

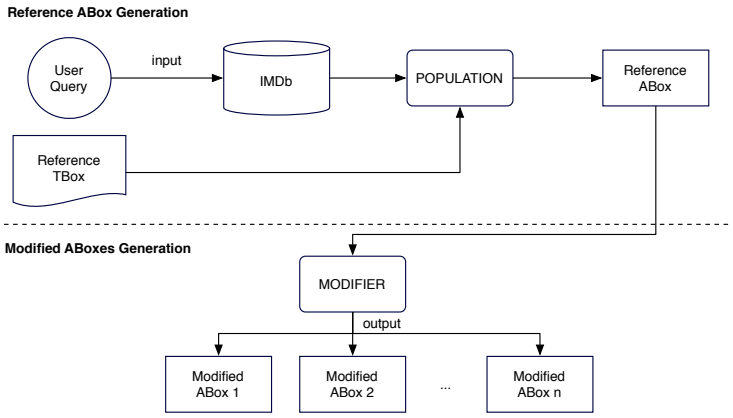

Fig. 2. Benchmarks generation

Q of the form:

$$SELECT * FROM\ movies\ WHERE\ title\ LIKE\ '\%X\%'$$

where X is a variable specifying a word of our choice. Thus, all selected movies contain the word X in their title. The corresponding individuals in the reference ABox are referred to similar objects, but each of them represents a distinct object in the real world. As a consequence, each instance can be univocally identified. In order to get our reference ABox, we put $X = Scarface$. The reference ABox obtained in that way contains 302 individuals, that is all the movie objects matching the query and all the actors in the movie cast.

3.2 Modified ABoxes Generation

Once the reference ABox is created, we generate a set of modified ABoxes, each consisting in a collection of instances obtained modifying the corresponding instances in the reference ABox. Transformations introduced in benchmark ABoxes can be distinguished into three main categories. In particular, each modification category simulates a specific problem that can be found when comparing ontologies' instances, that is the issues discussed in section 2. Modifications belonging to different categories are also combined together within the same ABox.

4 Generating Instance Modifications

In this section, we describe the *Modifier* module of our benchmarks generation procedure, that is the way the modified ABoxes of benchmarks are generated. Given the reference ABox as input, and a user specification of all the transformations to apply on it, the *Modifier* module automatically produces the corresponding modified ABoxes. In the following, all the modifications that can be applied on the reference ABox are presented.

4.1 Data Value Differences

The goal of this first category of modifications is to simulate the differences that can be found between instances referred to the same object at the property value level. Those include typographical errors, use of different standard formats to represent the same value, or a combination of both within the same value.

Typographical errors. Real data are often dirty. That is mainly due to typographical errors made by humans while storing data.
In order to simulate typographical errors, we use a function that takes as input a datatype property value and produces as output a modified value. This kind of transformation can be applied to each datatype property value (e.g., string value, integer value, date value). The modifications to apply on the input value are randomly chosen between the following:

- *Insert character.* A random character (or a random number, if the property has a numerical value) is inserted in the input value at a random position.
- *Modify character.* A random character (or a random number, if the property has a numerical value) is modified in the input value.
- *Delete character.* A random character (or a random number, if the property has a numerical value) is deleted in the input value.
- *Exchange characters' position.* The position of two adjacent characters (or two adjacent numbers, if the property has a numerical value) is exchanged in the input value.

For example, the movie title "Scarface" can be transformed in the modified value "Scrface", obtained deleting a random character from the original string.
In addition, it is possible to specify the level of severity (i.e., low, medium or high) in applying such transformations. Anyway, the number of transformations introduced in the input value is proportional to the value's length. If the number of transformations to apply is greater than one, the corresponding value can be modified combining different transformations.
Typographical modifications can be applied to "identifying properties", "non-identifying properties" or both. That classification is based on the analysis of the percentage of null and distinct values specified for the selected property. In particular, properties with an high percentage of distinct values and a low percentage of null values are classified as the most identifying.
Of course, the total amount of modifications applied to each modified ABox has to change the reference ABox in a way that it is still reasonable to consider the two ABoxes semantically equivalent. In other words, a modified ABox is included in the benchmark only if a human can understand that its instances are referred to the same real-world objects as the ones belonging to the reference ABox. Thus, in order to evaluate the distance between the reference ABox and each modified ABox, we introduce a measure that takes into account the number of modifications applied to the same ABox, the kind of the properties (i.e., "identifying properties" or "non-identifying properties") which have been

modified, and the level of severity of the modifications (i.e., low, medium or high). However, this measure does not affect the instance matching results in a deterministic way, since they depend on the weight that the tested algorithm gives to each kind of modification. Anyway, we assume that a modified ABox can be considered semantically equivalent to the reference ABox only if it changes no more than 20% of each instance description.

Use of different standard formats. The same data within different sources can be represented in different ways.
In order to simulate the use of different standards within different sources, we use a function that takes as input a property value which allows standard modifications (e.g., person name) and produces as output a modified value, using a different standard format. For example, the director name "De Palma, Brian" can be transformed in the modified value "Brian De Palma", which is another standard format to specify a person name.

4.2 Structural Heterogeneity

Another kind of situation that is simulated in our instance matching benchmark is the comparison between instances with different schemas. In fact, even assuming that concept mappings are available, the same individual feature (i.e., each instance property) can be modeled in different ways. Moreover, different descriptions of the same real-world object can specify different subsets, eventually empty, of all the possible values for that property. Combinations of different transformations belonging to this class of modification are also applied in the benchmark.

Use of different levels of depth for properties representation. A first example of this class of heterogeneity is shown in Figure 3. The two instances

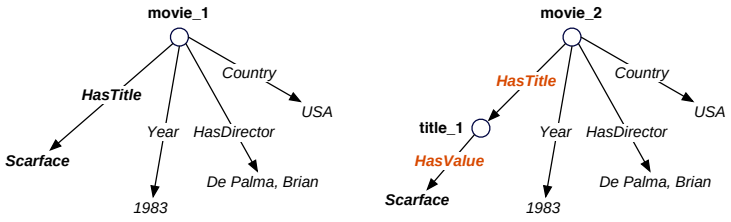

Fig. 3. Use of different levels of depth to represent the same property

*movie*_1 and *movie*_2 are both referred to the same film, but the movie title property is modeled in two different ways. In fact, the title of *movie*_1 is specified directly through a datatype property value, while the title of *movie*_2 is

specified through a reference to another individual which has a property with the same title value (i.e., "Scarface"). In particular, in the first representation, the property *HasTitle* is a datatype property, while in the second one it is an object property and its value is the reference to *title_1* instance.
In order to simulate the comparison between instances with different schemas, we use a function that takes as input a datatype property and produces as output an object property with the same name. Moreover, the function creates a new attribute to the generated object property, whose value is the same as the original datatype property.

Use of different aggregation criteria for properties representation. In an analogous way, the name of a person can be stored all within the same property, or it can be split into different properties such as, for example, *Name* and *Surname*. Figure 4 shows two different ways of modeling the name "Pacino, Al". In the first representation the whole value is stored within the property

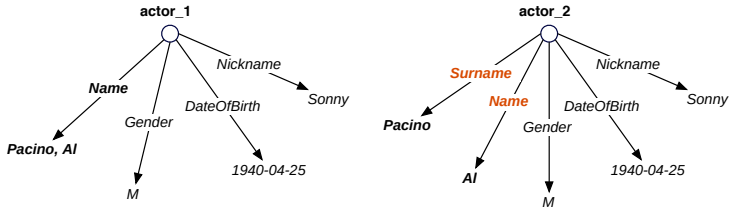

Fig. 4. Use of different aggregation criteria to represent the same property

Name, while in the second one the string is split into the two values "Pacino" and "Al", referred to the properties *Name* and *Surname* respectively.
In order to simulate the comparison between properties modeled in different ways, we use a function that takes as input a datatype property value that can be split and produces as output two new datatype properties, each specifying a different part of the original value.

Missing values specification. A further example of structural heterogeneity is shown in Figure 5. The two instances *movie_1* and *movie_2* are both referred to the same film, but the two different descriptions specify different subsets of values on the property *Genre*.
In order to simulate the comparison between different sets of values referred to the same property, we use a function that takes as input the set of values specified for a selected property and produces as output a subset, eventually empty, of it. This kind of transformation can be applied to each property. Moreover, if a property allows multiple values, it is possible to specify if deleting all the values of the selected property or a random number of them.

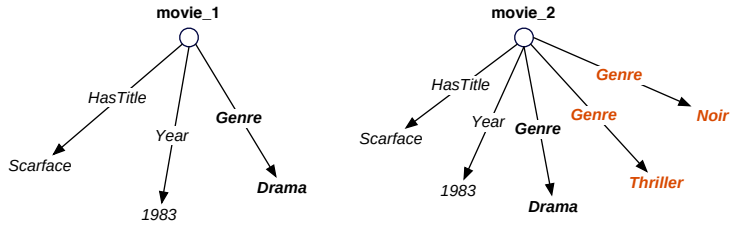

Fig. 5. Specification of different subsets of values on the same multi-values property

4.3 Logical Heterogeneity

Finally, instance matching process should take into account the need to use some kind of reasoning, in order to find out correctly instances to be compared. In fact, ontologies' individuals referring to the same entity can be instantiated in different ways within different ontologies. In the following we describe five kinds of situations that we develop in our benchmark, that can also be combined together. Each requires some kind of reasoning. Examples of those are shown in Figure 6.

Reference TBox

$Movie \sqsubseteq Item$	$Movie \sqcap Product \sqsubseteq \bot$
$Film \sqsubseteq Item$	$Movie \equiv \forall p.G$
$Product \sqsubseteq Item$	$SubM \equiv \forall p.SubG$
$Action \sqsubseteq Movie$	$SubG \sqsubseteq G$

Reference ABox	**Modified ABox**
$movie\_1 : Movie$	$movie\_1 : Film$
$movie\_2 : Movie$	$movie\_2 : Product$
$movie\_3 : Movie$	$movie\_3 : Action$
$movie\_4 : Movie$	$movie\_4 : SubM$
$movie\_5 : Movie$	$movie\_5 : Movie$
$(movie\_5, \text{“}Scarface\text{”}) : HasTitle$	$movie\_5 : (\exists HasTitle.\text{“}Scarface\text{”})$

Fig. 6. Examples of logical heterogeneity

Instantiation on different subclasses of the same superclass. This transformation is obtained instantiating identical individuals into different subclasses of the same class. For example, in our benchmark, all the movie objects are instances of class *Movie* in the reference ABox. Instead, in one of the modified ABoxes, we change the type of those individuals, making them instances of class *Film*. Classes *Movie* and *Film* are both subclasses of *Item*. In Figure 6, *movie_1*

is instance of *Movie* in the reference ABox, while it is instance of *Film* in the modified ABox. Instance matching algorithms are thus required to recognize that those two instances are referred to the same object, even if they belong to different concepts.

Instantiation on disjoint classes. This transformation is obtained instantiating identical individuals into disjoint classes. For example, in one of the modified ABoxes, we change the type of all the movie objects, making them instances of class *Product*. Classes *Movie* and *Product* are defined as disjoint classes in the reference TBox. In Figure 6, *movie_2* is instance of *Movie* in the reference ABox, while it is instance of *Product* in the modified ABox. In this case we want that tested algorithms would be able to recognize that instances belonging to disjoint classes cannot be referred to the same real-world object, even if they seem identical.

Instantiation on different classes of a class hierarchy explicitly declared. This transformation is obtained instantiating identical individuals into different classes on which an explicit class hierarchy is defined. For example, an individual representing a movie can be classified as an instance of the general concept *Movie*, as it is in the reference ABox, or it can be classified as an instance of a more specific subclass of it, such as *Action*, *Biography*, *Comedy* or *Drama*, depending on the value that the movie instances specify on the property *Genre*. In Figure 6, *movie_3* is instance of *Movie* in the reference ABox, while it is instance of its subclass *Action* in the modified ABox, since it is an action movie. Instance matching algorithms are thus required to recognize that those two instances are referred to the same object, even if they belong to different concepts within the class hierarchy. This explicit class hierarchy declaration can be recognized using a RDFS reasoner.

Instantiation on different classes of a class hierarchy implicitly declared. A further modification that we apply in the benchmark is the instantiation of identical individuals into different classes on which an implicit class hierarchy is defined. Such an implicit class hierarchy declaration can be obtained through the use of restrictions. For example, the restrictions specified on classes *Movie* and *SubM* in the reference TBox, implicitly declare that *SubM* is a subclass of *Movie*. In Figure 6, *movie_4* is instance of *Movie* in the reference ABox, while it is instance of *SubM* in the modified ABox. Instance matching algorithms are thus required to recognize that those two instances are referred to the same object, even if they belong to different concepts which are not explicitly related. This implicit class hierarchy declaration can be recognized using a DL reasoner.

Implicit values specification. Another use of restrictions that requires a reasoning process, is the comparison between an explicit specified value and an implicit specified one, that is using an *hasValue* restriction. This kind of situation

is simulated in our benchmark by adding a new type for each instance of the modified ABox. This type is a class that (implicitly) specifies property values through an *hasValue* restriction. In Figure 6, in the reference ABox, *movie_5* is instance of *Movie* and its value on the property *HasTitle* is "Scarface"; in the modified ABox, *movie_5* is as well instance of *Movie*, but it is also instance of the restriction class that implicitly specifies the value "Scarface" for its *HasTitle* property. Instance matching algorithms are thus required to recognize that those two instances are referred to the same object, even if some property values of the modified instance are implicitly defined.

5 Benchmark at Work

In this section, we describe how the generated benchmark is used to evaluate instance matching algorithms. Each execution of the evaluation process takes as input a couple of ABoxes, that is the reference ABox and one of the modified ABoxes, and produces the set of instance mappings found by the tested algorithm. The output mapping alignment is then compared with the expected one, which is given together with each modified ABox. That reference alignment is automatically generated by specifying a mapping for each couple of corresponding instances, that is the one belonging to the reference ABox and the one obtained by applying to it one or more of the modifications discussed in section 4.
Instance matching algorithms are evaluated according to the following parameters.

- *Precision:* the number of correct retrieved mappings / the number of retrieved mappings.
- *Recall:* the number of correct retrieved mappings / the number of expected mappings.
- *F-measure:* 2 · (precision · recall) / (precision + recall).
- *Fall-out:* the number of incorrect retrieved mappings / the number of non-expected mappings.
- *Execution time:* time taken by the tested algorithm to compare the two input ABoxes. This parameter measures how well the tested algorithm scales.

As an example, the results obtained by two instance matching algorithms are reported. Figure 7 shows the precision and recall evaluation of the two instance matching algorithms over the generated benchmark, distinguishing the results obtained in the three main classes of problems simulated in our benchmark (i.e., data value differences, structural heterogeneity, logical heterogeneity) and the ones obtained executing each algorithm without using any reasoner and using a (DL) reasoner (i.e., Pellet). The results obtained comparing the reference ABox with modified ABoxes simulating data value differences are higher than the ones obtained in the other categories, since string matching techniques are quite consolidated. The results obtained comparing the reference ABox with modified ABoxes simulating structural heterogeneity are not very high because neither the first nor the second algorithm can manage the use of different aggregation criteria

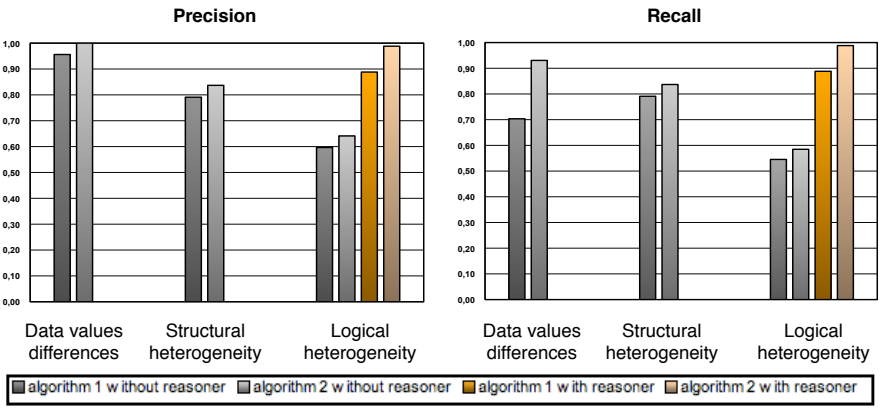

Fig. 7. Precision and recall evaluation

for properties representation. The results obtained comparing the reference ABox with modified ABoxes simulating logical heterogeneity are greatly affected by the use of a reasoner.
Finally, in Figure 8, the overall results obtained executing the two algorithms (with reasoner) on our benchmark are reported. That test had been executed on a Pentium 4 (2.00 GHz) with 512 MB of RAM. For each pair of compared

IM Algorithm	Precision	Recall	F-measure	Fall-out	Execution time
algorithm_1	0.88	0.79	0.81	0.05	50 sec
algorithm_2	0.94	0.92	0.93	0.01	31 sec

Fig. 8. Overall evaluation of two instance matching algorithms

instances, the first algorithm [10] analyzes all their property values, while the second algorithm [11] checks only the values specified for the "most identifying" properties. That is why the execution time of the first algorithm is greater than the execution time of the second one. Moreover, the recall of the second algorithm is higher than the recall of the first one due to the fact that all the modifications applied to "non-identifying" properties are ignored. A more detailed description of the two algorithms is available in [10, 11].

6 Concluding Remarks

In this paper, we provided a benchmark for instance matching, taking into account the main requirements that instance matching algorithms should address.

A contribution of our work is not only the definition of a specific benchmark, but also the definition of a semi-automatic procedure for the generation of several different benchmarks.
Future works include the creation of further benchmarks dealing with data belonging to different sources and different domains. In particular, we would like to create a benchmark in which data belonging to different sources but referred to the same real-world objects are compared. For example, it can include a mapping between movie descriptions in IMDb and Amazon. In that case, the expected alignments have to be done manually, so the benchmark dimension cannot be significant for a real benchmark. However, it would be interesting to compare the results obtained by the same algorithms executing that benchmark and our semi-automatically generated one, in order to evaluate the quality of our benchmark generation itself.
Another possible development would be the definition of a set of rules that automatically choose the modifications to apply, for each modified ABox, to the reference ABox.

References

1. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer-Verlag (2007)
2. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. Journal on Data Semantics (JoDS) (2005)
3. Bouquet, P., Stoermer, H., Niederee, C., Mana, A.: Entity name system: The backbone of an open and scalable web of data. In: Proceedings of the IEEE International Conference on Semantic Computing, ICSC 2008, IEEE (2008)
4. Fellegi, I., Sunter, A.: A theory for record linkage. J. Am. Statistical Assoc. (1969)
5. Winkler, W.: The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, Wachington, DC (1999)
6. Gu, L., Baxter, R., Vickers, D., Rainsford, C.: Record linkage: Current practice and future directions. Technical report, CSIRO Mathematical and Information Sciences, Canberra, Australia (2003)
7. Shvaiko, P., Euzenat, J., Noy, N., Stuckenschmidt, H., Benjamins, V., Uschold, M.: Proceedings of the 1st international workshop on ontology matching (om-2006) collocated with the 5th international semantic web conference (iswc-2006), athens, georgia, usa, november 5, 2006. In: OM. Volume 225., CEUR-WS.org (2006)
8. Hollink, L., van Assem, M., Wang, S., Isaac, A., Schreiber, G.: Two variations on ontology alignment evaluation: Methodological issues. ESWC 2008 (2008)
9. Isaac, A., Matthezing, H., van der Meij, L., Schlobach, S., Wang, S., Zinn, C.: Putting ontology alignment in contex: Usage scenarios, deployment and evaluation in a library case. ESWC 2008 (2008)
10. Castano, S., Ferrara, A., Montanelli, S.: Matching ontologies in open networked systems: Techniques and applications. Journal on Data Semantics (JoDS) (2006)
11. Bruno, S., Castano, S., Ferrara, A., Lorusso, D., Messa, G., Montanelli, S.: Ontology coordination tools: Version 2. Technical Report D4.7, BOEMIE Project, FP6-027538, 6th EU Framework Programme (2007)