An Efficient Matching Algorithm for Fuzzy RDF Graph* 

GUANFENG LI1 AND ZONG-MIN MA2, * 

1College of Computer Science and Engineering, Northeastern University 
Shenyang, 110819, China 

2College of Computer Science & Technology, Nanjing University of Aeronautics and Astronautics 
Nanjing, 211106, China 

The rapid growth of RDF (Resource Description Framework) data shows a steady trend of decentralization. Identify correspondences among these data sources is an important task. Concentrating on the RDF data with fuzzy information, in this paper, we propose a unified framework combined multiple measures of similarity for automatically solving the problem of fuzzy RDF data matching. For this purpose, syntactic and semantic similarities are calculated based on element labels of RDF graph. Motivated by the structural characteristics of fuzzy RDF graph, we introduce an effective similarity measure approach by interactively utilizing structural information, in which we take fuzzy edge values and edge similarities into consideration. It is proved that the iterative computation of the proposed similarity approach converges. Finally, we combine these results of the individual metrics and extract the correspondences on the basis of the total similarities. The experimental results show that our method can effectively identify correspondences among the fuzzy RDF graphs.

Keywords: fuzzy RDF, graph matching, similarity, iterative computation

1. INTRODUCTION

Data matching is the process of bringing data from different data sources together and comparing them in order to find out whether they represent the same real-world object in a given domain [6]. Data matching processes can be applied to many data management research areas, such as a query or search over web data sources, integration of heterogeneous data sources, and so on [43, 44, 53]. In the context of the Semantic Web, there has been a growing interest in using RDF (Resource Description Framework) data model, a recommendation by the World Wide Web Consortium (W3C) [13] for describing resources on the Web. The increasing amount of RDF data that are captured in different information stores has resulted in a strong need for automatically discovering correspondences among heterogeneous data source. Efficient RDF data matching has received more significant attention from both academic and industrial communities [20, 26, 10, 40].

RDF data have a natural representation in the form of labeled directed graphs, in which the vertices represent the resources and values (also called literals), and edges represent semantic relationships between resources. So, RDF data matching problem has been often addressed in terms of graph matching approach. Unfortunately, it has been proved that the complexity of the traditional graph matching algorithm based on isomorphic graph is NP-complete [25]. For this reason, the approaches to approximate matching on graph data mainly rely on similarity metrics to reduce the problem com-
plexity. Some existing RDF data matching methods are based on the linguistic information [47]. Because of the independent encoding system in different data sources and the insufficient linguistic information, directly applying existing linguistic-based techniques (evaluation of different string similarity metrics) to compare element (e.g., a vertex or edge) labels may obtain false mappings of RDF data. Furthermore, these approaches ignore the fact that a part of the vertices are blank vertices (vertices without URIs) in RDF graph. Structural similarity should be considered in addition to the classic typographical similarity. The basic idea is to construct RDF graph for describing the structural relationships among RDF resources and employ the topological properties of RDF graph for obtaining the correspondences pairs. There is a kind of structural similarity approaches [22], in which an element in graph $G_T$ and an element in graph $G_S$ are considered similar if their respective neighborhoods within $G_S$ and $G_T$ are similar. This kind of approaches is particularly suitable for the lack of syntactical information (blank vertices in our case) in graphs. But the computing of similarity degrees for the elements of these graphs needs an iterative processing, in which similarity degrees between elements propagate along to neighboring elements at each iterative step. Note that this kind of approaches concerns only the pure structural similarity, while the utilization of other evaluation methods such as string similarities is not considered. Furthermore, it does not take edges similarities into consideration, which are key properties in RDF graph.

In the real-world applications, information is usually vague or ambiguous. Some data are inherently fuzzy since their values are subjective in real applications [59]. Considering values representing the satisfaction degree for a film, clearly different individuals may have different satisfaction degrees. It is desired to represent and deal with fuzzy data with RDF. For this purpose, several recent efforts have devoted to develop fuzzy RDF model [46, 56]. However, these proposals lack enough expressive power to properly describe multiple granularity of data fuzziness in RDF. We argue that the semantics of an RDF graph relies on the connectivity of the resources described. This is not properly taken into account in a mere triple vagueness. Therefore, it appears natural to allow resources as a fuzzy concept and not only triple. In generally, the subject, the object, and predicate of a triple do not always need to be a crisp resource, but they can also be fuzzy concepts. Hence, we propose a general fuzzy RDF graph model that combines two common types of fuzziness based on fuzzy graph theory [24]. Specifically, we consider:

1. Fuzziness about the label values of vertices (i.e., vertices label value fuzziness).
2. Fuzziness about whether particular edges exist (i.e., edge existence fuzziness).

Like RDF data matching in the classical RDF data, fuzzy RDF data matching is a fundamental problem in the integration of fuzzy RDF data. Based on the fuzzy RDF data model, in this paper, we propose a graph matching approach over the fuzzy RDF graphs. To effectively tackle the graph matching problems mentioned above, we present a general framework for fuzzy RDF graph matching. The framework computes multiple measures of similarity among graph elements: syntactic, semantic and structural. These measures are composed in a principled manner for graph matching. In particular, an iterative similarity function is introduced with the consideration of structural information of fuzzy RDF graph. Moreover, we prove that the iterative computation of the proposed similarity function converges. To the best of our knowledge, the present paper is the first effort to investigate the issue of fuzzy RDF data matching.
The main contributions of this paper are summarized as follows:

(1) We introduce a general fuzzy RDF graph model that can capture fuzziness in the vertex and edge. We formalize the problem of fuzzy RDF graph and graph matching.

(2) We propose a graph matching approach based on a unified framework to generate graph alignments over the proposed fuzzy RDF graph model.

(3) We define the structural similarity function based on the idea of propagating similarities, which is iteratively computed by incorporating fuzzy edge similarity criteria in the iteration.

(4) We evaluate our approach on synthetic datasets. It is shown that our approach can effectively distinguish the differences of size and content among the RDF graphs. Experimental results demonstrate that our approach has significantly higher accuracy.

The rest of this paper is organized as follows. In Section 2, we give a brief overview of related work on RDF data matching and uncertain RDF data. In Section 3, we introduce some preliminary notions and definitions. In Section 4, we proposed a unified framework that combines multiple measures of computing similarity between fuzzy RDF graphs. We present the experimental evaluation of our approach in Section 5. In Section 6, we draw conclusions and sketch some future work.

2. RELATED WORK

The present work in this paper is closely related to the issues of classical RDF matching and uncertain RDF modeling.

2.1 RDF Graph Matching

There are a number of efforts in RDF data matching. Basically, we can identify three types of approaches for RDF data matching: schema-based mapping, instance-based matching and structure-based methods.

Schema-based methods rely on alignments or mapping rules to identify the same real-world objects. Nikolov et al. [2] proposed a data fusion method based on mapping rules to resolve the semantic heterogeneity of datasets. Zhu et al. [18] introduced an approach for semantic search by matching RDF graphs and the similarity definition was based on the ontology which consists of type hierarchies on vertices and edges.

Instance-based matching methods seek to identify the instances of a real-world object in two different datasets by using string similarities such as edit distance [47], Jaro similarity [1], and JaroWinkler similarity [51]. For instance, Araujo et al. [43] proposed an approach called SERIMI to solve the RDF instance-matching problem automatically. Without prior knowledge of the data, domain or schema of these datasets, SERIMI approximates the notion of similarity and match instances between a source and target datasets by pairing instances based on entity labels as well as structural context. Liu et al. [55] developed a RDF dataset fusion method based on the similarity of literal contents and the graph structure of the RDF dataset. Schultz et al. [40] provided a linked data integration framework based on Silk-Link Specification Language [41], which can identify the same real-world entities.

Structure-based methods usually employ the topological properties of a RDF graph. For testing RDF software, Carroll [20] proposed a RDF graph matching algorithm based
on vertices of classification and graph isomorphism. For the purpose of data dissemination, Wang et al. [26] described an effective algorithm to match RDF graph pattern and RDF graph exactly, and proposed an ontology-based publish/subscribe system. Zhang et al. [10] proposed a similarity-oriented RDF graph matching approach for ranking linked data, which considers the element-level and structure-level similarity of statements. Yves et al. [57] proposed an algorithm that considers both the similarity of the resources and the similarity of their neighbors for matching music-related datasets on the Web. Matching evidence was propagated through other resources via object properties, in a similar manner to the similarity flooding algorithm proposed in [22]. The basic idea is that the corresponding elements in a graph are similar when their adjacent elements are similar.

Several application-oriented algorithms, such as behavioral similarity [44] and heterogeneous events matching [53], also utilized the idea of recursively computing vertex similarity degrees based on the degree of neighboring vertices.

The matching approach proposed in the paper combines the semantic similarities of fuzzy RDF label data and structural heuristics to generate the overall similarities. As far as the semantic similarity measure of fuzzy RDF data, one of the most useful approaches is the general framework provided by Granular Computing (GrC) [42]. Granular computing is a general computation theory for effectively using granules such as classes, clusters, subsets, groups, and intervals to build an efficient computational model for complex applications with huge amounts of data and knowledge. This technology allows for a high level of man-machine cooperation by providing a framework in which concepts can be modeled in a manner amenable to both. Currently granular computing has been widely investigated in the literature [3, 5, 8, 9, 12, 14, 15, 17, 23, 27, 28, 29, 34, 38, 48, 49, 58, 60].

2.2 Uncertain RDF Models

All the aforementioned works assume that the RDF graph data are reliable and precise. Since RDF data may often contain inconsistent or imprecise information in the real-world applications, it is important to investigate the combination of RDF and uncertainty. Currently there have been few achievements in uncertain RDF, which mainly focus on the issues of representing and querying uncertain RDF data.

First, probability theory has recently been applied to represent and manipulate uncertain RDF data. Fukushige [54] extended the RDF vocabulary to allow probabilistic confidence existing in RDF data. Huang et al. [16] modeled uncertainty RDF data by a probabilistic database. Their model assumes that RDF triples (corresponding to edges in RDF graphs) have independent existence probabilities to appear in reality. Based on this assumption, their matching method is to decompose the query graph into triple patterns (edges), compute the probabilities of candidates for each triple pattern, and finally join the candidates by calculating their actual probabilities. In contrast, Lian et al. [52] proposed a different model of probabilistic RDF graphs, which has correlated and uncertain vertex labels rather than uncertain edges. They also proposed efficient matching algorithms with vertex attribute correlations. Moustafa et al. [50] identified three common types of uncertainties in attribute values, edge existences and identities, and proposed a general probabilistic graph model. They further developed techniques for efficiently answering subgraph pattern queries over the uncertain graphs.
Second, fuzzy set theory has recently been used for representing fuzzy RDF data [32, 33, 56, 44, 37, 4]. In [32], a syntactic and semantic extension of RDF was proposed to represent fuzzy data. The syntax is extended from a triple to a quadruple by adding a value to the triple. Such a value is applied to represent the fuzzy truth-value of the triple. The semantics for fuzzy RDF and fuzzy RDFS allows deriving truth values for derived statements as well. Straccia [44] presented a minimal deductive system for fuzzy RDF in a general setting, in which the triples were annotated with a degree of truth in [0, 1]. Hartig [35] defined semantics for fuzzy RDF graphs formed by associating trust degree to triplets and further proposed query evaluation algorithms for fuzzy RDF data with tSPARQL. Zimmermann et al. [4] described a generic framework for representing and reasoning with annotated Semantic Web data. They extended the works of [37] towards various annotations – not only temporal, but also fuzzy, confidence, provenance and etc. They also formalized the annotated language and the corresponding deductive system, and addressed the query answering problem with a query language named AnQL, which includes several features of SPARQL 1.1 along with the formal definitions of their semantics. A recent work [36] proposed a fuzzy extension of SPARQL to query fuzzy RDF data. They established a fuzzy RDF graph data model by using fuzzy graph theory.

Basically, the above-mentioned works of fuzzy RDF add a fuzzy degree to an edge, expressing the extent to which the fuzzy concept attached to the edge is satisfied. Although these works have addressed the problems of representing and querying fuzzy RDF data, they have a limitation in fuzzy RDF modeling. That is, a triple is associated with a fuzzy membership value, but the items of the triple have crisp values. A novel fuzzy RDF data model was proposed in [56], which takes the fuzziness in values of the triple’s items into consideration. This work focuses on how to represent fuzzy RDF data rather than efficient matching of fuzzy RDF data. Our work goes towards filling this gap.

3. FUZZY RDF GRAPH MODEL AND MATCHING SEMANTICS

In this section, we present the fuzzy RDF graph model and discuss the semantics of fuzzy RDF graph matching.

3.1 Fuzzy RDF

RDF is a data model for the Semantic Web, the contents of which are machine-readable. It uses URI references to identify things and uses RDF triple to make a statement. Herein, we give a brief walkthrough of the core concepts of RDF from a database point of view. More definitions can be found in the work [30]. Let \( U \) be a set of URI references, \( L \) a set of literals and \( P \) a set of properties, an RDF triples is defined as follows.

**Definition 1 (RDF triple).** An RDF triple is a triple \((s, p, o)\) in \( U \times P \times (U \cup L) \), where \( U, P \) and \( L \) are sets of URI references, properties and literals respectively. The element of the RDF triples \( s, p, \) and \( o \) is called the subject, property (also called predicate) and object respectively.

**Definition 2 (RDF graph).** A finite set of RDF triples \( G_{\text{F}} \) is called a RDF graph in which every triple \((s, p, o)\) describes a directed edge labeled with \( p \) from the vertex labeled with \( s \) to the vertex labeled with \( o \).
As the resource description language of the Semantic Web, RDF is playing an increasingly important role. In addition, imprecise data has become an emerging topic for various applications. In order to model imprecise information in real-world applications, we extend RDF to accommodate different types of fuzzy information applying fuzzy set theory. In a fuzzy RDF, there may be two kinds of fuzziness: one is the *fuzziness in triples*, i.e., a fuzzy membership degree associated with a RDF triple represents the amount of disagreement on the corresponding assertion (statement); the other one is the *fuzziness in the element level of triple*, i.e., we do not know the crisp value of an element, and the value of the element may be represented by a fuzzy set. Note that, we only consider the latter in the paper.

**Definition 3 (Fuzzy RDF triple).** A triple \((μ_s/s, μ_p/p, μ_o/o)\) is called a fuzzy RDF triple, where \(s\) is fuzzy subject and \(μ_s ∈ [0, 1]\) denote the membership degree of subject to the universe of a RDF graph, \(p\) is a fuzzy predicate and \(μ_p ∈ [0, 1]\) express the fuzzy degree to the property or relationship being described, \(o\) is fuzzy object and \(μ_o ∈ [0, 1]\) represent the fuzzy degree of the value for the property or relationship.

**Definition 4 (Fuzzy RDF data graph).** Fuzzy RDF data graph \(G\) is represented by a 6-tuple \((V, E, Σ, L, μ, ρ)\) where \(V\) is a finitude set of vertices, \(E ⊂ V_i × V_j\) is a set of directed edges, \(Σ\) is a set of labels, \(L: V ∪ E → Σ\) is a function assigning labels to vertices and edges respectively, \(μ: V → (0, 1]\) is a fuzzy subset, and \(ρ: E → (0, 1]\) is a fuzzy relation of on fuzzy subset \(μ\). Note that \(ρ(v_i, v_j) ≤ μ(v_i) ∧ μ(v_j)\) \(v_i, v_j ∈ V\).

In Definition 4, each vertex \(v_i ∈ V\) of graph \(G\) has one label, \(L(v_i)\), corresponding to either *subjects or objects* in RDF triples datasets. Moreover, \((v_i, v_j) ∈ E\) is a directed edge from vertex \(v_i\) to vertex \(v_j\), with an edge label \(L(v_i, v_j)\) that corresponds to the predicate in RDF triples. The label value of vertex is associated with a fuzzy degree indicating the possibility that vertex take the label, and the fuzzy value associated with each edge represents the amount of disagreement on the corresponding relationship between vertices. A fuzzy RDF data graph may contain both fuzzy vertices (resp. edges) and crisp vertices (resp. edges) as a fuzzy vertex (resp. edge) with a degree of 0 or 1 can be considered as crisp. Along the same line, a crisp RDF graph is simply a special case of fuzzy RDF data graph (where \(μ: V → \{0, 1\}\) for all \(v_i ∈ V\) and \(ρ: V × V → \{0, 1\}\) for all \((v_i, v_j) ∈ E\), and the fuzzy RDF graph is a generalization of the crisp RDF graph.

**Definition 5 (Predecessor set and successor set).** Let \(G\) be a fuzzy RDF graph, for any vertex \(v ∈ V\) of graph \(G\), \(pre(v) = \{v' | (v, v') ∈ E\}\) is the predecessor set (i.e., forward neighbors) of \(v\) and \(suc(v) = \{v' | (v', v) ∈ E\}\) is the successor set (i.e., backward neighbors) of \(v\).
AN EFFICIENT MATCHING ALGORITHM BASED ON FUZZY RDF GRAPH

For example, Fig. 1 (a) and (b) illustrates two fragments of fuzzy RDF data graph with some fuzzy elements, and crisp ones (equivalent to a corresponding degree of 1). The edge "pid2-has-address-addid2" associated with fuzzy degree 0.5 in target graph Fig. 1 (a) represents the fact that the person labeled pid2, whose address label is addid2. And the possibility of the fact is 0.5. In this example, the degree is based on a simple statistical notion, which can be made of subtler by fuzzy operations or the integration of expert knowledge. Note that opaque labels exist as shown in Fig. 1 (b). The resource "_:" is distinct from others, and it makes the resource name garbled. According to the RDF specification [11], a blank vertex can be assigned an identifier prefixed with "_:".

With the presence of dislocated matching [53], any vertices in a RDF graph can be a starting/ending vertex. That is, a path trace can start/end with any vertex \( v \) within it by ignoring those vertices before/after vertices \( v \). Furthermore, in order to conveniently process fuzzy degrees of vertex, we could convert the vertex fuzzy degree into the corresponding edge fuzzy degree in the case of retaining its fuzzy semantics. Based on these aforementioned intuitions, we add the following restriction on RDF graphs:

1. There is one and only one vertex in the RDF graph that is called the home vertex, denoted by \( \hat{v} \), which indicates the virtual beginning/end of all paths in a RDF graph. We specify that the label of the home vertex is "_:H", i.e., \( L(\hat{v}) = "_:H" \).  
2. There are paths from the home vertex to any other vertices in the fuzzy RDF graph. That is, for each vertex \( v \in V \) except \( \hat{v} \), we add two edges \((v, \hat{v})\) and \((\hat{v}, v)\). Thus, a path can begin or end with vertex \( v \) at all the locations where \( v \) occurs. Moreover, we associate \( \rho(v, \hat{v}) = \rho(\hat{v}, v) = \mu(v) \), i.e., we regard the fuzzy degree associated with each vertex represents the possibility that the vertex exists in the graph as the fuzzy degree of edge from home vertex to the vertex.

3.2 Fuzzy RDF Graph Matching Semantics
RDF graph matching refers to the process of determining whether two descriptions refer to the same real-world entity in a given domain. For the purpose of describing our approach clearly, we distinguish the two fuzzy RDF graph involved in the matching process, source graph (also input graph) and target graph (also template graph), and denoted by $G_S$ and $G_T$ respectively. For a fuzzy RDF graph matching, the objective is to investigate the syntactic and semantic occurrence of the target graph in the source graph.

**Definition 6 (Fuzzy RDF graph matching).** Given two fuzzy RDF graphs $G_S$ and $G_T$ from a given domain, the matching problem is to identify all correspondences between graphs $G_S$ and $G_T$ representing the same real-world object. The match result is typically represented by a set of correspondences, sometimes called a mapping. A correspondence $c = (id, E_s, E_t, m)$ interrelates two element $E_s$ and $E_t$ from graphs $G_S$ and $G_T$. An optional similarity degree $m \in [0, 1]$ indicates the similarity or strength of the correspondence between the two elements.

An essential requirement for any graph matching algorithm is a method for calculating similarity between elements of the target and source graphs. What criteria should we use for calculating similarity? How can we quantify these criteria? To obtain a satisfactory correspondence relation, RDF graph matching process employs a similarity function to compare elements values.

**Definition 7 (Similarity function).** Let $G_S$ and $G_T$ are two datasets, a similarity function is defined as: $F(s, t) \rightarrow [0, 1]$, where $(s, t) \in G_S \times G_T$, i.e., the function computes a normalized value for every pair $(s, t)$. The higher the score value, the more similar $s$ and $t$ are. The advantage of using similarity functions is to deal with a finite interval for the score values.

A matching algorithm computes the similarity of any two elements in $G_S$ and $G_T$ and selects the most similar element pairs. In the paper, we compute vertex similarity and structural similarity of the RDF graph and use the similarity obtains the correspondences pairs between two fuzzy RDF graphs. Thus, we can find the matching relationship between RDF datasets. Next, we present the approach to automatically generate the correspondences in the fuzzy RDF graphs.

### 4. FUZZY RDF GRAPH MATCHING APPROACH

This section presents the details of the fuzzy RDF graph matching approach.

#### 4.1 Framework

Our match algorithm uses a hybrid approach combining the static label and structural heuristics to generate the overall similarity degrees between vertices of the target and source graphs. Static label similarity is independent of the RDF semantics and combines typographic and semantic similarity degree between elements label of graph, respectively. Structural matching generates similarity degree between elements based on their structural semantics.

Before describing the details of the matching algorithm, we present in the subsection a general framework for matching fuzzy RDF graph. Matching procedure takes as input two fuzzy RDF graphs and outputs a set of correspondences of the two graphs. Fig. 2 illustrates an overview of the framework and it has three main stages: In the first phase,
the procedure is to compute a vertex-to-vertex similarity score using different similarity functions for each vertex in the target graph to each vertex in the source graph. These similarity functions are divided into two categories: label similarity function and structural similarity function. Label similarity functions adopt different computation strategies to compute multiple types of vertex label similarities and generate a vector of similarity scores. Structural similarity function iteratively computes a similarity score for every vertex pair by aggregating the edge similarity scores and the immediate neighbors of vertices. The algorithm iterates until either the similarity scores between all vertices pairs stabilize or a maximum number of iterations are reached. Then, in the second phase, we obtain the overall similarity by combining label similarity scores and structural similarity scores. After pair-wise similarities of vertices are computed, phase 3 of the procedure is to construct the correspondence based on the similarity scores. We select the potential correspondences and include them in the alignment.

The main question that arises is how to define the similarity function between elements of the RDF graph in the first phase. We compute labels similarity scores of elements of RDF graph by applying string similarity and semantic similarity. The inherent hardness in structural similarity function is how to aggregate edge label similarity scores and fuzzy properties of fuzzy RDF graph. Furthermore, the challenge is to combine the elements labels similarity, structural features and obtain a correspondence relation between input RDF graphs. We will describe the detailed steps in the following paragraphs.

4.2 Label Similarity Function

In the RDF world, there are two aspects that can be measured to produce a similarity measure between two RDF data: the syntactical and semantic aspects [43]. The syntactical aspect can be considered by applying string similarity functions or set similarity, while semantic similarity function uses external sources of knowledge during the process of matching. We use these two heuristics to define similarity metrics in the paper.

4.2.1 Syntactic Similarity

Intuitively, the element label denoting an element typically captures the most distinctive characteristic of the element in the RDF graph model. The syntactic similarity assigns a normalized similarity value to every pair \((s, t)\) by applying the Levenshtein distance [47] to the name labels of \(s\) and \(t\).

Formally the syntactic similarity \(\text{sim}_{\text{sy}}(s, t)\) between two name labels \(s\) and \(t\) is defined as follows.
Here \( s.label \) and \( t.label \) denote the name label of \( s \) and \( t \), respectively, \( \max(|s.label|, |t.label|) \) is the max length of the name string in \( s \) and \( t \), and \( LD(w_1, w_2) \) is the Levenshtein distance between two words \( w_1 \) and \( w_2 \).

The next limit shows that the width gets to 1 if the Levenshtein distance gets to 0:
\[
\lim_{LD(s,t) \to 0} \text{sim}_y(s,t) = 1 - 0 = 1
\]
The next limit shows that the width gets to 0 if the Levenshtein distance gets to 1:
\[
\lim_{LD(s,t) \to \infty} \text{sim}_y(s,t) = 1 - 1 = 0
\]
These two limits show that we have the width value \([0, 1]\).

For example, the result of syntactic similarity to some of the elements labels of the RDF graph in Fig. 1 is as following:
\[
\begin{align*}
\text{sim}_y("pid1", "pid2") &= 1 - 1/4 = 0.75 \\
\text{sim}_y("has-address", "address") &= 1 - 4/11 = 0.64 \\
\text{sim}_y("Furlan", "Furlan") &= 1 - 0/6 = 1.0 \\
\text{sim}_y("addid1", "_:_") &= 1 - 6/6 = 0
\end{align*}
\]

### 4.2.2 Semantic Similarity

For semantic similarity, we use WordNet::Similarity package [45] to get the semantic relatedness between element labels based on their linguistic correlations. In our work, we use Jaccard similarity [51] measures. In many case, the element labels whose relatedness is being measure are phrase or short sentence, e.g., “house-number” and “room number” in Fig. 1. In these cases, we need a new measure that computes degree for element labels expressed as a sentence or phrase. To this end, we use a simple measure from the natural language processing. The formula is shown as follows:
\[
\text{sim}_e(s,t) = 1 - \frac{1}{|s.tok|} \sum_j \max(Jaccard(s.tok_j), Jaccard(t.tok_j))
\]  

Here \(|s.tok|\) is the number of tokens in the name of \( s \), \( Jaccard \) denotes the Jaccard similarity between two sets, and \( syn(w) \) denotes the WordNet synset of a token \( w \).

Based on the Jaccard similarity, it is obviously proved that the semantic similarity in formula (2) is convergence. We summarize the procedure of computation of semantic similarity in Algorithm 1.

---

**Algorithm 1. Semantic similarity algorithm**

Input: two name labels \( s \) and \( t \)  
Output: semantic similarity \( \text{sim}_e \)  
1: \( s.tok \leftarrow \text{Tokenize}(s) \)  
2: \( t.tok \leftarrow \text{Tokenize}(t) \)  
3: for each \( s.tok_i \in s.tok \) do  
4: \( s.tokw_i \leftarrow \text{syn}(s.tok_i) \)  
5: for each \( t.tok_j \in t.tok \) do  
6: \( t.tokw_j \leftarrow \text{syn}(t.tok_j) \)  
7: \( \text{sim} \leftarrow 0 \)  
8: for \( i = 1 \) to \(|s.tok|\)
The algorithm takes two name labels as input and computes their semantic similarity as output. In Algorithm 1, we begin by tokenizing the names of both labels in lines 1-2. We use tokenize to split a phrase or short sentence into a sequence of words, in which \(s_.tok\) means the \(i\)-th token of the name label \(s\). Then we call the function syn to search the synset of each token by applying WordNet in lines 3-6, in which \(t_.tok\) means the WordNet synset of a token \(t\). We further use the Jaccard similarity to calculate the maximum semantic similarity on the synsets of each pair of tokens in lines 8-10. Finally, we calculate the average-max semantic similarity in line 12, and return the average-max Semantic similarity as the result in line 13.

Let us look at an example. Suppose that we would calculate the similarity degree between two labels “house-number” and “room number”. First, two name labels are tokenized into two sets \{“house”, “number”\} and \{“room”, “number”\}, respectively. Second, we retrieve the synsets of “house”, “number” and “room” from WordNet. Then we calculate the Jaccard similarity on the synsets of each pair of tokens. Finally, we find the maximum Jaccard similarity for each token in the label and average the similarity over all tokens in the first label name. We have \(sim_{se} = (Jaccard(syn("house"), syn("room")) + Jaccard(syn("number"), syn("number"))) / 2 \approx 0.85\).

### 4.3 Structural Similarity Function

Computing similarity iteratively based on the graph structure has been proposed in \([44, 53]\). These works deal with the pairing of vertices across two graphs and rely on domain-specific metadata (e.g., vertex and edge labels) as well as structural relationships. The principle of the algorithm is that the similarity between two vertices must depend on the similarity between their adjacent vertices. Inspired by these works, in this section, we propose a structural similarity function for matching RDF data, in which we further take edge similarity and edge fuzzy value into consideration.

#### 4.3.1 Structural Similarity Function

Intuitively, to provide a natural way to match fuzzy RDF graph, our matching approach draws on the notion of bi-similarity between vertices. Bi-similarity is a recursive notion and can be defined in a forward and backward way \([39]\). Let \(v_i\) be a vertex of fuzzy RDF graph \(G_i\), i.e. \(v_i \in G_i\), \(v_i \in \text{pre}(v_i)\), and \(e_i \in E_i\) is a directed edge from vertex \(v_i\) to vertex \(v_i\). Our matching method iteratively computes a similarity degree for every vertex pair \((v_i, v_i)\) of two fuzzy RDF graphs by aggregating the similarity degrees between the immediate neighbors of \(v_i\) and \(v_i\). By neighbors, we mean either successor or predecessor, depending on which bi-similarity notion is being used. The method iterates until either the similarity degree between all vertex pairs stabilizes or a maximum number of iterations is reached. In the remainder of this section, we describe the method for the forward case, and the backward case is similar.

Although the forward neighbors \(v_i\) and \(v_i\) of \(v_i\) and \(v_i\), respectively, have high sim-
ilarity degree, if the similarity of edge \( e_i \) is different from the similarity of \( e_t \) or the fuzzy membership \( \rho_i \) deviates far from the fuzzy membership \( \rho_t \), the similarity degree of \( v_t' \) and \( v_i' \) will have less effect on the similarity degree of \( v_t \) and \( v_i \). For proper aggregation of similarity degrees between vertices of two fuzzy RDF graphs, we further take edge labels similarity and edge fuzzy membership into consideration. We compare edge labels using label similarity functions defined in section 4.2. These similarity functions assign a similarity degree \( \text{sim}_s(e_s, e_t) \) to every edge labels pair \((e_s, e_t)\).

The notion of approximation of fuzzy values is important in applications of fuzzy sets. In our work, we use a measure of similarity of fuzzy values \[7\], which is based on the difference as well as the sum of corresponding grades of membership. Let \( \rho_i \) and \( \rho_t \) are the fuzzy membership degrees of edges \( e_i \) and \( e_t \), respectively. The similarity of fuzzy values \( \rho_i \) and \( \rho_t \) is defined by

\[
\text{sim}_s(\rho_i, \rho_t) = 1 - \frac{(\rho_i \lor \rho_t)-(\rho_i \land \rho_t)}{\rho_i + \rho_t}
\]  \hspace{1cm} (3)

Here \((\rho_i \lor \rho_t) = \max(\rho_i, \rho_t)\), \((\rho_i \land \rho_t) = \min(\rho_i, \rho_t)\).

Having described edge comparison, we now describe the computation of forward similarity. For every vertex pair \((v_i, v_t)\), the similarity degree \( \text{Sim}(v_i, v_t) \) is computed from 1) similarity degrees between \( v_i \) and \( v_t \) after step \( i-1 \), i.e., \( \text{Sim}^{-1}(v_i, v_t) \); 2) similarity degrees between the forward neighbors of \( v_i \) and those of \( v_t \) after step \( i-1 \), i.e., \( \text{Sim}^{-1}(v_i', v_t') \); 3) similarity degrees between the edge labels relating \( v_i \) and \( v_t \) to their forward neighbors, i.e., \( \text{sim}_s(e_t, e_i) \); 4) similarity degrees between the edges fuzzy values of \( e_t \) and those of \( e_i \).

To find the best match for \( v_i \) among the forward neighbors of \( v_t \), we need to maximize the value \( \text{sim}_s(e_t, e_i) \times \text{sim}_s(\rho_i, \rho_t) \times \text{Sim}^{-1}(v_i', v_t') \). Hence, similarity degrees between the forward neighbors of \( v_i \) and their best matches among the forward neighbors of \( v_t \) after the \( i \)th iteration is computed by

\[
\text{sim}(v_i, v_t) = \frac{1}{|\text{pre}(v_t)|} \sum_{v_i' \in \text{pre}(v_t)} \max_{v_i \in \text{pre}(v_t)} \text{sim}_s(e_t, e_i) \times \text{sim}_s(\rho_i, \rho_t) \times \text{Sim}^{-1}(v_i', v_t') \hspace{1cm} (4)
\]

Here \( v_t \in \text{V}_t \), \( v_i \in \text{pre}(v_t) \), \( e_i \in \text{E}_i \) is the edge of vertices pair \((v_i, v_t)\), and \( \rho_i \) is the fuzzy membership of \( e_i \), \( \text{sim}_s(e_t, e_i) \) and \( \text{sim}_s(\rho_i, \rho_t) \) are the similarity degree of edge label and fuzzy membership, respectively.

And another side similarity degree, i.e., the similarity degrees between the forward neighbors of \( v_t \) and their best matches among the forward neighbors \( v_i \) of after iteration \( i \) are computed by

\[
\text{sim}(v_t, v_i) = \frac{1}{|\text{pre}(v_i)|} \sum_{v_t' \in \text{pre}(v_i)} \max_{v_t \in \text{pre}(v_i)} \text{sim}_s(e_t, e_i) \times \text{sim}_s(\rho_i, \rho_t) \times \text{Sim}^{-1}(v_i', v_t) \hspace{1cm} (5)
\]

Note that this \( \text{sim} \) measure is asymmetric, i.e., \( \text{sim}(v_t, v_i) \neq \text{sim}(v_i, v_t) \).

The value of \( \text{Sim}(v_i, v_t) \) is computed by first taking the average of \( \text{sim}(v_t, v_i) \) and \( \text{sim}(v_i, v_t) \), and then computing its average with \( \text{Sim}^{-1}(v_i, v_t) \). In conclusion, we define the forward similarity degrees of vertex pair \((v_i, v_t)\) after the \( i \)th iteration as follows:
\[ Sim'(v_i, v_j) = \frac{((Sim'(v_i, v_j) + Sim'(v_j, v_i)) / 2 + Sim^{i-1}(v_i, v_j)) / 2}{2} \]  

In the above formula, The larger the \(Sim'(v_i, v_j)\), the greater the structural similarity of \(v_i\) and \(v_j\). For backward similarity degrees calculating, we perform the above formula for vertex \(v_i\) and \(v_j\), but consider their backward neighbors instead of their forward neighbors.

### 4.3.2 Iterative Computation

So far, we only considered the basic structural similarity formulas between two fuzzy RDF graphs. And we have yet to characterize its solution, which we take to be a measure of similarity. We now explain how these formulas presented in Section 4.3.1 implement our intuition in this subsection. A solution to these formulas can be reached by iteration. To calculate \(sim(v_i, v_j)\) from forward neighbors, we present an iteration method by iteratively applying these formulas. The computation has two phases: the initialization phase which assigns \(sim^0(v_i, v_j)\) for every vertex pair \((v_i, v_j)\), and the iteration phase which update the degree of \(sim^i(v_i, v_j)\) by using \(sim^{i-1}(v_i, v_j)\) according to Formula (6), when \(i \geq 1\).

In the initialization phase, for the home vertices \(\hat{v}_i\) and \(\hat{v}_j\), the initial similarities \(sim^0(\hat{v}_i, \hat{v}_j)\) are set to 1.0 since both of them are defined as the beginning and ending of traces. \(sim^0(\hat{v}_i, v_j)\) (resp. \(sim^0(v_i, \hat{v}_j)\)) are set to 0 if one vertex is real vertex but the other is home vertex. For every other vertex pair \((v_i, v_j)\), we assign the similarity degrees \(sim^0(v_i, v_j) = 0\) as initial similarities. For the edge pair formed by the home vertices \(\hat{v}_i\) or \(\hat{v}_j\) is respectively connected with other real vertex, the similarity \(sim^0(\hat{v}_i, \hat{v}_j)\) is set to 1.0 since their edges labels are inexistence.

In each iteration \(i+1\), we can update the similarity degree for each vertex pair \((v_i, v_j)\) based on the similarities of their neighbors in the previous iteration \(i\) according to iterative Formula (6). Similarity degrees can be thought of as “flowing” from its forward neighbors to vertex. Each iteration computation propagates similarity degrees one step backward along the direction of the edges. The value \(Sim^i(v_i, v_j)\) is non-decrease as \(i\) increase, and the method iterates until either the similarity degree between all vertex pairs stabilize or a maximum number of iterations is reached. Note that, the similarities between home vertices and real vertices are not refreshed during the iteration.

In conclusion, the principle of the algorithm is that the similarities between two vertices must depend on the similarities between their adjacent vertices. We summarize the procedure of iterative computation of forward similarity in the Algorithm 1 as following.

---

**Algorithm 1. Structural similarity algorithm**

Input: two fuzzy RDF graphs \(G_S\) and \(G_T\), and a constant \(\varepsilon\)

Output: matching similarity \(Sim\)

1: for each \(v_i \in V_S, v_j \in V_T\) do
2: \(\rho(\hat{v}_i, v_i) \gets \mu(v_i)\) and \(\rho(\hat{v}_j, v_j) \gets \mu(v_j)\)
3: for each \(e_i \in V_S \times \{\hat{v}_i\}, e_j \in V_T \times \{\hat{v}_j\}\) do
4: \(sim(e_i, e_j) \gets 1\)
5: for each \(v_i \in V_S\) and \(v_j \in V_T\) do
6: if \((v_i = \hat{v}_i)\) and \((v_j = \hat{v}_j)\) then
7: \(Sim^0(v_i, v_j) \gets 1\)
Algorithm 2 is an iteration method by iteratively applying Formula (6). We begin by assigning the fuzzy degree $\mu(v)$ associated with each vertex $v \in V$ except $\hat{v}$ to edges $(v, \hat{v})$ and $(\hat{v}, v)$ in lines 1-2. Then we initialize the similarity $\text{sim}^0(\hat{e}_i, \hat{e}_j)$ for the edge pair $(\hat{e}_i, \hat{e}_j)$ formed by the home vertices $\hat{v}_i$ or $\hat{v}_j$ connected with other real vertex in lines 3-4. The similarity $\text{sim}^0(\hat{e}_i, \hat{e}_j)$ is set to 1.0 since their edges labels are inexistence.

At the same time, we initialize the similarity for vertex pair in lines 5-9. For every home vertex pair $(\hat{v}_i, \hat{v}_j)$, we set $\text{sim}^0(\hat{v}_i, \hat{v}_j) = 1.0$ in line 7. For every other vertex pair $(v_i, v_j)$, we assign the similarity degrees $\text{sim}^0(v_i, v_j) = 0$ as initial similarities in line 9. We further use iteration method to calculate the matching similarity in lines 10-14. The matching algorithm iterates until either the similarity degree between all vertex pairs stabilizes or a maximum number of iterations is reached. Finally, we return the matching similarity $\text{Sim}$ as the result in line 15.

### 4.3.3 Convergence

Having established the model of iterative calculating the structural similarity between elements of two fuzzy RDF graphs, we now show that the iterative computation of the proposed similarity function converges.

**Lemma 1.** The iterative formula (6) is bounded in the interval $[0, 1]$.

**Proof:** This follows the analysis and the definitions of the iterative formula. It is quite straightforward. For all $v_i \in V_s$, $v_j \in V_t$, $i \geq 1$, $\text{Sim}^i(v_i, v_j) \in [0, 1]$, i.e., iterative formula is bounded.

**Lemma 2.** The iterative formula (6) is monotone non-decreasing.

**Proof:** This can be simply proved by mathematical induction.

**Basis:** Let show that the monotone holds for $i = 1$. If $\hat{v}_i \in \text{pre}(v_i)$ and $\hat{v}_j \in \text{pre}(v_j)$, we have $\text{sim}^1(v_i, v_j) \in [0, 1]$, otherwise, $\text{sim}^1(v_i, v_j) = 0$.

Similarly, we have $\text{sim}^1(v_i, v_j) \in [0, 1]$. According to the iterative formula (3), $\text{Sim}^1(v_i, v_j) \in [0, 1]$.

Since the initial similarities $\text{sim}^0(v_i, v_j) = 0$, we have $\text{Sim}^0(v_i, v_j) = 0$. Thus $\text{Sim}^0(v_i, v_j) \leq \text{Sim}^1(v_i, v_j)$. Therefore, the monotone non-decreasing holds for $i = 1$.

**Inductive step:** Assume $\text{Sim}^{k-1}(v_i, v_j) \leq \text{Sim}^k(v_i, v_j)$ holds for $i = k$. According to the above iterative formula definitions, we have

$$
\text{Sim}^k(v_i, v_j) = \frac{1}{|\text{pre}(v_i)|} \sum_{v'_i \in \text{pre}(v_i)} \max_{v'_j \in \text{pre}(v_j)} \text{sim}_s(e_i, e_j) \times \text{sim}_s(p_i, p_j) \times \text{Sim}^{k-1}(v'_i, v'_j)
$$

$$
\leq \frac{1}{|\text{pre}(v_i)|} \sum_{v'_i \in \text{pre}(v_i)} \max_{v'_j \in \text{pre}(v_j)} \text{sim}_s(e_i, e_j) \times \text{sim}_s(p_i, p_j) \times \text{Sim}^k(v_i, v_j)
$$

$$
= \text{Sim}^{k+1}(v_i, v_j).
$$
Thus we have $Sim^k(v_s, v_t) \leq Sim^{k+1}(v_s, v_t)$, that is, the monotone non-decreasing holds for $i = k + 1$.

Since both the basis and the inductive step have been performed, by mathematical induction, the monotone non-decreasing holds for all $i \geq 1$.

**Theorem 1.** Iterative formula (6) is convergence.

*Proof:* Based on the above Lemma 2 and Lemma 2, it is obviously proved.

### 4.4 Combining Similarities and Alignment Extraction

The matching goal is identifying a satisfactory set of correspondences between fuzzy RDF graphs. A similarity measure between the data of both graphs provides a large set of correspondences. Those which will be part of the resulting alignment remain to be extracted on the basis of the similarity. Since we determine similarity by different heuristics, we first need to combine them to obtain the total similarity. Then, we extract the correspondences on the basis of the total similarity.

In order to obtain the overall similarity degrees between RDF datasets, we need to aggregate the similarity degrees of different similarity functions. There are several approaches to this, including linear averages, nonlinear averages and machine learning techniques. In our works, we use a simple approach based on linear averages. Firstly we obtain the label similarity ($Sim_L$) by taking an average of the syntactic similarities ($sim_{sy}$) and semantic ($sim_{se}$) similarities. Then, total similarity ($Sim$) is calculated by combining the label similarity and structural similarity ($Sim_S$). In order to more accurately distinguish between similarity scores that are close to the median, we use a non-linear function, sigmoid function [31], to compute each similarity score. The idea behind using a sigmoid function is quite simple: it allows reinforcing similarity scores higher than 0.5 and to weaken those lower than 0.5. That is to say, the sigmoid function provides high values for the best matches and lower ones for the worse matches. This treatment is meant to clearly separate two zones: the positive and negative correspondences. In this way, the general formula for this combination task can be given as following:

$$Sim(v_s, v_t) = \omega \cdot \text{sig}(Sim_L(v_s, v_t)) + (1-\omega) \cdot \text{sig}(Sim_S(v_s, v_t))$$

(7)

Here $\omega$ is a pre-defined weight, and $\text{sig}(x) = \frac{1}{1 + e^{-\alpha(x-0.5)}}$, here $\alpha$ being a parameter for the slope.

Identifying correspondences between different RDF graphs is difficult. A similarity measure between the elements of both graphs provides a large set of correspondences. To obtain a correspondence relation between input RDF graphs, we set a threshold $\delta_m$ for translating the overall similarity degrees into a binary relation. Pairs of data with similarity degree above the threshold are included in and the rest are left out. The output of the alignment extraction operator is a set of correspondences in the form of 4-tuples: $(id, E_s, E_t, m)$.

The main problem of similarity-based approaches is the identification of a right threshold, in a way that distinguishing matching from non-matching elements is reasonable. For example, the problem is to decide if two elements having a similarity measure of 0.5 have to be considered as matching or not. So how to choose the reasonable threshold is very significant. However, the choice of the thresholds $\delta$ is difficult: an increment of $\delta$...
results in increased matching quality (i.e., a low number of false positives), but simultaneously reduces the matching coverage (i.e., a low number of false negatives). Similarly, a smaller $\delta$ decreases the matching quality along with a higher matching coverage (see the experiments in Section 5.3 for the effect of varying $\delta$). In practice, we expect to produce a small decrease in the matching quality if it can bring about a comparable increase in the matching coverage. Because it is easier for us to remove incorrect matches rather than find the missing ones in the process of data matching.

4.5 Complexity of Matching

We analyze the time and space complexity of the matching algorithm. Let us denote the number of vertices in the source graph $G_s$ and the target graph $G_t$ as $|V_s|$ and $|V_t|$, respectively. The time and space complexity of computing labels similarity scores between vertex pairs are negligible. Since the set of vertex labels is finite and determined, we can compute the largest value of similarity degrees of the vertex labels and bound it as a constant.

The time complexity of computing label similarity is $O(|V_s| \times |V_t|)$ and structural similarity $O(c \times |V_s| \times |V_t| \times \text{deg})$, where $c$ is the maximum number of iterations for the computing forward similarity algorithm and $\text{deg}$ is the average degree of all the vertices in the fuzzy RDF graph. When the density of the fuzzy RDF graph as well as the numbers of iterations is high (i.e., $\text{deg}$ and $c$ are high), the iterative computation is time-consuming. The space complexity of matching is the storage needed for keeping a vertex label similarity matrix and an edge label similarity matrix and is $O(|V_s| \times |V_t| + |E_s| \times |E_t|)$.

5 EXPERIMENTAL EVALUATIONS

We have proposed an algorithm for computing similarity degrees between vertices of a fuzzy RDF graph. And the structural similarity approach is an iterative processing that depends on the graph structure and labels of vertices and edges of fuzzy RDF graph. In this section, we first describe the employed data sets and evaluation metrics. In the main part of this section, we report and discuss the obtained experimental results. In the evaluation, we will consider both match quality and runtime efficiency.

5.1 Data Sets

We generate the synthetic dataset based on the Person-Restaurant collection proposed in the instance-matching track of the Ontology Alignment Evaluation Initiative [19]. The collection contains 3 pairs of datasets. Two of these pairs describe people and the other pair describes restaurants. The Person 1 dataset is created with the help of the Febrl project example datasets. It contains original records of people and modified duplicate records of the same entries. Person 2 is created as Person 1, but with a maximum of 3 modifications per attribute, and a maximum of 10 modifications per record. The third dataset (Restaurant) is created from two different data sources (Fodor and Zagat restaurant guides). Restaurants are described by name, street, city, phone and restaurant category. These datasets contain instances from one class, and together sum up a few
AN EFFICIENT MATCHING ALGORITHM BASED ON FUZZY RDF GRAPH

thousands of RDF resources. Table 1 provides importing information for any dataset: number of vertices ($|V_s|$ column), number of generated edges ($|E_s|$ column), and number of correct matches. Moreover, in order to describe fuzzy information, fuzzy degrees are assigned to each element. We make use of a random data generation method that transforms the ordinary RDF into a fuzzy data set. The properties of the resulting fuzzy data set (e.g., the number of fuzzy vertices and edges) could be controlled by setting parameters.

We implement all the algorithms in Java and run on Window 7 system with 2.5GHz Intel(R) Core(TM) i5 processor and 8GB RAM.

Table 1. Importing information for any datasets evaluated.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Source Graph</th>
<th>Target Graph</th>
<th>Number of reference alignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person11-Person12</td>
<td>58</td>
<td>43</td>
<td>38</td>
</tr>
<tr>
<td>Restaurant1-Restaurant2</td>
<td>34</td>
<td>29</td>
<td>25</td>
</tr>
<tr>
<td>Person21-Person22</td>
<td>48</td>
<td>39</td>
<td>31</td>
</tr>
</tbody>
</table>

5.2 Evaluation Metrics

A matching approach is effective if it neither produces too many false positives nor misses too many false negatives. This paper uses the classical criterion [21]: Precision, Recall and f-measure to evaluate the answer quality. Let $|tp|$ be the number of correct matches produced by a matching approach, $|fp|$ be the number of matches found by matching approach that does not exist in the correct matches, and $|fn|$ be the number of matches not found by matching approach. We can define Precision, Recall and f-measure as follows:

$$\text{Precision} = \frac{|tp|}{|tp| + |fp|}$$  (8)

$$\text{Recall} = \frac{|tp|}{|tp| + |fn|}$$  (9)

$$f - \text{measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$  (10)

5.3 Evaluation Results

In this subsection, we show the results of our experiments. To analyze the effectiveness of our proposed approach, we perform three groups of experiments. We first investigate the performances of the combination matching (abbr. CM) which combine these results of label similarity matching (abbr. LM) and structural similarity matching (abbr. SM). And we obtain a desired threshold $\delta_m$, for a good trade-off between effectiveness, by using different thresholds ranging from 1 down to 0.5. Then we compare the performance of these three methods in different data sets. Besides the match quality, we also evaluate time cost of the matching algorithm.
Since the choice of the threshold $\delta$, in a way that distinguishing matching from non-matching elements is reasonable, plays an important role during the matching process, our first set of experiments focuses on determining an optimal value for $\delta$.

In the experiment, we study how $\delta$ affect the overall performance of CM by using different threshold for parameters $\delta$. As introduced earlier in the end of Section 4.4, threshold refers to the cutoff value used for determining the correspondence relation from the similarity degrees. A reasonable threshold is vital in the process of matching. In order to determine an appropriate value for $\delta$, we calculate CM’s Precision, Recall and $f$-measure for thresholds ranging from 0.5 to 0.95, in increments of 0.05. The results of the CM for each dataset in terms of Precision (P), Recall (R) and $f$-measure (f) are presented in Table 2.

Table 2 shows overall performance of CM when we vary the parameter $\delta$. As we can observe, the standard deviation of the precision and recall are close to 0.1, meaning that the different data sets, that is, the size of source resources, does not affect much the CM’s overall performance. Moreover, we use the median, which is a commonly used measure for the properties of a data set in statistics, to define an appropriate value for parameters $\delta$. As we can see, the overall performance of CM is relatively well when threshold $\delta_{m}=0.75$. Although not optimal, the use of the $\delta_{m}$ as threshold performs relatively well in all cases. Therefore, for all other experiments shown in this paper, we will use in our evaluation: $\delta_{m}=0.75$.

<table>
<thead>
<tr>
<th>Data</th>
<th>Person1-Person12</th>
<th>Restaurant1-Restaurant2</th>
<th>Person2-Person22</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>P</td>
<td>R</td>
<td>f</td>
</tr>
<tr>
<td>1</td>
<td>0.706</td>
<td>0.756</td>
<td>0.730</td>
</tr>
<tr>
<td>0.95</td>
<td>0.743</td>
<td>0.386</td>
<td>0.508</td>
</tr>
<tr>
<td>0.9</td>
<td>0.71</td>
<td>0.525</td>
<td>0.621</td>
</tr>
<tr>
<td>0.85</td>
<td>0.778</td>
<td>0.687</td>
<td>0.730</td>
</tr>
<tr>
<td>0.8</td>
<td>0.659</td>
<td>0.72</td>
<td>0.688</td>
</tr>
<tr>
<td>0.75</td>
<td>0.833</td>
<td>0.833</td>
<td>0.833</td>
</tr>
<tr>
<td>0.7</td>
<td>0.704</td>
<td>0.683</td>
<td>0.693</td>
</tr>
<tr>
<td>0.65</td>
<td>0.663</td>
<td>0.781</td>
<td>0.717</td>
</tr>
<tr>
<td>0.6</td>
<td>0.51</td>
<td>0.88</td>
<td>0.646</td>
</tr>
<tr>
<td>0.55</td>
<td>0.535</td>
<td>0.817</td>
<td>0.647</td>
</tr>
<tr>
<td>0.5</td>
<td>0.527</td>
<td>0.882</td>
<td>0.660</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.106</td>
<td>0.149</td>
<td>0.081</td>
</tr>
</tbody>
</table>

In the second group experiments, we compare the match quality of LM, SM and CM three methods over the above three data sets. For match quality, we evaluate the usual measures Precision, Recall, and $f$-measure. Fig. 3 presents the experiment results. The diagram of Fig. 3(a) represents the Precision values of three methods for the three data sets in Table 1, and the diagram of Fig. 3(b) represents the Recall values for those case studies. Bars refer each method, using different gray scales, i.e., from LM, white bars, to CM, black bars. The horizontal axis displays the range of Precision or Recall values, while the vertical axis displays the data sets. For the convenience of diagram representations, in the Fig. 3, we abbreviate the data sets name Person1-Person12, Rest-
It can be seen that the precision of LM is higher than SM in the P11-P12 and R1-R2 data pairs, where elements with typographically similar names are likely to correspond. However, LM misses several correct matches, and hence has low recall. SM, in contrast, has lower precision, but high recall. Furthermore, the performance of LM is poor than SM in the P21-P22 pairs. This observation can be explained by the fact that these datasets were built by adding spelling mistakes or null value to the elements and literals values of their original datasets. While SM is able to exploit better the similarity in misspelled data or blank vertices by structural information. Moreover, we can see that the precision rates of our proposed CM are higher than all the existing evaluated approaches to all the data sets. For example, the precision values for the LM, SM, and CM in the R1-R2 are 0.703, 0.682, and 0.902, respectively. This is explained by the fact that LM and CM approaches are filtering out each other’s false positives. Recall remains high in the CM approach, as LM and SM find many complimentary high-quality matches.

Finally, in the third group of experiments, we compare the execution time of CM against LM and SM. Results for these approaches are reported in Fig. 4. The meaning of Fig. 4 (a) is similar to the one of Fig. 3. And the only difference is the vertical axis which represents f-measure values. As observed in Fig. 4 (a), we find that a structural similarity metric based on graph matching achieved the highest retrieval quality (f-measure). However, the metric is hindered by the fact that the iterative computation is time-consuming.
The diagrams of Fig. 4 (b) represent the time costs of LM, SM, and CM in different data sets. Most of the features of the diagrams, such as the bar or the horizontal axis, should be similar to those of Fig. 3. The difference is the vertical axis which represents corresponding time cost. As we can see, corresponding time cost of SM is no more than the double of LM’s and lowers than that of CM. It is not surprising owing to the high complexity of computing graph structural similarities. Therefore, we need improvement in terms of time by decreasing the number of iterations that strike a tradeoff between computational complexity and precision of matching in practice.

In summary, a number of experimental results confirm the effectiveness of the proposed measure. Although, scalability is not the focus of this paper, the fact that we were able to run experiments on relatively large data sets shows the general feasibility of our approach.

6. CONCLUSIONS

RDF data matching in the context of fusion and interlink of RDF datasets is the key task of determining if two resources are referred to the same entity in the real world. In this paper, we propose an efficient approach based on fuzzy RDF graph matching. The proposed approach is based on the similarity measure, which takes textual characteristics and structural features into account while matching the fuzzy RDF graph. Firstly, label matching is calculated by combining syntactic and semantic similarities of labels of vertices or edges of the RDF graph. We then propose a function computing structural similarity iteratively for matching RDF data, in which we further take edge similarity and edge fuzzy value into consideration. Next, we combine these results of the individual metrics to obtain the total similarities. Finally, we extract the correspondences on the basis of the total similarities. The experimental results show that the proposed approach can effectively measure the similarity between fuzzy RDF graphs in various situations.

While the evaluation results demonstrate the effectiveness of our approach, it has a limitation. As shown in Section 4.5, the time complexity of computing structural similarity depends on several factors such as the size and the density of the fuzzy RDF graph, and the numbers of iterations. We intend to investigate the evaluation of convergence so that the iteration number in calculating similarity can be reduced, which is a matter of obvious importance when handling large RDF graphs. We also plan to implement a prototype to assess our approach on real-world RDF datasets. Finally, it is worth of future research to apply Granular Computing techniques to solve the problem of fuzzy RDF data matching.

REFERENCES


