An Effective Rule Miner for Instance Matching in a Web of Data

Xing Niu, Shu Rong, Haofen Wang and Yong Yu
APEX Data & Knowledge Management Lab
Shanghai Jiao Tong University
Shanghai, P.R.China
{xingniu, rongshu, whfcarter, yyu}@apex.sjtu.edu.cn

ABSTRACT

Publishing structured data and linking them to Linking Open Data (LOD) is an ongoing effort to create a Web of data. Each newly involved data source may contain duplicated instances (entities) whose descriptions or schemata differ from those of the existing sources in LOD. To tackle this heterogeneity issue, several matching methods have been developed to link equivalent entities together. Many general-purpose matching methods which focus on similarity metrics suffer from very diverse matching results for different data source pairs. On the other hand, the dataset-specific ones leverage heuristic rules or even manual efforts to ensure the quality, which makes it impossible to apply them to other sources or domains. In this paper, we offer a third choice, a general method of automatically discovering dataset-specific matching rules. In particular, we propose a semi-supervised learning algorithm to iteratively refine matching rules and find new matches of high confidence based on these rules. This dramatically relieves the burden on users of defining rules but still gives high-quality matching results. We carry out experiments on real-world large scale data sources in LOD; the results show the effectiveness of our approach in terms of the precision of discovered matches and the number of missing matches found. Furthermore, we discuss several extensions (like similarity embedded rules, class restriction and SPARQL rewriting) to fit various applications with different requirements.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability; H.2.8 [**Database Management**]: Database Applications; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

Keywords

Instance matching, semi-supervised learning, EM algorithm, association rule mining

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
Copyright 2012 ACM 978-1-4503-1156-4/12/10 ...$15.00.

1. INTRODUCTION

The Linking Open Data (LOD) project, also known as Linked Data [4] or the Web of data, is an ongoing effort to connect structured open data published on the Web. Encouraged by the Linked Data principle[1], these data are represented in a Resource Description Framework (RDF[2]) and interlinked using typed links to constitute a single global data space. With these links, we can navigate from a data item in one data source to related items in other sources.

Currently, LOD already contains abundant data sources of various domains: media, geography, life science, publications, government, etc. Since these data sources are usually designed and built independently, they often contain instances (entities) that refer to the same real-world thing with descriptions using heterogeneous schemata. According to the Linked Data principles, instances use URIs as their identifiers and different data sources contain instances of different URIs even though some instances might represent the same entity. Due to the decentralized nature of LOD, while it is difficult or even impossible to assign a unique URI for equivalent instances from different data sources, we can still interlink these instances through equivalence relations (i.e. `owl:sameAs`, a special link from the OWL standard[3]) [8].

The LOD space is continuously evolving; when a data source is newly added or is updated, we should connect instances to their possible correspondences in other existing data sources. The procedure for addressing this problem is often called *instance matching*, also known as *entity matching* or *link discovery* in the field of the Semantic Web.

Most practical instance matching methods for LOD are either domain-specific [30, 28] or dataset-specific [12, 3], so they cannot be adapted to other domains or data sources. In recent years, general-purposed approaches have also been proposed. On one hand, link discovery frameworks leverage manually defined dataset-specific matching rules (i.e. link specifications) to discover matched instances [13, 32]. On the other hand, other approaches focus on similarity metrics on instance descriptions, properties and property values to find correspondences automatically [16, 6, 2, 18, 19, 25]. Their performance depends heavily on the quality of underlying data and thus suffers from very diverse matching results for different data source pairs.

In this paper, we offer another choice to tackle the instance matching problem. We try to relieve the burden on user

[1] `http://www.w3.org/DesignIssues/LinkedData.html`
[2] `http://www.w3.org/TR/rdf-concepts/`
[3] `http://www.w3.org/TR/owl-ref/`

by automatically discovering dataset-specific matching rules while preserving high accuracy and good coverage. These derived rules have the capacity to find the most discriminative data characteristics for a given data source pair; this is similar to methods considered in [14, 15]. We list the main differences between our approach and the above two in detail in the related work section (Section 6).

Specifically, we design a semi-supervised learning algorithm (based on an Expectation-Maximization (EM) framework) to iteratively refine matching rule sets and find new matches of high confidence using these rules. A small number of existing matches in the form of `owl:sameAs` properties are used as seeds and the matching rules are treated as parameters (to be estimated) for maximizing the likelihood function. In addition, we proposed a graph-based metric to estimate the matching precision. The metric serves as the likelihood function. We also introduce Dempster's rule to combine confidence values of different matching rules. Furthermore, we discuss several extensions to fit various application scenarios of different requirements. Our approach is evaluated with real-world large scale data sources in LOD. The experimental results demonstrate the effectiveness of our approach in terms of the high precision of newly discovered matches and the large number of missing matches it found.

The rest of this paper is organized as follows. In Section 2, we give an overview of the framework of our proposed approach and describe some relevant concepts. Then, in Section 3, we introduce the workflow of our semi-supervised instance matching approach and discuss its convergence and complexity. Some straightforward extensions are also introduced in Section 4. Section 5 reports the experimental results. We discuss related work and compare some other approaches with ours in Section 6, and finally, we give a conclusion and outline future work in Section 7.

2. OVERVIEW AND RELEVANT CONCEPTS

The problem we want to solve is discovering instance matches, so firstly, we introduce some basic concepts.

An instance that refers to a real-world thing is usually described by several property-value pairs. A property-value pair along with an instance constitute a triple $\langle s, p, o\rangle$. Generally, the subject (s) is the instance mentioned above, the predicate (p) stands for the property and the object (o) stands for the value, which can be a literal or another instance. The set of triples in a given source constitute a graph G.

Definition 1. **Equivalence of Instances**, denoted by $\sim_{\mathcal{I}}$, is an equivalence relation. It indicates that two instances are actually the same thing in the world. Two instances, e_1 and e_2, which are identified by different URIs are equivalent iff. $(e_1, e_2) \in \sim_{\mathcal{I}}$.

Definition 2. A **Correspondence** found by instance matchers can be represented as $\langle e_1, e_2, \text{conf}\rangle$ in this paper, where e_1 and e_2 are two instances and they satisfy $(e_1, e_2) \in \sim_{\mathcal{I}}$ at the confidence value of conf (conf $\in [0, 1]$).

Instance matchers usually set a threshold of confidence to determine what correspondences can be output as the final matches.

The framework of our proposed approach is outlined in Figure 1. After preprocessing, a vector of property-value pairs, similar to document description, is attached to each instance. Then, initial matches (i.e. seeds, we will discuss how to get seeds in Section 5.1) are imported into the system to drive the discovery of new matches. After that, the set of seeds is expanded with newly found high-quality matches and the system enters another loop. The same procedure is repeated until the termination condition is satisfied.

The main method of discovering matches in our approach is learning rules to deduce instance equivalences. First, rules are mined in known matches given potential property equivalences (see Section 3.1.1). Then, these rules are applied to unmatched instances to find new equivalent ones (see Section 3.1.2).

Here we give an typical example to illustrate this idea. Suppose we have mined three property equivalences from descriptions of known matches as follows,

$$\begin{aligned} \texttt{rdfs:label}\ (p_{11}) &\approx (p_{21})\ \texttt{gs:hasCommonName} \\ \texttt{foaf:name}\ (p_{12}) &\approx (p_{22})\ \texttt{gs:hasCanonicalName} \\ \texttt{dbpedia:phylum}\ (p_{13}) &\approx (p_{23})\ \texttt{gs:inPhylum}. \end{aligned}$$

In practice, a property may be equal to another one. For example,

The property `dbpedia:phylum` defined by dbpedia.org and the property `gs:inPhylum` defined by geospecies.org have the same intensional meaning: their values both refer to the same taxonomic rank in biology when given a certain instance (species). We say such properties are connotatively equivalent.

Based on these property equivalences and known instance equivalences, we can mine a rule like this: if two instances e_1 and e_2 satisfy

$$\bigwedge_{i=1}^{3} \left(p_{1i}(e_1, o_1) \wedge p_{2i}(e_2, o_2) \wedge o_1 \simeq o_2\right) \tag{1}$$

($p(e, o)$ is the function expression of $\langle e, p, o\rangle$ and $o_1 \simeq o_2$ means both o_1 and o_2 refer to the same instance or literal), then we have $(e_1, e_2) \in \sim_{\mathcal{I}}$.

Meanwhile, our data set contains two instances from DBpedia and GeoSpecies respectively,

dbpedia:ep	rdfs:label	"X"
	foaf:name	"Y"
	dbpedia:phylum	⟨Z⟩
gs:ep	gs:hasCommonName	"X"
	gs:hasCanonicalName	"Y"
	gs:inPhylum	⟨Z⟩.

So the rule above can be applied to them and can deduce that these two instances are equivalent. If the confidence of the correspondence representing this equivalence is high enough, this correspondence will be seen as a known match in the next iteration.

For convenience, we use $\langle Z\rangle$ in our example to denote a shared instance linking to both `dbpedia:ep` and `gs:ep` (i.e. a known match). But we need a formal expression of comparing property-value pairs.

Definition 3. **Equivalence of Property-Value Pairs**, denoted by $\sim_{\mathcal{P}}$, is an equivalence relation. Given two connotatively equivalent properties (p_1 and p_2) and two values (o_1

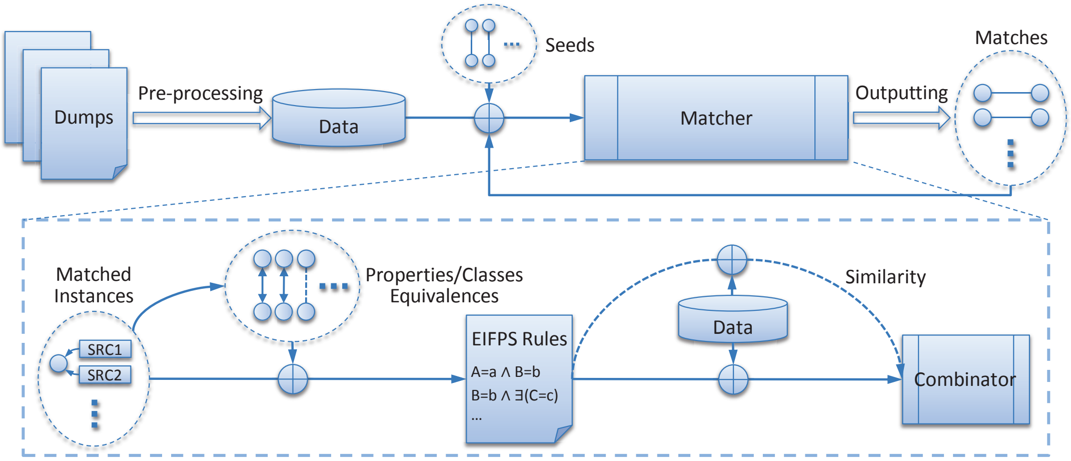


Figure 1: Overview of the framework

and o_2), the two property-value pairs ($\langle p_1, o_1\rangle$ and $\langle p_2, o_2\rangle$) are equivalent iff. $(o_1, o_2) \in\sim_{\mathcal{I}}$ (values are both instances) or $o_1 = o_2$ (values are both literals).

Now we extend the equivalence relation to property-value pair sets. Given an instance e and a set of properties (property suite) P, we have a property-value pair set $\text{PV}_{e,P} = \{\langle p, o\rangle | p \in P, \langle e, p, o\rangle \in G\}$.

Definition 4. **Equivalence of Property-Value Pair Sets**, denoted by $\sim_{\mathcal{S}}$, is an equivalence relation. Given two instances (e_1 and e_2) and a set of equivalent property pairs ($\langle P_1, P_2\rangle$), two property-value pair sets (PV_{e_1,P_1} and PV_{e_2,P_2}) are equivalent iff. there exists a bijection f from PV_{e_1,P_1} to PV_{e_2,P_2} and $f \in\sim_{\mathcal{P}}$.

Some properties have great discriminating power so that their values can uniquely identify corresponding instances. They are defined by OWL as inverse functional properties (IFP). However, using a single property to uniquely identify instances is quite difficult in most cases. Instead, as the above example shows, using combinations of properties can be an alternative choice. We call such combinations inverse functional property suites (IFPS).

Definition 5. A set of equivalent property pairs eps is an **Inverse Functional Property Suite** iff. it satisfies: if $(\text{PV}_{e_1,\text{eps}_1}, \text{PV}_{e_2,\text{eps}_2}) \in\sim_{\mathcal{S}}$, then $(e_1, e_2) \in\sim_{\mathcal{I}}$.

Once the IFPS is determined, we can make rules similar to Rule (1) in the above example. We call such rules IFPS rules.

Definition 6. An **IFPS Rule** is based on an Inverse Functional Property Suite eps. For all property pairs $\langle p_{i1}, p_{i2}\rangle$ in eps, an IFPS rule has the form:

$$\forall e_1\forall e_2.\Bigg(\bigwedge_{i=1}^{|\text{eps}|}\Big(\forall o_1\forall o_2.(p_{i1}(e_1,o_1)\wedge p_{i2}(e_2,o_2)) \wedge \sim_{\mathcal{P}} (\langle p_{i1},o_1\rangle,\langle p_{i2},o_2\rangle))\Big) \rightarrow\sim_{\mathcal{I}} (e_1,e_2)\Bigg). \tag{2}$$

Note that the universal quantification $\forall o_1\forall o_2.$ in Rule (2) indicates that if a property has more than one value, all of its values should have equivalents. Practically, this restriction is too strict in most cases, so we change it to the existential quantification. Therefore, we have the concept of an Extended IFPS (EIFPS) rule.

Definition 7. For all property pairs $\langle p_{i1}, p_{i2}\rangle$ in eps, an **EIFPS Rule** has the form:

$$\forall e_1\forall e_2.\Bigg(\bigwedge_{i=1}^{|\text{eps}|}\Big(\exists o_1\exists o_2.(p_{i1}(e_1,o_1)\wedge p_{i2}(e_2,o_2)) \wedge \sim_{\mathcal{P}} (\langle p_{i1},o_1\rangle,\langle p_{i2},o_2\rangle))\Big) \rightarrow\sim_{\mathcal{I}} (e_1,e_2)\Bigg). \tag{3}$$

The original universal quantification can be realized by calculating the proportion of overlapping values of the given property for two instances and we will discuss this method in Section 4.

3. THE SEMI-SUPERVISED INSTANCE MATCHING APPROACH

The procedure of discovering rules as well as matches in an iterative way follows the Expectation-Maximization algorithm.

3.1 The Expectation-Maximization Algorithm

The EM algorithm is an iterative procedure to estimate missing data by estimating the model parameter(s) for which the observed data are the most likely.

The EM iteration alternates between performing an expectation step (E-step), and a maximization step (M-step). The E-step estimates the missing data (i.e. correspondences) using the observed data and the current estimate for the parameters. In our scenario, we regard the EIFPS Rules, denoted by $\boldsymbol{\theta}$, as the model parameters. The M-step computes parameters maximizing the likelihood function as the data estimated in E-step are used in lieu of the actual missing data.

Algorithm 1: Wrapper algorithm

```
input : A set of triples, a set of known matches
        (seeds), and a threshold for selecting
        correspondences with high confidence.
output: A set of newly discovered matches and a set of
        Inverse Functional Property Suites (IFPSs).
1 candidates,matches ← $\emptyset$;
2 repeat
3     IFPSs ← MineIFPSs(triples,seeds,candidates);
4     candidates ← GetCorrespondences(triples,IFPSs);
5     if candidates is large enough then
6         new ← $\{c | c \in$ candidates$, c.\text{conf} >$ threshold$\}$;
7         seeds ← seeds $\cup$ new;
8         matches ← matches $\cup$ new;
9 until new = $\emptyset$ (new $\neq$ null);
```

Before giving the definition of our specific likelihood function, we should introduce the concept of a Correspondence Graph (CG).

Definition 8. A **Correspondence Graph** is an undirected graph, in which a vertex represents an instance and an edge links two vertices if their corresponding instances are supposed to be equivalent. If the confidence values of correspondences are considered as weights of edges, a set of correspondences can map to a unique weighted CG.

The likelihood function is considered to be a function of $\boldsymbol{\theta}$ given the CG $\boldsymbol{M}$. So we define it as,

$$L(\boldsymbol{\theta}; \boldsymbol{M}) = \Pr(\boldsymbol{M}|\boldsymbol{\theta}). \tag{4}$$

Naturally, the probability of $\boldsymbol{M}$ is reflected in the proximity of $\boldsymbol{M}$ and the CG built by real matches. For a set of given correspondences, the proximity is reflected in two aspects: correctness and completeness. Precision and recall are two widely used measures to evaluate these characteristics [10]. However, without complete reference matches, it is difficult to evaluate either of them. Hence we propose an alternative measurement, that is optimizing the precision takes priority (ensuring the precision is higher than a given threshold) and obtaining all potential matches on the premise of that precision value. So the Equation (4) can be continued as,

$$L(\boldsymbol{\theta}; \boldsymbol{M}) \approx \text{Precision}(\boldsymbol{M}|\boldsymbol{\theta}). \tag{5}$$

Given an CG, we can estimate an approximate precision value by evaluating the divergence of this graph:

$$\begin{aligned} \text{Precision}_{approx}(M) =& \text{Divergence}(M) \\ =& \frac{|\text{ConnectedComponent}(M)|}{|\text{Edge}(M)|}. \end{aligned} \tag{6}$$

Considering an CG, M, with which instances (vertices) come from two data sources and assuming that no equivalent instances exist in a single data source, we can infer that an instance is equivalent to at most one other from the other data source and thus Divergence(M) should be 1. Incorrect matches in M may result in a vertex connecting to more than one other vertices, which is contrary to the assumption, and decrease the divergence of M. We should pay attention to the statistical significance of the estimate: the approximate precision is meaningful only if the size of M is large enough.

For each parameter θ (i.e. an EIFPS rule), we can construct an M according to Rule (3), and put it into the final united $\boldsymbol{M}$. To maximize Precision($\boldsymbol{M}|\boldsymbol{\theta}$), setting a high threshold for each Precision($M|\theta$) is our chosen solution.

Once the precision (threshold) is determined, the number of Ms, as well as the EIFPS rules are also determined. Note that this does not mean that the likelihood will not change, because $\boldsymbol{\theta}$ changes in each iteration due to the different inputs (line 3 in Algorithm 1).

Algorithm 1 demonstrates the wrapper that implements the EM approach. The first several iterations when the set of candidates is not large enough, as well as the following pass of line 3 can be seen as the startup of the EM algorithm, where the EIFPS rules are initialized. Then, E-step (line 4) and M-step (line 3) execute alternately until the termination condition (new= $\emptyset$) is satisfied. Note that the conditional statement in line 5 implements the control over the size of $\boldsymbol{M}$. In practice, this condition is satisfied on every iteration except the first.

Here we introduce the procedures of the M-step and E-step in detail.

3.1.1 M-step: Mining IFPSuites

The M-step mines EIFPS rules maximizing the likelihood function (Equation 5) given the candidates and seeds obtained in the E-step. See Algorithm 2.

Algorithm 2: Mining IFPSuites (MineIFPSs)

```
input : A set of triples, a set of known matches
        (seeds), and candidate matches (candidates).
output: A set of Inverse Functional Property Suites
        (IFPSs) and the modified triples
foreach $c \in$ seeds do
    Replace every $\langle s, p, c.e_2 \rangle$ with $\langle s, p, c.e_1 \rangle$ in triples;
trans ← $\emptyset$;
foreach $c \in$ seeds $\cup$ candidates do
    trans ← trans $\cup \{\{\langle p, o \rangle | \langle s, p, o \rangle \in$ triples$, s \in \{c.e_1, c.e_2\}\}\}$;
    // trans also takes cs' information
pEquivalents ←MineAssociationRules(trans);
suite ← $\emptyset$;
foreach $t \in$ trans do
    foreach $P \subseteq$ pEquivalents do
        if $(\text{PV}_{t.c.e_1,P_1}, \text{PV}_{t.c.e_2,P_2}) \in \sim_S$ then
            Assign a hash code (hc) computed using
            $\text{PV}_{t.c.e_1,P_1}$ and $\text{PV}_{t.c.e_2,P_2}$ to $t.c$;
            suite ← suite $\cup P$;
            Put $t.c$ into $P.M$;
            if P.M already contains a match with the
            same hash code of hc then
                Merge a vertex of this match with a
                vertex of $t.c$;
IFPSs ← $\emptyset$;
foreach $P \in$ suite do
    if Divergence($P.M$) > threshold and
    Divergence($P.M$) > all Divergence($p.M$) where
    $p \subset P$ and $p \in$ suite then
        IFPSs ← IFPSs $\cup P$;
```

The newly discovered equivalences (matches with high confidence) from the previous step can be exploited to boost the discovery of potential matches. To make use of them, we should first give equivalent instances a unified identity. The operations in line 2 replace e_2s with e_1s.

For each pair of matched instances, their property-value pairs are merged to form a transaction. The procedure of mining association rules in line 6 only uses the properties in transactions. This method, proposed by Völker *et al.* [31], acquires property subsumption axioms by mining binary association rules. According to Definition 5, property pairs found here should be connotatively equivalent property pairs. The equivalence relation can be denoted by subsumption relations on both sides. Parundekar *et.al* [26] used this idea to mine relations between classes. They restricted the confidence values (called P and R) of subsumptions to be larger that 0.9. In practice, some weakly related properties that have overlaps in semantics are also considered to be involved in building rules. For example, there is no inclusion relation between `dbpedia:synonym` and `gs:hasCanonicalName`, but there exist certain species in DBpedia that have canonical names as their synonyms. Such information should not be ignored. So we set a relatively low threshold, 0.1, to output more property pairs with potential relations (weak equivalences).

CGs are constructed for all EIFPS rules. An ideal CG contains only connected components that have one edge each. Different instances connected in one component result from sharing the same property-value set on a given property suite (line 15). In order to save storage space, property-value sets can be hashed. In our experiments, hash collisions can be neglected if the hash functions is well designed.

The maximization of Precision($\boldsymbol{M}|\boldsymbol{\theta}$) is realized in line 18 by setting a threshold (0.95 in our experiments). Note that the support of an EIFPS rule is also considered although it is not reflected in the algorithm. Sometimes, additional restrictions in a rule are not necessary. For example, we may have $S_1 \rightarrow A$ with confidence c and also have $S_2 \rightarrow A$ with the same confidence c where $S_2 \supset S_1$. Naturally, additional restrictions in S_2 are redundant for deducing A, so we neglect such rules.

3.1.2 E-step: Getting Candidate Correspondences

The E-step estimates the correspondences (i.e. candidates) using the current estimate for the EIFPS Rules. See Algorithm 3.

In line 3, each EIFPS rule is used to find all corresponding initial correspondences. Every IFPS brings a confidence value with it. This value is calculated in the M-step (line 18 in Algorithm 2) and it indicates the precision of the CG obtained using that rule in the previous EM iteration.

A correspondence may be derived from several EIFPS rules. As mentioned above, different rules have different confidence values and thus the correspondence will inherit these confidence values. We have to combine them and assign the combined confidence to that correspondence (line 5).

Algorithm 3: Getting candidate correspondences (`GetCorrespondences`)

```
input : A set of triples and a set of Inverse Functional
        Property Suites (IFPSs).
output: Candidate matches (candidates).
1 iniCorrespondences ← $\emptyset$;
2 foreach eps ∈ IFPSs do
3     iniCorrespondences ← iniCorrespondences ∪
      $\{\langle e_1, e_2, \text{eps.conf}\rangle | \{e_1, e_2\} \times \text{PV}_{\{e_1,e_2\},\text{eps}} \in$
      triples, $\langle e_1, e_2\rangle$ satisfies Rule (3) given eps};
4 foreach unique $\langle e_1, e_2\rangle$ within iniCorrespondences do
5     Calculate final confidence cc by combining
      $\{\text{conf} | \langle e_1, e_2, \text{conf}\rangle \in$ iniCorrespondences} using
      Dempster's rule (7);
6     candidates ← candidates ∪ $\{\langle e_1, e_2, \text{cc}\rangle\}$;
```

Choosing the maximum is a conservative method. But intuitively, rules can been regarded as evidence and the more evidence we have, the more likely true the conclusion is. Here we consider two ways to combine such evidence. One is using basic probability theory. That is regarding a confidence value of a correspondence as the probability of this correspondence being true, the probability that it is false is 1-confidence. So we have the combination rule,

$$\text{conf}_1 \oplus \text{conf}_2 = 1 - (1 - \text{conf}_1)(1 - \text{conf}_2).$$

The other method is to use Dempster-Shafer theory, a mathematical theory of evidence [29]. Dempster's rule of combination is a generalization of a special case of Bayes' theorem. In the case where only two conditions (true and false) exist, the rule has the form,

$$\text{conf}_1 \oplus \text{conf}_2 = \frac{\text{conf}_1 \cdot \text{conf}_2}{1 - \text{conf}_1 - \text{conf}_2 + 2 \cdot \text{conf}_1 \cdot \text{conf}_2}. \quad (7)$$

Although different pieces of evidence cannot be guaranteed to be independent, which is required by both of the last two methods, these rules have been proven effective in our experiments. Here we give an example of matching instances from DBpedia and GeoNames[4].

Consider a test case consisting of an IFPS eps and two subset of eps (namely eps_1 and eps_2 and $\text{eps}_1 \cup \text{eps}_2 = \text{eps}$). Confidence values of eps_1 and eps_2 are combined using the three combination rules mentioned above. The absolute error equals the combined value minus the reference value which is the confidence of eps. In this example, we have a total of 248 test cases, and the results are shown in Figure 2. Translucent gray boxes representing certain absolute errors are plotted in the graph, so the more often an error value appears, the darker its corresponding position's color is. We can see that most error values of **Probability** and **Dempster** are near 0.00. In fact, the actual mean error values also indicate that these two rules are better than **Maximum**. Results of using **Probability** and **Dempster** are very close if two confidence values to be combined are relatively high, above 0.95 (a commonly used threshold in our experiments) for example. We choose Dempster's rule of combination finally empirically.

3.2 Convergence

Since the output set of matches is expanded in each iteration, its size monotonically increases. Meanwhile, the number of instances is finite so the number of equivalent instances (including incorrect ones) is also finite. Thus, the termination condition can always be satisfied when no matches are found in a certain iteration.

[4] Detailed information will be introduced in Section 5.

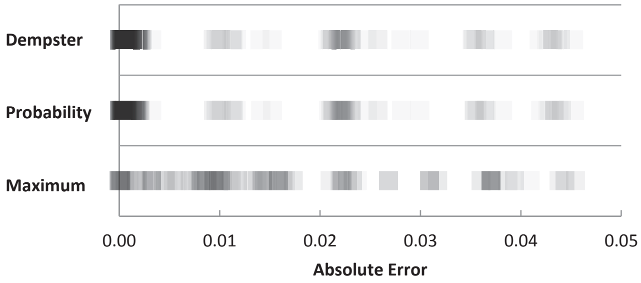


Figure 2: Comparison on absolute error among three combination rules (**Maximum**, **Probability** and **Dempster**)

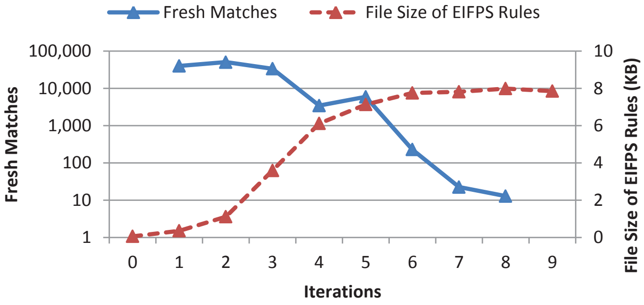


Figure 3: The trends of fresh matches and EIFPS rules

We give a practical example[5] in Figure 3. With the procedure going, mined EIFPS rules are reaching a plateau and the set of fresh matches (newly found matches in each iteration) gradually becomes smaller, becoming empty in the ninth iteration. Note that the curve of fresh matches rebounds in the fifth iteration probably because some important instance equivalences are found in the fourth iteration and given unified identities, which makes many instances share these values from then on.

3.3 Complexity and Parallel Implementation

Within each iteration, the phases can be classified into four categories:

Data pre-processing. The first seven lines of Algorithm 2 fall into this category. The complexity is linear and implementing them in parallel is quite straightforward.

Data post-processing. Getting final output from initial candidates/rules in both the E-step and M-step fall into this category. Sorting the initial data is a rational choice. The Map/Reduce framework [7] is suitable for solving such problems.

Association rule mining. Mining property relations in Algorithm 2 can directly use mature techniques, such as FP-Growth [11] and its parallel implementation [27]. We just implement the simple Apriori algorithm [1] using Map/Reduce and this method is efficient enough.

Join of data and rules. The other phases in both the M-step and the E-step fall into this category. In the M-step, we enumerate all possible rules (K) for each known match (N) to select suitable rules, while in the E-step we enumerate all determined rules (K) for each instance (N) to find matched instances. So we have the time complexity: $O(K \times N)$. Note that if a property has more than one value, the corresponding rule should be joined repeatedly according to the existential quantification in Rule 3, which will expand K dramatically in certain cases. Fortunately, such iterations can be processed in parallel because the joined results are independent.

The outermost loop in Algorithm 1 should be centrally controlled. The Map/Reduce framework as well as external memory algorithms are adequate for this task because the number of iterations is relatively small in practice (see Section 5.4).

4. EXTENSIONS

What we discussed in the last section is a general core algorithm. In order to improve its performance in certain scenarios, we propose several (potential) ways to extend our approach.

4.1 Similarity Embedded EIFPS Rules

The property-value pair equivalence measurement currently used is based on Definition 3. Literals are matched only when their characters are exactly the same. This method is strict, especially for numeric values. For example, the latitude and longitude of a location are usually decimal numbers, and they cannot be sensibly compared without taking account of allowable errors.

If an IFPS contains properties whose values are numeric, the instances can be indexed by other non-numeric values. Instances with the same index are further compared by their numeric values. Under such circumstances, confidence values of candidate correspondence should be multiplied by the similarities of numeric values. The idea of indexing instances before further comparison is used by several instance matching systems such as SILK [32] and Zhishi.links [23]. Note that we do not consider the case that an IFPS contains only properties with numeric values. Some methods have been proposed to solve this problem [17].

Besides numeric values, many other data types (e.g. Date) can also be compared. Even the similarity of two lists of values can be calculated in this way. As we mentioned in Section 2, universal quantification can be realized by calculating the similarity of lists of values for two instances. Here the similarity is defined as the proportion of overlapping values in two lists.

In fact, we have implemented the similarity embedded EIFPS rules in discovering matches between DBpedia and GeoNames. We will discuss it in detail in Section 5.

4.2 Considering Class Restrictions

One difficult issue of instance matching is disambiguation. For example, we can hardly classify “Harry Potter” to a series of novels, a film series or a character. But if the type (class restriction) of “Harry Potter” is provided, everything becomes easier.

So we can regard the class restriction as one kind of special property and add it to the EIFPS rules. To this end, we should modify the procedure of data pre-processing slightly.

In practice, we find that adding class restrictions has little effect on the final results because several properties have connotative relations with class restrictions. For example, an instance having the property “release date” indicates that it is a work that has a release date like “Film”. However, this information may be of great importance for some data sources.

[5] This example comes from a task of matching DBpedia to GeoNames.

4.3 Online Algorithm

The approach we implemented is an offline one. However, since the EIFPS rules are very intuitive, they can be rewritten as SPARQL queries[6]. Reusing the example in Section 2, if we have instance `dbpedia:ep` and want to find an equivalent instance in GeoSpecies, we can construct a SPARQL query based on Rule 1,

```
SELECT ?species
WHERE
  { ?species gs:hasCommonName    "X" ;
             gs:hasCanonicalName "Y" ;
             gs:inPhylum         <Z> . }
```

The execution time of a completely online algorithm may be unacceptable, but it is meaningful to apply it to continually growing data sources when relatively high-quality EIFPS rules have been obtained offline.

5. EXPERIMENTS AND DISCUSSIONS

Our approach is implemented in Java. All procedures are performed with the Hadoop Map/Reduce framework. All the tests were carried out on a Hadoop cluster which contains 40 nodes. Each node is a PC (Intel Core 2 Quad 2.66GHz CPU, 2GB RAM) can run 3 Maps + 3 Reduces simultaneously. This is a shared cluster and we occupy 50 slots in most cases.

5.1 Data Sets and Evaluation Methodology

In our experiments, we considered a general data source DBpedia[7], and three domain-specific data sources: GeoNames[8], LinkedMDB[9] and GeoSpecies[10]. They are typical representative data sources in Linked Open Data. The tasks in our experiments is to discover matches between DBpedia and the other three data sources.

DBpedia is a hub data source in LOD. It structures Wikipedia knowledge and make this structured information available on the Web [5]. The DBpedia community occasionally release data dumps and the latest version (3.7) is used in our experiments. From numerous datasets, we chose labels, external links, geo-coordinates and refined ontology information including instance types and mapping-based properties. Note that we removed instances appeared as objects in "Redirect" and "Disambiguation Links" datasets, but used their labels as aliases.

GeoNames is a geographical database that integrates tens of data sources and allows users to improve and correct data. The dump version we used was 2012-03-03. GeoNames contains links to Wikipedia pages (`gn:wikipediaArticle`). These links were established manually or by heuristic rules. We converted such links to reference matches.

LinkedMDB is an open semantic web database dedicated to movie-related information [12]. The dump version we used was 2010-01-29. In our experiments, we only considered instances whose type was film since they are the core of this data source and have sufficient descriptive information. The reference matches are obtained from `owl:sameAs` links. These links were found by ODDLinker, which employs state-of-the-art similarity join techniques.

Table 1: Statistics of Data Sets

Data Sets	Instances	Triples	References
DBpedia	4,071,600	36,565,123	-
GeoNames	8,147,136	86,409,073	317,433
LinkedMDB (Film)	97,471	1,269,572	16,447
GeoSpecies	20,939	997,753	11,490

GeoSpecies was started to help tie together disparate data about species. The dump version we used was 2010-04-11. Note that some properties which can be seen as cheating information such as `skos:closeMatch` were removed. GeoSpecies also contains links to Wikipedia pages (`gs:has WikipediaArticle`) and reference matches are converted from them. The mechanism of finding these links is not described by the project owner.

Statistics of these datasets are illustrated in Table 1.

We measured the precisions, newly found matches, and running times (number of iterations) on different tasks. Before showing the evaluation results in following sections, the evaluation methodology and some basic settings should be discussed.

Judgements and repeats. Since in many cases, reference matches are far from complete, we have to judge whether a match is correct or not manually. For each matching task, we randomly selected seeds and ran the program three times to measure the average performance. After each execution, 100 randomly selected matches (1000 for DBpedia-GeoSpecies tasks) were judged.

Seeds. Since our approach is based on the idea of semi-supervised learning, the choice of labeled data (seeds) is an issue of concern. Here we selected different proportions of reference matches as seeds to observe the impact of the number of seeds on algorithm performance. In situations where no references exist, seeds can be found by using some strict heuristic rules or inferred based on the OWL semantics [15].

Parameters. Algorithm 1 needs a threshold (line 6) for selecting outputs. In practice, we set it to 0.98 unless otherwise specified. Meanwhile, all supports for association rules and EIFPS rules in our approach were set to a fixed value, 10. Not many parameters are involved in our approach, and they can be tuned easily using a small amount of training data. We reused the same settings for all data sources, which proved the good stability of our approach in a way [24].

5.2 Precision

The precision is the ratio of correct matches to total matches found. We estimate this value by sampling a certain number of output matches as mentioned in the last section. The experimental results are shown in Figure 4.

The X-axis indicates the number of seeds and the proportions of selected seeds in complete reference matches. We conceal the absolute seed numbers in the top figure because two matching task series share the same plot.

When DBpedia was matched to GeoSpecies and LinkedMDB, the precision values of newly found matches were relatively high at any number of initial seeds (Figure 4a). In the case of matching DBpedia to GeoNames, the precision decreased slightly (#0.98 in Figure 4b). The precision varied

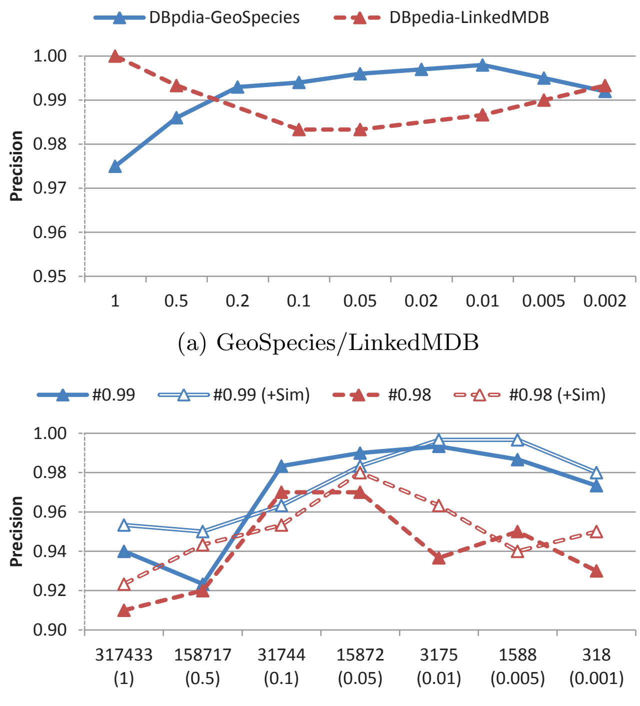
[6] `http://www.w3.org/TR/rdf-sparql-query/`

[7] `http://dbpedia.org/`

[8] `http://www.geonames.org/ontology/`

[9] `http://linkedmdb.org/`

[10] `http://lod.geospecies.org/`



(a) GeoSpecies/LinkedMDB

(b) DBpedia-GeoNames

Figure 4: Precisions versus the proportions of selected seeds in complete reference matches

partly because differences existed in quality of data sources. Recall that we assume no equivalent instances exist in one single source, however, we have found that several distinct instances may refer to the same place. This naturally affected the estimation of precision in the M-step. Predictably, the precisions were higher when we changed the threshold of outputting matches to 0.99 (#0.99 in Figure 4b).

We also tested an extended version of our approach using similarity embedded EIFPS rules (see Section 4.1). This extended version only applied to matching DBpedia to GeoNames because some important numeric values such as latitudes and longitudes were involved. The results show that considering similarities had almost no effect on the precision (#0.99(+Sim) and #0.98(+Sim) in Figure 4b).

Note that more seeds did not necessarily lead to higher precision. If things of a certain domain need more property-value information to be uniquely identified, then it is easier to have trouble with over-fitting. Determining a proper support for EIFPS rules will be a solution to this problem.

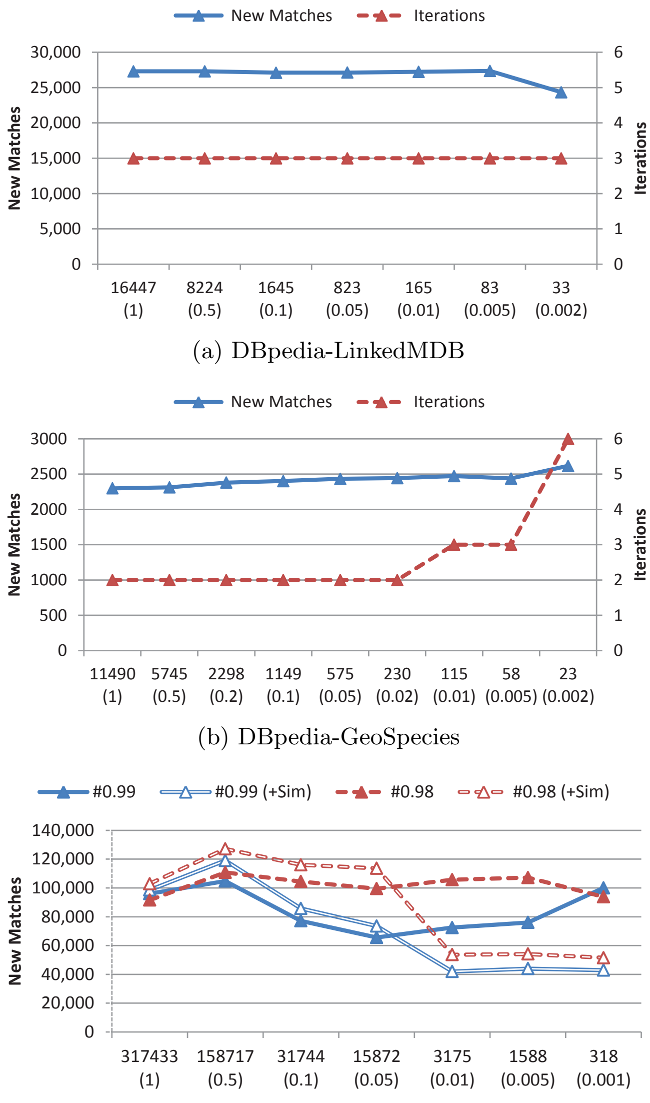
5.3 Newly Found Matches

The newly found matches (new matches for short) are defined as the correct matches apart from ones contained in references. So we calculate their number by

$$|\text{OutputMatches}| \times \text{Precision} - |\text{TruePositive}|$$

where TruePositive means correctly proposed matches with regard to reference matches. The experimental results are shown in Figure 5. Note that the secondary vertical axes on the right side and corresponding curves in first two plots are prepared for the next section (to save space).

The number of new matches is relatively constant for any number of initial seeds in the cases of matching DBpedia to LinkedMDB, GeoSpecies and GeoNames without using similarity embedded EIFPS rules.



(a) DBpedia-LinkedMDB

(b) DBpedia-GeoSpecies

(c) DBpedia-GeoNames

Figure 5: Newly found matches versus the proportions of selected seeds in complete reference matches

However, in the case of considering similarity in DBpedia-GeoNames tasks, fewer initial seeds tend to give fewer newly found matches (#0.9X(+Sim) in Figure 5c). This is understandable because comparing numeric values needs prior indexing by non-numeric values, so an EIFPS rule containing properties with both numeric and non-numeric values needs more known matches to support it.

Note that we did not measure recalls and only considered matches out of the set of references so far, because even the reference matches were far from complete. We show the match space constituted by reference matches and newly found matches in Figure 6 to illustrate how many matches can be found by our approach.

The whole match space is divided into three parts: New Matches, Overlaps and Missing Matches. The last two parts

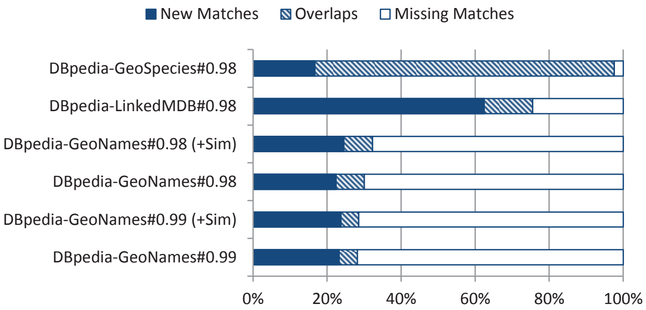


Figure 6: The match space constituted by reference matches and newly found matches

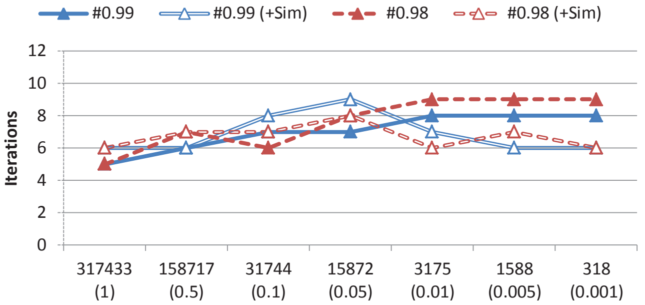


Figure 7: Number of iterations versus the proportions of selected seeds in complete reference matches (DBpedia-GeoNames)

(streaky and white blocks in Figure 6) together constitute the match space of references. Since we repeated our algorithm with different numbers of seeds, we choose the median number of recalls to compute the proportions of "Overlaps". Naturally, remaining matches that were not proposed by our approach fell into "Missing Matches". The dark blocks that represent "New Matches" are using the results under the condition that all references are involved as seeds.

From the results we can see that our approach is qualified for matching DBpedia to GeoSpecies and it also performs better than ODDLinker in matching DBpedia to LinkedMDB. While in the DBpedia-GeoNames matching task, our automatic method cannot discover as many instance equivalences as manual methods, but it is complementary to them.

5.4 Time

The numbers of iterations we need to accomplish the matching tasks are shown in Figure 5a, 5b and 7. We chose the mode of three results since each task was repeated three times. Not too many iterations were needed as the results tell us.

Generally speaking, the number of iterations did not decrease when fewer seeds were provided. Exceptions existed similar to the situations when we talked about newly found matches in the case of considering similarity in DBpedia-GeoNames tasks. Recall that fewer new instance equivalences are found because similarity embedded EIFPS rules have insufficient known matches to support them. Thus, insufficient new matches have difficulty driving the next iteration.

Besides the number of iterations, we are also interested in the time cost of each iteration. We sample some typical running times for a single iteration in Figure 8. The most time-consuming phases are "data pre-processing" and "join of data and rules" in the E-step (i.e. GetIniCorrespondences) which have been discussed in Section 3.3.

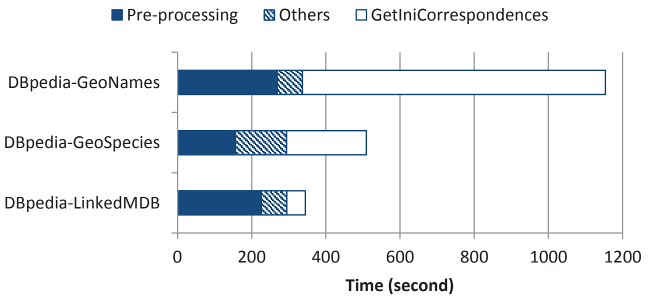


Figure 8: Typical time cost for one iteration in three matching tasks

6. RELATED WORK

A related problem called *record linkage* or *duplicate detection* has been investigated for decades in the realm of databases [9, 20]. It focuses on dealing with the matching problems w.r.t. entity-relationship model. Another highly related problem is *ontology alignment* [10]. Relevant methods can be used for connecting LOD sources, but they focus on schema-level matching and rely on richly structured ontologies.

Our proposed approach belongs to the category of instance matching in the context of LOD. It has attracted much attention and extensively studied by the research community. Some researchers focused on matching instances for specific domains, such as people (FOAF) [30] and music [28]. Some dataset creators designed heuristic methods to discover instance equivalents between their data (e.g. LinkedGeoData [3] and LinkedMDB [12]) and DBpedia. Some reusable link discovery frameworks are favored due to the fact that they are domain-independent and customizable (users specify link specifications). SILK [32], LIMES [21] and the framework proposed by Hassanzadeh *et al.* [13] are typical examples.

Since the similarity of property-value pairs of different instances can be used for finding similar instances, various kinds of similarity metrics are employed by many general-purpose (semi-)automatic methods. For example, Castano *et al.* [6] designed several domain-dependent similarities, Albertoni *et al.* [2] combined different kinds of similarities, Li *et al.* [19] used multiple strategies, Ngomo *et al.* [22] learned weights of pairwise property-values, and there are many others [16, 18, 25].

Some other researchers address the instance matching problem with inference based on OWL semantics. A simple survey can be found in [15]. Specifically, because inverse functional properties along with their values have great discriminating power to identify an instance, IFPs play an important role in inferring instance equivalents. Based on this idea, Hogan *et al.* [14] measured the static discriminating power values of property-value pairs globally and then used these values to compute how likely two instances are matched. Inspired by this work, Hu *et al.* [15] proposed a self-training algorithm to quantify how discriminating the property-value pairs were dynamically and choose the most discriminating property-value pairs to output current matchable instances iteratively. This method further considered the matchability between properties and frequent property combinations. The idea of these two approaches is similar to ours, but we choose a quite different and more flexible way to implement it. Unlike [15], we did not pop the most discriminating property-value pairs but the most matchable instances in each iteration, and we did not face the problem of determining the termination condition. At the same time, we used newly found matches in each iteration as new matchable values, while [14] and [15] did not since they focused more on literal values. Furthermore, the cardinality of our EIFPS is unlimited but [14] and [15] compared at most two properties on each side.

7. CONCLUSION

In this paper, we proposed a general-purpose approach to automatically mine dataset-specific matching rules and this approach is based on the EM algorithm. We introduced a graph-based metric to estimate likelihood (precision) and Dempster's rule to combine confidence values. Our proposed approach discovers new matches by iteratively refining matching rules and integrating known equivalent instances. The number of iterations to accomplish the matching task is relatively small, and the whole process can be implemented parallelly.

Moreover, we discussed some extensions to our approach in order to fit the different requirements of various practical applications. We carried out experiments on several real-world datasets. The results demonstrated the correctness of matches discovered by our approach, which achieves the high precision (>0.96 in most cases). We also shown more matches are found than existing references (established by dataset-specific methods), which indicates that our approach is qualified for matching instances in LOD.

Since the proposed approach is language independent, we plan to handle the cross-lingual matching problem and perform more comprehensive experiments on more datasets of different domains in LOD. In addition, we plan to extend the expressivity of our EIFPS rules by considering the negation operator, and also more sophisticated similarity measures will be taken into account.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, 1994.

[2] R. Albertoni and M. D. Martino. Asymmetric and context-dependent semantic similarity among ontology instances. *Journal on Data Semantics*, 10:1–30, 2008.

[3] S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData: Adding a Spatial Dimension to the Web of Data. In *ISWC*, 2009.

[4] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[5] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[6] S. Castano, A. Ferrara, S. Montanelli, and D. Lorusso. Instance matching for ontology population. In *SEBD*, 2008.

[7] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[8] L. Ding, J. Shinavier, Z. Shangguan, and D. L. McGuinness. SameAs Networks and Beyond: Analyzing Deployment Status and Implications of owl:sameAs in Linked Data. In *ISWC*, 2010.

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

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

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, 2000.

[12] O. Hassanzadeh and M. Consens. Linked Movie Data Base. In *I-SEMANTICS*, September 2008.

[13] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. A framework for semantic link discovery over relational data. In *CIKM*, 2009.

[14] A. Hogan, A. Polleres, J. Umbrich, and A. Zimmermann. Some entities are more equal than others: statistical methods to consolidate linked data. In *NeFoRS Workshop*, 2010.

[15] W. Hu, J. Chen, and Y. Qu. A self-training approach for resolving object coreference on the semantic web. In *WWW*, 2011.

[16] A. Isaac, L. van der Meij, S. Schlobach, and S. Wang. An empirical study of instance-based ontology matching. In *ISWC/ASWC*, 2007.

[17] R. Isele, A. Jentzsch, and C. Bizer. Efficient multidimensional blocking for link discovery without losing recall. In *WebDB*, 2011.

[18] Y. R. Jean-Mary, E. P. Shironoshita, and M. R. Kabuka. Ontology matching with semantic verification. *Journal of Web Semantics*, 7(3):235–251, 2009.

[19] J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1218–1232, 2009.

[20] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[21] A.-C. N. Ngomo and S. Auer. LIMES: A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In *IJCAI*, 2011.

[22] A.-C. N. Ngomo, J. Lehmann, S. Auer, and K. Höffner. RAVEN - active learning of link specifications. In *OM Workshop*, 2011.

[23] X. Niu, S. Rong, Y. Zhang, and H. Wang. Zhishi.links results for OAEI 2011. In *OM Workshop*, 2011.

[24] X. Niu, H. Wang, G. Wu, G. Qi, and Y. Yu. Evaluating the stability and credibility of ontology matching methods. In *ESWC*, 2011.

[25] J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt. Leveraging terminological structure for object reconciliation. In *ESWC*, 2010.

[26] R. Parundekar, C. A. Knoblock, and J. L. Ambite. Linking and building ontologies of linked data. In *ISWC*, 2010.

[27] I. Pramudiono and M. Kitsuregawa. Parallel FP-Growth on PC Cluster. In *PAKDD*, 2003.

[28] Y. Raimond, C. Sutton, and M. Sandler. Automatic interlinking of music datasets on the semantic web. In *LDOW Workshop*, 2008.

[29] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[30] J. Sleeman and T. Finin. A Machine Learning Approach to Linking FOAF Instances. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.

[31] J. Völker and M. Niepert. Statistical schema induction. In *ESWC*, 2011.

[32] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *ISWC*, 2009.