International Journal of Database Theory and Application
Vol.8, No.4 (2015), pp.191-206
http://dx.doi.org/10.14257/ijdta.2015.8.4.20

Property Alignment of Linked Data Based on Similarity between Functions

Yu Liu[1, 2], Shi-Hong Chen[1] and Jin-Guang Gu[2]

[1]*Computer School, Wuhan University, Luo-Jia-Shan Road 16, Wuhan 430072, China*
[2]*College of Computer Science and Technology, Wuhan University of Science and Technology, Heping Road 947, Wuhan 430081, China;*

Abstract

Owing to the complex structure and multi-meaning, property alignment is generally regarded as a challenging problem in the context of linked data. In this paper, we propose a novel method to align properties between datasets of linked data. Considering the role of properties in RDF triples, we regard all properties of linked data as property functions, and convert the problem of property alignment to the similarity evaluation between property functions, while the equivalent instances as inputs of property functions. Based on the similarity of property functions, the property alignment process of linked data is introduced. In order to prove the validity, we use the method to align properties in five representative domains between DBpeida and YAGO, DBpedia and LinkedGeoData respectively. The experimental results show that our method is independent of the property naming rules and can retrieve some matching properties ignored by other methods. In addition, our method requires fewer entity co-reference links than the link statistical approach.

Keywords: *Property Alignment; Linked Data; Property Function; Similarity between Functions.*

1. Introduction

Since linked data was proposed by Chris Bizer and Richard Cyganiak in 2007, more and more datasets following the principle of linked data are published on the web. These datasets cover a diversity of areas so that many researchers of different backgrounds try to develop some intelligent systems by taking advantage of these massive data. For example, the Traffic LarKC combined DBpeida with the datasets of two Milano municipalities to implement a question answering system about the traffic [1]; the music site in BBC pulls music metadata, from Musicbrainz and fetches introductory text from Wikipedia via DBpeida interlinking [2]. Although linked data is shining a bright light on the road forward for engineers to build intelligent information system, there is an inevitable roadblock, how to retrieve information from multiple datasets, that always have distinct schemas. In order to solve the above problem, ontology alignment has been widely employed.

Ontology alignment, or named ontology matching, is the process of determining a set of relationships (for example, subsumption and equivalence relationships) between entities (classes, properties, and individuals) in two ontologies [3]. The result of ontology alignment can be used for various tasks, such as ontology merging, query answering, or data translation [4]. Due to the importance of ontology alignment, a lot of achievements on ontology alignment have been carried out. According to the information that the methods of ontology alignment refer to, these methods fall into three categories: schema-based (H-Match [5], COMA & COMA++ [6, 7], CtxMatch & CtxMatch2 [8, 9]), instance-based (T-tree [10], ProbaMap [11]), mixed (Falcon-AO [12], RiMOM [13]). In order to assess these methods and measure the progress of ontology matchers, the OAEI (Ontology Alignment Evaluation Initiative) annually releases the benchmark test library, resulting in the purpose of many techniques is to achieve better results on the test library. Considering the characteristics of linked data, lots of researchers have realized that the techniques aiming at the benchmark may not suitable for linked data, so several alignment methods for linked data have been proposed in recent years.

In the context of linked data, ontology alignment mainly comprises three parts: class (concept) alignment, property alignment, and instance alignment [14]. The techniques for class alignment of linked data can be divided into two categories according to the types of information used for alignment: (1) external hierarchies and knowledge presented in lexical databases (BLOOM & BLOOM+ [15, 16]), (2) instance-level information (PARIS [17] and the method introduced in [18]). Instance alignment also is a research hotpot, and many systems have been proposed, such as SILK [19], SERIMI [20] and EAGLE [21]. Because properties have more complex structure and meaning than classes and instances, property alignment is regarded as a challenging problem in the ontology alignment field. Basing on similarity metrics, clustering, machine learning, and other technologies, many methods have been proposed in the last decade [14]. Among them, there are two methods that are similar with our method. The first one is introduced in [22, 33], the original purpose of which is to construct an upper level ontology from LOD (linked open data). Despite this method can find some similar properties, it has an apparent imperfection – properties having different semantic meaning may be aggregated into the same group (for example, "birthPlace" and "deathPlace"). The second one uses the entity co-reference (ECR) links to count the numbers of matching subjects and subject-object pairs between two datasets, and then find matching properties by analyzing the statistical results [24]. The method can eliminate the mismatches that happened in [22, 33], but it also has an obvious limitation – a large number of subjects and objects in the datasets should be related with ECR links.

By regarding properties as functions, we propose a novel method to align properties of linked data in this paper. On the basis of "owl:sameAs" links between subjects, the method collects all output sets of two property functions that take the equivalent instance as input, and measures the similarity between two properties by evaluating the similarity between these output sets. The higher similarity between the output sets means that two properties match each other with the higher probability. The main contributions of this paper are as follows:

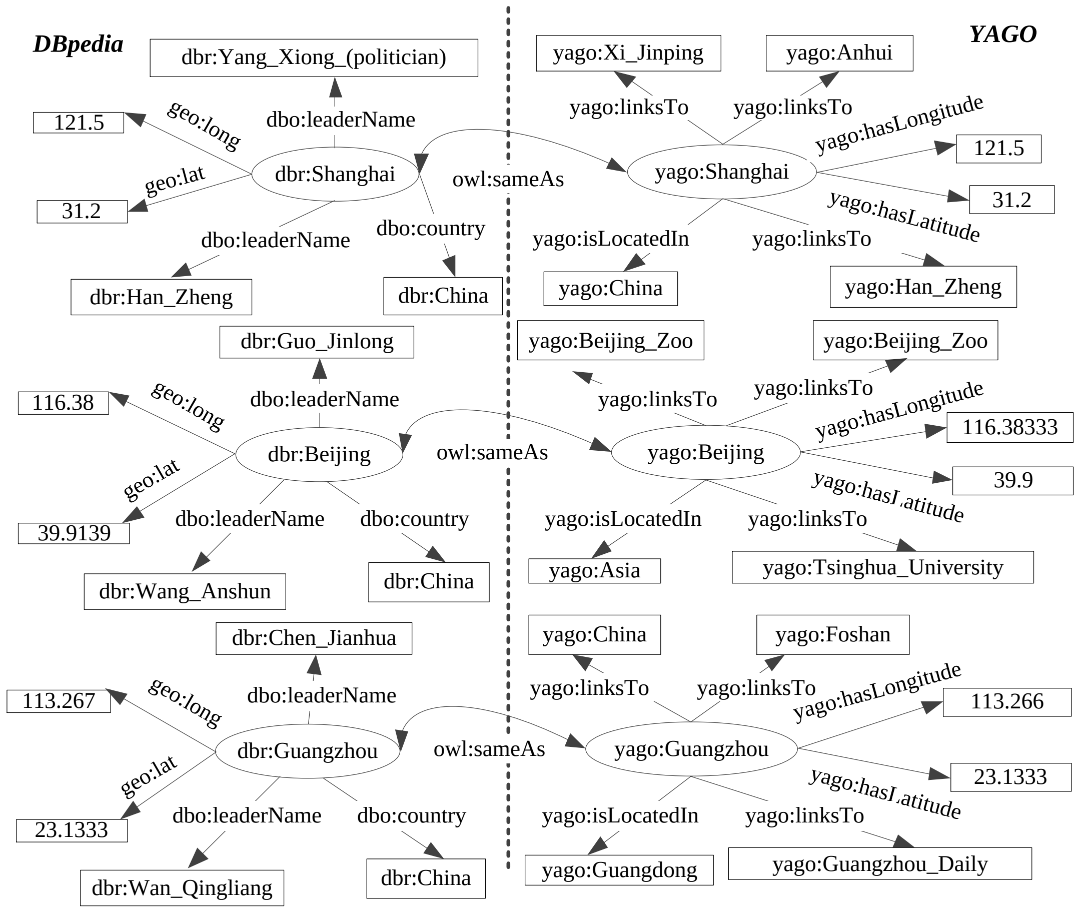
1. We define the property function on the context of linked data and propose a method to measure the similarity between two property functions. As the property functions have multiple types of outputs, we also propose several methods to measure the similarities between sets and between elements.

2. We illustrate the process of property alignment, which is composed of triple collecting, data preprocessing, property pair construction, similarity computing, filtering, etc.

3. In order to verify the effectiveness, we use several methods to align properties in five representative domains between DBpedia and YAGO, DBpedia and LinkedGeoData (LGD) respectively. The experimental results show that our method has distinctive characteristics and advantages compared with other methods, such as the independence on the naming rules of property and the fewer requirements for ECR links.

The remainder of this paper is structured as follows: Section 2 defines the property function in the context of linked data. Section 3 describes how to compute the similarity between property functions. In Section 4, the process of property alignment is explained in detail. The experiments and results are discussed in Section 5. Finally, the conclusion is presented in Section 6.

2. Property Function



geo: http://www.w3.org/2003/01/geo/wgs84_pos# foaf: http://xmlns.com/foaf/0.1/ dbp: http://dbpedia.org/property/
owl: http://www.w3.org/2002/07/owl# yago: http://yago-knowledge.org/resource/ dbo: http://dbpedia.org/ontology/
dbr: http://dbpedia.org/resource/ lgd: http://linkedgeodata.org/ontology/ w3: http://www.w3.org/2000/01/rdf-schema#

Figure 1. Several RDF Triples in DBpedia and YAGO

The basic component of linked data is RDF (Resource Description Framework) triple, which can be formally expressed as $(s, p, o) \in (I \cup B) \times (I \cup B) \times (I \cup B \cup L)$, where I is a set of IRIs (International URIs), B a set of blank and L a set of literals. In the triple, s can be looked as subject, p the property, and o the object or property value [25]. The subject denotes the resource, and the property denotes traits of instances or a relationship between the subject and the object. Owing to the vague meaning of the triples with blank, providing none useful information, our method does not take them into account in the process of property alignment. Therefore, we present the definition of property function, the key concept for the following content.

Definition 1 (Property Function) Suppose that (s, p, o) is a triple in the dataset $D1$, p can be defined as a property function, where the domain of p is the subject set $S=(s_1,s_2,...,s_n)$ and the range of p is the set of object/value set $O=(o_1,o_2,...,o_m)$, such that $s \in S$ and $\exists i \in [1,m], p(s) = o_i, o \in o_i$.

Note that the output of property function is a set, which contains one or more than one value/object, such as dbo:leaderName(dbr:Shanghai) in Figure 1 include two objects. In addition, the property functions have multiple types of outputs. As shown in Figure 1, the output elements of dbo:leaderName are URI type, and the output elements of geo:long are number type. In [23], property-object pairs are classified into five distinct types: Class, String, Date, Number, and URI. In this paper, we modify this classification for the subsequent computation of similarity between property functions. As all classes must be expressed as URI according to the principles of linked data, it is not necessary to set Class as a separate type, so we divide the property functions into four types: String, Date, Number, and URI. Table 1 presents our classifications and some samples. Obviously, it is easy to identify the types of property functions. For example, the elements in the output elements of URI property functions all start with "http://", the elements in the output set of String property functions contain "@".

Table 1. Property Function Classifications and Some Samples

Type	Property	Object
String	dbo:postalCode	"100000–102629"@en
	yago: isPreferredMeaningOf	"Beijing"@en
Date	dbo:foundingYear	"1955-01-01T00:00:00+02:00"^^ <http://www.w3.org/2001/XMLSchema#gYear>
	yago: wasCreatedOnDate	"1868-##-##"^^ <http://www.w3.org/2001/XMLSchema#date>
Number	dbo:areaTotal	"1.68012e+10"^^ <http://www.w3.org/2001/XMLSchema#double>
	geo:lat	"39.9139"^^ <http://www.w3.org/2001/XMLSchema#float>
	yago:hasArea	"1.680125E10"^^ <http://yago-knowledge.org/resource/m^2>
	yago:hasLatitude	"39.9"^^<http://yago-knowledge.org/resource/degrees>
URI	dbo:country	http://dbpedia.org/resource/China
	dbo:leaderName	http://dbpedia.org/resource/Han_Zheng
	yago:linksTo	http://dbpedia.org/resource/Tinghua_University
	yago:isLocatedIn	http://yago-knowledge.org/resource/Asia

3. Similarity between Property Functions

The similarity measure between functions is a widely studied issue, related with several pattern recognition problems, such as classification, clustering, and retrieval problems [26]. Considering lots of equivalent instances are connected by owl:sameAs in linked data, it is a reasonable to use the similarity between property functions to align the properties coming from the different datasets. Suppose that *D1* and *D2* are two datasets in linked data, *p1* and *p2* are properties in *D1* and *D2* respectively, *S1* is the subject set of *p1* and *S2* is the subject set of *p2*, such that the similarity between *p1* and *p2* (denoted as $Sim_{Property}(p1,p2)$) can be calculated by the following formula.

$$Sim_{Property}(p1,p2)=\begin{cases}\dfrac{\sum_{(s_1,s_2)\in S1\otimes S2} Sim_{Set}(p1(s_1),p2(s_2))}{|S1\otimes S2|} & |S1\otimes S2|\neq 0\\ 0 & |S1\otimes S2|=0\end{cases} \tag{1}$$

$S1\otimes S2$ = $\{(s_1,s_2)\,|\,s_1\in S1\wedge s_2\in S2\wedge(s_1,owl:sameAs,s_2)\wedge\exists o_1(s_1,p_1,o_1)\in D1\wedge\exists o_2(s_2,p_2,o_2)\in D2\}$, and $|S1\otimes S2|$ is the size of subject pair set. $Sim_{Set}(p1(s_1),p2(s_2))$ is on behalf of the similarity between the output sets of $p1(s_1)$ and $p2(s_2)$, which is introduced in the subsequent content. The intuition behind the Formula (1) is that the similarity between properties can be converted to the similarity between the outputs of property functions. The greater value the similarity between outputs gets, the greater value the similarity between property functions gets, and these properties are more likely to match with each other. In Figure 1, yago:hasLatitude is more similar with geo:lat than yago:hasLongtitude, because the output set of geo:lat(dbr:Shanghai) ({31.2}) equals with the output set of yago:hasLatitude(yago:Shanghai) ({31.2}), the output set of geo:lat(dbr:Beijing) ({39.9139}) approximately equals with the output set of yago:hasLatitude(yago:Beijing) ({39.9}), and the output set of geo:lat(dbr:Guangzhou) ({23.1333}) equals with the output set of yago:hasLatitude(yago:Guangzhou) ({21.1333}). At the same time, there are some big gaps between the output sets of geo:lat and yago:hasLongtitude. Hence, geo:lat and yago:hasLatitude are more likely to have the same meaning than geo:lat and yago:hasLongtitude. The example above is very intuitive and the conclusion can be easily drawn. However, the quantitative technique for the evaluation of similarity between sets is indispensable to implement the property alignment automatically and precisely.

Table 2. The Similarity Matrix of dbo: leaderName (dbr: Shanghai) and yago: linksTo (yago: Shanghai)

		dbo:leaderName(dbr:Shanghai)	
		dbr:Yang_Xiong_(politician)	dbr:Han_Zheng
yago:linksTo (yago:Shanghai)	yago:Xi_Jinping	0.087	0.299
	yago:Han_Zheng	0.269	1.0
	yago:Anhui	0.236	0.289

Given $RS1(rs_1^1,...rs_1^n)$ and $RS2(rs_2^1,...rs_2^m)$ are the output sets of $p1(s_1)$ and $p2(s_2)$ in Formula (1) respectively, the similarity matrix of $RS1$ and $RS2$ can be constructed with similarity values between corresponding object/value pairs. It is clear that the size of similarity matrix of $RS1$ and $RS2$ is $n*m$. Table 2 describes the similarity matrix of dbo:leaderName(dbr:Shanghai) and yago:linksTo(yago:Shanghai), in which each similarity value can be calculated according to the similarity measurement of URIs introduced later. In order to compute the similarity between output sets of property functions, we propose the definition of best matching collection.

Definition 2 (Best Matching Collection) Suppose that *M* is a similarity matrix of $RS1$ and $RS2$, the best matching collection is composed of several object/value pairs, and each pair (denoted as (rs_1^x,rs_2^y)) has the maximum similarity value compared to other pairs, which are in the same row and same column of (rs_1^x,rs_2^y). The definition of the best matching collection of $RS1$ and $RS2$ can be formally described as the formula below.

$$\begin{aligned}BMC(RS1,RS2) = \{(rs_1^x,rs_2^y) \mid x \in [1,n] \wedge y \in [1,m] \wedge \\ \forall i \in [1,n], Sim_{Elem}(rs_1^x,rs_2^y) \geq Sim_{Elem}(rs_1^i,rs_2^y) \wedge \\ \forall j \in [1,m], Sim_{Elem}(rs_1^x,rs_2^y) \geq Sim_{Elem}(rs_1^x,rs_2^j)\}\end{aligned} \quad (2)$$

In Formula (2), the function $Sim_{Elem}(rs_1^x,rs_2^y)$ returns the similarity between rs_1^x and rs_2^y, which are the elements in $RS1$ and $RS2$ respectively. Once the best matching collection is solved out, the similarity between the output sets can be calculated by the following formula.

$$Sim_{Set}(p1(s_1),p2(s_2)) = Sim_{Set}(RS1(rs_1^1,...rs_1^n),RS2(rs_2^1,...rs_2^m)) \\ = f(n,m,BMC(RS1,RS2))\frac{\sum_{(brs1,brs2)\in BMC(RS1,RS2)} Sim_{Elem}(brs1,brs2)}{|BMC(RS1,RS2)|} \tag{3}$$

Obviously, two factors in Formula (3) determine the similarity between the output sets of $p1(s_1)$ and $p2(s_2)$. The first factor is the average of similarity between elements in all pairs of the best matching collection, the second is $f(n,m,BMC(RS1,RS2))$, which is employed to evaluate how much the relationship between n, m and $BMC(RS1,RS2)$ influences the similarity between output sets. In this paper, we take the evaluation function as follow.

$$f(n,m,BMC(RS1,RS2)) = \frac{|BMC(RS1,RS2)|}{m+n-|BMC(RS1,RS2)|} \tag{4}$$

Formula (4) reflects a phenomenon – the greater size of best matching collection means that two output sets might be more similar, and vice versa. Hence, it can be calculated out that the similarity between the output sets shown in Table 2 is 0.2. It should be noted that the size of $BMC(RS1,RS2)$ may exceed the sum of *m* and *n* when lots of the same elements exist in the output sets. To avoid this unreasonable situation, the duplicate triples should be removed from the output sets before the construction of similarity matrix.

According to the explanation above, it is evident that the similarity between elements is the fundamental for the similarity evaluation between output sets. Nevertheless, the diversity of elements shown in Table 1 makes it impossible to use a single approach to calculate the similarity between all kinds of objects/values. The computation method relies on the several concrete situations. Consequently, we introduce these situations and the corresponding computation methods one by one. As the similarity between strings is the basis of similarity between elements of other types, the computation method for string similarity is given firstly. To evaluate the similarity between strings accurately, literal similarity and semantic similarity are taken into account. The literal similarity of our method is the mean value of three string-based similarity measures -- Jaro_Winkler distance, Levenshtein distance, and N-gram, which are introduced in [27]. The similarity library introduced in [28], an extension of the JWSL (Java WordNet Similarity Library), is employed to evaluate the semantic similarity between strings. Consequently, the mean value of literal similarity and semantic similarity is the similarity between strings, just as shown in Formula (5) and Formula (6).

$$Sim_{String\_String}^{Literal}(s1,s2) = \frac{Jaro\_Winkler(s1,s2)+Levenshtein(s1,s2)+N\_gram(s1,s2)}{3} \tag{5}$$

$$Sim_{String\_String}(s1,s2) = \frac{Sim_{String\_String}^{Literal}(s1,s2)+Sim_{String\_String}^{Semantic}(s1,s2)}{2} \tag{6}$$

Where, $Sim_{String\_String}^{Semantic}(s1,s2)$ adopts FaITH (Feature and Information Theoretic) [28] to measure the semantic similarity between *s1* and *s2*.

$$Sim_{URI\_URI}(u1,u2) = \begin{cases} \dfrac{Sim_{String\_String}(tail(u1),tail(u2))+Sim_{String\_String}(u1,u2)}{2} & if\ hasTail(u1)\wedge hasTail(u2) \\ Sim_{String\_String}(u1,u2) & else \end{cases} \tag{7}$$

As far as URI is concerned, the evaluation of similarity between URIs relies on the similarity between strings. Given *u1* and *u2* are two instances of URI type, the similarity between them can be calculated by applying Formula (7). Here, $tail(u1)$ is to obtain the substring of *u1*, which starts at the character following the last '/' of *u1*. For example, the result of $tail(http://dbpedia.org/resource/China)$ is "China". Therefore, the evaluation of similarity between *u1* and *u2* consider not only the similarity between two original strings but also the similarity between their tails.

$$Sim_{Number\_Number}(n1,n2)=\begin{cases}1-\dfrac{|n1-n2|}{|n1|+|n2|} & if\ !(n1=0\wedge n2=0)\\ 1 & else\end{cases} \tag{8}$$

$$Sim_{Date\_Date}(d1,d2)=Sim_{Number\_Number}(toNumber(n1),toNumber(n2)) \tag{9}$$

Formula (8) explains how to compute the similarity between numbers. The method fixes the bug in [23], which may cause the problem when *n1* and *n2* are opposite in sign. Since a date can be cased into a number, the similarity between dates can refer to Formula (8) after both dates are cast to numbers. Hence, the similarity between dates can be computed by Formula (9).

The evaluation of similarity between elements introduced above only considers the elements that share the same type, but two elements belonging to different types may refer to the equivalent objects/values in the real world. For instance, "Beijing"@en and yago:Beijing both mean the city of Beijing. In this paper, the similarity between String and URI is processed separately. Suppose that *u* is a URI instance and *s* is a String instance, the similarity between *u* and *s* can be calculated by Formula (10).

$$Sim_{String\_URI}(s,u)=\begin{cases}\dfrac{Sim_{String\_String}(s,tail(u))+Sim_{String\_String}(s,u)}{2} & if\ hasTail(u)\\ Sim_{String\_String}(s,u) & else\end{cases} \tag{10}$$

With regard to other pairs composed by elements belonging to different types, Formula (5) and Formula (6) are employed to evaluate the similarity between them after all elements in pairs are cast to strings. The purpose of this strategy is to find similarity between elements as much as possible, because the elements belonging to different types may be potentially similar. For example, the type of elements in the output set of dbp:establishedDate is http://www.w3.org/2001/XMLSchema#integer, but http://www.w3.org/2001/XMLSchema#date is the type of elements in the output set of yago:wasCreatedOnDate, so that the naive strategy only considering elements belonging to the same type may overlook the matching properties. In order to explain our strategy more clearly, the calculation methods of similarity for different property types are summarized in Table 3.

4. Property Alignment Process

Just as most of ontology matching systems introduced in [29], the property alignment proposed in this paper also is a multi-step process, which is shown in Figure 2. First of

Table 3. The Calculation Methods of Similarity for Different Property Types

	String	Date	Number	URI
String	Sim_{String_String}	Sim_{String_String}	Sim_{String_String}	Sim_{String_URI}
Date	Sim_{String_String}	Sim_{Date_Date}	Sim_{String_String}	Sim_{String_String}
Number	Sim_{String_String}	Sim_{String_String}	Sim_{Number_Number}	Sim_{String_String}
URI	Sim_{String_URI}	Sim_{String_String}	Sim_{String_String}	Sim_{URI_URI}

all, several "owl:sameAs" RDF triples in two datasets D1 and D2 are collected. These triples should meet the following requirements: subject coming from D1 and object coming from D2, or vice versa. Then, the triples that describe the subjects and objects in these triples are retrieved from D1 and D2. Second, the data preprocessing is to deal with the collected RDF triples for the later alignment, such as duplicate triples removal, string replacement, tokenization, and normalization. The aim of duplicate triples removal is to construct a reasonable similarity matrix, which is explained in the above section. The operation of string replacement can solve the imperfection in the original value, such as "1868-##-##" in table 1. In this paper, we use the string “01” to replace “##”. As [27], tokenization is to split strings into their component words based on delimiters and camel case, and normalization is the elimination of stylistic differences due to capitalization, punctuation, word order, and characters not in the Latin alphabet. In the third step, property pairs and the set of subjects shared by them are retrieved by traversing all “owl:sameAs” triples and RDF statements collected in previous steps. In Figure 2, *p1* and *p1’* are properties in D1, *p2* and *p2’* are properties in D2, *SS* is the set that contains the subjects shared by *p1* and *p2*, and *SS’* has similar meaning. Once the property pairs and the set of subjects shared by properties are prepared, the similarity between these properties can be calculated out by applying the corresponding calculation methods introduced above, so that each property pair can get a value indicating the similarity between properties in the pair. At the end, the filtering operation is executed to identify matching properties from all property pairs. Obviously, it is a key issue what kind of strategy should be employed in the filtering operation, which is discussed in the next section.

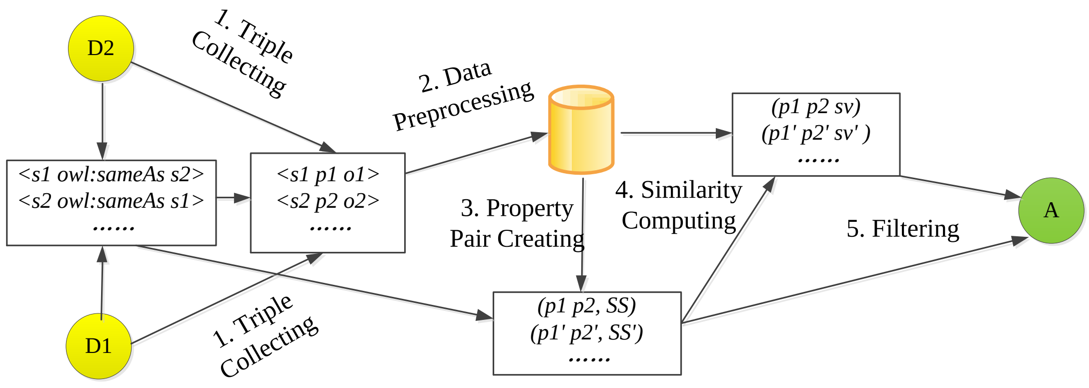


Figure 2. The Property Alignment Process of Linked Data based on Similarity between Functions

5. Experiments and Discussions

In order to verify the performance of our method, we use four similarity measurement methods (JaroWinkler, Levenshtein, FaITH and SimFun) to find the matching properties between three real-world datasets – DBpedia, YAGO and LGD. SimFun is the abbreviation of property alignment based on similarity between functions. Among the datasets in LOD, DBpeida is an interlining hub because there are a mass of “owl:sameAs” links between DBpedia and other datasets, which is an attractive feature for our method. Owing to the vast number of RDF triples in DBpedia, it is a great deal of work to align all properties in DBpeida with other datasets so that the experiments in this paper only cover five representative domains: airport, city, country, island and school. According to the property alignment process introduced above, all RDF triples required for alignment are collected from the SPARQL endpoints of the corresponding datasets. Table 4 lists some statistics of the experimental data, such as the number of RDF triples, the number of distinct subject in these triples, and the number of property belonging to different types. It is evident that the types of most properties in these datasets are String and Number, and the proportion of URI properties is relatively less, that means the method introduced in [24] may miss most of matching properties between these datasets because it only concern the objects of URI type. Considering the original purpose of the method introduced in [22, 33] and its apparent imperfection, we do not compare SimFun with the method in this paper.

Table 4. The Statistics of the Experimental Data

Data Sources		#RDF	#Subject	#Property			
				String	Date	Number	URI
Airport	DBpedia	648933	7233	252	8	353	75
	YAGO	268952	6718	3	2	6	7
	LGD	135981	7061	192	3	40	10
City	DBpedia	2373467	22205	716	15	897	79
	YAGO	770649	14124	6	5	8	13
	LGD	485864	22926	572	2	133	28
Country	DBpedia	81977	202	186	5	399	79
	YAGO	100122	199	5	2	17	15
	LGD	27966	194	357	2	14	7
Island	DBpedia	41367	316	145	3	297	90
	YAGO	24561	283	3	1	11	11
	LGD	5603	332	55	2	17	7
School	DBpedia	201377	2259	452	11	221	130
	YAGO	109374	2251	4	2	2	7
	LGD	35226	1985	37	3	16	9

Before comparing SimFun with other methods, the filtering strategy of SimFun should be settled down firstly. Although most of ontology alignment methods based on the similarity evaluation generally use a single threshold to identify the matching properties, this naive strategy may be not suitable for our method because there are several computational methods for similarity evaluation, which can be broadly divided into two categories: string-based method (Sim_{String_String} , Sim_{String_URI} , Sim_{URI_URI}) and number-based method (Sim_{Date_Date} , Sim_{Number_Number}). Hence, SimFun should have at least two thresholds to filter the results of different methods: one is for the results of string-based methods; the other is for the results of number-based methods. Obviously, these threshold values have great effects on the performance of SimFun. In order to choose the optimal values of string threshold and number threshold, performance metrics of different threshold values are recorded while DBpeida and YAGO are aligned by SimFun. In this paper, the performance metrics that we concern include precision, recall and F1-measure, which are defined as the formula (11), (12) and (13) respectively.

$$Precision = \frac{|Machting\_Property\_Pairs \cap Property\_Pairs\_In\_Result|}{|Property\_Pairs\_In\_Result|} \tag{11}$$

$$Recall = \frac{|Machting\_Property\_Pairs \cap Property\_Pairs\_In\_Result|}{|Machting\_Property\_Pairs|} \tag{12}$$

$$F1\text{-}measure = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall} \tag{13}$$

Where, *Machting_Property_Pairs* and *Property_Pairs_In_Result* both are the sets of property pair. The difference between two sets is that *Machting_Property_Pairs* is obtained by manual, and *Property_Pairs_In_Result* is the result of SimFun. In Figure 3, (a1), (a2) and (a3) show the changes of precision, recall and F1-measure respectively while the string threshold values ranging from 0.7 to 1.0 are used to filter the results of string-based methods; (b1), (b2) and (ba3) show the changes of precision, recall and F1-measure respectively when the number threshold values ranging from 0.9 to 0.98 are used to filter the results of number-based methods. Considering F1-measure is the comprehensive metric, the optimal threshold value should ensure that the F1-measure value is the greatest. In Figure 3, it is not hard to find that the optimal value of string threshold is around 0.9 and the optimal value of number threshold is around 0.98.

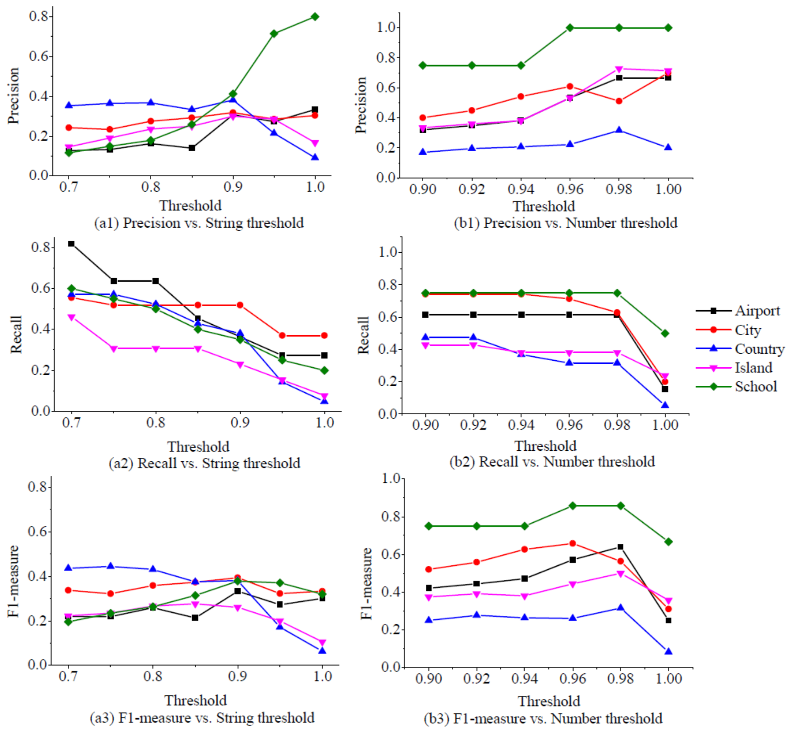


Figure 3. The Performances of SimFun with different Threshold Values

By applying other methods to align the properties between DBpeida and YAGO, the threshold values of these methods are determined according to the same principle (the greater F1-measure is, the better threshold value is). As the result, the threshold values of JaroWinkler, Levenshtein and FaITH are around 0.8, 0.5 and 1 respectively. While these methods with the corresponding threshold values are used to align the properties between DBpedia and YAGO, we record the performances of different methods and present them in Figure 4. It is clear that these methods have their respective advantages: JaroWinkler is good at precision; FaITH can recall more results; The F1-measure of SimFun is greater than others in the most cases. Taken together, the experimental results show that SimFun is more suitable for the property alignment between DBpedia and YAGO. The note about these results is that the performances of all methods are not very good, which reflects the reality that the property alignment of real-world datasets really is a difficult problem.

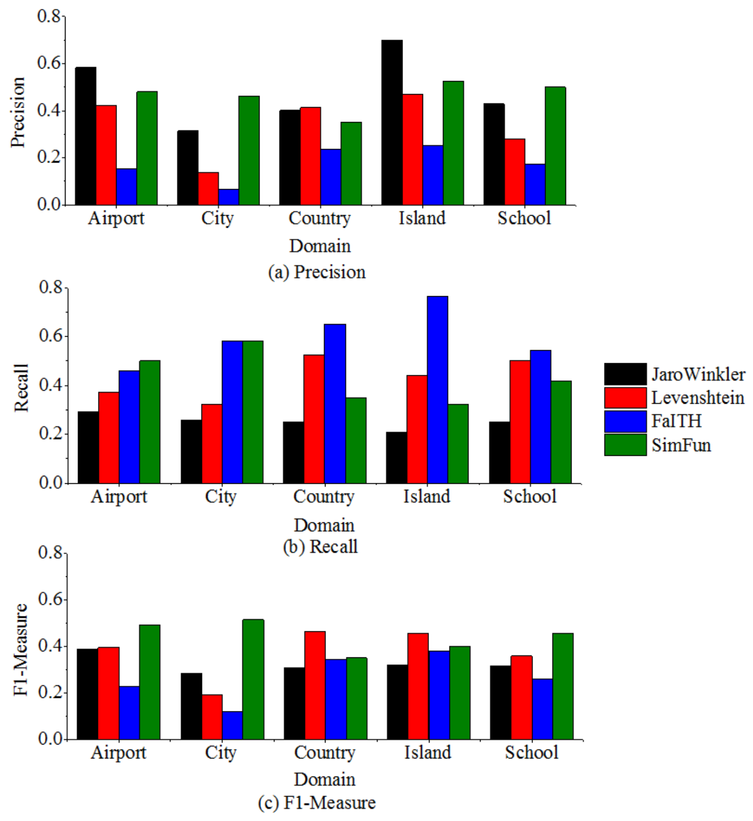


Figure 4. The Performances of different Methods (DBpedia vs. YAGO)

To more fully compare the performances of different methods, we also apply these methods with the same threshold values to align the properties between DBpeida and LGD. The results shown in Figure 5 illustrate that the alignment methods based on the property self-expression (JaroWinkler, Levenshtein, FaITH) obtain some completely different performances comparing to the performances shown in Figure 4. Although these methods adopt the same threshold values that are used in the alignment of DBpedia and YAGO, they are apt to achieve the higher recall and the much smaller precision. At the same time, SimFun can achieve the stable performances no matter what datasets are aligned. By analyzing the dataset of LGD, it is discovered that a mass of properties in LGD start with several same substrings, such as “lgd:name%3Aale”, “lgd:name%3Aprefix”, “lgd:name%3Apam”, and so on. Although these properties are literally similar, they actually have very different meanings. As the result, this kind of naming rule tends to cause plenty of mismatches when LGD is aligned with DBpedia, and then decrease the performance of precision. As far as the naming rules of object/value concerned, DBpedia, YAGO and LGD share some similar rules, especially number and date. Consequently, we can draw a conclusion that the performances of methods based on the property self-expression rely on the naming rules of property, which always have some differences between datasets, but the performance of SimFun is related with the naming rules of object/value of datasets, which usually are similar in many datasets.

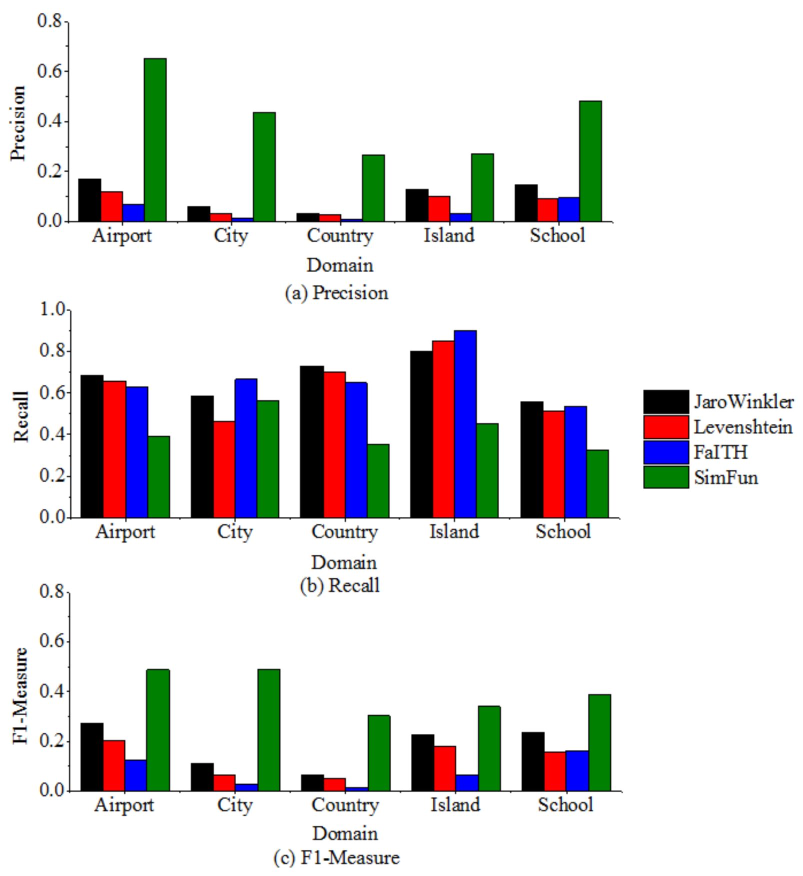


Figure 5. The Performances of different Methods (DBpedia vs. LGD)

In addition to the characteristics mentioned above, SimFun has other prominent advantages—it can retrieve the matching properties that other methods cannot find, and can identify the unmatched properties that are returned by other methods. Table 5 shows some property pairs and their similarity computed by different methods. In the header of Table 5, J, L, F, S stand for JaroWinkler, Levenshtein, FaITH and SimFun respectively. The property pairs in No.1-4 lines are regarded as the matching properties by SimFun, but are overlooked by other methods. Moreover, all methods except SimFun return the property pairs in No.5-8 lines as the matching properties, but they actually have very different meanings. Despite of these advantages, SimFun has its own drawbacks too. First, the "owl:sameAs" links still play an important role in the process of property alignment, though SimFun do not need that the objects in the triples are connected by "owl:sameAs", which is the essential condition of the method introduced in [24]. For example, a mismatch may occur when the number of subjects shared by two properties is too small, such as (dbp:sepPrecipitationInch, lgd:ele) in the domain of island. These two properties have completely different meanings, but the similarity calculated by SimFun is 0.984, because they have only one input shared by them and the values in two triples happen to be similar. More than that, some properties that are literally same are not regarded as the matching properties by SimFun, because they share none of equivalent subjects, such as (dbp:url, lgd:url) in the domain of school. Second, SimFun takes much more time to compute the similarity between properties than the alignment methods based on the property self-expression. The reason is that two properties in a property pair may share many equivalent subjects and a property function with even one input could produce lots of objects/values, which ultimately results in the large computational quantity of similarity evaluation.

Table 5. Some Property Pairs and their Similarity Computed by different Methods

No.	Property Pair	J	L	F	S
1	(dbp:country, lgd:is_in%3Anation)	0.406	0.154	0	1
2	(dbp:popEstAsOf, lgd:population%3Adate)	0.694	0.375	0.116	1
3	(geo:lat, yago:hasLatitude)	0.472	0.25	0.035	0.999
4	(dbp:population, yago:hasNumberOfPeople)	0.367	0.1	0.601	0.982
5	(dbp:countryCode, lgd:contry_code_fips)	0.941	0.706	1	0.205
6	(dbp:labelType, w3:lable)	0.9	0.5	1	0.104
7	(dbp:populaitionDensity, lgd:population)	0.911	0.556	1	0.0004
8	(dbp:officialLanguage, lgd:officialName)	0.909	0.706	1	0.019

6. Conclusion

In this paper, we propose a novel method to align properties between datasets of linked data. With regard to properties in RDF triples as functions, the problem of similarity evaluation between properties is transformed into the similarity measurement between the object/value sets, which are the output sets of property functions that take the equivalent instances as inputs. Then, we propose several calculation methods to evaluate the similarity between sets and elements of different types respectively. Besides that, the property alignment process of linked data based on similarity between functions is introduced in detail. After collecting the RDF triples and the “owl:sameAs” statements from DBpeida, YAGO and LGD, some properties in five representative domains are aligned with JaroWinkler, Levenshtein, FaITH and our method. The experimental results show that our method can guarantee freedom from the interference of property naming rules in different datasets, so that the property alignment based on similarity between functions can retrieve some matching properties that other methods cannot find, and can identify the unmatched properties that are returned by other methods. Besides that, our method requires fewer entity co-reference links than the link statistical approach introduced in [24]. Meanwhile, our method also has some deficiencies, such as the dependence on “owl:sameAs” links and the large calculating quantity. In the future, we will continue to investigate the property alignment of linked data from the following two directions. The first direction is to combine SimFun with other methods so as to improve the performance of property alignment by taking advantage of their superiorities. The second direction is to find some ways to decrease the computational quantity on the premise of approximation performances.

Acknowledgement

This work was partly supported by the National Science Foundation of China (No. 61303117), the Key Projects of National Social Science Foundation of China (No.11&ZD189), the Science Guidance Project of Education Department of Hubei Province (No. B2014085), the Science Foundation for Yong Teachers of Wuhan University of Science and Technology (No. 2012XZ015).

References

[1] I. Celino, D. D. Aglio and E. D. Valle, “Integrating Machine Learning in a Semantic Web Platform for Traffic Forecasting and Routing”, In Proceedings of the 3rd International Workshop on Inductive Reasoning and Machine Learning for the Semantic Web. Springer Berlin Heidelberg, **(2011)**.

[2] S. Tu, “Exploiting linked data to build web applications”, IEEE internet computing, vol.13, no.4, **(2009)**, pp.68-73.

[3] O. Udrea, L. Getoor and R J. Miller, “Leveraging data and structure in ontology integration”, In Proceedings of the 2007 international conference on Management of data. ACM, **(2007)**, pp.449-460.

[4] P. Shvaiko and J. Euzenat, “Ontology matching: state of the art and future challenges”, IEEE Transactions on Knowledge and Data Engineering, vol.25, no.1, **(2013)**, pp.158-176.

[5] S. Castano, A. Ferrara and S. Montanelli, “H-MATCH: an Algorithm for Dynamically Matching Ontologies in Peer-based Systems”, In Proceedings of the First Workshop on Semantic Web and Databases, **(2003)**.

[6] H. H. Do and E. Rahm, “COMA: a system for flexible combination of schema matching approaches”, In Proceedings of the 28th international conference on Very Large Data Bases. VLDB Endowment, **(2002)**.

[7] D. Aumueller, H. H. Do and S. Massmann, “Schema and ontology matching with COMA++”, In Proceedings of the 2005 international conference on Management of data. ACM, **(2005)**.

[8] L. Serafini, P. Bouquet and B. Magnini, “An algorithm for matching contextualized schemas via SAT”, In Proceedings of CONTEXT, vol.3, **(2003)**, pp.126-132.

[9] P. Bouquet, L. Serafini and S. Zanobini, “Bootstrapping semantics on the web: meaning elicitation from schemas”, In Proceedings of the 15th international conference on World Wide Web. ACM, **(2006)**.

[10] J. Euzenat “Brief overview of T-tree: the Tropes taxonomy building tool”, Advances in Classification Research Online, vol.4, no.1, **(1993)**, pp.69-88.

[11] R. Tournaire, J. M. Petit and M. C. Rousset, “Discovery of probabilistic mappings between taxonomies: Principles and experiments”, Journal on data semantics, **(2011)**, pp.66-101.

[12] W. Hu and Y. Qu, “Falcon-AO: A practical ontology matching system”, Web Semantics: Science, Services and Agents on the World Wide Web, vol.6, no.3, **(2008)**, pp.237-239.

[13] J. Li, J. Tang and Y. Li, “Rimom: A dynamic multi-strategy ontology alignment framework”, IEEE Transactions on Knowledge and Data Engineering, vol.21, no.8, (**2009)**, pp.1218-1232.

[14] K. Gunaratna, S. Lalithsena and A. Sheth, “Alignment and dataset identification of linked data in Semantic Web”, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol.4, no.2, **(2014)**, pp.139-151.

[15] P. Jain, P. Hitzler and A. Sheth, “Ontology alignment for linked open data”, In Proceedings of the International Semantic Web Conference, **(2010)**.

[16] P. Jain, P. Yeh and K. Verma, “Contextual ontology alignment of LOD with an upper ontology: a case study with proton”, In Proceedings of The Semantic Web: Research and Applications, **(2011)**.

[17] F. Suchanek, S. Abiteboul and P. Senellart, “PARIS: probabilistic alignment of relations, instances, and schema”, In Proceedings of VLDB Endow 2011, **(2011)**.

[18] R. Parundekar, C. Knoblock and J. Ambite, “Discovering concept coverings in ontologies of linked data sources”, In Proceedings of the International Semantic Web Conference, **(2012)**; Springer Berlin Heidelberg.

[19] J. Volz, C. Bizer and M. Gaedke, “Silk–a link discovery framework for the web of data”, In Proceedings of the 2nd Linked Data on the Web Workshop, **(2009)**.

[20] S. Araujo, J. Hidders and D. Schwabe, “SERIMI resource description similarity, RDF instance matching and interlinking”, Journal of Computing Research Repository, **(2011)**, pp.236-248.

[21] N. A. Ngonga and K. Lyko, “EAGLE: efficient active learning of link specifications using genetic programming”, In Proceedings of The Semantic Web: Research and Applications, **(2012)**.

[22] L. Zhao and R. Ichise, “Mid-ontology learning from linked data”, The Semantic Web, Springer Berlin Heidelberg, **(2012)**, pp.112-127.

[23] L. Zhao and R. Ichise, “Ontology Integration for Linked Data”, Journal on Data Semantics, **(2014)**, pp.1-18.

[24] K. Gunaratna, K. Thirunarayan and P. Jain, “A statistical and schema independent approach to identify equivalent properties on linked data”, Proceedings of the 9th International Conference on Semantic Systems. ACM, **(2013)**.

[25] H. Huang, C. Liu and X. Zhou, “Approximating query answering on RDF databases”, World Wide Web, vol.15, no.1, **(2012)**, pp.89-114.

[26] S. H. Cha, “Comprehensive survey on distance/similarity measures between probability density functions”, International Journal of Mathmatical Models and Methods in Applied Sciences, vol.1, no.2, **(2007)**, pp.300-307.

[27] M. Cheatham and P. Hitzler, “String similarity metrics for ontology alignment”, In Proceedings of the International Semantic Web Conference, **(2013)**; Springer Berlin Heidelberg.

[28] P. Giuseppe and E. Jérôme, “A Feature and Information Theoretic Framework for Semantic Similarity and Relatedness”, In Proceedings of the 9th International Semantic Web Conference, **(2010)**; Springer.

[29] J. Euzenat and P. Shvaiko, “Overview of matching systems”, Ontology Matching, **(2007)**, pp.153-192.

Authors

Yu Liu, he received the master degree in School of Computer Science and Technology of Huazhong University of Science and Technology. He is a PhD candidate in Computer School of Wuhan University. His research interests are semantic web and knowledge engineering.

Shihong Chen, he is a professor in Computer School of Wuhan University, China. He also is the deputy director of the national engineering research center for multimedia software. His research interests are software engineering and knowledge engineering.

Jin-Guang Gu, he is a professor at College of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan, China. His current research interests include distributed computing, intelligent information processing, semantic web and software engineering.