Large Scale Instance Matching via Multiple Indexes and Candidate Selection

Juanzi Li[a], Zhichun Wang[a,b,*], Xiao Zhang[a], Jie Tang[a]

[a]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*
[b]*College of Information Science and Technology, Beijing Normal University, Beijing, China*

Abstract

Instance Matching aims to discover the linkage between different descriptions of real objects across heterogeneous data sources. With the rapid development of Semantic Web, especially of the linked data, automatically instance matching has been become the fundamental issue for ontological data sharing and integration. Instances in the ontologies are often in large scale, which contains millions of, or even hundreds of millions objects. Directly applying previous schema level ontology matching methods is infeasible. In this paper, we systematically investigate the characteristics of instance matching, and then propose a scalable and efficient instance matching approach named VMI. VMI generates multiple vectors for different kinds of information contained in the ontology instances, and uses a set of inverted indexes based rules to get the primary matching candidates. Then it employs user customized property values to further eliminate the incorrect matchings. Finally the similarities of matching candidates are computed as the integrated vector distances and the matching results are extracted. Experiments on instance track from OAEI 2009 and OAEI 2010 show that the proposed method achieves better effectiveness and efficiency (a speedup of more than 100 times and a bit better performance (+3.0 to 5.0% in terms of F1-score) than top performer RiMOM on most of the datasets. Experiments on Linked MDB and DBpedia show that VMI can obtain comparable results with the SILK system (about 26,000 results with good quality).

Keywords: semantic web, instance matching, ontology matching, linked data.

1. Introduction

Ontology is one of the key components to realize the Semantic Web. With the rapid development of the Social Web, a lot of ontologies especially lightweight ontologies have been widely used, and a huge number of instances were annotated according to the ontologies. For example, the FOAF vocabulary (schema) is comprised of 13 classes and 60 properties while LiveJournal website alone provides approximately 15,000,000 FOAF profiles (instances). The DBpedia ontology, which covers 273 classes described by 1,300 different properties, contains

*Corresponding author. Tel.: +86 01062773618; fax: +86 010 62781461
Email addresses: ljz@keg.cs.tsinghua.edu.cn (Juanzi Li), zcwang@bnu.edu.cn (Zhichun Wang), zhangxiao@keg.cs.tsinghua.edu (Xiao Zhang), jietang@tsinghua.edu.cn (Jie Tang)

about 1,600,000 instances. GeoNames provides RDF descriptions of more than 6,500,000 geographical features worldwide. The Linking Open Data (LOD) project already has a data set of more than 4.7 billion RDF triples and around 142 million RDF links between instances [6].

As the number of published ontologies grows, a increasing number of ontology-based applications have also been proposed, such as Question Answering [14], Query Expansion [28], Knowledge Support [1] and Web Services [26]. The rapid usage of ontologies arises the ontology heterogeneity problem. In the last decade, ontology matching has been widely studied as the key technology to reach interoperability over ontologies [20, 8, 24]. Traditionally, ontology matching approaches focus on finding semantic correspondences between complex ontology schemas. Recently, as the number of ontology instances grows rapidly, the problem of instance matching attracts increasingly more research interest [35]. The yearly ontology matching competition OAEI (Ontology Alignment Evaluation Initiative)[1] has set up instance matching campaigns since 2009. Several systems, such as RiMOM [36], HMatch [7], and FBEM [30], have participated in the instance matching tasks of OAEI. The problem of instance matching involves handling large number of instances, which raise new challenges: 1) How to deal with large scale input? Shvaiko and Enzuenat [29] point out that the scalability is important for ontology matching approaches. Widely used matching techniques such as Edit Distance [15], KNN [16], Google Distance [12] will take much running time when applied to large number of instances. Suppose we apply the Edit Distance, one of the most efficient similarity metrics, to match two ontologies with 1,000,000 instances. Even on a server with 32 Gigabyte memory and 3.2 GHz CPU, the running time of calculating all the potential matches will be up to 2 days. 2) How to trade off between precision and recall? Most approaches use strict measures to find matching results with very high precision but only a very small part of the potential matches are obtained. By investigating information contained in ontology instances, we observe that there are several different characteristics of instance matching compared with the traditional ontology matching in schema level. Firstly, instance data is usually with large scale. Secondly, instance data contains devious semantic information. Usually concepts and properties in ontology schema are described with labels and comments. However, for instances or individuals in an ontology, every property is given a specific value and represented in various ways. For example, the e-mail address of a person, the ISBN number of a book, the DNA sequence of a gene is consisted of a large number of different values of validated types. It is difficult to take full advantages of the information. Thirdly, the concepts and properties in the ontology schema construct a connected graph structure. The ontology can be viewed as a whole ontology graph and some graph-based algorithms are employed in ontology matching. However, a concept may have lots of instances and all the instances are with almost the same structure. The graph algorithm with the whole ontology graph is not suitable for the instance matching task.

To address the above two challenges, we propose a large scale instance matching method (named VMI) by using multiple indexes and candidate selection. VMI aims at matching large scale instance datasets efficiently and generates as many matching results as possible with high quality. In particular, VMI uses the vector space model to represent instances' descriptive information. VMI creates two types of vectors for each instance, one for names and labels of the instance and the other for descriptive information and information from neighboring instances. We build inverted indexes for these types of vectors and select matching candidates according to the indexes. In this way, VMI is able to avoid pair-wise comparison and reduces the matching

[1]http://oaei.ontologymatching.org/

space greatly so that the matching efficiency can be improved. Then VMI compares the value pairs from user specified properties to filter the primary candidates and improve the precision. Experimental results on the datasets from the Instance Matching track of OAEI 2009 and OAEI 2010 show that VMI is much faster than existing methods (100 times faster than the participants) while achieves better (+3.0-5.0% in terms of F1-score on most of datasets) accuracy performance than the top performer RiMOM. Experimental results on LinkedMDB and DBpedia dataset show that VMI can generate matching results with almost the same amount and quality as ones from the SILK system [33].

Contributions of this work can be summarized as:

- We formally define the problem of large scale instance matching.
- We propose an efficient and accurate instance matching method by using the inverted indexing and candidate matching selection rules.
- We validate the proposed VMI approach on three datasets from the Instance Matching track of OAEI 2009 and 2010 as well as datasets from Linked Open Data. Experimental results show that VMI can achieve a more than 100× speedup than the best performer in OAEI 2009 and a comparable precision and recall performance with the prevalent SILK system.

The rest of this paper is organized as follows. In section 2, some related work are summarized. In section 3, we give some preliminary and definitions. In section 4, we show an overview of VMI algorithm and a detailed presentation of VMI is given in section 5. Experimental results are illustrated in section 6 with discussions. Finally, conclusion and future work are given in section 7.

2. Related Work

There has been already several approaches dealing with the instance matching problem. Most of them focus on achieving high precision and recall, the problem of matching large scale instances has not been well studied. We summarize some of these methods and systems compare them with our proposed approach VMI.

COMA++ [3] is an schema and ontology matching tool utilizing a composite approach to combine different match algorithms. In the enhanced eversion of COMA++, it uses two methods to matching instances [10]: one is the content-based similarity, the other is constraint-based similarity. COMA++ needs to compare all the instances between two ontologies, and it also uses a similarity propagation algorithm to transfer similarities from instances to their surrounding ontology elements. Our approach generates a virtual document for each instance to include its neighboring information, and computes the similarity by using Vector Space Model; it is more efficient than the iterative similarity propagation. Furthermore, our approach selects the matching candidates based on two inverted indexes, it does not need to compare all the instance pairs. HMatch [7] is a ontology matching suite which provides a component for instance matching. In HMatch, each instance is represented as a tree where role fillers are nodes and roles are labeled edges. Matching is performed by traversing the instances trees of the two instance in postorder, and recursively executing filler similarity. Filler similarities of different properties are combined by weighted averaging with manually weights. RiMOM [31][21] uses a systematic approach to quantitatively estimate the similarity characteristics for each matching task and employs a strategy selection method to automatically combine the matching strategies based on two estimated factors. For instance matching, RiMOM chooses some data-type properties as the ”necessary” and ”sufficient” attributes manually. ”sufficient” attributes are use to find the initial alignment while the ”necessary” attributes for refinement, and a similarity propagation is employed in the last step. DSSim[23] is an ontology mapping system used with a multi-agent ontology mapping framework in the context of question answering. In order to improve the matching quality, it incorporates the Dempster Shafer theory of evidence into the mapping process. DSSim assesses similarity of all the entities from two different ontologiesthe belief combination process of DSSim is also computationally expensive. Therefore, DSSim employs an multi-agent architecture to enable distributed execution of the approach. Our approach uses inverted indexes to select matching candidates, therefore reduces the computation time; it is different from the distributed execution of DSSim that needs multiple machines. FBEM[30] is a feature based instance matching system. It does not need any kind of schema or strong typing information of the instances. FBEM supports a complete generic way to match instances. Given two instances, FBEM first computes the Levenstein similarity between all the features of them, and then calculates the combined similarity score by summing all the maximum similarity feature combinations between two instances. FBEM also implemented a ”brute-force” matching, similarity of any instance pairs need to be computed to get the matching results. Being different from FBEM, our approach allows users to specify instance types and important properties to improve the accuracy and efficiency of VMI. SILK [33] is a link discovery engine which automatically finds RDF links between datasets. Users should specify which type of RDF links should be discovered between the data sources as well as which conditions data items must fulfill in order to be interlinked. These link conditions can apply different similarity metrics to multiple properties of an entity or related entities that are addressed using a path-based selector language. The resulting similarity scores can be combined using various similarity aggregation functions.

Comparing with the above approaches, VMI shows several advantages. Firstly, it is an generic instance matching algorithm while some of these works are customized for specific domains and schemas. Secondly, instead of simple string comparison on names and property values, it makes more comprehensive use of instance information. Thirdly, with the help of multiple indexes and candidate selection rules, VMI is more scalable than these systems, especially on instance matching.

Some other closely related work can be divided into two groups, ontology schema matching and data integration. We also briefly reviewed some of these work as follows:

Schema Matching ASMOV [19] is a novel algorithm that uses lexical and structural characteristics of two ontologies to iteratively calculate a similarity measure between them, derives an alignment, and then verifies it to ensure that it does not contain semantic inconsistencies. Wang et al. [34] compute instance similarity by annotations first and use Markov Random Fields as classifier to generate concept mappings. Hu et al. [18] uses a structure-base partitioning algorithm to partition ontology into clusters and construct blocks by assigning RDF sentences to those clusters. Two matchers, V-Doc and GMO are employed to discover alignments between selected block candidates. PRIOR+[22] also uses vector space model to generate linguistic and structural similarity for schema entities. It aggregates the similarities with an adaptive method based on their harmonies and then activate a neural network to search for a solution that can satisfy schema constraints. iMatch [2] employs Markov networks for schema matching. It constructs the network with relations between concepts of the schema to perform probabilistic reasoning and generate one to one alignments.

Data Integration Instance Matching task is also known as identity recognition, object consolidation, or record linkage in the databases world. Bilke et.al[5] detects duplicates from different data sources to align schemas. They compute the tuple similarity using Vector Space Model , use the Whirl algorithm to calculate the upper bounds for each tuple of one database, and refine these bounds by combining those tuples with promising tuples of the other database. Then duplicates are detected by their similar attribute tuples. Hogan et.al[17] propose a method for performing large-scale object consolidation to merge identifiers of equivalent instances occurring across data sources. The algorithm is based on the analysis of defined inverse functional properties: properties which have values unique to an instance. Probabilistic models and machine learning techniques are also widely employed in duplicate detection[11, 4, 32]. Elmagarmid et.al gives an comprehensive survey on this research area in [9].

3. Preliminaries and Definitions

In this section, we define the problem of instance matching and then analyze the information that can be used in instance matching. An ontology is a formal, explicit specification of a shared conceptualization [13][25]. Currently there are many formal representations of ontology, we choose the following one, which describes ontology as a 6-tuple.

Definition 1. *An ontology is a 6-tuple $O = \{C, P, H^C, H^P, A^O, I\}$, where C and P are the sets of classes and properties respectively. H^C defines the hierarchical relationships between classes. $H^C \subset C \times C$. (c_i, c_j) denotes that class c_i is the subclass of class c_j. Similarly, H^P defines the hierarchical relationships between each property and its subproperties, $H^P \subset P \times P$. A^O is a set of axioms. I is a set of instances of classes. We call concepts, properties and instances entities in ontologies.*

```
<owl:Class rdf:about="http://dbpedia.org/ontology/Film">
  <rdfs:label xml:lang="en">Film</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://dbpedia.org/ontology/Work"/>
</owl:Class>
<owl:DatatypeProperty rdf:about="http://dbpedia.org/ontology/imdbid">
  <rdfs:label xml:lang="en">imdbId</rdfs:label>
  <rdfs:domain rdf:resource="http://dbpedia.org/ontology/Film"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:about="http://dbpedia.org/ontology/starring">
  <rdfs:label xml:lang="en">starring</rdfs:label>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:class rdf:about="Film"/>
        <owl:class rdf:about="TelevisionShow"/>
        <owl:class rdf:about="FilmFestival">
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
<rdf:Description rdf:about="http://dbpedia.org/resource/Star_Wars:The_Clone_Wars_%28film%29">
  <rdf:type rdf:resource="http://dbpedia.org/resource/Film"/>
  <foaf:name>Star Wars: The Clone Wars</foaf:name>
  <dbpedia:imdbid>1185834</dbpedia:imdbid>
  <dbpedia:starring rdf:resource="http://dbpedia.org/resource/Matt_Lanter"/>
  <dbpedia:starring rdf:resource="http://dbpedia.org/resource/James_Arnole_Taylor"/>
  <dbpedia:producer rdf:resource="http://dbpedia.org/resource/George_Lucas"/>
  <rdfs:comment>Star Wars: The Clone Wars is a 2008 CGI animated science fiction/action film
        that takes place within the Star Wars saga, leading into the TV series of the same
        name.</rdfs:comment>
</rdf:Description>
```

Figure 1: Snippet of the DBpedia Ontology in OWL

Figure 1 is a snippet from the DBpedia Ontology in OWL. We have the following information in it:

- Film is a class and it is a subclass of Work;
- imdbid is a datatype property. It has the domain of Film and the range of xmls:string;
- starring is a object property. Its domain is the combination of three classes Film, TelevisionShow and FilmFestival;
- Star Wars:The_Clone_Wars(film) is an instance of Film. Its value of property imdbid is 1185834. Its values of property starring are two other instances Matt_Latner and James_Arnold_Taylor.

Although ontology aims to make web data sharable, ontologies themselves are heterogeneous and distributed. When trying to interoperate data under different ontologies, it is necessary to find the matchings among these ontologies. We define the problem of Ontology Matching as:

Definition 2. *Given two input ontologies O_s and O_t, an ontology matching task is defined to find corresponding entities in O_t for each entity in O_s. O_s is called the source ontology and O_t is called the target ontology. The matching results can be represented as:*

$$\begin{aligned} Align(O_s, O_t) =&\{(e_{is}, e_{it}, con_i, relation_i)|e_{is} \in O_s, e_{it} \in O_t, con_i \in [0, 1], \\ & relation_i exact, narrower, broader, overlap\} \end{aligned} \tag{1}$$

Each 4-tuple (e_{is}, e_{it}, con_i, $relation_i$) in $Align(O_s, O_t)$ indicates that entity e_{is} in O_s is matched to entity e_{it} in O_t with the confidence con_i and the matching type $relation_i$. The alignment type can be exact matching (exact), narrowing matching (narrower:e_{is} is a sub-entity of e_{it}), broadening matching (broader:e_{is} is a super-entity of e_{it}) and partially overlapping alignment. con_i is a numeric value. The higher the con_i value, the more reliable the matching result is.

Particularly, the matching problem restricted only to instances is Instance Matching. The results of instance matching can be represented as:

$$InstAlign(O_s, O_t) = \{(i_{js}, i_{jt}, con_j)|i_{js} \in I_s, i_{jt} \in I_t, con_i \in [0, 1]\} \tag{2}$$

Because there is only exact matching in instance matching, there is no matching type in instance matching. The matching problem on concepts and properties is called Schema Matching. Both instance matching and schema matching are sub-tasks of ontology matching. Previous research pays a lot of interest on schema matching. But as we introduced in Section 1, instances have different characteristics from the schema, which makes instance matching a more challenging problem. To solve this problem, it is necessary to leverage all the information contained in instances. We summarize information in an instance into six categories:

- URI,URI is the unique identifier on the web for the instance. If two instances have the same URI, we can simply claim they are the same one;
- Meta, the schema information of the instance, including the classes the instance belongs to, the properties the instance has and so on;

- Name, the names and labels people use to refer to the instance in real world. Names of an instance may come from values of RDFS:label property, other ontology specified property (like foaf:name) or fragment of its URI. When distinguishing two objects, first choice of most people is to check the names and in many cases it works, so names is very important information for instance matching.
- Descriptive Property Values, these property values are the descriptions of the instance in natural language. Typical example of descriptive property is RDFS:comment.
- Discriminative Property Values, these property values are not descriptions of the instance but characteristics of instance which can be used directly to distinguish them. For example, a Person instance with Male value on Sex property is not likely to be matched to a person instance with Female value on the property while two persons' instance with the same value on e-mail property is highly likely to be the same one.
- Neighbors, because instances are related to each other through object properties, so neighboring information is a complement of instances. In many ontologies, instances are lacking in descriptive information but rich in neighboring information, so it is necessary to take neighboring information into account in instance matching.

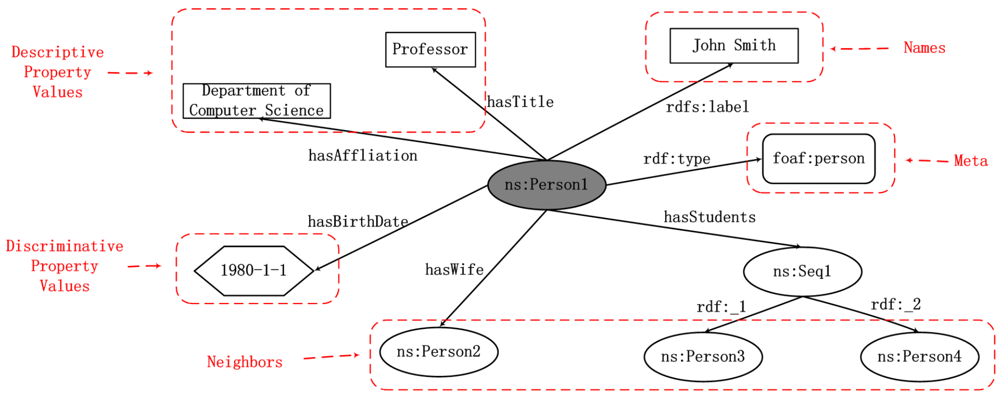


Figure 2: An Example of Information within an Instance.

Figure 2 illustrates the six kinds of information in a given instance. Its name is John Smith. The meta information contains the type of the instance and the properties. The descriptive information has two parts, the affiliation and the title, which describe the instance as a professor in the department of computer science. The descriptive property information is the birth date of the instance. The given instance has three neighbors. The first one is the property value of hasWife, the other two are connected through the RDF black node as the property value of hasStudents.

4. Approach Overview

The major challenge we are to solve in VMI is to improve the efficiency of instance matching as well as keeping high quality of the matchings. Given an instance *i* from the source ontology

O_s, traditional methods usually compute similarity between i and every instance in target ontology O_t in a brute-force way. In fact, there may be only a few possible instances in O_t that match i. If VMI can select the matching candidates at first, the matching process will be accelerated greatly. On the other hand, to gain matching results with good quality, it is necessary to correctly use all the information in an instance as possible as we can. Among the six categories of information, both Names and Descriptive Property Values are meaningful natural language segments. With consideration on these two aspects, we find that Vector Space Model plus Inverted Index perfectly fit in the scenario. Vector Space Model is a good way to process text information and with Inverted Index we can easily find candidates. Now we have a few more questions to answer: 1) Should VMI use this method directly? 2) How to use the neighboring information? 3) How to use the Discriminative Property Values? When trying to distinguish two instances, people usually compare their names at first, then look at their descriptions and property values. Therefore VMI should follow a similar procedure and should not put all the information in only one vector. Therefore, VMI builds two different vectors for each instance, one contains the Name information, called Name Vector, and the other contains the Descriptive Property Values and the Neighboring information, called Virtual Document Vector. According to the two vectors, VMI generates two inverted indexes for instances and terms in vectors and then selects primary matching candidates based on rules. VMI first puts the local Descriptive Property Values in the Virtual Document Vector, then takes Name Vector and Local Description of all neighbors as the neighboring information, adds their weighted sum into the Virtual Document Vector to complete it. Trying to compare the Discriminative Property Values, we need to know the property matchings before hand and the mappings are vital for instance matching. We cannot compare the birth date of a person in the source ontology to the wedding date of a person in the target ontology, which will result in wrong results. Hence VMI asks users to input property matchings at first, not complete but with good precision guarantee. If primary candidates do not have matching values on these properties, VMI eliminates them from the candidate set. This step is a further refinement on the candidates to improve the precision of final results. The full process of VMI is shown in Figure 3.

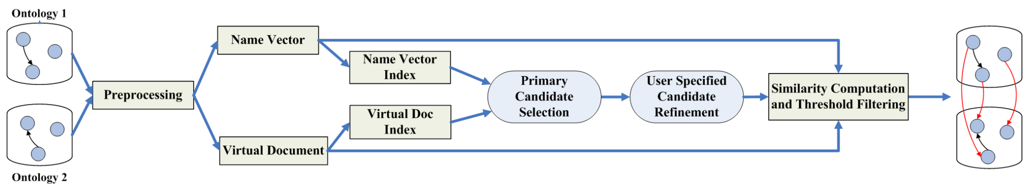

Ontology 1
Ontology 2
Preprocessing
Name Vector
Virtual Document
Name Vector Index
Virtual Doc Index
Primary Candidate Selection
User Specified Candidate Refinement
Similarity Computation and Threshold Filtering

Figure 3: The Matching Process of VMI.

The first step of VMI is to preprocess the instance file, load the RDF graph into memory, and collect the local information and neighboring information for every instance. In step 2, VMI builds the Name Vector and the Virtual Document Vector for each instance. In step 3 VMI adapts the candidate selection rules based on the vectors and indexes to generate primary matching candidates. In step 4 VMI uses the user specified property pair and value patterns to refine the primary matching candidates. The similarity of the candidate instance pairs is computed by their cosine distance based on Name Vector and Virtual Document. At last a threshold filter is used to produce the final matchings. We will give a detailed explanation of each step in the next section.

5. The VMI Algorithm

At the beginning, we give some symbols used in this section. i refers to a given instance and O is the ontology i belongs to. i_s and i_t are instances from source ontology O_s and target ontology O_t. After preprocessing, we can collect all the information for instance i, $Names(i)$ denotes the set of its names, $SPV(i)$ the set of its string typed values, $NSPV(i)$ the set of its non-string typed values of i and $NB(i)$ denotes the set of its neighboring instances. The two types of vectors we build for i are Name Vector $NV(i)$ and Virtual Document $VD(i)$ respectively.

5.1. Building Vectors and Indexes

To get $Names(i)$, we first check if there are rdfs:label property or other ontology specified name property values in i and put them into the set $Names(i)$. Otherwise $Names(i)$ contains only the fragment of the URI. Then we go on to segment strings in $Names(i)$ and eliminate stop words. The terms in the result are put into $NV(i)$ and their weights are assigned as their occurrence times in the strings. Because we believe that the words in $NV(i)$ are equally important for i, we build an inverted index for all the terms in the Name Vector, i.e. for every term t, we maintain a list of instances whose Name Vectors contain t.

The construction of $VD(i)$ is a little more complex because the information comes from both its local string typed property values and its neighboring information. In the pre-processing stage, we put all the string typed values in $SPV(i)$, then we can built a vector $LD(i)$ for the instance with just the same procedure when building $NV(i)$. Then the neighboring information for i is defined as the sum of both Name Vectors and Local Description of its neighbor instances:

$$NBI(i) = \sum_{i' \in NB(i)} (NV(i') + LD(i')) \tag{3}$$

And $VD(i)$ is defined as

$$VD(i) = LD(i) + \gamma \cdot NBI(i) \tag{4}$$

where γ is the neighboring factor which indicates the strength of neighboring information in the Virtual Document.

Compared with Name Vector, Virtual Document contains more information while each term in the vector is less important than the terms in Name Vector, so there is no need for us to build an inverted index for every term in the Virtual Document. Instead, it is necessary to choose some important terms from each vector to represent the instance. For evaluating the significance of terms in a Virtual Document, the prevalent tf-idf [27] weight is introduced into the method. Let w_t be the weight of a given term t in a Virtual Document $VD(i)$, W be the sum of weight of all the words in the document, n be the number of documents which contains t, and N be the number of all documents. The term frequency (tf) and inverted document frequency (idf) and final weight of the term t in a document is defined as:

$$tf = \frac{w_t}{W} \tag{5}$$

$$idf = log\frac{N+1}{n+1} \tag{6}$$

$$weight(t) = tf \times idf \tag{7}$$

After the tf-idf computation, we build the inverted indexes for Name Vectors and Virtual Documents respectively. The inverted index [37] is a widely used data structure used by search engines. Typically, the inverted index uses a dictionary store all the unique terms in the documents. For each term t in the dictionary, there is a link to a postings list $L_t =< d^t_1, d^t_2, ..., d^t_{n_t} >$. The element d^t_i denotes the document identifier of the i'th document containing t. By using the inverted index, we can quickly find the documents that contain a given term. Here we build two inverted indexes for Name Vectors and Virtual Documents of instances, respectively. In our approach, only terms whose weights are higher than a threshold t_{vd} are added to the dictionary of the inverted index. Primary matching candidate selection methods will be described in the following subsection.

5.2. Primary Candidate Selection

Given two sets of instances I_1 and I_2, our approach does not compare all the instance pairs between them to decide the matching results. Instead, VMI first selects a set of primary matching candidates before computing the similarities of instance pairs. For each instance $i_s \in I_1$, instance $i_t \in I_2$ that satisfies one of the following rules is selected as matching candidate of i_s:

- If $|NV(i_s)| \geq 5$ and $|NV(i_t)| \geq 5$ and the two vectors have at least 2 terms in common;
- If $|NV(i_s)| \leq 5$ or $|NV(i_t)| \leq 5$ and they have at least 1 term in common;
- If $VD(i_s)$ and $VD(i_t)$ have at least 1 common keyword.

The first two rules originate from the idea that we are willing to compare instances with similar names. When instances have short names, one name term in common could mean a candidate pair. But when instances have longer names, it is very possible if they are matched, they should have at least two common terms in the Name Vector. The third rule is a supplement for the first two rules to make sure that if the instances are not similar in names but have similar semantic descriptions, they will be chosen as candidates, too. These rules select most of the correct matching pairs are contained in the candidates (in our experiment data sets, more than 95%) while cutting off a very large portion of the pair-wise comparison.

In the process of selecting matching candidates, the two inverted indexes are used to generate candidate instance pairs. Take the inverted index of Name Vectors as an example, VIM generates candidate instance pairs that have at least 1 common term in their names as follows:

- For each unique term t, get its postings list L_t from the inverted index of Name Vectors;
- Generate the set of instance pairs of term t: $P_t = \{< i_s, i_t > | i_s \in L_t \cup I_1, i_t \in L_t \cup I_2\}$;
- Get all the desired instance pairs by Merging the instance pairs generated from all the terms in inverted index: $P = \cup_{t \in T} P_t$, here T denotes the set of all unique terms.

All the instance pairs in P have at least 1 common term, those pairs having at least 2 common terms can also be found within P. The inverted index of Virtual Documents is also used in the same way to find instance pairs that have at least 1 common term in their Virtual Documents.

5.3. User Specified Candidate Refinement

As we look into the selection rules in the above part, we will find that they are relatively simple so the process is just a rough filtering on impossible matching pairs. To get more precise results, VMI needs a further filtering on primary candidates using the Discriminative Property Values. Since it is really difficult to get the property matching automatically, VMI relies on the user's input as the prior information. We regard rdf:type as a special discriminative property in VMI. Instances not matched on the type (either under the same concept or one concept is the subconcept of the other) will be removed. And for the user specified properties, there are two kinds of scenarios. One is to check whether the property values are in the user's given set and the other one is to check if the property values from two instances are identical. Because different users may have different presentations on the same information, VMI permits users to fetch particular part of the property value. All the property matchings, allowed value set and the ways of fetching values are specified by users in the configuration file of VMI.

5.4. Similarity Computation and Threshold Filtering

Once the candidate instance pairs have been selected, the similarity between them is computed using name vector and virtual document respectively according to the Vector Space Model. We use the COSINE distance between two vectors to get the similarity. It is defined as:

$$Sim(V_s, V_t) = \frac{\sum_{i=1}^{V}(v_{si} \cdot v_{ti})}{\sqrt{(\sum_{i=1}^{V}(v_{si}^2))(\sum_{j=1}^{V}(v_{tj}^2))}} \tag{8}$$

where V is the dimension of the vector, the elements in V correspond to the weights of terms computed by formula 7. Then we combine the two similarities to a root mean square as the initial similarity of the candidate pair.

$$Sim(i_s, i_t) = \sqrt{w_n \cdot Sim_{nv}(i_s, i_t)^2 + (1 - w_n) \cdot Sim_{vd}(i_s, i_t)^2} \tag{9}$$

where w_n is the weight of the name vector similarity. After the final similarity is computed, VMI takes a threshold filtering on the similarity to extract the final results.

6. Evaluation

In this section we employ VMI algorithm on the datasets from the Instance Matching track of OAEI 2009 and OAEI 2010 campaigns and evaluate its effectiveness and efficiency on large scale instance files.

6.1. Data Sets and Experiment Setup

Data Sets We choose two datasets, the instance matching track of OAEI and the LinkedMDB-DBpedia data set on the movie domain. OAEI[2] is a yearly international ontology matching competition. The Instance Matching track is introduced into the OAEI from 2009 aiming at evaluating instance data matchers. We use the A-R-S benchmark and T-S-D benchmark from OAEI 2009 and the Data Interlinking dataset from OAEI 2010 to test VMI. The A-R-S benchmark includes three ontologies named eprints, rexa and dblp within the domain of scientific publications in the same Opus schema. T-S-D benchmark includes three data sets covering broader domains which are structured according to different schemata. The Data Interlinking dataset chooses several data sources from LOD. There are reference matchings for the A-R-S benchmark, so we use it to demonstrate the accuracy of our approach and use the other two for testing scalability of

[2]http://oaei.ontologymatching.org

VMI. Table 1 shows the information about the sizes of the data sets and the number of instances contained in them. The scale of the three data sets in A-R-S benchmark varies greatly. Three data sets, dblp in A-R-S benchmark, swetodblp and dbpedia in T-S-D benchmark are really large so that matching these data sets efficiently becomes a major concern. The LinkedMDB repository[3] is the largest semantic movie database created by researchers from University of Toronto. There are also a number of movie-related instances in DBpedia which are extracted and stored in one file. The details of the LinkedMDB-DBpedia data set are shown in Table 2. We match the movie instances from the two data sources to compare the performance of VIM and the prevalent system SILK.

Table 1: The sizes of datasets in OAEI (number of instances)

A-R-S Benchmark		T-S-D Benchmark		Data Interlinking	
eprints	847	TAP	65,246	Sider	2,684
rexa	14,771	swetodblp	813,287	Dailymed	10,014
dblp	1,642,945	DBpedia	2,091,003	Drugbank	19,700

Table 2: The detail of LinkedMDB-DBpedia data set

Data	Movie Count	Actor Count	Total Count
LinkedMDB	85,620	50,603	450,000
DBpedia	35,240	26,010	200,000

Experiment Setup We implement VMI in Java. The ontologies are parsed by Jena API[4]. Because the processing of large instance files consume a large amount of memory, we run the normal version of VMI on a server with 32GB memory running Ubuntu Server 8.10.

Measurement To evaluate the accuracy of matching algorithms, we choose the standard measurement Precision, Recall and F1-Measure to evaluate the efficiency of VMI, we compare the running time of VMI with two baseline methods, Edit Distance on instance names referred to as EDist and the champion RiMOM system in A-R-S benchmark.

6.2. *Parameter Analysis*

We have three parameters in VMI: γ is the neighboring factor, indicates the strength of neighboring information in the Virtual Document; w_n is the weight of Name Vector in the computation of final similarity and t is the threshold in final threshold filtering. We will analyze the performance of VMI with different parameters settings. The following analysis is made on the eprints-rexa dataset, one of the datasets in A-R-S benchmark.

Neighboring Factor First, we check the performance of VMI with different amount of neighboring information. We suppose $\gamma = 0$ mean to eliminate the neighboring information in the Virtual Document. We compare the performance of VMI with different γ on the eprints-rexa dataset with $t = 0.45$ and $w_n = 0.6$. The result is shown in Table 3. It is obvious that VMI can not

[3]http://www.linkedmdb.org
[4]http://jena.sourceforge.net/

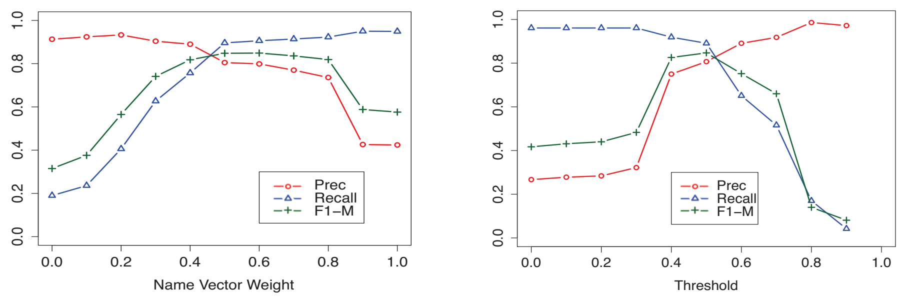
get desired performance without sufficient neighboring information, because the local feature does not contain all the instances information. People may simply believe that instances with similar neighbors are of high possibility to be similar, too. It is important to add the neighboring information to the Virtual Document.

Threshold Then we let $w_n = 0.6$ and $\gamma = 0.5$ and evaluate the threshold t on the performance of VMI algorithm. The result is shown in Figure 4(a). The F1-Measure gets to its peak at about some points around $t = 0.5$, which is the default value for t. In this case, the matchings with low similarity are definitely wrong, because the recall increases as the threshold increases from 0 to 0.4.

Name Vector Weight Similar experimental results on w_n with $t = 0.45$ and $\gamma = 0.5$ are shown in the right chart in Figure 4(b). Based on the growing recall curve, it is convincible that the Name information is very significant to find possible matchings and to promote the absolute value of the similarity between candidates. However, the relatively low precision with high w_n indicates that only relying on the Name information is not a good way. We choose as 0.6 the default w_n value for the A-R-S benchmark.

Table 3: The effect of neighboring information

γ	Precision	Recall	F1-Measure
0	0.653	0.883	0.751
0.25	0.799	0.906	0.849
0.5	0.799	0.906	0.849



(a) (b)

Figure 4: Effects of Two Major parameters in VMI.

6.3. Efficiency Performance

In order to evaluate the efficiency of VMI, we compare the runtime of VMI with Edit Distance, RiMOM and DSSim. Here Edit Distance is a baseline approach we develop; it computes the Edit Distance based similarity between instances' names, and decides the results with a threshold filtering method. All of these three approaches are coding in Java, and we run each of them on the same server with 32GB memory and Ubuntu Server 8.10 operating system. The A-R-S benchmark dataset is used for efficiency analysis, and the result is shown in Table 4. We can see that VMI performs much better than the three baseline methods in terms of efficiency. On the rexa-dblp dataset which is approximately 10,000 ×1,000,000 size, VMI is about 100 times faster than RiMOM. As we can learn from Table 6, VMI generates comparable matchings with RiMOM, so VMI is a much better solution for large scale inputs. The great improvement of efficiency comes from the using of two inverted indexes and candidate selection rules. By candidate selection, VMI cuts off large portion of candidates in pair-wise way. We give the runtime of VMI on some large scale input in Table 5. With the increase of input scale, the runtime does not increase with the same speed. VMI is able to finish the matching process of input in million-scale within a few hours. But when it comes to the million-scale, the runtime is going up faster, so maybe more strict rules should be applied to get a smaller amount of candidates.

Table 4: Runtime on A-R-S benchmark

Method	eprints-rexa	eprints-dblp	rexa-dblp
VMI	10s	16min10s	19min52s
Edist	2min8s	3h15min	>10h
RiMOM	1min33s	4h18min	36h34min
DSSim	18min7s	3h23min	20h32min

Table 5: The Runtime of VMI on Some Large Scale Input

Data	Scale	Time
drugbank-DBpedia	10,000×2,000,000	14min18s
rexa-dblp	15,000×1,600,000	19min52s
dailymed-DBpedia	20,000×2,000,000	21min3s
tap-DBpedia	65,000×2,000,000	29min42s
sweto-DBpedia	800,000×2,000,000	114min37s
dblp-DBpedia	1,600,000×2,000,000	181min9s

Table 6: VMI Performance on A-R-S benchmark compared with OAEI participants

	eprints-rexa			eprints-dblp			rexa-dblp		
System	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
VMI	0.80	0.90	0.85	0.62	0.70	0.66	0.71	0.78	0.76
RiMOM	0.94	0.59	0.80	0.93	0.70	0.73	0.78	0.67	0.73
Hmatch	0.95	0.46	0.62	0.65	0.65	0.65	0.42	0.48	0.45
DSSim	0.60	0.28	0.38	0.11	0.15	0.13	0.00	0.00	0.00
FBEM	0.94	0.10	0.18	0.98	0.16	0.28	0.99	0.12	0.21

6.4. Accuracy Performance

We compare VMI with other systems participated A-R-S benchmark in OAEI 2009. The result is shown in Table 5. The results show that VMI explores the semantic information in instances effectively and generates the matchings as good as the OAEI 2009 champion RiMOM system. We can see on eprints-rexa and rexa-dlbe, VMI gets the best F1-Score. However, VMI gets a lower precision on eprints-dblp dataset. By analyzing the false matchings in the results, we find out that many of the false ones come from the isolated person instances without any descriptions and neighboring information. By only using the constraint refinement, it is difficult to filter out all the false alignments generated from Name Vector with the huge amount of instances in the dblp dataset. In all three tasks, VMI achieves the best recall. This fact indicates that the candidate selection strategy of VMI indeed filters out the impossible pairs while keeping the possible ones.

We compare the results of VMI with SILK system on the LinkedMDB-DBpedia dataset. SILK is able to produce 26059 sure results and 1856 results to be confirmed. VMI is also able to generate 25979 results. We randomly choose 200 ones from the results and check them manually and 182 out of them are correct. So the results of VMI are in the same level as the ones of SILK in both quantity and quality.

In summary, we have following conclusion of VMI with the experiment results: 1) VMI explores the semantic information with instances effectively and can generate matching results with high quality. It achieves slightly better F1-Score than the champion RiMOM system in OAEI 2009. 2) The multiple indexing and candidate selection strategy improves the efficiency of matching process greatly. Compare with the existing methods, VMI shortens the running time greatly on datasets with millions of instances.

7. Conclusion and Future Work

In this paper we have proposed an approach VMI to match large scale instances via multiple indexing and candidate selection. VMI builds two different vectors for each instance, the Name Vector and the Virtual Document Vector. Two inverted indexes are produced for indexing all the two groups of vectors. In the matching process, matching candidates are selected based on several heuristic rules defined on the indexes. By using multiple indexing and candidate selection, VMI greatly reduces the number of similarity computations, and therefore performs efficiently on matching large scale instance sets.

In the future, more carefully designed candidate selection rules could be introduced into the algorithm. With even larger scale of data, the memory consumption will becomes the bottleneck of VMI, the vectors and indexes may have to be stored on harddisks instead of in the memory and a better structure of indexes in company with good index compression is in need. Moreover, a complete implementation of the algorithm in parallel is a promising solution for further scalability.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61035004, 61202246, 61222212, 61073073), NSFC- ANR (No. 61261130588), 863 High Technology Program (2011AA01A207), FP7-288342, and THU-NUS NExT Co- Lab.

References

[1] Y. Afacan and H. Demirkan. An ontology-based universal design knowledge support system. *Knowledge-Based Systems*, 24(4):530 – 541, 2011.

[2] S. Albagli, R. Ben-Eliyahu-Zohary, and S. E. Shimony. Markov network based ontology matching. *Journal of Computer and System Sciences*, (0):–, 2011.

[3] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *Proceedings of Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 906–908, 2005.

[4] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2003.

[5] A. Bilke and F. Naumann. Schema matching using duplicates. In *Proceedings of the 21st International Conference on Data Engineering,*, pages 69–80, 2005.

[6] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[7] S. Castano, A. Ferrara, S. Montanelli, and D. Lorusso. Instance matching for ontology population. In *Proceedings of the 16th Italian Symposium on Advanced Database Systems*, pages 121–132, 2008.

[8] N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *ACM SIGMOD Record*, 35(3):34–41, Sept. 2006.

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

[10] D. Engmann and S. Maßmann. Instance matching with COMA++. In *Proceedings of Datenbanksysteme in Business, Technologie and Web(BTW 07)*, pages 28–37, 2007.

[11] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[12] R. Gligorov, W. ten Kate, Z. Aleksovski, and F. van Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 767–776, New York, NY, USA, 2007. ACM.

[13] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, (2):199–220, 1993.

[14] Q. Guo and M. Zhang. Question answering based on pervasive agent ontology and semantic web. *Knowledge-Based Systems*, 22(6):443 – 448, 2009.

[15] D. Gusfield. *Algorithms on strings, trees, and sequences.* Cambrige University Press, 1997.

[16] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27:83–85, 2005.

[17] A. Hogan, A. Harth, and S. Decker. Performing object consolidation on the semantic web data graph. In *Proceedings of the WWW2007 Workshop I3: Identity, Identifiers, Identification, Entity-Centric Approaches to Information and Knowledge Management on the Web*, 2007.

[18] W. Hu, Y. Qu, and G. Cheng. Matching large ontologies: A divide-and-conquer approach. *Data and Knowledge Engineering*, 67(1):140–160, 2008.

[19] Y. R. Jean-Mary, P. Shironoshita, and M. R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, (3):235–251, 2009.

[20] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(01):1–31, 2003.

[21] J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: A dynamic multistrategy ontology alignment framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1218–1232, 2009.

[22] M. Mao, Y. Peng, and M. Spring. An adaptive ontology mapping approach with neural network based constraint satisfaction. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(1):14 – 25, 2010.

[23] M. Nagy, M. Vargas-Vera, and P. Stolarski. Dssim results for oaei 2009. In *Proceedings of International Workshop on Ontology Matching*, pages –1–1, 2009.

[24] S. Pavel and J. Euzenat. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, PP(99), 2011.

[25] J. Pulido, M. Ruiz, R. Herrera, E. Cabello, S. Legrand, and D. Elliman. Ontology languages for the semantic web: A never completely updated review. *Knowledge-Based Systems*, 19(7):489 – 497, 2006.

[26] M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt. Learning domain ontologies for semantic web service descriptions. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):340 – 365, 2005. ¡ce:title¿World Wide Web Conference 2005——Semantic Web Track¡/ce:title¿ ¡xocs:full-name¿World Wide Web Conference 2005——Semantic Web Track¡/xocs:full-name¿.

[27] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.

[28] N. A. Segura, Salvador-Sánchez, E. García-Barriocanal, and M. Prieto. An empirical analysis of ontology-based query expansion for learning resource searches using merlot and the gene ontology. *Knowledge-Based Systems*, 24(1):119 – 133, 2011.

[29] P. Shvaiko and J. Euzenat. Ten challenges for ontology matching. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008*, volume 5332 of *Lecture Notes in Computer Science*, pages 1164–1182. Springer Berlin / Heidelberg, 2008.

[30] H. Stoermer and N. Rassadko. Results of okkam feature based entity matching algorithm for instance matching contest of oaei 2009, 2009.

[31] J. Tang, J. Li, B. Liang, X. Huang, Y. Li, and K. Wang. Using bayesian decision for ontology mapping. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(4):243 – 262, 2006.

[32] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 350–359, 2002.

[33] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 650–665. Springer Berlin / Heidelberg, 2009.

[34] S. Wang, G. Englebienne, and S. Schlobach. Learning concept mappings from instance similarity. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 339–355. Springer Berlin / Heidelberg, 2008.

[35] T. Zhang, D. Xu, and J. Chen. Application-oriented purely semantic precision and recall for ontology mapping evaluation. *Knowledge-Based Systems*, 21(8):794 – 799, 2008.

[36] X. Zhang, Q. Zhong, F. Shi, J. Li, and J. Tang. RiMOM results for oaei 2009. In *Proceedings of International Workshop on Ontology Matching*, 2009.

[37] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), July 2006.