SERIMI: Class-based Matching for Instance Matching Across Heterogeneous Datasets

Samur Araujo, Duc Thanh Tran, Arjen P. de Vries and Daniel Schwabe

Abstract—Based on a detailed analysis, we observed that state-of-the-art instance matching approaches do not perform well when used for matching instances *across heterogeneous datasets*. This is because they are built upon *direct matching*, which involves a direct comparison of a source dataset with a target dataset. This is not suitable when the overlap between the datasets is too small. Aiming at this problem, we propose a new paradigm called *class-based matching*. Given a class of instances from the source dataset, called the *class of interest*, and a set of candidate matches retrieved from the target, class-based matching helps to refine the candidates by filtering out those that do not belong to the class of interest. For this refinement, only data in the target is used, i.e., no direct comparison between source and target is involved. Based on extensive experiments using public benchmarks, we show our approach greatly improves the results of state-of-the-art systems especially on hard matching tasks.

Index Terms—Data integration, Class-based matching, Direct matching, Instance matching, Semantic Web.

1 Introduction

A large amount of datasets have been made available on the Web as a result of initiatives such as Linking Open Data. As a general graph-structured data model, RDF[1] is widely used especially for publishing Web datasets. In RDF, an entity, also called an instance, is represented via $\langle subject, predicate, object \rangle$ statements (called *triples*). Predicates and objects capture *attributes* and *values* of an instance, respectively (terms that are used interchangeably here). Fig. 1 shows an example of a RDF graph representing two instances (Belmont in France and Belmont in California) by the four triples: ⟨`db:Belmont_France`, `rdfs:label`, 'Belmont'⟩, ⟨`db:Belmont_France`, `db:country`, `db:France`⟩, ⟨`db:Belmont_California`, `rdfs:label`, 'Belmont'⟩ and ⟨`db:Belmont_California`, `db:country`, `db:Usa`⟩.

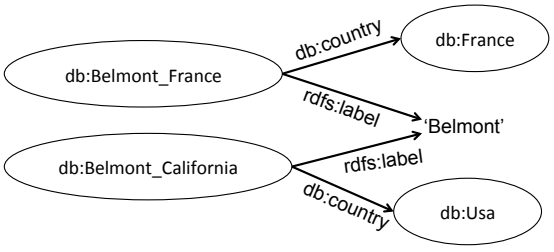

Fig. 1. An Example of a RDF Graph.

Besides RDF, OWL[2] is another standard language for knowledge representation, especially for capturing the "same-as" semantics of instances. Using `owl:sameas`, data providers can make explicit that two distinct URIs actually refer to the same real world entity. The task of establishing these same-as links is known under various names such as entity resolution and *instance matching*.

There are *semantic-driven approaches* that uses specific OWL semantics, such as explicit `owl:sameas` statements, to allow the same-as relations to be inferred via logical reasoning. Complementary to this, there are *data-driven approaches* that derive same-as relations mainly based on attribute values of instances. Namely, two instances are considered the same when they have many attribute values in common. While they vary with respect to the selection and weighting of features, all data-driven approaches are built upon the same paradigm of *direct matching*, namely they directly compare the instance representations. Hence, they produce only high quality results when there is sufficient overlap between instance representations. Overlaps however, might be small in heterogeneous datasets; especially, because the same instance represented in two distinct datasets may not use the same schema.

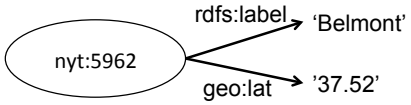

Fig. 2. Another Example of a RDF Graph.

For example, the instances `nyt:5962` (from Fig. 2), `db:Belmont_France` and `db:Belmont_California` (from Fig. 1) shares the same `rdfs:label` value (i.e., the string 'Belmont'). However, `rdfs:label` is the only attribute in which overlaps can be found as the source and target graphs use distinct schemas. This overlap alone is not sufficient to determine whether `nyt:5962` is the same than `db:Belmont_France` (or `db:Belmont_California`). In this scenario of *instance matching across heterogeneous datasets*, direct matching

1. http://www.w3.org/RDF/
2. http://www.w3.org/TR/owl-features/

alone often cannot deliver high quality results.

Contributions. We provide a (1) *detailed analysis* of many datasets and matching tasks investigated in the OAEI 2010 and 2011 [1] instance matching benchmarks. We show that tasks greatly vary in their complexity. There are difficult tasks with a small overlap between datasets that cannot be effectively solved using state-of-the-art direct matching approaches. Aiming at these tasks, we propose to use direct matching in combination with (2) *class-based matching*.

Given a class of instances from the source dataset, called the *class of interest*, and a set of candidate matches retrieved from the target via direct matching, class-based matching helps to refine the candidates by filtering out those that do not map to the class of interest. However, it does not assume that the class semantics is explicitly given so that a direct matching at the class level is possible between the source (e.g. Drugs) and target (e.g. Medication). Instead, class-based matching uses the idea that the correct matches to a set of source instances that form a class, should also form a class, i.e. should be similar among themselves (e.g. share some data attribute / value). Then, by comparing the candidates in a non-trivial way, class-based matching can leverage a subset of target candidates that are more likely to be the positive matches to the source instances. During this process, there is no comparison between source and target but only data from the target is used for matching.

For example, in Table 1, the instances `nyt:2223` and `nyt:5962` from the source dataset belong to the (implicit) class "cities in California". The candidates matches from the target dataset are `db:San_Francisco`, `db:Belmont_France` and `db:Belmont_ California`, as depicted in Fig. 3.

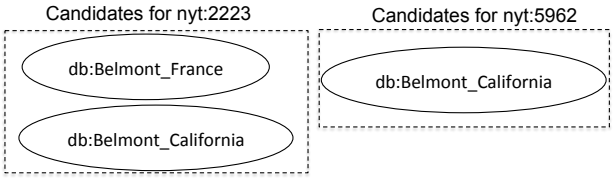

Fig. 3. Candidates for nyt:2223 and nyt:5962.

In this example, class-based matching would select `db:Belmont_California` and `db:San_ Francisco` as correct matches because this subset of instances are the most similar among the candidates. In this example, they have the predicate `db:country` and value `db:Usa` in common, as shown in Table 1.

This matching does not involve any direct comparison between instances in the source and target datasets. Also, it does not assume the class to which `db:Belmont_California` and `db:San_Francisco` belong to is explicitly given, so that it can be directly compared with "cities in California". Instead, a latent instance-based representation is inferred from the three candidates retrieved from the target in this example.

TABLE 1
Instances represented as RDF triples.

Source Dataset		
Subject	Predicate/Attribute	Object/Value
nyt:2223	rdfs:label	'San Francisco'
nyt:5962	rdfs:label	'Belmont'
nyt:5962	geo:lat	'37.52'
nyt:5555	rdfs:label	'San Jose'
nyt:4232	nyt:prefLabel	'Paris'
geo:525233	rdfs:label	'Belmont'
geo:525233	in:country	geo:887884
geo:525233	geo:lat	'37.52'
Target Dataset		
Subject	Predicate/Attribute	Object/Value
db:Usa	owl:sameas	geo:887884
db:Paris	rdfs:label	'Paris'
db:Paris	db:country	db:France
db:Belmont_France	rdfs:label	'Belmont'
db:Belmont_France	db:country	db:France
db:Belmont_California	rdfs:label	'Belmont'
db:Belmont_California	db:country	db:Usa
db:San_Francisco	rdfs:label	'San Francisco'
db:San_Francisco	db:country	db:Usa
db:San_Francisco	db:locatedIn	db:California
db:San_Jose_California	rdfs:label	'San Jose'
db:San_Jose_California	db:locatedIn	db:California
db:San_Jose_Costa_Rica	rdfs:label	'San Jose'
db:San_Jose_Costa_Rica	db:country	db:Costa_Rica

We (3) evaluated this approach called SERIMI using data from OAEI 2010 and 2011, two reference benchmarks in the field, and compared the results with OAEI results as well as those obtained from other state-of-the-art systems. These *extensive experiments* show that SERIMI yields superior results. Class-based matching achieved competitive results when compared to the direct matching; most importantly, it was complementary to it; i.e. achieved good performance when direct matching's performance was bad. Thus, using only a simple combination of the two, our approach could greatly improve the results of existing systems. Considering all tasks in OAEI 2010, it increases average F1 result of the second best by 0.21 (from 0.76 to 0.97). For 2011 data, SERIMI also greatly improves the results of recently proposed approaches (*PARIS* [2] and *SIFI-Hill* [3]). Compared to the best system participated at OAEI 2011, SERIMI achieved the same performance. However, while that system leverages domain knowledge and assumes manually engineered mappings, our approach is generic, completely automatic and does not assume any training data.

Outline. This paper is organized as follows: In Section 2, we introduce some preliminary definitions. In Section 3, we provide an overview of the instance matching process implemented by SERIMI. In Section 4, we discuss class-based matching problem and in Section 5 we propose a solution. In Section 6, we present a measure of complexity and a detailed analysis of matching tasks based on this measure. This section also contains the results of our experiments, where we compare SERIMI with state-of-the-art approaches. In Section 7, we discuss related works. Finally, we conclude in Section 8.

2 Preliminary Definitions

In this section, we present some important definitions.

Data. We use an RDF-based graph-structured model to accommodate different kinds of structured data.

Definition 1 (Data Graph): The data is conceived as a set of graphs **G**. Let U denote the set of Uniform Resource Identifiers (URIs) and L the set of literals, every $G \in \mathbf{G}$ is a set of triples of the form $\langle s, p, o \rangle$, where $s \in U$ (called subject), $p \in U$ (predicate) and $o \in U \cup L$ (object).

Every instance (set of instances) is represented as a set of triples.

Definition 2 (Instance Representation): It is defined as: $IR(G, S) = \{\langle s, p, o \rangle \mid \langle s, p, o \rangle \in G, s \in S\}$, where G is a graph and S a set of instances in G. It yields a set of triples in which $s \in S$ appears as the subject. We denote the set of objects associated with an instance s over the predicate p in G as $O(s, p, G)$, with $O(s, p, G) = \{o | \langle s, p, o \rangle \in G\}$.

The representation of a single instance s is $IR(G, \{s\})$.

Features. First, we introduce the concept of features of a set of instances X, and then, we define a class, an important concept used throughout this paper.

Definition 3 (Features): Let G be a dataset and X be a set of instances in G. The features of X are:

- $A(X) = \{p | (s, p, o) \in IR(G, X) \wedge s \in X\}$,
- $D(X) = \{o | (s, p, o) \in IR(G, X) \wedge s \in X \wedge o \in L\}$,
- $O(X) = \{o | (s, p, o) \in IR(G, X) \wedge s \in X \wedge o \in U\}$,
- $T(X) = \{(p, o) | (s, p, o) \in IR(G, X) \wedge s \in X\}$.

The combined set of features of X is

$$F(X) = A(X) \cup D(X) \cup O(X) \cup T(X)$$

Intuitively, $A(X)$ is the set of predicates, $D(X)$ the set of literals, $O(X)$ the set of URIs, and $T(X)$ is the set of predicate-object pairs that appear in the representation of X.

Considering X ={`db:Belmont_California`}, its features would be: $A(x)$ ={`rdfs:label`, `db:country`}, $D(x)$ = { 'Belmont'}, $O(x)$ ={`db:Usa`}, and $T(x)$ ={(`rdfs: label`, 'Belmont'), (`db:country`, `db:Usa`)}. Consequently, $F(X)$ ={ `rdfs:label`, `db:country`, 'Belmont', `db:Usa` (`rdfs: label`, 'Belmont'), (`db:country`, `db:Usa`)}.

To avoid misconception, we define a class such as:

Definition 4 (Class): Let G be a dataset and X a set of instances in G, X is a class if $\forall x \in X : F(\{x\}) \cap F(X - \{x\}) \neq \emptyset$.

A class should be understood as a set of instances where each instance in this set must share at least one feature in common to any other instance in this set. This definition considers any feature relevant because we cannot assume that instances (in the heterogeneous setting) will share any class-related feature ($A(\bullet)$). For example, in a heterogeneous dataset, two distinct instances in the same class may have different predicates with the same semantics. One may have the predicate `locatedIn` with value "UK" and another the predicate `placedIn` with value "UK". In this case, $A(\bullet)$ features does not define their syntax similarity (our approach only consider the syntactical similarities), but still we can consider they are similar based only on the value "UK". Especially in heterogeneous data, this definition is important because instances of a class may not necessarily share the same predicates but at least their values.

In addition, one purpose of class-based matching is to find a set of instances (among all candidates) that form a concise class, i.e. where the similarity (w.r.t. to $F(\bullet)$) of its constituent instances is maximized. Also, class-based matching tries to maximize the number of instances in the class that matches to the source instances. In this process, only the candidate instances are considered.

3 Overview of the Approach

In this section, we present an overview of SERIMI, our solution for instance matching.

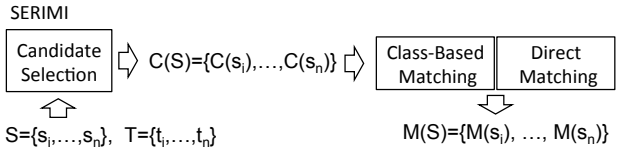

Fig. 4. The instance matching in SERIMI.

The process of instance matching performed by SERIMI is illustrated in Fig. 4. SERIMI focuses on the problem of *instance matching across heterogeneous datasets*. In particular, the inputs are conceived to be partitioned into two datasets, the source S and target T. For every instance in $s \in S$, the goal is to find matching instances $t \in T$, i.e. s and t refer to the same real-world object. This matching is performed in two main steps, candidate selection and match refinement.

Candidate Selection. For each $s \in S$, we firstly perform a low cost candidate selection step to obtain a candidate set $C(s) \subset T$. The union of all candidate sets is denoted as $C(S) = \{C(s) | s \in S\}$. This step reduces the number of comparisons needed to find matches between the source and target, i.e., from a maximum of $|S| \times |T|$ comparisons to $|S| \times |C(s)^{max}|$, where $C(s)^{max}$ is the set with the largest number of candidates among all candidate sets in $C(S)$.

Existing blocking techniques [4], [5], [6] can be used to quickly select candidates. Typically, a predicate (a combination of predicates) that is useful in distinguishing instances is chosen, and its values are used as blocking keys. In this setting of cross-dataset matching, a predicate in the source is chosen (e.g. `rdfs:label`) and its values (e.g. 'San Francisco'.) are used to find target candidate instances that have similar values in their predicates. Using the current example, the candidates matches for S ={`nyt:2223`, `nyt:5962`, `nyt:5555`} would be C(`nyt:2223`) = {`db:San_Francisco`} , C(`nyt:5962`) = {`db:Belmont_California`, `db:Belmont_ France`} and C(`nyt:5555`) = {`db:San_Jose_ California`, `db:San_Jose_Costa_Rica`}, in all sets the candidates

share the same value of the predicate rdfs:label with the source instance.

Particularly in this work, to generate the candidates, we used Boolean queries over the tokens of candidate labels, where the tokens of the source labels where keywords in the queries. Standard data processing was used (e.g. to make tokens lowercase) and stop words were removed (e.g. the, an, a, etc.). These Boolean queries guarantees that candidates are retrieved when at least one token in the source and target values are the same. Although effective, this method can be improved even further, for example, by using techniques of [7].

After the candidates are determined, a more refined matching step is performed to find correct matches, $M(s) \subseteq C(s)$. For this, it is applied state-of-the-art approaches that perform more complex *direct matching*. Usually, instead of a simple blocking key, they use a combination of weighted similarity functions defined over several predicate values [3], [2].

Direct Matching. In direct matching, two given instances s and t are directly compared. They are considered as a match when their similarity, $sim(s,t)$, exceeds a threshold δ. Typically, $sim(s,t)$ is captured by an *instance matching scheme*, which is a weighted combination of similarity functions (Edit Distance, Jaccard, ect.) defined over the predicate values of s and t [3], [2]:

$$sim(s,t) = \sum_{p \in P} w_p \cdot sim(O(s,p,S), O(t,p,T)) > \delta \quad (1)$$

The above scheme assumes that s and t have some common predicates p based on which they can be directly compared (e.g. rdfs:label, db:incountry). In the heterogeneous setting, S and T may exhibit differences in their schemas. Instead of assuming p, more generally, we can define the instance matching problem in this setting based on the notion of comparable predicates $\langle p_s, p_t \rangle$. The predicate p_s is a predicate in S, whose values can be compared with those of p_t, a predicate in T.

For example, the instance nyt:4232 does not share any predicate with the target instances but we can assume that the predicate nyt:prefLabel (p_s) is comparable to the predicate rdfs:label (p_t) because they have a similar range of values. Solutions, which specifically target this setting of cross-datasets matching, employ automatic schema matching or manually find the pairs of comparable predicates [2], [8], [9]. Let P_{st} be the set of all comparable predicates, we can define the instance matching scheme for this setting as follows:

$$sim(s,t) = \sum_{\langle p_s, p_t \rangle \in P_{st}} w_{\langle p_s, p_t \rangle} sim(O(s,p_s,S), O(t,p_t,T)) > \delta \quad (2)$$

The direct matching paradigm has proven to be useful in the homogeneous setting where instances share some common predicates p. In the heterogeneous setting, we showed above that this paradigm would require some pairs of comparable predicates. Since these direct overlaps at the level of predicates (or values) between instances may be too small to perform matching in the heterogeneous setting, we propose class-based matching.

In combination with direct-matching, class-based matching can be applied on top of the candidate selection step. As illustrated in Fig. 4, candidate selection yields a set of candidates $C(S)$, which is then further refined by a module that combines class-based and direct matching to obtain $M(S) = \{M(s_i), ..., M(s_n)\}$, where $\forall i. M(s_i) \subseteq C(s_i) \in C(S)$.

Considering our example, the set $M(S)$ for those candidate sets would be: $M(S)$ ={M(nyt:2223), M(nyt:5962), M(nyt:5555)}.

With:

- M(nyt:2223) = {db:San_Francisco},
- M(nyt:5962) = {db:Belmont_California} and
- M(nyt:5555) = {db:San _Jose_California}.

SERIMI. This paper focuses on class-based matching. Existing works are adopted for the candidate selection and direct matching components of SERIMI. As discussed before, the candidate sets $C(s) \in C(S)$ are determined for each instance $s \in S$ using a predicate value of s as key. The predicate is selected automatically based on the notion of coverage and discriminative power of predicates, also employed by [9]. Then, for direct matching, we use simple schema matching to compute comparable predicates P_{st}. The matching between a source instance s and a target instance t is then performed using values of predicates in P_{st}. As $sim(s,t)$, we use Jaccard similarity. The main difference to existing works lies in the selection of the threshold: for this, we use the same method that we propose for class-based matching.

We observe in the experiments that a simple combination of direct- and class-based matching was sufficient to produce good results. In SERIMI, they are treated as black boxes that yield two scores. SERIMI multiplies and normalizes these scores to obtain a value in [0,1].

4 Class-Based Matching: The Problem

4.1 The problem

Given the source instances S and their candidate sets $C(S)$, class-based matching it the problem of finding the correct matches $M^*(S)$ to S when $s \in S$ and $t \in C(s) \in C(S)$ cannot be directly compared because they do not share any common feature (syntactically or semantically). Departing from this, the only information that is considered to find correct matches to $s \in S$ are the candidate instances $t \in C(s) \in C(S)$.

Particularly, class-based matching is build upon the observation that matching is performed for a class of source instances. That is, all $s \in S$ belongs to a specific class[3]. Our assumption is that if S is a class (they share common features), then the set $M^*(S)$ of correct matches for $s \in S$ should also belong to a class, i.e.,

3. Notice that when the input S captures different classes, it can be partitioned into sets of instances representing specific classes [10].

the target correct matches $M^*(S)$ should share some common features among themselves.

Using this assumption, the challenge of finding $M^*(S)$ sums up to find a subset $M(S)$ of the instances $t \in C(s) \in C(S)$ that forms the most concise class, i.e. where the similarities of the instances in $M(S)$ is maximized or in the optimal case, $M(S) = M^*(S)$.

In the current example, the set {db:Belmont_California, db:San_Francisco and db:San_Jose_California} form a more concise class than the set {db:Belmont_France, db:San_Francisco and db:San_Jose_California}. Precisely, the candidate db:Belmont_California shares the predicate db:country and value db:Usa with the instance db:San_Francisco, which shares the predicate db:locatedIn and value db:California with the instance db:San_Jose_California. Consequently, class-based matching would consider the former set as more likely to contain the correct matches for the source instances than the latter set.

At this point, should be clear that class-based matching does not directly compare s with a candidate $t \in C(s)$. Rather, it determines whether t is a match or not based on its class membership, i.e. whether it "belongs to" $M^*(S)$. Here, $M^*(S)$ acts as an idealized instance-based representation of the target class of interest. In practice, $M^*(S)$ is not given but stands for the actual result to be determined by instance matching.

4.2 Formal Definition

The class-based matching resembles the unweighted set-cover problem. They have substantial differences but in the core, class-based matching also wants to find a set that minimize (or maximize) some criteria, subject to some constraints.

In the unweighted set cover problem, given a set $E = \{e_1, \ldots, e_n\}$, some subsets of those elements $B_1, \ldots, B_m$ where each $B_j \subseteq E$. The goal is to find a collection of subsets that covers all E; that is, we wish to find an $I \subseteq \{1, \ldots, m\}$ that minimizes $\sum_{j \in I} 1$ subject to $\bigcup_{j \in I} B_j = E$.

In our case, we assume $E = C = (M^*(S) \cup M^*(S)^-)$, C is the set of candidates, which is the union of $M^*(S)$ (the set of matches to S) and $M^*(S)^-$ (the set of non-matches to S). Also, we assume $B_j = M(s_j)$ where each $M(s_j) \subseteq C(s_j) \subseteq C$ and $C(s_j) \in C(S)$. The goal is to find a collection of subsets that cover only $M^*(S)$, that is, we wish to find an $I \subseteq \{1, \ldots, m\}$ that minimizes $\sum_{j \in I} 1$, subject to $\bigcup_{j \in I} B_j = M^*(S)$.

This problem can be rephrased into the maximization problem: the goal is to find a collection of subsets that cover only $M^*(S)$, that is, we wish to find an $I \subseteq \{1, \ldots, m\}$ that maximizes $\sum_{j \in I} |M(s_j) \cap M^*(S)|$, subject to $\bigcup_{j \in I} M(s_j) \cap M^*(S)^- = \emptyset$.

Differently from the set-cover problem, the universe that we want to cover ($M^*(S)$) is undefined. It is a subset of the entire universe E. Then, to accommodate the fact that $M^*(S)$ is undefined but we have a heuristics that approximates $M(S)^*$, the maximization problem consists entirely in finding this approximation, i.e. in finding $M^*(S)$. Therefore, the main idea behind class-based matching can be formalized as follows:

Definition 5 (Class-based Matching): To find the solution for the class-based matching problem consists of computing

$$M^*(S) \approx \underset{M(S) \in \mathbf{M}}{\operatorname{argmax}} \frac{\sum_{M(s) \in M(S)} \sum_{t \in M(s)} Sim(t, M(S))}{\sum_{M(s) \in M(S)} |M(s)|} + Z(S)$$
$$\text{Subject to:}$$
$$Sim(t, M(S)) > \delta \text{ and}$$
$$M(S) = \{M(s) | s \in S : M(s) \subseteq C(s) \in C(S)\} \tag{3}$$

where $\mathbf{M}$ is the set containing all possible $M(S)$ as elements. The term $Sim(t, M(S)) > \delta$ captures the heuristic that avoid non-matches, which would map to the initial constraint $\bigcup_{j \in I} M(s_j) \cap M^*(S)^- = \emptyset$. Precisely, $Sim(t, M(S))$ is an arbitrary function (e.g., Jaccard) that returns the similarity between an instance t and an instance-based class representation $M(S)$, δ is a similarity threshold and

$$Z(S) = \frac{\sum_{s \in S} H(s)}{|S|}, \text{ with } H(s) = \begin{cases} 0 & \text{if } |M(s)| = 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

Note that $Sim(t, M(S))$ operates over *features* extracted from the instance t and instances in the sets in $M(S)$. This will be detail further, in our proposed solution to this problem.

Also note $Z(S)$ is simply an auxiliary term introduced to deal with the general case where $M(s)$ might be empty. It helps to score higher a solution set $M(S) \in \mathbf{M}$ where the majority of $M(s) \in M(S)$ has cardinality higher than zero; therefore, avoiding solution sets with many empty matches. This term is not needed in the setting where there always exists at least one candidate for a given source instance s. In this case, we can simplify Eq. 3 removing $Z(S)$ and adding the constraint that $|M(s)| > 0$.

Intuitively, the idea is to find an approximate solution $M^*(S) \in \mathbf{M}$, which contains at least one candidate for every source instance s (considering $|M(s)| > 0$ in Eq. 3). Comparing to all the other candidate solutions in $\mathbf{M}$, $M^*(S)$ is the most similar to the instances it consists of (c.f. $Sim(t, M(S))$ in Eq. 3). That is, $M^*(S)$ is not only the result but at the same time, acts as the class that is compared with the candidates instances $t \in C(s) \in C(S)$.

Solving this problem requires finding the threshold δ, enumerating of all possible sets $\mathbf{M}$, and determining the optimal $M^*(S)$. Since this enumeration could be very large, i.e., $|\mathbf{M}| = 2^{\sum_{s \in S} |C(s)|}$ (because $|M(S)| = |S|$), we propose an approximate solution to this that does not require a full enumeration. Also, we show how to obtain a more compact representation of $M^*(S)$ and to automatically choose δ.

5 Class-based Matching: A Solution

We will first present the main idea and then discuss extensions to this basic solution.

5.1 Basic Solution

Class-based Matching. Given a set of instances S and the candidate sets $C(S) = \{C(s_1), \ldots, C(s_n)\}$, we formulate class-based matching as the one of finding the instances t from each candidate set (i.e. $t \in C(s) \in C(S)$) that are similar to the candidate sets $C(S)$.

Our method starts computing a score of similarity between $t \in C(s)$ and $C(S)$ itself, i.e., $Sim(\{t\}, C(S))$. In this process $C(S)$ is considered the class of interest but not the solution set $M(S)$; differently from the formal problem definition where $M(S)$ is both the class of interest and a solution set. In this approximation, we depart from $C(S)$ to obtain the solution set $M(S)$, therefore avoiding to enumerate all possible $M(S) \in \mathbf{M}$, as discussed before.

Intuitively, given t and any candidate set $C(s) \in C(S)$, if $F(\{t\})$ does not share any feature with $F(C(s))$, then t is not similar to any instance in this candidate set. If t is not similar to any candidate set $C(s) \in C(S)$, it cannot form a class with any candidate instance; therefore, based on our assumption, it cannot be a correct match for s. Contrarily, a candidate t that are more similar to other candidate sets are more likely to be form a class to other candidates, and therefore, can be a correct match. This heuristic is implemented as follows.

The computation of $Sim(\{t\}, C(S))$ obtains a score for each individual instance $t \in C(s) \in C(S)$. Then, the final solution set $M(S)$ is composed of $M(s) \subseteq C(s)$, where for all $t \in M(s)$, $Sim(\{t\}, C(S)) > \delta$. Below, we define Sim and further we describe how we compute the threshold δ.

$$Sim(t, C(S)) = \sum_{C(s') \in C(S)^-} \frac{SetSim(\{t\}, C(s'))}{|C(s')|} \quad (5)$$

where $t \in C(s)$ and $C(S)^- = C(S) \setminus C(s)$.

Observe that in Eq. 5, $t \in C(s)$ is only compared to the complement sets of $C(s)$. This avoids candidates that are dissimilar to other candidate sets to obtain larger scores when their features are abundant in $C(s)$. Intuitively, Eq. 5 captures the comparisons between t and candidate sets in $C(S)^-$ where the individual score $SetSim(\{t\}, C(s'))$ is weighted by the cardinality of $C(s')$ such that a $C(s')$ with high cardinality has a smaller impact on the aggregated similarity measure. We do this to capture the intuition that small sets containing only a few representative instances (captured by only a few features) represent better the class of interest.

We further normalize the result of Eq. 5 by the maximum score among all instances in $C(s)$ as

$$Sim(t, C(s), C(S)) = \frac{Sim(t, C(S))}{MaxScore(C(s), C(S))} \quad (6)$$

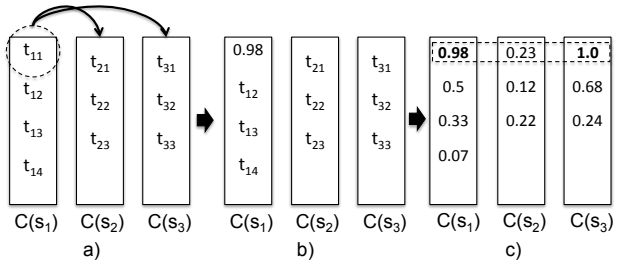

Fig. 5. (a) Class-based similarity score for the candidate t_{11} is obtained by comparing it with $C(s_2)$ and $C(s_3)$, (b) the score for t_{11} and (c) the scores for all other candidates.

where

$$MaxScore(C(s), C(S)) = MAX\{Sim(t', C(S)) | t' \in C(s) \in C(S)\} \quad (7)$$

This yields a class-based similarity score that is in $[0, 1]$. Using this function, an instance t is considered as a correct match for s, if $Sim(t, C(s), C(S))$ is higher than a threshold δ or when it is the top-1 result. We will refer to these two variants as the Threshold and the Top-1 approach, respectively.

The Top-1 approach makes sense for those cases in the heterogeneous scenario, where datasets are duplicate-free or one-to-one mapping between a source and a target instance can be guaranteed. In this case, as every instance in every dataset stands for a distinct real-world entity, there exist at most only one correct match in the target for every instance in the source (i.e. likely the top-1). In the other cases where there are one-to-many matches, the Threshold approach is used. Notice that the Threshold approach can also be used in the one-to-one matching scenario. As we will show empirically, it yields competitive accuracy to the Top-1 in these cases.

Class-based matching is illustrated in Fig. 5 for the instance t_{11}, where it is compared to the candidate sets $C(s_2)$ and $C(s_3)$, where $C(S)^- = \{C(s_2), C(s_3)\}$. Notice that, in the end, $Sim(t_{11}, C(S))$ compares the features of $F(\{t_{11}\})$ to $F(C(s_2))$ and to $F(C(s_3))$. This is done for all instances in $C(s_1)$ and the one with the highest score Sim is assumed to be the correct match for s_1. Notice that for $C(s_2)$, $C(S)^-$ is defined as $C(S)^- = \{C(s_1), C(s_3)\}$. Alg. 1 illustrates the computation of Sim.

Similarity Function. Now, we introduce $SetSim(X_1, X_2)$, from Eq. 5, which captures the similarity between two sets of instances X_1 and X_2 based on their set of features $F(X_1)$ and $F(X_2)$:

$$SetSim(X_1, X_2) = FSSim(F(X_1), F(X_2)) \quad (8)$$

where $FSSim(F(X_1), F(X_2))$ is the function capturing the similarity between the two sets of features $F(X_1)$ and $F(X_2)$.

Early work such as the one by Tversky [11] shows that the similarity of a pair of items depends both on their commonalities and differences. This intuition is applied in similarity functions used for instance matching, which

Algorithm 1 SimScores($C(S)$).

1: $scores \leftarrow \emptyset$
2: **for** $c(s) \in C(S)$ **do**
3: $C(S)^- \leftarrow C(S) \setminus C(s)$
4: $score_{c(s)} \leftarrow \emptyset$
5: **for** $t \in C(s)$ **do**
6: $score_t \leftarrow 0$
7: **for** $c(s)' \in C(S)^-$ **do**
8: $score_t \leftarrow score_t + \frac{SetSim(\{t\},C(s)')}{|c(s)'|}$
9: **end for**
10: $score_{c(s)} \leftarrow score_{c(s)} \cup score_t$
11: **end for**
12: $scores \leftarrow scores \cup score_{c(s)}$
13: **end for**
14: $maxscore \leftarrow max(scores)$
15: **for** $score_{c(s)} \in scores$ **do**
16: **for** i in $1..|score_{c(s)}|$ **do**
17: $score_{c(s)}[i] \leftarrow \frac{score_{c(s)}[i]}{maxscore}$
18: **end for**
19: **end for**
20: **return** $scores$

like Jaccard similarity, gives the *same weight* to commonalities and differences. This is suitable for matching instances because commonalities help to infer that two instances might be the same while differences support the conclusion that they are not.

For class-based matching, we depart from this to give a *greater emphasis on commonalties*. We do so because the amount of features that a class of instances have in common is typically small compared the amount of features that are specific to individual instances. For deciding whether an instance belong to a class or not, we consider the common features to be more characteristic for the class. Also, they are more distinctive due to their scarcity. Features that are specific to individual instances are less representative for the class, and also convey more noise, due to their abundance. We propose the following function to support this intuition:

$$FSSim(f_1, f_2) = \begin{cases} 0 \text{ if } |f_1 \cap f_2| = 0 \\ |f_1 \cap f_2| - (\frac{|f_1 - f_2| + |f_2 - f_1|}{2|f_1 \cup f_2|}) \quad \text{otherwise} \end{cases} \tag{9}$$

where f_1 and f_2 stand for $F(X_1)$ and $F(X_2)$, respectively. $FSSim(f_1, f_2)$ only considers f_1 and f_2 to be similar when there exist some commonalities (i.e. $FSSim(f_1, f_2)$=0 if $|f_1 \cap f_2| = 0$). The first term $|f_1 \cap f_2|$ has a much larger influence, capturing commonalities as the number of overlaps between f_1 and f_2, which is always larger than 1. While the second term $(\frac{|f_1 - f_2| + |f_2 - f_1|}{2|f_1 \cup f_2|})$, capturing the differences, is always smaller than 1. In fact, given t_j and t_k that have n and $n-1$ features in common with t_i, respectively, $FSSim$ always returns a higher score for t_j. Notice that $FSsim$ does not aim to capture any class semantics, it is a simple set similarity function tailored towards the commonalities, for the reason that we discussed.

For example, assuming $f_1 = F(\{\texttt{db:Belmont\_California}\})$, $f_2 = F(\{\texttt{db:Belmont\_France}\})$ and $f_3 = F(C(\texttt{nyt:5555}))$; then, $FSSim(f_1, f_3) = 3.65$, while $FSSim(f_2, f_3) = 1.5$. The scores reflect the fact that f_1 has 4 features in common with f_3, while f_2 only 2.

This bias towards commonalities is captured by the following theorem, which does not hold for the Jaccard function (see Appendix A):

Theorem 1: If $|f_i \cap f_j| > |f_i \cap f_k|$ then $FSSim(f_i, f_j) > FSSim(f_i, f_k)$.

Note the proposed function does not completely neglect the role of differences. In particular, when two instances have the same number of overlaps with a class, their differences to that class decide which ones is a better match.

Importantly, to avoid to bias the solution towards a specific feature set, we considered all features ($A(\cdot)$, $D(\cdot)$, $O(\cdot)$ and $T(\cdot)$) equally determinant in our setting. As we observed empirically, on average, this strategy produced better results than the settings where we removed any feature set. We acknowledge that a fine-grained weighting of the features may improve the method; however, this requires an non-trivial solution, to be consider as future research.

5.2 Reducing the Number of Comparisons

In order to compute a score for every instance in each candidate set $C(s) \in C(S)$, our class-based matching approach requires a maximum of $|C(S)| \times |C(s)^{max}|$ comparisons, where $C(s)^{max} \in C(S)$ denotes the candidate set with the largest number of instances. Since $|C(S)|$ can be large, we propose to reduce the number of comparisons by reducing $C(S)$ to a minimal subset $C(S)^*$ such that the feature distribution of $C(S)^*$ differs only within an error margin ϵ from the feature distribution of $C(S)$. Then, $C(S)^*$ is used in the line 3 of Alg. 1 instead of $C(S)$, i.e., $C(S)^- = C(S)^* \backslash C(s)$. We define the feature set and the distribution over elements in that set as follows:

Definition 6 (Feature Set): The feature set of $C(S)$ is $F(C(S)) = \bigcup_{C(s) \in C(S)} F(C(s))$.

Definition 7 (Feature Distribution): A distribution over the feature set $X = F(C(S))$, denoted by $Pr(X)$, assigns a probability $p(x)$ to every feature x, i.e. the probability of observing a feature x through the repeated sampling of features from X:

$$p(x) = Pr\{X = x\} = \frac{\sum_{C(s) \in C(S)} |\{x\} \cap F(C(s))|}{|F(C(S))| \times |C(S)|}$$

where

1) $p(x) \geq 0$ for all $x \in X$ and
2) $\sum_{x \in X} p(x) = 1$.

In the ideal case, $C(S)^*$ contains a much smaller amount of candidate sets compare to $C(S)$, i.e. $|C(S)^*| << |C(S)|$, while carrying the same amount of information such that the similarity scores computed for $C(S)^*$ and $C(S)$ are the same. In order to capture the differences in the provided information content, we use the *z-test*, which is a standard method for analyzing the similarity/difference between the distribution of a

sample and the distribution of the original population:

$$z\text{-}test = \frac{(\mu(sample) - \mu(population))}{(\frac{\sigma(population)}{\sqrt{size(sample)}})}$$

where $\mu(\cdot)$, $\sigma(\cdot)$ and $size(\cdot)$ denote the mean, the standard deviation and the size, and $population = Pr(F(C(S)))$ and $sample = Pr(F(C(S)^*))$.

A brute force algorithm to solve this problem is to enumerate all possible subsets of $C(S)$, i.e., its power set $2^{C(S)}$. Then, for each set in $2^{C(S)}$, it picks the minimal set $C(S)^*$ that has a distribution equivalent to the one of $C(S)$. In the worse case, this algorithm takes $O(2^{C(S)})$ verifications to find $C(S)^*$, which is prohibitive even for small $C(S)$.

We note the attempt to find an optimal solution to this problem may goes against our goal. We need to find the set $C(S)^* \subseteq C(S)$ at very low cost so that the time spent is smaller than the gain that can be achieved by using $C(S)^*$ instead of $C(S)$. We thus use an efficient greedy algorithm that exploits the following intuition: a sample is more similar to its population when it contains more data from the population. Without enumerating and evaluating each subset, it iteratively extracts and adds a subset $C(s) \in C(S)$ to the sample $C(S)^*$ until the *z-test* between $Pr(F(C(S)))$ and $Pr(F(C(S)^*))$ approach the confidence value commonly used in the literature,[4] or all $C(s) \in C(S)$ is added to $C(S)^*$. For faster convergence, only features that occur more than once in the data are considered in $F(C(S))$.

The procedure to obtain $C(S)^*$ is summarized in Alg. 2. In Sec. 6, we compare the time performance and accuracy of Alg. 1 with and without this procedure.

Algorithm 2 CandidateSetsReduction($C(S)$).

```
1: C(S)* ← ∅
2: μ ← mean(p(C(S)))
3: σ ← stdv(p(C(S)))
4: for all C(s) ∈ C(S) do
5:    C(S)* ← C(S)* ∪ C(s)
6:    n ← |C(S)*|
7:    M ← mean(p(C(S)*))
8:    SE ← σ/√n
9:    z ← (M−μ)/SE
10:   if z is in the confidence interval then
11:      return C(S)*
12:   end if
13: end for
14: return C(S)*
```

5.3 Selecting the Threshold

As discussed, the Top-1 approach can be used when the datasets are duplicate-free. In all other cases, a threshold selection method should be employed. Then, only instances with similarity score above the computed threshold δ are selected as matches. State-of-the-art methods [3], [12] are supervised, relying on training data to find the best threshold. We propose an unsupervised method, which only uses statistics that can be derived from the computed scores. We cast the problem of threshold selection as the one of finding the statistical outliers among the similarity scores. In particular, we use two bags of scores, one containing only the maximum scores and the other contain all scores.

Definition 8 (Bag of Scores): Given the candidates $C(S)$ and the CoI $C(S)$, the bag of *all scores* contains a score for every $t \in C(s) \in C(S)$, i.e., $Scores_{all} = \{Sim(t, C(S)) \mid t \in C(s), C(s) \in C(S)\}$. The bag of *maximum scores* contains a score for every $C(s) \in C(S)$, i.e., $Scores_{max} = \{MaxScore(C(s), C(S)) | C(s) \in C(S)\}$.

The maximum scores constitute the starting point for threshold selection. Intuitively speaking, two cases can be distinguished: First, (1) we have maximum scores that all are close to 1, and differences among them are small. (2) In the second case, there are large variations among scores. Some of them are low, approaching 0.

Note the first case corresponds to the setting where correct matches are easy to find, i.e., at least one candidate with score close to 1 could be found for every source instance. In this case, δ is simply defined based on the minimum score in $Score_{max}$. In this way, all candidates with score in $Score_{max}$ are selected. This strategy works for this "easy setting" because due to the use of set-based similarity in class-based matching, score differences among correct matches tend to be small while differences between correct and incorrect ones are much larger. Thus, incorrect matches typically have scores much lower than the minimum score in $Score_{max}$.

In the second "harder setting", "bad" candidates were detected, i.e., those with low scores in $Score_{max}$. This indicates that for some source instances, no correct candidates exist or could be found. However, we cannot use the minimum score as before to filter these "bad" candidates. It could be too low, or generally, not precise enough to separate correct from incorrect matches. To find δ in this case, we propose to detect outlier scores. For finding outliers more precisely, we use the bag of all scores, $Score_{all}$, instead of $Score_{max}$. Intuitively, candidates that have an outlier score share fewer features in common with the class of interest, thus can be regarded as incorrect.

As a mechanism to implement the ideas discussed above, we propose to use a method based on the Chauvenet's criterion [13], a statistical technique for outlier detection.

Definition 9 (Chauvenet's Criterion): Given the mean μ and the standard deviation σ of the scores in $Score_{all}$, a score $x \in Score_{all}$ is an outlier if $Chauvenet(x) < c_1$, where

$$Chauvenet(x) = p(\frac{\mu - x}{\sigma}) \times |Score_{all}|,$$

c_1 is a confidence level[5] and $p(\frac{\mu-x}{\sigma})$ is the probability[6] of observing a data point x that is $\frac{\mu-x}{\sigma}$ times standard

4. which, under our assumption of normal distribution, is in $[-1.96, 1.96]$

5. Typically, it is set to 0.5 when using Chauvenet's criterion.

6. We assume a normal distribution.

deviations away from the mean.

According to the Chauvenet's criterion, there are no outliers when $\sigma < c_2$, another confidence level that is typically set close to 0.[7]

Our procedure for threshold selection first extracts the maximum score of each candidate set $C(s) \in C(S)$ to form $Score_{max}$. When there are no outliers according to the Chauvenet's criterion, we set the threshold as the minimum score in $Score_{max}$. Otherwise, we iteratively apply the Chauvenet's criterion over $Score_{all}$ until no further outliers can be detected: in every iteration, if outliers are found and δ is the highest score among all outliers, we remove all scores that are smaller than δ from $Score_{all}$; this pruned bag of scores is then used in the next iteration. The maximum δ found during this process is used as the threshold. Alg. 3 describes this algorithm.

Algorithm 3 ThresholdBasedSelection(C).

```
1: Y ← getMaxScores(C)
2: L ← getAllScores(C)
3: δ ← Array
4: if Y.standardDeviation < c2 then
5:    return Y.min
6: end if
7: for all x ∈ L do
8:    if L.mean − x < 0 then
9:       continue;
10:   end if
11:   if chauvenet(L, x) then
12:      C′ ← remove all scores ≤ x from C
13:      δ.add(x)
14:      δ.add(ThresholdBasedSelection(C′))
15:      return δ.max
16:   end if
17: end for
18: return 0
```

For example, for the scores in Fig. 5, the list of maximum scores $Score_{max}$={0.98, 0.23, 1.0} has a standard deviation much higher than the confidence level c_2; therefore, the algorithm is applicable. Considering all scores $Score_{all}$= {0.98, 0.5, 0.33, 0.07, 0.23, 0.12, 0.22, 1.0, 0.68, 0.24 }, this algorithm would select as threshold $\delta = 0.68$; therefore, all instances with scores smaller than 0.68 would be automatically rejected as a correct match. Notice that 0.68 is much higher than 0.22, the minimal of $Score_{max}$.

6 Evaluation

Our experiments are based on the OAEI 2010 and 2011 instance-matching track. We observed that SERIMI with the proposed candidate set reduction algorithm was 20% faster than SERIMI without it. Also, class-based matching was useful and complementary to direct matching. For OAEI 2010, this combination increased average F1 result of the second best by 0.21; and, for OAEI 2011 data, SERIMI improves the results of recently proposed approaches, *PARIS* [2] and *SIFI-Hill* [3], by 0.44 and 0.09, respectively. Compared to the best system participated at OAEI 2011, SERIMI achieved the same performance. However, as opposed to that, SERIMI does not assume domain knowledge and manually engineered mappings.

Evaluation Metrics. We used the standard metrics precision (proportion of correct matches among matches found), recall (proportion of matches identified among all actual matches) and F1 (harmonic mean between precision and recall) to measure the result accuracy (also employed by OAEI). To compute these metrics, the provided reference mappings were used as the ground truth.

$$F1 = 2 \times \frac{Recall \times Precision}{Recall + Precision} \tag{10}$$

Data. We used all the OAEI 2010 data employed by participants, which include the life science (LS) collection containing DBPedia, Sider, Drugbank, Dailymed, Tcm and Diseasome and the Person-Restaurant (PR) dataset. From OAEI 2011, the datasets used were New York Times (Nyt), DBPedia, Geonames and Freebase. Given a pair of datasets, the task was to match instances in one dataset to instances in the other. The source class of instances for each dataset was defined by the OAEI. Detailed information can be found in their website [8]. Table 2 and 3 show some relevant statistics related to the datasets and matching tasks, respectively.

TABLE 2
Number of triples in each dataset.

Dataset	Triples	Dataset	Triples
Nyt	350.349	Person11	9.000
Freebase	3.554.824	Person12	7.270
DBPedia	>10.000.000	Person21	10.800
Geonames	>10.000.000	Person22	5.944
Sider	96.204	Rest1	1.130
Tcm	111.021	Rest2	7.520
Dailymed	131.068	Drugbank	507.500
Diseasome	69.545	-	-

Systems. All computed results were done using an Intel Core 2 Duo, 2.4 GHz, 4 GB RAM, using a FUJITSU MHZ2250BH FFS G1 248 GB hard disk. The SERIMI implementation used in these experiments is available for download[9] at GitHub. It was implemented in Ruby. Except for SIFI and PARIS, we copied all available results as published in the OAEI benchmarks. We used the available authors implementation[10] for PARIS, and the best effort implementation in Java for SIFI-Hill (SIFI).

6.1 Task Analysis

The suitability of direct matching and class-based matching for a task is related to the complexity of the matching task itself. So far, there is no method that suits all kinds of matching tasks, because data are imperfect in this heterogeneous setting. As we will show, the widely employed assumption that attributes between datasets largely overlaps is not true for all matching tasks, or for

7. In literature, $\sigma < 0.011$ is typically used.

8. http://oaei.ontologymatching.org
9. https://github.com/samuraraujo/SERIMI-RDF-Interlinking
10. http://webdam.inria.fr/paris/

all instances within a matching task. We observed the accuracy of each matching technique largely depends on the distribution of the predicates and values in the source and target dataset. In order to obtain a better understanding of how these distributions affect the accuracy of a matching technique, below we propose the use of coverage (Cov) and discriminative power (Disc) as measures for analyzing the task complexity.

$$Cov(p, S, G) = \frac{|\{s|\langle s, p, o\rangle \in G \wedge s \in S\}|}{|S|} \quad (11)$$

$$Disc(p, S, G) = \frac{|\{o|\langle s, p, o\rangle \in G \wedge s \in S\}|}{|\{t|t = \langle s, p, o\rangle \in G \wedge s \in S\}|} \quad (12)$$

where S is the given set of instances in the dataset G.

The coverage of a predicate p measures the number of instances in S that p occurs as a predicate. A predicate p with low coverage indicates that p occurs in a few instances; therefore, when utilizing values of p for finding matches, we may miss some candidates. The discriminative power measures the diversity of predicate values. A predicate p has low discriminative power when many instances have the same values for p; therefore, using values of p for matching, results in larger candidate sets. Consequently, datasets with many predicates that have low coverage and low discriminative power are harder to match.

Using these two measures, we introduce a task complexity measure TC that defines the complexity of matching a set of instances S with T, where $T = \bigcup_{c \in C(S)} c$. First, we introduce the *predicate complexity measure* (PCM) that measures the complexity of matching a set of instances X based on coverage and discriminative power of a set of predicates P in G.

$$PCM(P, X, G) = \frac{\sum_{a \in P} Cov(a, X, G) + Disc(a, X, G)}{2|P|} \quad (13)$$

The size of the candidates sets in $C(S)$ is also an indication of complexity because sets with more candidates may have more ambiguous candidates to filter out. Therefore, we define the function $Card(S)$. Smaller values for $Card(S)$ indicate that $C(S)$ has bigger candidate sets.

$$Card(S) = \frac{|C(S)|}{\sum_{c \in C(S)} |c|} \quad (14)$$

Finally, we introduce TC, defined as:

$$TC = 1 - PCM(P_s, S, G_s) \times PCM(P_t, T, G_t) \times Card(S) \quad (15)$$

where TC is a value in the interval [0,1], where 0 is less complex and 1 more complex. Table 3 shows the characteristics of each matching task. Fig. 9 shows the tasks ordered by TC. With respect to that, Nyt-Geonames is the most complex task, which on average has around six candidate matches per instance. In this table, some tasks are easier tasks because most of the candidate sets contain only correct matches, or one instance per candidate set (e.g. Sider-Tcm).

TABLE 3
Dataset pairs representing matching tasks, number of comparable predicates (CP) for every task, number of correct matches (Match), number of candidate matches obtained from candidate selection (Cand), mean (MEAN) and standard deviation (STDV) of the number of candidates per instance.

Dataset Pairs	CP	Match	Cand	MEAN	STDV
Nyt-DB-Corp	3	1965	3839	2.0	2.01
Nyt-DB-Geo	4	1920	9246	4.87	7.9
Nyt-DB-Per	5	4977	7937	1.61	1.02
Nyt-Freebase-Corp	2	3044	3398	1.15	0.37
Nyt-Freebase-Geo	3	1920	2234	1.19	0.43
Nyt-Freebase-Per	3	4979	5090	1.04	0.19
Nyt-Geonames	4	1789	10782	6.18	9.21
Dailymed-Sider	8	1592	1591	1.0	0.03
Diseasome-Sider	4	238	163	1.0	0.08
Drugbank-Sider	8	284	283	1.0	0.06
Sider-Dailymed	2	1634	1915	2.93	2.43
Sider-DB-Drugs	2	734	742	1.05	0.22
Sider-DB-SideEffect	2	775	960	1.25	0.56
Sider-Diseasome	4	173	192	1.2	0.57
Sider-Drugbank	8	1140	881	1.04	0.21
Sider-Tcm	2	171	168	1.0	0.08
Person11-Person12	6	500	1501	3.23	2.28
Person21-Person22	6	400	476	5.06	3.2
Rest1-Rest2	2	112	117	1.06	0.5

Fig. 6 shows the coverage and discriminative power of predicates in the target datasets. In all these datasets, there exist at least one predicate with 100% coverage (e.g. `drugbank:brandName`, `freebase:name`). However, only in some cases, their discriminative power is maximal (e.g. `drugbank:brandName`). The DBPedia, Geonames and Freebase datasets seem to be the hardest to match, as both coverage and discriminative power of their predicates are the lowest. In these cases, many predicates have to be used, which is only possible when there are many corresponding predicates in the source. Contrarily, the higher the coverage, the easier is the task because more instances can be covered with fewer predicates (the discriminative power of source predicates is, however, irrelevant because only target predicate values are used for finding matches). Fig. 7 shows predicates in the source datasets that are comparable to target predicates, and their coverage. It indicates there are always some comparable predicates that can be used (Table 3 explicitly shows the number of comparable predicates for every task), and that their coverage is always maximal (except for Nyt). In summary, comparable predicates exist for all the given tasks. However, direct matching is harder for some tasks such as Nyt-Geonames and Nyt-DB-Geo as they require using several predicates due to low coverage and discriminative power of target predicates. As the coverage is different for different target instances in those tasks, direct matching may not achieve its full performance due to the lack of comparable predicates at instance level.

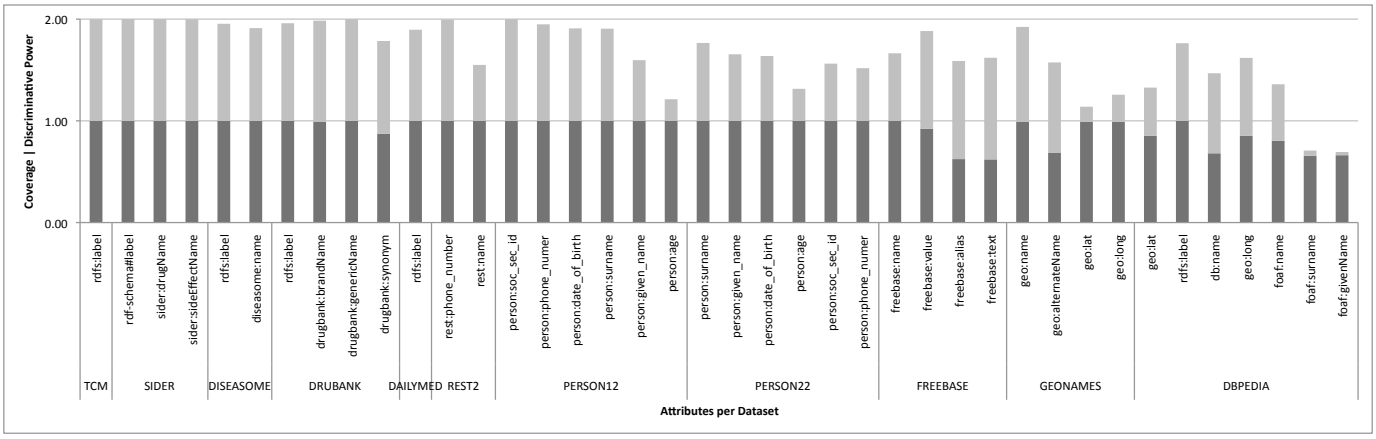

Fig. 6. Coverage and discriminative power of predicates in the target datasets.

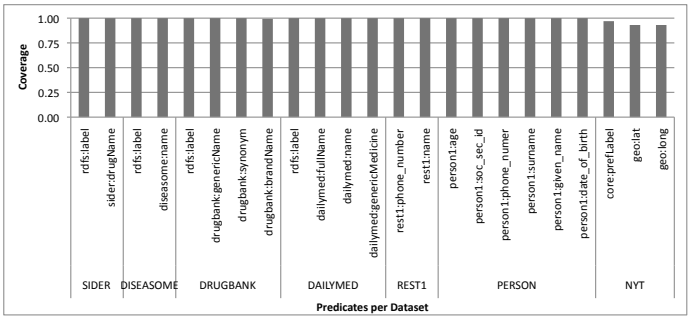

Fig. 7. Coverage of predicates in the sources.

6.2 SERIMI Configurations

We evaluated 5 different configurations of SERIMI: (1) We evaluated SERIMI's performance without and with candidate set reduction (algorithm in Sec. 5.2), referred to as *S* and *S+SR*, respectively. (2) We removed different features proposed for class-based matching, namely predicates (S+SR-P), datatype properties (S+SR-D), object properties (S+SR-O) and tuples (S+SR-T). (3) We evaluated SERIMI's performance with the top-1 approach (S+SR+TOP1) and the threshold approach (S+SR+TH). (4) Further, direct matching is used (DM), which is compared with SERIMI's performance (class-based matching) combined with direct matching (S+SR+DM). (5) Finally, S+SR+DM+J uses Jaccard instead of FSSim (Eq. 9). Except for S+SR+TOP1 and S+SR+TH, top-1 was used instead of the threshold for matching tasks with one-to-one matching. We measured time efficiency and result accuracy for every configuration, using all mentioned data collections in OAEI 2010 and 2011. The results are shown in Table 4 and Table 5, respectively.

Candidate Set Reduction. We observed that with candidate set reduction, SERIMI is 20% faster (average performance of S is 61s vs. 49s for S+SR). The number of candidate sets used in class-based matching could be considerably reduced. Consequently S+SR performed a much smaller number of comparisons. S+SR did not compromise accuracy as average results for S and S+SR were almost the same (F1 of 0.89 vs. 0.9).

Feature Removal. We could see that the performance improvement resulting from using less features (S+SR vs. S+SR-P, -D, -O and -T) is consistent but small in most cases. Removing predicates (S+SR-P) has the largest impact, where performance increased by 20%. This type of features represented a large part of all features used. Hence, processing was much faster without them. Removing features, however, also had a small but consistently negative impact on the accuracy. S+SR-P had the greatest impact on efficiency as well as accuracy; without predicates, F1 is 0.88 (a 0.02 loss in F1). Thus, while removing this type of features yielded an efficiency gain similar to using S+SR, it incurred greater loss in accuracy (it exhibited lower gain to loss ratio). In general, the results suggest that all proposed features are useful as they contributed to higher accuracy.

Top-1 vs. Threshold. There were no significant differences in time between the top-1 and the threshold approach (S+SR+TOP1 and S+SR+TH performances were similar). This suggests that selecting the threshold using the method in Sec. 5.3 requires little effort and can be done very efficiently. In terms of accuracy, S+SR+TOP1 had better average performance (86% F1) than S+SR+TH (84% F1). More specifically, S+SR+TOP1 yielded better results for tasks with one-to-one mappings between source and target instances, i.e. when there was only one correct match for every source instance. However, S+SR+TOP1 exhibited lower performance than S+SR+TH in two cases (50% F1 for Person21-Person22 and 56% F1 for Sider-Dailymed, compared to 86% and 81%, respectively), in which one-to-many mappings were needed.

Direct Matching vs. Class-based Matching. The DM (20s) approach was the fastest, followed by S+SR (50s), S+SR+DM (55s) and S (61s). Class-based matching as performed by S was expensive, requiring a much larger number of comparisons than direct matching (DM). Using candidate set reduction (S+SR), performance could be improved; S+SR is only 2.5 times slower than DM.

TABLE 4
Time performance for different SERIMI configurations, in seconds.

Datasets	S	S+SR	S+SR+DM	S+SR+DM+J	DM	S+SR+TH	S+SR+TOP1	S+SR-P	S+SR-D	S+SR-O	S+SR-T
Dailymed-Sider	22.6	14.57	20.41	29.87	17.34	14.15	14.15	15.33	14.44	14.03	13.93
Diseasome-Sider	1.75	1.38	1.45	1.86	1.71	1.46	1.37	1.36	1.41	1.44	1.36
Drugbank-Sider	8.85	8.12	8.84	10.79	8.33	7.79	7.64	7.67	8.62	7.84	7.56
Nytimes-DB-Corp	64.08	57.52	62.37	73.68	18.03	56.34	58.62	48.06	55.39	52.76	49.74
Nytimes-DB-Geo	606.43	440.71	470.12	435.19	78.63	441.12	437.56	365.62	421.33	430.98	413.31
Nytimes-DB-Per	159.13	167.14	196.53	190.75	96.07	172.58	167.27	145.19	163.92	163.29	162.24
Nytimes-Freebase-Corp	47.43	41.39	47.11	44.86	27.15	40.34	47.19	37.92	37.79	38.97	38.84
Nytimes-Freebase-Geo	33.41	33.34	39.44	38.81	21.99	32.25	35.93	28.38	31.91	34.59	34.05
Nytimes-Freebase-Per	78.76	74.68	91.79	87.95	57.56	75.29	73.65	70.04	77.12	71.94	72.7
Nytimes-Geonames	73.75	47.85	54.13	45.02	15.72	51.51	47.95	35.67	46.28	43.29	35.45
Person11-Person12	3.29	3.11	3.7	3.21	1.34	3.04	3.26	2.58	2.86	2.8	2.45
Person21-Person22	2.73	2.86	3.08	2.38	0.47	2.86	2.84	2.28	2.46	2.51	1.79
Rest1-Rest2	0.32	0.36	0.38	0.31	0.14	0.33	0.34	0.27	0.33	0.29	0.27
Sider-Dailymed	20.53	11.92	13.02	12.72	9.58	12.87	11.64	11.3	12.24	12.72	9.99
Sider-DB-Drugs	9.52	8.3	9.63	8.81	7.89	8.35	8.05	7.55	8.04	7.64	8.51
Sider-DB-SideEffect	4.37	4.1	3.63	3.38	2.3	3.5	3.35	2.42	2.72	2.68	2.67
Sider-Diseasome	1.05	0.56	0.71	0.67	0.48	0.54	0.55	0.48	0.54	0.56	0.52
Sider-Drugbank	22.77	18.05	16.76	17.53	10.84	14.42	14.09	12.94	13.41	14.24	12.99
Sider-Tcm	0.41	0.14	0.17	0.19	0.15	0.14	0.14	0.15	0.13	0.13	0.13
AVERAGE	61.11	49.27	54.91	53.05	19.77	49.41	49.24	41.85	47.42	47.51	45.71

Their combination (S+SR+DM) is slightly slower than S+SR. In return, S+SR+DM achieved the best F1 performance (93%). That is, SERIMI achieved the highest accuracy when direct and class-based matching are combined. S+SR+DM improved upon S+SR because DM could reinforce the similarity between instances when there was a direct overlap between the source and target. In fact, overlaps existed in all matching tasks as there were always some comparable predicates between source and target. Thus, there were no cases in which DM performed poorly. However, in some cases such as Nyt-DB-Geo, S+SR achieved much higher F1 than DM (81% vs. 69%). The combination of the two, S+SR+DM, could leverage evidences used by both approaches to further improve the results (82%). While this simple combination led to better results on average, there was one exception where DM yielded better performance (Person11-Person12), and several cases in which S+SR produced better results (Sider-Dailymed, Sider-DB- Side-Effect, Sider-DIASEASOME).

Particularly, S and S+SR performed poorly in Person11-Person12 (49% and 47%, respectively) because features of the candidate instances (all of them) were very similar (e.g. all contained phone, address and were of the type person). Due to this, the class-based matching produced similar scores to all candidates, which were not sufficiently distinct to distinguish the correct matches from the incorrect ones. In this task, the DM performed better, because there were sufficient overlapping between the source and target instances to identify the correct matches.

Jaccard Similarity vs. Set-based Similarity. Observe also that the use of Jaccard in S+SR+DM+J as set similarity decreased the average F1 from 93% to 87%. This confirms our intuition that the commonalities are more relevant than the differences to define similarity in our problem setting. Regarding performance, S+SR+DM+J (53s) was slightly better than S+SR+DM (54s), in average.

Task Complexity. Fig. 8 shows the connection between time performances for S+SR, S+SR+DM and DM and the number of triples in the candidate sets, which captures the amount of data that has to be processed. Clearly, more time was needed when more candidates and data have to be processed. Time performance for all 3 configurations increased quite linearly with a larger amount of data. To assess the complexity from the viewpoint of accuracy, we used the TC measure discussed before. Fig. 9 shows the connection between F1 performances for S+SR, S+SR+DM and DM and TC. We observed there was a trend between complexity and F1: F1 decreased as complexity increased. Interestingly, we could see many cases, including Person21-Person22 and Nyt-DB-Geo, where S+SR and DM are complementary, i.e. S+SR had a higher performance when DM had a lower performance, and vice-versa. S+SR+DM was most helpful in these cases as it could leverage the complementary nature of these two approaches to improve the results.

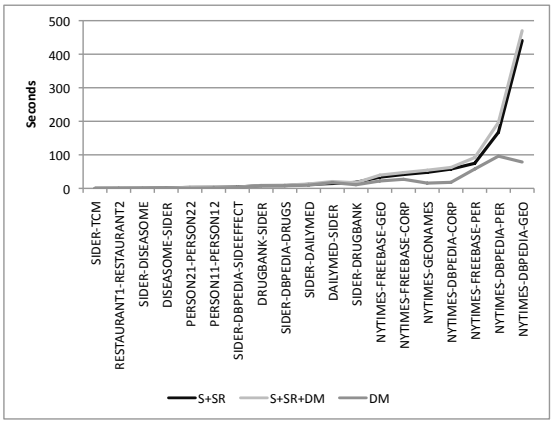

Fig. 8. Time performance; tasks are ordered according to the number of triples in the candidate sets.

Concluding, the highest accuracy is achieved by combining class-based matching with direct matching. Further, candidate set reduction helps to improve time efficiency. In the following experiments, we will use

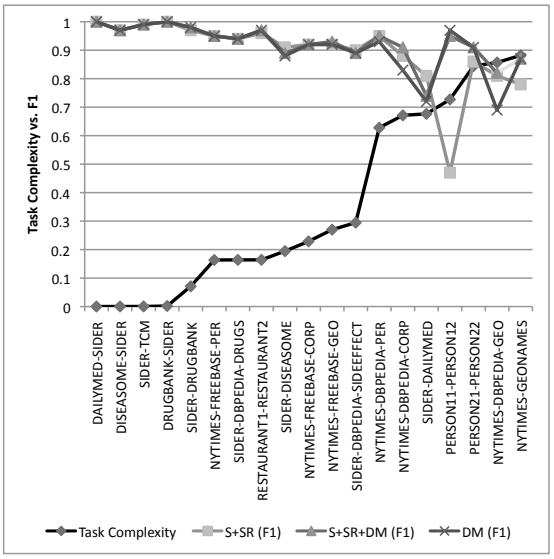

Fig. 9. F1 for tasks with increasing complexity.

S+SR+DM, in combination with the top-1 approach where there is an one-to-one mapping or the threshold approach otherwise.

6.3 SERIMI vs. Alternative Approaches

We compared SERIMI with state-of-the-art approaches. We carefully selected in the literature systems that reported the best performance in the benchmark that they participated. Those systems represents a large number of approaches used for instance matching.

We compared SERIMI with *RIMOM* and *ObjectCoref2010* (OC2010) using the data and results of OAEI 2010 [14]. To ensure the validity of this evaluation, we also included recently published results for ObjectCoref [15], called ObjectCoref2012 (OC2012). Using OAEI 2011 data and published results [1], we compared SERIMI with *AgreementMaker* (AM) and *Zhishi.links* (Zhi). Using the same data, we also compared SERIMI with the latest state-of-the-art approaches for instance matching, which did not participate at OAEI: *PARIS* [2] and *SIFI-Hill* [3].

OAEI 2010. Table 6 shows results for OAEI 2010. Missing values in the table indicates that the results were not published by the authors at OAEI. On average, SERIMI largely outperformed both systems. As shown in Table 6, SERIMI (93% F1) largely outperformed RIMON (72% F1) on average. SERIMI achieved considerable performance gain for the life science collection. Here, class-based matching played an important role because source and target instances often belong to different classes. In Sider-Dailymed for instance, there were instances of the types Drug and Ingredient sharing the same name that were incorrectly identified as candidate matches; these false positives were rejected by SERIMI thanks to class-based matching.

SERIMI was outperformed by OC2010 and RIMON in the Person collection. One reason is that this data involves artificially generated spelling mistakes. OC2010 and RIMON employed special direct matching strategies to deal with that. More importantly, SERIMI could not yield better results because class-based matching has limited impact when all candidates belong to the same class and the data schema is well-defined. In this scenario, all instances belong to the class Person and the source and target schema completely overlap. Thus, instances did not greatly vary in terms of class related information.

Also compared to OC2012, which only published results for the easiest matching tasks, SERIMI achieved better average performance (97% F1).

TABLE 6
F1 performance for SERIMI, OC2010, RIMON, OC2012 over OAEI 2010 data; some results were not available for OC2010, RIMON OC2012.

Datasets	SERIMI	OC2010	RIMON	OC2012
Sider-Dailymed	**0.74**	-	0.62	-
Sider-Diseasome	**0.89**	-	0.45	-
Sider-Drugbank	**0.98**	-	0.50	-
Sider-Tcm	**0.99**	-	0.79	-
Dailymed-Sider	**1.0**	0.70	0.62	-
Diseasome-Sider	**0.97**	0.74	-	-
Drugbank-Sider	**1.0**	0.46	-	-
Person11-Person12	0.95	**1.0**	**1.0**	**1.0**
Person21-Person22	0.91	0.95	**0.97**	0.95
Restaurant1-Rest.2	**0.97**	0.73	0.81	0.89
Average (OC2010)	**0.97**	0.76	-	-
Average (RIMON)	**0.93**	-	0.72	-
Average (OC2012)	**0.97**	-	-	0.95

OAEI 2011 As shown in Table 7, SERIMI had the same average performance as Zhi. In particular, Zhi performed better in tasks involving the location datasets (DB-Geo and GeoNAMES) because as opposed to SERIMI, it made use of domain knowledge and location-specific similarity functions. SERIMI largely outperformed SIFI (91% vs. 82%). SIFI had slightly better performance than SERIMI for Nyt-DB-Per and Nyt-Freebase-Per. With these tasks, SIFI was able to obtain more fine-tuned thresholds, which led to better results. As opposed to SIFI, which relies on training data for this threshold tuning, SERIMI is completely unsupervised. For SIFI, we used 10% of the OAEI ground truth as positive examples, and 10% of wrong alignments in the candidate sets as negative examples. PARIS obtained average performance of 47% F1, which was considerably worse than SERIMI. PARIS used both schema- and data-level features for matching. However, it only employed exact matching, i.e. it considers instances as matches when their features exactly match. In PARIS's authors experiments, good results could be achieved because exact matching was sufficient for the tasks involved. With the tasks studied here, exact matching led to very low performance.

Overall, the results show that SERIMI achieved the best accuracy results. Further, there is room for improvement as SERIMI so far neither uses training data nor exploits domain knowledge. Training data, for instance, could be exploited to fine tune the threshold (as implemented by SIFI) or to train a more optimized combina-

TABLE 5
F1 performance for different SERIMI configurations.

Datasets	S	S+SR	S+SR+DM	S+SR+DM+J	DM	S+SR+TH	S+SR+TOP1	S+SR-P	S+SR-D	S+SR-O	S+SR-T
Dailymed-Sider	1.0	1.0	1.0	1.0	1.0	0.99	1.0	1.0	1.0	1.0	1.0
Diseasome-Sider	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97
Drugbank-Sider	1.0	1.0	1.0	1.0	1.0	0.98	1.0	1.0	1.0	1.0	1.0
Nytimes-DB-Corp	0.88	0.88	0.91	0.85	0.83	0.78	0.88	0.87	0.88	0.88	0.88
Nytimes-DB-Geo	0.81	0.81	0.82	0.63	0.69	0.36	0.81	0.79	0.81	0.81	0.81
Nytimes-DB-Per	0.95	0.95	0.95	0.94	0.93	0.91	0.95	0.95	0.95	0.95	0.95
Nytimes-Freebase-Corp	0.92	0.92	0.92	0.84	0.92	0.88	0.92	0.86	0.92	0.92	0.92
Nytimes-Freebase-Geo	0.92	0.92	0.93	0.83	0.92	0.87	0.92	0.88	0.92	0.93	0.93
Nytimes-Freebase-Per	0.95	0.95	0.95	0.93	0.95	0.94	0.95	0.95	0.95	0.95	0.95
Nytimes-Geonames	0.78	0.78	0.87	0.49	0.87	0.4	0.78	0.64	0.78	0.78	0.78
Person11-Person12	0.47	0.47	0.95	0.95	0.97	0.49	0.47	0.46	0.48	0.46	0.46
Person21-Person22	0.86	0.86	0.91	0.91	0.91	0.86	0.5	0.86	0.86	0.86	0.86
Rest1-Rest2	0.96	0.96	0.97	0.97	0.97	0.94	0.96	0.96	0.96	0.96	0.96
Sider-Dailymed	0.83	0.81	0.74	0.55	0.72	0.81	0.56	0.73	0.8	0.79	0.79
Sider-DB-Drugs	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94
Sider-DB-SideEffect	0.9	0.9	0.89	0.89	0.89	0.89	0.9	0.9	0.9	0.9	0.9
Sider-Diseasome	0.91	0.91	0.89	0.9	0.88	0.91	0.91	0.92	0.91	0.91	0.91
Sider-Drugbank	0.97	0.97	0.98	0.99	0.98	0.96	0.97	0.97	0.97	0.97	0.97
Sider-Tcm	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
AVERAGE	0.90	0.89	0.93	0.87	0.91	0.84	0.86	0.88	0.89	0.89	0.89

TABLE 7
F1 performance for OAEI 2011.

Datasets	SERIMI	AM	Zhi	SIFI	PARIS
Nyt-DB-Corp	**0.91**	0.74	**0.91**	0.84	0.65
Nyt-DB-Geo	0.82	0.69	**0.92**	0.82	0.03
Nyt-DB-Per	0.95	0.88	0.97	**0.98**	0.06
Nyt-Freebase-Corp	**0.92**	0.80	0.87	0.80	0.82
Nyt-Freebase-Geo	**0.92**	0.85	0.88	0.64	0.60
Nyt-Freebase-Per	0.95	0.96	0.93	**0.97**	0.66
Nyt-Geonames	0.87	0.85	**0.91**	0.72	0.46
Average	**0.91**	0.82	**0.91**	0.82	0.47

tion of direct matching and class-based matching.

7 Related Work

Instance matching across datasets involves the use of similarity functions, thresholds and comparable attributes. They are captured by a matching scheme. While the majority of approaches use a flat representation of instances based on attribute values, other features might be applied. We will discuss existing approaches along these dimensions of features, similarity functions and matching schemes.

Matching Features. Instance features are derived from flat attributes, structure information (e.g. relations between RDF resources) [16], [17], [18] or semantic information extracted from ontologies. ObjectCoref [8] for instance, exploits the semantics of OWL properties such as *owl:Inverse FunctionalProperty* and *owl:FunctionalProperty*. Also, the combination of instance-level and schema-level features have been explored by PARIS [2], which jointly solve the problem of instance and schema matching.

SERIMI targets the heterogeneous scenario, where no structure, semantic or schema information is available in the worst case. It is based on a simple *flat representation*, where instances are captured as a set of attribute values. This representation is employed for single instances as well as for class of instances, which are needed for class-based matching.

Similarity Functions. The choice of similarity functions depends on the nature of the features. For strings, character-based (e.g. Jaro, Q-grams), token-based (e.g. SoftTF-IDF, Jaccard) and document-based functions (e.g. cosine similarity) were used. In addition to using syntactic information, special similarity functions have also been proposed to exploit different kinds of (lexical) semantic relatedness [19], [20].

Also along this dimension, we pursued a simple approach where only tokens are employed. However, for our new problem of class-based matching, which involves comparing sets of instances, we propose a *set-based similarity function* that take the token overlaps between sets into account.

Matching Schemes. With approaches relying on a flat representation of instances, i.e., attribute values, the employed schemes contain the similarity functions, thresholds and comparable attributes. Comparable attributes are either computed via automatic schema matching or assumed to be manually defined by experts [21]. Then, techniques with different degrees of supervision are employed for learning the scheme. Knofuss+GA[22] is an unsupervised approach that employs a genetic algorithm for learning. SIFI [3] and OPTrees [12] represent supervised approaches that learn the schemes from a given set of examples. Others approaches such as Zhishi.links [21], RIMON [23] and Song et. al [9] assume matching schemes that for the most part, were manually engineered, i.e., the similarity functions and thresholds were defined manually. They focus on the problem of learning the best comparable attributes.

The above solutions focus on direct matching. As oppose to that, class-based matching does not rely on a complex scheme. It uses a special similarity function we specifically design for this matching task. The problem of finding the threshold is cast as the one of detecting outliers, for which we propose an unsupervised solution.

Overall, our solution can be characterized as an unsupervised, simple, yet effective solution, which employs a

novel class-oriented similarity function, matching technique and threshold selection method to exploit the space of class-related features never studied before in the literature.

There are other systems in the literature that propose to tackle the same problem. For instance, Linda [24] is an entity matching system for web scale that was evaluated over a small subset of the datasets that we consider here. The reported results have a lower accuracy compared to the systems used on our evaluation.

8 Conclusion

In this work, we propose an unsupervised instance matching approach that combines direct-based matching with a novel class-based matching technique to infer Sameas relation over heterogeneous data. We evaluated our method using two public benchmarks: OAEI 2010 and 2011. The results show that we achieved good and competitive results compared to several representative systems focused on instance matching over heterogeneous data.

References

[1] J. Euzenat, A. Ferrara, W. R. van Hage, L. Hollink, C. Meilicke, A. Nikolov, D. Ritze, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Sváb-Zamazal, and C. T. dos Santos, "Results of the ontology alignment evaluation initiative 2011," in *OM*, 2011.

[2] F. M. Suchanek, S. Abiteboul, and P. Senellart, "Paris: Probabilistic alignment of relations, instances, and schema," *PVLDB*, vol. 5, no. 3, pp. 157–168, 2011.

[3] J. Wang, G. Li, J. X. Yu, and J. Feng, "Entity matching: How similar is similar," *PVLDB*, vol. 4, no. 10, pp. 622–633, 2011.

[4] M. A. Hernández and S. J. Stolfo, "The merge/purge problem for large databases," pp. 127–138, 1995.

[5] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *KDD*, 2000, pp. 169–178.

[6] G. Papadakis and W. Nejdl, "Efficient entity resolution methods for heterogeneous information spaces," in *ICDE Workshops*, 2011, pp. 304–307.

[7] A. Arasu, S. Chaudhuri, and R. Kaushik, "Learning string transformations from examples," *PVLDB*, vol. 2, no. 1, pp. 514–525, 2009.

[8] W. Hu, Y. Qu, and X. Sun, "Bootstrapping object coreferencing on the semantic web," *J. Comput. Sci. Technol.*, vol. 26, no. 4, pp. 663–675, 2011.

[9] D. Song and J. Heflin, "Automatically generating data linkages using a domain-independent candidate selection approach," in *International Semantic Web Conference (1)*, 2011, pp. 649–664.

[10] Y. Ma and T. Tran, "Typifier: Inferring the type semantics of structured data," in *ICDE*, 2013.

[11] A. Tversky, "Features of similarity," *Psychological Review*, vol. 84, no. 4, pp. 327–352, July 1977. [Online]. Available: http://dx.doi.org/10.1037/0033-295X.84.4.327

[12] S. Chaudhuri, B.-C. Chen, V. Ganti, and R. Kaushik, "Example-driven design of efficient record matching queries," in *VLDB*, 2007, pp. 327–338.

[13] W. Chauvenet, *A Manual of Spherical and Practical Astronomy V.II*. Dover, 1960.

[14] J. Euzenat, A. Ferrara, C. Meilicke, A. Nikolov, J. Pane, F. Scharffe, P. Shvaiko, H. Stuckenschmidt, O. Svb-Zamazal, V. Svtek, and C. Trojahn dos Santos, "Results of the ontology alignment evaluation initiative 2010," in *Proc. 5th ISWC workshop on ontology matching (OM), Shanghai (CN)*, P. Shvaiko, J. Euzenat, F. Giunchiglia, H. Stuckenschmidt, M. Mao, and I. Cruz, Eds., 2010, pp. 85–117. [Online]. Available: http://oaei.ontologymatching.org/2010/results/oaei2010.pdf

[15] W. Hu, J. Chen, and Y. Qu, "A self-training approach for resolving object coreference on the semantic web," in *WWW*, 2011, pp. 87–96.

[16] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in *ICDE*, 2002, pp. 117–128.

[17] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *J. Data Semantics IV*, 2005, pp. 146–171.

[18] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov, "Discovering and maintaining links on the web of data," in *International Semantic Web Conference*, 2009, pp. 650–665.

[19] A. Budanitsky and G. Hirst, "Evaluating wordnet-based measures of lexical semantic relatedness," *Computational Linguistics*, vol. 32, no. 1, pp. 13–47, 2006.

[20] X. Han and J. Zhao, "Structural semantic relatedness: A knowledge-based method to named entity disambiguation," in *ACL*, 2010, pp. 50–59.

[21] X. Niu, S. Rong, Y. Zhang, and H. Wang, "Zhishi.links results for oaei 2011," in *OM*, 2011.

[22] A. Nikolov, M. d'Aquin, and E. Motta, "Unsupervised learning of link discovery configuration," in *ESWC*, 2012, pp. 119–133.

[23] Z. Wang, X. Zhang, L. Hou, Y. Zhao, J. Li, Y. Qi, and J. Tang, "Rimom results for oaei 2010," in *OM*, 2010.

[24] C. Böhm, G. de Melo, F. Naumann, and G. Weikum, "Linda: distributed web-of-data-scale entity matching," in *CIKM*, 2012, pp. 2104–2108.

SUPPLEMENTAL PAGE

APPENDIX A

Jaccard Vs. FSSim

Amos Tversky[11] proposed the *ratio model* as a measure of similarity between two sets A and B. The ratio model is given below:

$$S(A,B) = \frac{|A \cap B|}{|A \cap B| + \alpha|A - B| + \beta|B - A|}, \alpha, \beta \geq 0 \qquad (16)$$

The parameters α and β balance the weight of the differences in the equation. This equation normalizes the similarity so that S is between 0 and 1. We can show that this model generalizes several set-theoretical models of similarity proposed in literature. If $\alpha = \beta = 1$ it reduces to the Jaccard coefficient, i.e. $Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|}$:

Proof: Replacing $\alpha = \beta = 1$ in E.q. 16, we have $S(A,B) = \frac{|A \cap B|}{|A \cap B| + |A - B| + |B - A|}$. Then, using the identity $|AUB| = |A \cap B| + |A - B| + |B - A|$, we have: $\frac{|A \cap B|}{|A \cap B| + |A - B| + |B - A|} = \frac{|A \cap B|}{|A \cup B|} = Jaccard(A,B)$ □

It is easy to show that when FSSim is replaced by Jaccard it violates the Theorem 1.

Proof: By counterexample: Let $|A \cap B| = 20$ and $|A \cap C| = 10$, and let $|A \cup B| = 40$ and $|A \cup C| = 20$. Then $|A \cap B| > |A \cap C|$ but $\frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap C|}{|A \cup C|} \Rightarrow \frac{20}{40} = \frac{10}{20} \Rightarrow Jaccard(A,B) = Jaccard(A,C)$. □

Now we proof that Theorem 1 is valid for FSSim(A,B).

Lemma 1: If $|A \cap B| > 0$ then $\frac{|A-B|+|B-A|}{2|A \cup B|} < 1$

Proof: Proof of Lemma 1: If $|A \cap B| > 0$ then $|A-B| + |B-A| < |A \cap B| + |A-B| + |B-A| < 2(|A \cap B| + |A - B| + |B - A|)$. Applying identity mentioned in Proof A, we have: $|A-B| + |B-A| < 2|A \cup B| \Rightarrow \frac{|A-B|+|B-A|}{2|A \cup B|} < 1$ □

Proof: Proof of Theorem 1: If $|A \cap B| > |A \cap C|$ then $|A \cap B| > 0$. Let a positive integer $\delta < 1$ and $\omega < 1$, then $|A \cap B| > |A \cap C| + (\delta - \omega) \Rightarrow |A \cap B| - \delta > |A \cap C| - \omega$. By Lemma 1 $\delta = \frac{|A-B|+|B-A|}{2|A \cup B|} < 1$ and $\omega = \frac{|A-C|+|C-A|}{2|A \cup C|} < 1$, then $|A \cap B| - \frac{|A-B|+|B-A|}{2|A \cup B|} > |A \cap C| - \frac{|A-C|+|C-A|}{2|A \cup C|} \Rightarrow FSSim(A,B) > FSSim(A,C)$. □