Data & Knowledge Engineering 67 (2008) 140–160

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Matching large ontologies: A divide-and-conquer approach

Wei Hu *, Yuzhong Qu, Gong Cheng

School of Computer Science and Engineering, Southeast University, Nanjing 210096, PR China

ARTICLE INFO

Article history:
Received 28 January 2008
Received in revised form 9 June 2008
Accepted 11 June 2008
Available online 20 June 2008

Keywords:
Ontology matching
Data integration
Semantic heterogeneity

ABSTRACT

Ontologies proliferate with the progress of the Semantic Web. Ontology matching is an important way of establishing interoperability between (Semantic) Web applications that use different but related ontologies. Due to their sizes and monolithic nature, large ontologies regarding real world domains bring a new challenge to the state of the art ontology matching technology. In this paper, we propose a divide-and-conquer approach to matching large ontologies. We develop a structure-based partitioning algorithm, which partitions entities of each ontology into a set of small clusters and constructs blocks by assigning RDF Sentences to those clusters. Then, the blocks from different ontologies are matched based on precalculated anchors, and the block mappings holding high similarities are selected. Finally, two powerful matchers, V-Doc and Gmo, are employed to discover alignments in the block mappings. Comprehensive evaluation on both synthetic and real world data sets demonstrates that our approach both solves the scalability problem and achieves good precision and recall with significant reduction of execution time.

1. Introduction

The Semantic Web is an effort by the W3C Semantic Web Activity to achieve data integration and sharing across different applications and organizations. Ontologies, as a new representational form of knowledge, play a prominent role in the development and deployment of the Semantic Web. To date, the popularity of ontologies is rapidly growing, and the amount of available ontologies continues increasing. For example, an ontology search engine, Swoogle [12], reports that it has collected more than 10,000 ontologies on the Web so far.

Due to the decentralized nature of the Web, there usually exist multiple ontologies from overlapped application domains or even within the same domain. In order to establish interoperability between (Semantic) Web applications that use different but related ontologies, ontology matching has been proposed as an effective way of handling the semantic heterogeneity problem. It is typically useful to some data management applications, such as information integration and distributed query processing.

Generally, the well-known "80/20" rule applies in ontology matching [1]. Automatic matching algorithms and tools can automate 80% of the work, covering common cases and creating results that are close to correct. As surveyed in [16], a wide range of characteristics in ontologies to be matched are exploited by existing approaches, such as linguistic descriptions (e.g., *rdfs:label*), structural information (e.g., *rdfs:subClassOf*) and data instances. In addition, some of them also utilize background knowledge from thesauri or third parties' ontologies. The rest 20% requires manual contribution to perfect portions of the results. Past work mainly focuses on human collaboration [40,65] and matching visualization [1,9,17,36], which significantly facilitate matching understanding and refinement. In this paper, we aim at proposing automatic approaches for ontology matching.

* Corresponding author. Tel: +86 25 5209 0908; fax: +86 25 5209 0880.
E-mail addresses: whu@seu.edu.cn, whu1982@gmail.com (W. Hu), yzqu@seu.edu.cn (Y. Qu), gcheng@seu.edu.cn (G. Cheng).

0169-023X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2008.06.003

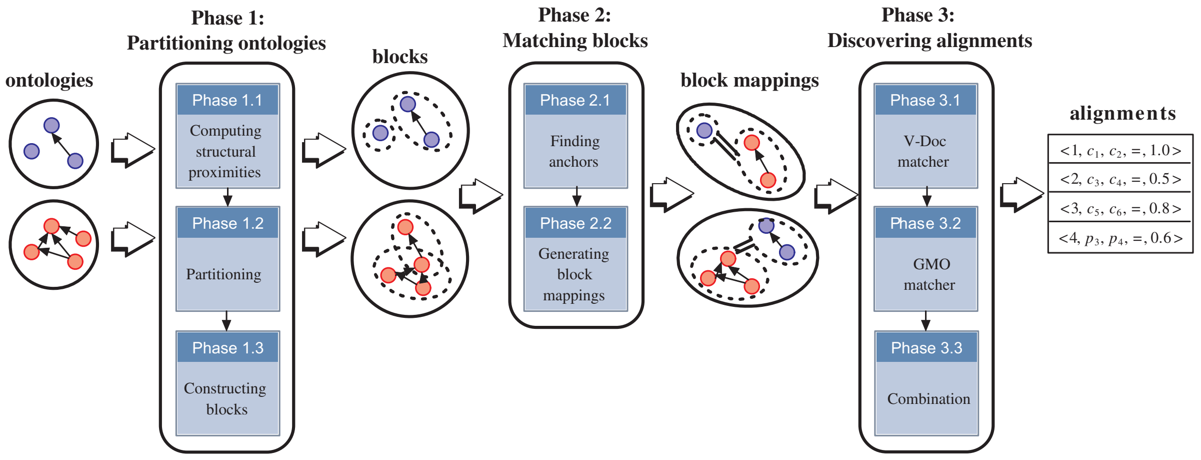


Fig. 1. Overview of the approach.

Based on our investigation, most of the state of the art automatic approaches are merely applicable for small-scale ontologies, and the effectiveness of these approaches decreases for large ontologies. As concluded in the latest Ontology Alignment Evaluation Initiative 2007 (OAEI) [15], 17 ontology matching tools joined the campaign, but only four tools (including the one described in this paper) accomplished all the three matching tasks with large ontologies (`Anatomy`, `Food` and `Library`). But, the other three tools, namely DSSim [43], RiMOM [58] and Prior+ [39], do not really solve the large ontology matching problem. DSSim manually partitions large ontologies into several smaller pieces, while RiMOM and Prior+ use simple string comparison techniques as alternatives.

As of today, there are quite a lot of applications that require matching large ontologies. For example, in library management, large book classification categories, such as `Brinkman` and `GTT` thesauri,[1] are needed to be integrated to remove redundant books. For another example, in the medicine and biology domains, large life-science ontologies, e.g., `GALEN`[2] and `FMA` [50], are required to be matched in order to provide a uniform service for access and manipulation [63]. Therefore, these emerging demands on matching large ontologies bring a new challenge to the state of the art ontology matching technology.

It is our goal in this paper to explore ways of matching large ontologies. The main principle of our approach is based on the divide-and-conquer philosophy. The framework of the approach is illustrated in Fig. 1. Specifically, the input is two large ontologies to be matched. The approach firstly partitions the entities (i.e., classes and properties) of each ontology into a set of small clusters by measuring their structural proximities, and then constructs blocks via assigning RDF Sentences [64] (i.e., a kind of integrated units over RDF triples) to the clusters (**Phase 1**). Those blocks are matched based on precalculated anchors (i.e., matched entities gained beforehand), and block mappings (i.e., matched blocks with high similarities) are selected (**Phase 2**) to be further matched by two powerful matchers, V-Doc [46] and Gmo [27], for outputting alignments (**Phase 3**). In this paper, we focus on the ontologies written in RDFS [5] or OWL [45], because they are the latest common formats recommended by W3C for publishing ontologies. Furthermore, the work in [8] has analyzed 3959 ontologies available on the Web, in which 3258 (nearly 82%) are expressed in RDFS or OWL.

Our approach has two major advantages. Firstly, it solves the scalability problem of matching large ontologies, i.e., it does not suffer from lack of memory (e.g., the "out of memory" error, which is caused by not enough memory to be allocated). According to the OAEI 2007 report, most matching tools can only work on small ontologies with hundreds of entities, while our approach can deal with very large ontologies with ten thousands of entities. Secondly, it significantly decreases the execution time with pretty good quality. For example, two large ontologies are provided in the `Anatomy` track of OAEI 2007 [15], each one contains about 3000 entities. DSSim, RiMOM and Prior+ spend 75 min, 4 h and 23 min to complete the task, respectively, while our approach merely takes 12 min. More importantly, as compared to DSSim, RiMOM and Prior+, our approach achieves better (or at least comparable) precision and recall.

It is also worthy of noting that the approach presented in this paper substantially improves our previous conference paper [30] with the following aspects: (1) it utilizes RDF Sentences [64] as basic units to construct blocks from the clusters. RDF Sentences provide more integrated syntactic and semantic structures than RDF triples (i.e., statements), since they encapsulate blank nodes. As a result, the completeness of blank nodes, which is the lowest requirement for blocks, is preserved; (2) it extends the partitioning algorithm for properties in ontologies, while our previous work only copes with class hierarchies; (3) it integrates two powerful matchers, V-Doc and Gmo, to discover alignments; and (4) we comprehensively evaluate our approach on both synthetic and real world data sets. In the synthetic test, two new kinds of metrics (the partitioning quality

[1] http://stitch.cs.vu.nl/ – semantic interoperability to access cultural heritage.
[2] http://www.opengalen.org/.

and the mapping quality) are defined to assess the effectiveness of our approach. In the real world test, we attended OAEI 2007 and compared our approach with other participants.

The remainder of this paper is organized as follows: Section 2 defines the terminologies and notations used in this paper. Section 3 discusses related work. Section 4 presents the computation of structural proximities, the partitioning algorithm and the construction of blocks. Section 5 introduces the approaches to anchor generation and block matching. Section 6 summarizes two matchers V-Doc, Gmo and our combination strategy. Section 7 reports experimental results on both synthetic and real world data sets. Finally, Section 8 concludes the paper with future work.

2. Problem formulation

An *ontology* is a formal, explicit specification of a shared conceptualization [21]. In this paper, we use a basic definition of ontologies based on the RDF graph model [35,45], which is general enough to cover most of the state of the art large ontologies.

Definition 1. (ontology) An ontology $\mathcal{O}$ is a set of RDF triples T. Any RDF triple t $(t \in T)$ denotes a statement of the form $\langle \text{subject}, \text{predicate}, \text{object} \rangle$. Any node in an RDF triple may be a URI with an optional local name (URIref), a literal, or a blank node (having no separate form of identification). A predicate is always a URIref, and a literal cannot be a subject.

Without explanation, ontologies shown in this paper are expressed in RDFS or OWL. An OWL ontology can easily be transformed to an RDF graph [45]. In our notation, we use c to represent a *class* and p to represent a *property*. The specification [5] provides the basic mechanism to recognize classes and properties from nodes in RDF triples. For simplicity, classes and properties are uniformly called *entities* (d denotes an entity). In this paper, we consider an ontology containing more than one thousand entities as a *large* ontology.

Ontology matching (also called mapping or aligning) aims at discovering *alignments* (also named mappings, correspondences or matches) between semantically similar entities in different ontologies. These alignments might stand for equivalence as well as other relationships, such as subsumption or disjointness, between entities [16]. Alignments can be used in a variety of applications, such as ontology merging, query answering, data translation or for browsing the Semantic Web. Inspired by the definitions in [54], we define ontology matching as follows.

Definition 2. (ontology matching) Let $\mathcal{O}, \mathcal{O}'$ be two ontologies. Matching $\mathcal{O}$ with $\mathcal{O}'$ finds a set of alignments $A = \{a_1, a_2, \ldots, a_n\}$. Each a_i $(i = 1, 2, \ldots, n)$ is a 5-tuple: $\langle id_1, d, d', u, v \rangle$, where id_1 is a unique identifier; d is an entity in $\mathcal{O}$, and d' is an entity in $\mathcal{O}'$; u is an equivalence (=), subsumption ($\sqsubseteq$ or $\sqsupseteq$) or disjointness ($\perp$) relationship holding between d and d'; and v is a similarity between d and d' in the $[0, 1]$ range.

In this paper, matching large ontologies specifically indicates each of the ontologies to be matched is large, rather than matching a large amount of small ontologies (called holistic matching in [25]). Furthermore, we only consider the equivalence relationship between entities.

Precisely defining blocks requires the notion of the RDF Sentence, so we first give the definition of the RDF Sentence [64]. An equivalent definition of the RDF Sentence is the minimum self-contained graph [60]. Following the theorems in [60], an RDF graph can be decomposed into a unique set of RDF Sentences.

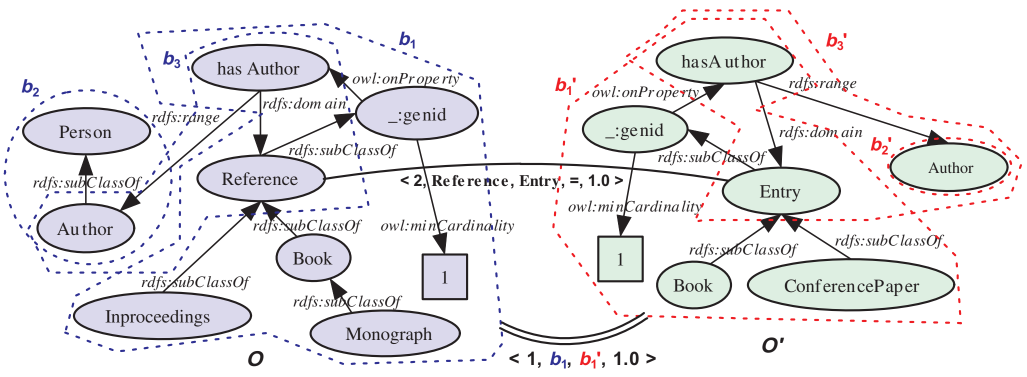
In RDF (also RDFS and OWL), there is a special kind of nodes, called *blank nodes* (or *bnodes*), which are not identified by URIs. Blank nodes are a sort of existentially quantified resource whose meaning is in the scope of the triples it appears. RDF triples sharing a common blank node provide an integrated structure indicating a joint context of the blank node, and if such triples are separated, the context is broken. For example, in the left part of Fig. 2, the ontology $\mathcal{O}$ has a blank node `_:genid`, which is shared by three RDF triples: t_1, t_2, t_3. Those three triples form a restriction structure, which constrains a `Reference` at least `has` one `Author`. At present, RDF semantics [35] does not provide any intrinsic mechanism to guarantee this kind of structure. We define it as an *RDF Sentence*.

We say that two RDF triples are b-connected if they share blank nodes. An RDF Sentence is the maximum closure of the b-connected RDF triples.[3]

Definition 3. (RDF Sentence) Let $\mathcal{O}$ be an ontology. An RDF Sentence s is a set of RDF triples, which satisfies the following conditions:

(1) $s \subseteq \mathcal{O}$;
(2) $\forall t_i, t_j \in s$, $i \neq j$, t_i, t_j are b-connected;
(3) $\forall t_i \in s, t_j \notin s$, t_i, t_j are not b-connected.

[3] In implementation, ontology parsing tools usually assign system-generated internal names to blank nodes. We transform an RDF graph into an undirected triple graph, where each vertex represents an RDF triple in the original RDF graph. An edge exists between two RDF triples iff they contain blank nodes having the same name. We call that the two RDF triples share a blank node. Then, each connected component in the triple graph forms an RDF Sentence.



b_1	t_1: < Reference, *rdfs:subClassOf*, _:genid >
	t_2: < _:genid, *owl:onProperty*, hasAuthor >
	t_3: < _:genid, *owl:minCardinality*, "1" >
	t_4: < Inproceedings, *rdfs:subClassOf*, Reference >
	t_5: < Book, *rdfs:subClassOf*, Reference >
	t_6: < Monograph, *rdfs:subClassOf*, Book >
b_2	t_7: < Author, *rdfs:subClassOf*, Person >
b_3	t_8: < hasAuthor, *rdfs:domain*, Reference >
	t_9: < hasAuthor, *rdfs:range*, Author >

rdfs = "http://www.w3.org/2000/01/rdf-schema#"
owl = "http://www.w3.org/2002/07/owl#"

b_1'	t_1': < Entry, *rdfs:subClassOf*, _:genid >
	t_2': < _:genid, *owl:onProperty*, hasAuthor >
	t_3': < _:genid, *owl:minCardinality*, "1" >
	t_4': < ConferencePaper, *rdfs:subClassOf*, Entry >
	t_5': < Book, *rdfs:subClassOf*, Entry >
b_3'	t_6': < hasAuthor, *rdfs:domain*, Entry >
	t_7': < hasAuthor, *rdfs:range*, Author >

Fig. 2. A toy example.

An RDF triple whose subject is not a blank node is defined as a *main* RDF triple. In principle, an RDF Sentence is composed of a main RDF triple, and all the other triples in the RDF Sentence are b-connected to it. The *subject* of an RDF Sentence is the subject of its main RDF triple.

Now, we give the definition of blocks by using RDF Sentences as basic units instead of RDF triples.

Generally speaking, a *block* can be viewed as a subset of an ontology (i.e., a set of triples). Compared to a module [20] or a cluster [22], a block is neither strictly complete in logics nor merely a set of entities. For example, in [20], an ideal module guarantees to capture the meanings of the entities in an OWL DL ontology. That is, when answering arbitrary queries against an ontology, importing the module would give us exactly the same answers as if we had imported the whole ontology. A *cluster* is comprised of a set of similar entities grouped together. A *partitioning* breaks all the entities in an ontology into a set of clusters, satisfying: (1) for any two different clusters, they are disjoint; and (2) the union of all the clusters equals to the entire set of the entities.

Definition 4. (block) Let $\mathcal{O}$ be an ontology and D be a set of all the entities in $\mathcal{O}$. A partitioning, G, of D, breaks D into a set of clusters $\{g_1, g_2, \ldots, g_n\}$, which satisfies: (1) $\forall g_i, g_j$, $i, j = 1, 2, \ldots, n$ and $i \neq j$, $g_i \bigcap g_j = \emptyset$; and (2) $g_1 \bigcup g_2 \bigcup \cdots \bigcup g_n = D$. Let b_i be the block corresponding to a cluster g_i $(i = 1, 2, \ldots, n)$. b_i is a union of RDF Sentences ($b_i = s_1 \bigcup s_2 \bigcup \cdots \bigcup s_m$), where each s_k $(k = 1, 2, \ldots, m)$ satisfies $subject(s_k) \in g_i$. $\forall b_i, b_j$, $i, j = 1, 2, \ldots, n$ and $i \neq j$, $b_i \bigcap b_j = \emptyset$.

In this paper, we carefully differentiate between clusters and blocks. A cluster is a set of entities, while a block is a set of RDF triples. For any two different blocks, the triples in the two blocks are disjoint. But, the entities contained in these triples may overlap. Furthermore, the union of all the blocks occasionally does not equal to the entire ontology, because a kind of RDF triples is missing: those triples with blank nodes as subjects that have never been objects (e.g., *owl:AllDifferent*).

A *block mapping* is a pair of matched blocks from two ontologies. We refer to the process of discovering block mappings as *block matching*. In this paper, please note that we allow a block to appear in several block mappings and do not specify the relationship of two blocks within a block mapping.

Definition 5. (block matching) Let B, B' be two sets of blocks derived from two ontologies $\mathcal{O}, \mathcal{O}'$, respectively. Block matching B with B' finds a set of block mappings $\mathrm{BM} = \{bm_1, bm_2, \ldots, bm_n\}$. Each bm_i $(i = 1, 2, \ldots, n)$ is a 4-tuple: $\langle id_2, b, b', f \rangle$, where id_2 is a unique identifier; b, b' are two blocks in B, B' respectively; and f is a similarity between b and b' in the $[0, 1]$ range.

To help understanding, we illustrate a toy example here. Let us see the two ontologies shown in Fig. 2 (typing triples are not depicted in the figure). The ontology $\mathcal{O}$ in the left part includes six classes `Reference`, `Book`, `Inproceedings`, `Monograph`, `Person` and `Author`; and one property `hasAuthor`. We can generate three blocks b_1, b_2, b_3 (enclosed by dotted lines) from three clusters $g_1 = \{$`Reference`, `Book`, `Inproceedings`, `Monograph`$\}, g_2 = \{$`Author`, `Person`$\}, g_3 = \{$`hasAuthor`$\}$. Note that the triples among b_1, b_2 and b_3 are disjoint, not the entities. For example, `hasAuthor` is shared by b_1 and b_3, but the RDF triple t_8 only belongs to b_3. The ontology $\mathcal{O}'$ in the right part has five entities: `Entry`, `Book`, `ConferencePaper`, `Author` and `hasAuthor`. We may also generate three blocks b'_1, b'_2, b'_3 (enclosed by dotted lines as well). Some block mappings may be found such as $\langle 1, b_1, b'_1, 1.0\rangle$. Some alignments might also be discovered such as $\langle 2, \texttt{Reference}, \texttt{Entry}, =, 1.0\rangle$.

3. Related work

Although quite a lot of work has addressed the ontology matching problem, there are few generic approaches that raise the issue of matching large ontologies. In this section, we will focus on discussing related work on large ontology matching. Regarding matching small ontologies, it has been well surveyed in literature, and we refer the reader to [14,16,31,44,48,54].

3.1. Large ontology matching

To match large ontologies, a few light-weight ontology matching approaches, on the one hand, can be directly used. For example, edit-distance based methods [37,55] compute the similarity between two strings (e.g., *rdfs:label*(s)) by counting the number of operations required to transform one of them into the other. Because they only compare two strings in each time, they do not have the scalability limit and can be applied to match ontologies with any size. On the other hand, some work adapts the matching algorithms to reduce memory consumption. For example, the work in [42] simplifies a graph matching algorithm to match two large medical taxonomies by only considering the direct children and grandchildren of entities. Besides, using proper data structures (e.g., sparse matrices) and secondary storage (e.g., hard disks) might also facilitate large ontology matching. Compared with these methods, our approach addresses the problem from another direction based on the divide-and-conquer strategy.

In addition, there exist some domain-specific approaches that address matching large ontologies in particular areas (e.g., in biology [34,63] and geography [9]). Usually, they utilize certain light-weight methods to gain initial candidates and use domain knowledge to infer alignments. Aoas [63] is a representative tool that is heavily specialized on matching biomedical ontologies and makes extensive use of medical background knowledge. For the `Anatomy` track in OAEI 2007, it takes a broader ontology `FMA` as the background knowledge to further improve the initial results. Because these approaches heavily depend on domain knowledge, they are not general-purpose solutions and easily fail when such knowledge is unavailable. But, if there is proper background knowledge, they can find some complex and interesting alignments.

In OAEI 2007, three generic ontology matching tools, DSSim [43], RiMOM [58] and Prior+ [39], took part in all the three tracks with large ontologies. DSSim designs a query-like framework, which expands each entity from one of the two ontologies to be matched to a query fragment in terms of WordNet.[4] RiMOM is based on the risk minimization model and integrates seven different ontology matching strategies, each one aims at a specific ontological information. Prior+ is an automatic ontology mapping tool based on propagation theory, information retrieval technique and artificial intelligence model. Although the three tools showed their results for the large ontology matching tracks, they do not propose methods to solve the problem. DSSim manually divides large ontologies into small blocks, while RiMOM and Prior+ adopt edit-distance based methods as alternatives.

3.2. Ontology partitioning

In the first stage of our approach, two ontologies are partitioned into two sets of blocks, so it may be broken down into the category of ontology modularization. Among the literature, some representative work is in [20,52,56]. The work in [52] introduces a way of constructing stand alone fragments by exhaustively traversing various links (i.e., properties) from a specified entity. The work in [56] presents a framework based on Distributed Description Logics, which uses bridge rules to support modularization. But, the mechanism for manipulating the sizes of the modules is not clear. The work in [20] studies the problem of extracting minimal modules by approximation in the context of collaborative ontology development and controlled integration, which makes the sizes of the modules relatively small. These ontology modularization approaches have realized the importance of scalability and made progress in reducing the sizes of modules, however, they still have the scalability problem and are inapplicable in the context of matching large ontologies yet. For example, by applying the approach in [20] to `GALEN`, the maximum module still owns more than 7000 entities, which is not an easily manageable scale for the state of the art matching approaches. Our partitioning approach aims at controlling the sizes of blocks, while it still satisfies the lowest requirement for modules, i.e., the completeness of blank nodes in blocks [60]. However, please note that, if such modularization approaches could flexibly control the sizes of modules in the future, they might be used to replace our partitioning approach.

Although the major contribution of the work in [59] is ontology visualization, it also gives a way of partitioning large ontologies by using a data clustering algorithm (the Force Directed Placement algorithm). The basic ideas of the approach in [59] and our approach are the same, and both of them transform ontologies into graphs and use clustering algorithms to partition the graphs. However, compared to it, not only does our approach partition entities in an ontology to disjoint clusters hierarchically, but it also uses RDF Sentences to recover RDF triples to the clusters, while the method in [59] does not provide any mechanism to preserve the triples that contain blank nodes.

[4] http://wordnet.princeton.edu/.

3.3. Block matching

To the best of our knowledge, Bmo [28] and Coma [2,13,49] are the only two work we know so far that consider the block matching problem. Bmo matches two ontologies from a linguistic perspective and generates a similarity matrix. Afterwards, it applies a spectral partitioning algorithm to that similarity matrix to obtain block mappings directly. Because it requires to match two full ontologies, it may suffer from the scalability problem and be not feasible to match large ontologies. Besides, it uses a linguistic measure to calculate the distances between entities, so the quality of its generated blocks is not good.

Coma is a generic schema and ontology matching tool, providing a collection of elementary matchers and a flexible infrastructure to combine those matchers. It develops a fragment-based matching method to solve the large schema matching problem. It transforms each ontology into a directed acyclic graph and decomposes the graph from top to bottom. During decomposition, when the number of entities in any block exceeds a pre-defined value, a compaction function would be called to the bottom entities in the block. As a result, the descendants of a bottom entity are all represented by the entity, and another block starts construction. It is originally used to partition database schemas or XML schemas, which are usually represented as trees rather than complex graphs, so it does not well fit ontologies that have complex structures. Another problem is that it ignores the different features of classes and properties. So, the partitioning quality of Coma is not very good. Besides, when matching fragments, it merely uses the local descriptions of roots to represent the whole fragments, therefore some matched fragments may be skipped.

In schema matching (see [14,48]), iMap [10] semi-automatically finds both 1:1 and very complex alignments (e.g., $\text{room}-\text{price}=\text{room}-\text{rate}\cdot(1+\text{tax}-\text{rate})$). It embeds two kinds of domain knowledge (overlapped data and external data) to infer complex alignments. However, it is hard to specify the domain knowledge in some cases, and the granularity of those complex alignments is low. Artemis [6] is another work that vaguely presents the idea of block matching. It computes the 1:1 alignments between two schemas by using WordNet and generates block mappings from the 1:1 alignments via a clustering algorithm. This is similar to the framework of Bmo. Hence, it also suffers from the high computational complexity for finding the 1:1 alignments.

4. Partitioning ontologies

In this section, we will first present the measure of structural proximities (see **Phase 1.1** in Fig. 1). Then, we will introduce an agglomerative algorithm to partition entities into a number of clusters based on the structural proximities (**Phase 1.2**). Finally, we will describe the construction of blocks by assigning RDF Sentences to the clusters (**Phase 1.3**).

4.1. Computing structural proximities

According to the investigation in [3], a large number of ontology design patterns are employed during the development of ontologies. For large ontology construction, we highlight two design principles. The first principle is named the classification principle, which categorizes and specifies concepts (e.g., in FMA [50]) so that the taxonomy feature is depicted; the second one is the modularization principle [56], which provides different levels of granularity to support maintenance and re-use. These two kinds of principles are mainly revealed in the hierarchies of large ontologies, i.e., embedded in the *rdfs:subClassOf* and *rdfs:subPropertyOf* relationships between entities. They inspire us to measure the hierarchical distances in order to characterize their structural proximities.

Structural proximities between classes are defined based on how closely they are related in the hierarchies of the *rdfs:subClassOf* relationships. Let c_i, c_j be two classes in a given ontology $\mathcal{O}$, the structural proximity between c_i and c_j is:

$$\text{prox}(c_i, c_j) = \frac{2\cdot\text{depth}(c_{ij})}{\text{depth}(c_i)+\text{depth}(c_j)}, \tag{1}$$

where c_{ij} is the common superclass of c_i and c_j, and $\text{depth}(c_k)$ gets the depth of c_k in the original class hierarchy (i.e., without inferencing). Please note that these class hierarchies of large ontologies are usually represented as directed acyclic graphs (if there exists any cycle, we break up the cycle by arbitrarily removing an edge) rather than trees, so the depth of c_k is not unique. In this paper, we choose the maximum one as the depth of c_k, since it is most specific in semantics. Also, the common superclass of c_i and c_j is not unique, so we choose the one, which has the maximum depth, as the common superclass of c_i and c_j.

Formula (1) gives two meanings: (1) the deeper the common superclass of two classes is, the closer the two classes are; and (2) the structural proximity of two classes strengthens as their depths increase. For example, let us consider five classes: `Object`, `Human`, `Animal`, `Woman` and `Man`. `Woman` and `Man` are two subclasses of `Human`, while `Human` and `Animal` are two subclasses of `Object`. Hence, `Woman` and `Man` are closer than `Human` and `Animal`. (If we assume that the depth of `Object` is 1, then $prox(\texttt{Man},\texttt{Woman}) = 2\cdot 2/(3+3) > 2\cdot 1/(2+2) = prox(\texttt{Human},\texttt{Animal})$.)

Formula (1) has also been presented in [59] for ontology browsing. In practice, for many popular ontology visualization tools such as Protégé,[5] class hierarchies are the most important structure for browsing. Also, in large ontologies, the number

[5] http://protege.stanford.edu/.

of the *rdfs:subClassOf* relationships is considerably dominant as compared to others. For instance, `GALEN` contains nearly 10,000 classes and 30,000 RDF triples excluding typing triples and annotations, in which more than 18,000 triples are about *rdfs:subClassOf*. Moreover, for thesauri such as `Brinkman`, they only have the *rdfs:subClassOf* relationships.

The hierarchies of the *rdfs:subPropertyOf* relationships between properties are not so prevalent as the ones between classes. Therefore, structural proximities between properties are computed not only by how closely they are placed in the hierarchies but also by whether they have overlapped *rdfs:domain*(s). The properties with the overlapped *rdfs:domain*(s) serve for the same class, so they are related. Let p_i, p_j be two properties in a given ontology $\mathcal{O}$, the structural proximity between p_i and p_j is:

$$\text{prox}(p_i, p_j) = \frac{\text{depth}(p_{ij})}{\text{depth}(p_i) + \text{depth}(p_j)} + \frac{|\text{dom}(p_i) \bigcap \text{dom}(p_j)|}{|\text{dom}(p_i)| + |\text{dom}(p_j)|}, \tag{2}$$

where p_{ij} is the common super-property of p_i and p_j, depth(p_k) gets the depth of p_k in the original property hierarchy, and dom(p_k) gets the domain(s) of p_k.

Besides, two optimizations are adopted: (1) some entities are identified as the same in a preprocessing stage, such as two entities are explicitly declared identical by using the *owl:equivalentClass* or *owl:equivalentProperty* relationships; and (2) only the structural proximities of entities having adjacent depths are measured in order to reduce the computational complexity, i.e., only computing the structural proximities of entities satisfying $|\text{depth}(d_i) - \text{depth}(d_j)| \leqslant 1$. It is worthy of noting that this approximation is often used in structure-based partitioning approaches (e.g., in [42]).

Let us see the example shown in Fig. 3. Given an ontology in the left part of the figure, after executing **Phase 1.1**, the structure proximities between the entities are depicted in the right part of the figure.

4.2. Partitioning

The objective of a partitioning algorithm is to partition a set of vertices V into a set of disjoint clusters $g_1, g_2, \ldots, g_n$, where, by certain measure, the cohesiveness among the vertices in a cluster g_i is high; meanwhile the coupling crossing different clusters g_i, g_j is low. In the context of this paper, we seek to partition entities of an ontology into several clusters, so that the structural proximities among the entities in a cluster are high, while those crossing different clusters are low. Please note that the computed clusters satisfy the two conditions given in Section 2, i.e., any two different clusters are disjoint; and the union of all the clusters equals to the complete set of the entities.

The partitioning algorithm proposed in this paper is an agglomerative (i.e., bottom-up) partitioning algorithm mainly extended from Rock [22], which is a very scalable agglomerative clustering algorithm in the area of data mining. The main difference between Rock and our algorithm is that we use the *cut*() function as the criterion function in order to improve the efficiency. Another difference is that we consider floating point values (i.e., structural proximities) between entities instead of binary values.

As the criterion function, cut() is designed to calculate both the cohesiveness and the coupling, which measures the distance of two clusters by considering the aggregate inter-connectivity of them. Let g_i, g_j be two clusters and W be the structural proximity matrix between entities, the cut() between g_i and g_j is defined as follows:

$$\text{cut}(g_i, g_j) = \frac{\sum_{d_i \in g_i} \sum_{d_j \in g_j} W(d_i, d_j)}{|g_i| \cdot |g_j|}, \tag{3}$$

when g_i and g_j are the same, it computes the cohesiveness of the cluster, i.e., $\text{cohesive}(g_i) = \text{cut}(g_i, g_i)$; when g_i and g_j are different, it computes the coupling between them, i.e., $\text{coupling}(g_i, g_j) = cut(g_i, g_j)$.

Similar *cut*() functions are often introduced in spectral clustering [11,32,53] and information retrieval (e.g., cosine similarity [47]). In our experiment, we also evaluate two popular variations $|g_i| + |g_j|$ and $\log(|g_i| \cdot |g_j|)$ instead of the

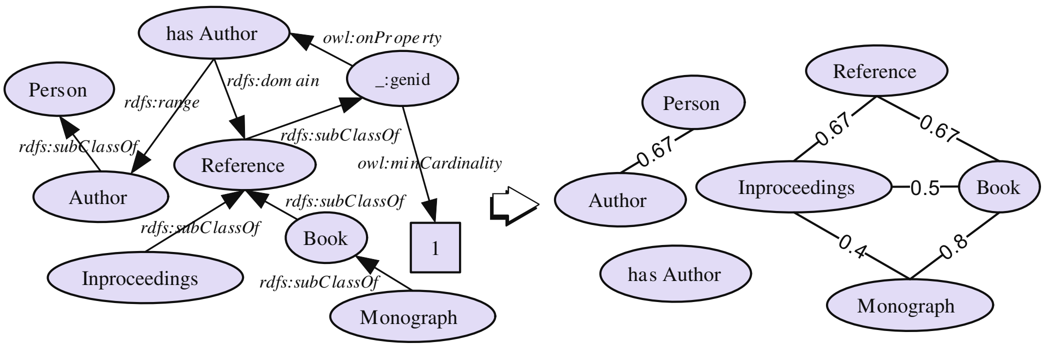


Fig. 3. An example about computing structural proximities.

denominator in Formula (3). The experimental results show that our choice is better than the other two denominators, because they sometimes make the sizes of clusters heavily imbalanced (i.e., the skewed partitioning). In addition, using a uniform function to calculate both cohesiveness and coupling is also proposed in [51], called silhouette coefficient.

Our partitioning algorithm is illustrated in Fig. 4, which follows the popular framework of agglomerative partitioning [23]. It accepts as input a set of entities to be clustered. Initially, it constructs a cluster for each entity. The cohesiveness of an entity is its depth in the ontology hierarchy (Line 5). As a result, the more specific entities can be selected earlier for merging. During an iteration, it selects the cluster g_s having the maximum cohesiveness (Line 9) and searches for the cluster g_t that has the maximum coupling with g_s (Line 10). After merging g_s and g_t to create a new cluster g_p (Line 16), it updates the cohesiveness of g_p as well as its coupling with other ones (Lines 17–20). If g_s is fully isolated (Line 13), we set its cohesiveness to zero (a trick in implementation), which implies that selecting it to merge further provides no benefit. The algorithm terminates (Line 11) if the maximum number of the entities in any cluster exceeds ϵ or there is no cluster whose cohesiveness is larger than zero (indicating that it is unnecessary to continue merging, since every cluster is completely separated). In practice, ϵ is determined based on the memory requirements of the following matchers such as GMO [27].

One may think that it is costly to update the cohesiveness and coupling when a new cluster is created. In fact, it is not necessary to re-sum the structural proximities between the entities in the new cluster. For instance, let g_p be the new cluster merged from g_s and g_t, the sum of the structural proximities in g_p can be computed as follows:

$$\sum_{d_i \in g_p} \sum_{d_j \in g_p} W(d_i, d_j) = \sum_{d_i \in g_s} \sum_{d_j \in g_s} W(d_i, d_j) + \sum_{d_i \in g_t} \sum_{d_j \in g_t} W(d_i, d_j) + \sum_{d_i \in g_s} \sum_{d_j \in g_t} W(d_i, d_j). \tag{4}$$

Recall Formula (3), the sum of the structural proximities in g_s (or g_t) could be generated by multiplying the cohesiveness of g_s (or g_t) to $(|g_s| \cdot |g_s|)$ (or $(|g_t| \cdot |g_t|)$). Also, the sum of the structural proximities between g_s and g_t can be got by multiplying the coupling between g_s and g_t to $(|g_s| \cdot |g_t|)$. In the past steps, the cohesiveness and coupling have already been computed. In our algorithm shown in Fig. 4, we use $\dotplus$ to represent this optimization (Line 17 and Line 19).

Algorithm: partition(D, W, ϵ).

Input: a set of entities D, a matrix W indicating the structural proximities between the entities in D and a parameter ϵ limiting the maximum number of the entities in each cluster $(\epsilon < |D|)$.

Output: a set of clusters G.

```
01  for each entity $d_i \in D$  // Initialization
02      $g_i = \text{create}(d_i)$;
03      $G = G \bigcup \{g_i\}$;
04  for each cluster $g_i \in G$
05      $\text{cohesive}(g_i) = depth(d_i)$;  // $depth(root) = 1$
06      for each cluster $g_j \in G$ satisfying $j \neq i$
07          $\text{coupling}(g_i, g_j) = W(d_i, d_j)$;
08  while(true)  // Partitioning
09      $g_s = \arg\max(\text{cohesive}(g_i))$;  // $g_i \in G$
10      $g_t = \arg\max(\text{coupling}(g_s, g_j))$;  // $g_j \in G,\ g_j \neq g_s$
11      if $|g_s| + |g_t| > \epsilon$ or $\text{cohesive}(g_s) == 0$  // Termination condition
12          return $G$;
13      else if $\text{coupling}(g_s, g_t) == 0$  // Isolated cluster
14          $\text{cohesive}(g_s) = 0$;
15      else  // Merging
16          $g_p = g_s \bigcup g_t$;
17          $\text{cohesive}(g_p) = \text{cohesive}(g_s) \dotplus \text{cohesive}(g_t) \dotplus \text{coupling}(g_s, g_t)$;
18          for each cluster $g_i \in G$ satisfying $i \neq p, s, t$
19              $\text{coupling}(g_p, g_i) = \text{coupling}(g_s, g_i) \dotplus \text{coupling}(g_t, g_i)$;
20              $\text{coupling}(g_i, g_p) = \text{coupling}(g_p, g_i)$;
21          $G = G \bigcup \{g_p\} \setminus \{g_s, g_t\}$;
```

Fig. 4. Partitioning algorithm.

Compared to some other data clustering approaches [23], our algorithm has two strengths: (1) it is efficient in our context of large ontology matching. The time complexity of our algorithm is $O(n^2)$, where n is the number of entities ($n = |D|$). The maximum times of iterations for partitioning is n. In each iteration, the most time-consuming step is updating the coupling of g_p with others (Line 18). It takes at most k times, where k is the number of clusters in that iteration ($k \leq n$). Assuming that we do not sort clusters by the cohesiveness (or coupling), thus selecting g_s (or g_t) spends no more than k times. Totally, the time complexity is $O(n^2)$. In general, the time complexity of agglomerative partitioning is at least $O(n^2)$ [23], hence our algorithm has already achieved that lower bound. The time complexity of spectral clustering [11,32,53] (a kind of divisive partitioning) is even higher due to the high computational cost of singular value decomposition ($O(n^3)$); and (2) it is a hierarchical algorithm, which can form a dendrogram (a typical type of tree structure) consisting of layers of clusters at different levels of granularity by saving the merging history. Thus, we can flexibly extract clusters with different scale according to running environment. However, partitional clustering approaches such as K-MEANS [38] require to specify the number of clusters in advance, which is difficult to decide in our scenario. Please note that extensive evaluation on clustering algorithms is out of the scope of this paper.

Let us consider Fig. 5. Based on the structural proximities depicted in Fig. 3, by running the partitioning algorithm illustrated in Fig. 4 (**Phase 1.2**), the seven entities in Fig. 5 are partitioned into three different clusters g_1, g_2, g_3. Every cluster is just a set of entities. From the figure, we find that the partitioning result is reasonable: the four entities about publication are grouped into g_1, while classes and properties are separated into disjoint clusters.

4.3. Constructing blocks

In the previous subsection, all the entities in an ontology are partitioned into a set of disjoint clusters. However, these clusters cannot be directly used for existing ontology matching approaches, because the RDF triples connecting these entities are missing. Hence, we need to recover the triples to the clusters, i.e., constructing blocks from the clusters (**Phase 1.3** in Fig. 1).

A very simple approach is by assigning each RDF triple to a cluster in which at least two entities are contained. For instance, let us see the cluster $g_1 = \{\texttt{Reference}, \texttt{Book}, \texttt{Inproceedings}, \texttt{Monograph}\}$ in Fig. 2, three triples

t_4: $\langle \texttt{Inproceedings}, \textit{rdfs}:\textit{subClassOf}, \texttt{Reference}\rangle$.
t_5: $\langle \texttt{Book}, \textit{rdfs}:\textit{subClassOf}, \texttt{Reference}\rangle$.
t_6: $\langle \texttt{Monograph}, \textit{rdfs}:\textit{subClassOf}, \texttt{Book}\rangle$.

should be assigned to g_1. But, we would immediately recognize that an important restriction structure is omitted: a `Reference` at least `has` one `Author`. In fact, this restriction is useful for many ontology matching approaches [16].

Missing this restriction is caused by leaving out blank nodes. As introduced in Section 2, a blank node is a node that has no intrinsic name [35]. Adopting RDF Sentences [64] rather than RDF triples can preserve more semantic information (e.g., the restriction structure) when constructing blocks from the clusters, even the nested list structure. The entities between different blocks are no longer disjoint, which may lead to additional computational cost. Since each RDF triple belongs to one and only one RDF Sentence [60], such duplicate part at most equals to the number of RDF triples (every cluster includes only an entity). However, the number of overlapped entities in practice is small (less than 3% in our experience), because the number of blocks is much less than the number of entities.

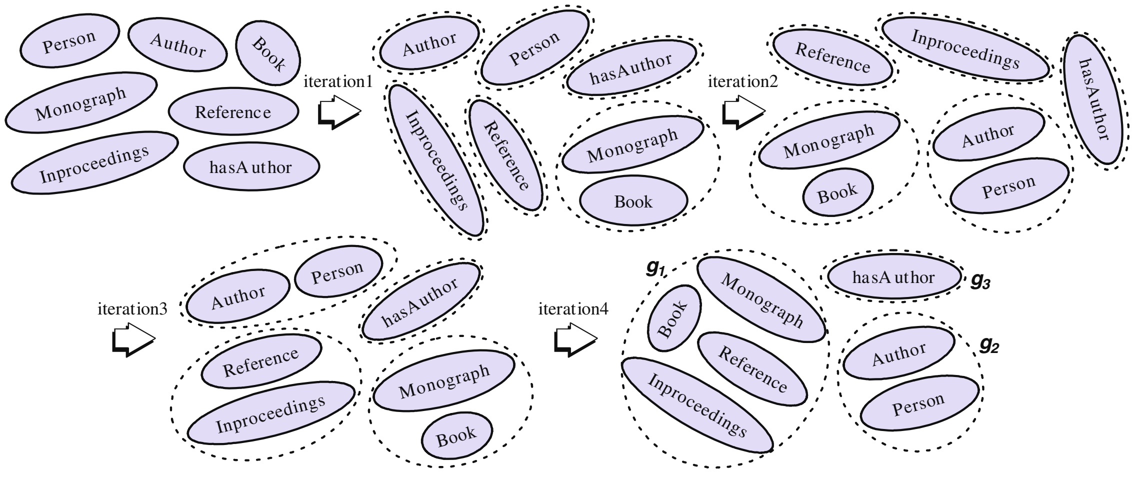


Fig. 5. An example about partitioning.

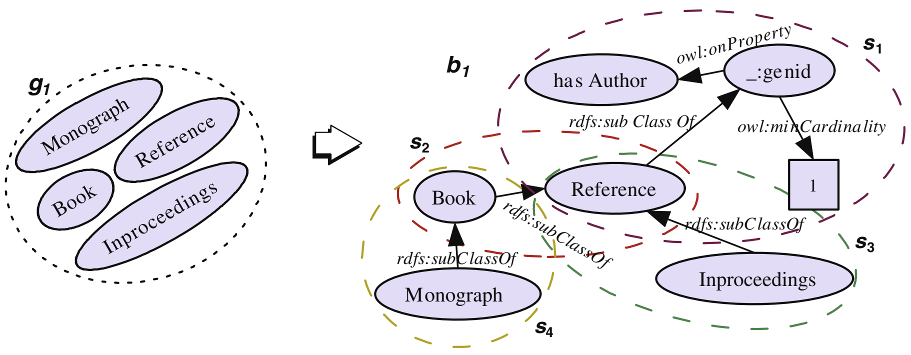


Fig. 6. An example about constructing a block from a cluster.

Let us see the example shown in Fig. 6. The block b_1 in the right part of the figure includes four RDF Sentences s_1, s_2, s_3, s_4 (encircled by dashed lines). s_1 is composed of three RDF triples, and its subject is `Reference`. If we break down the context of `_:genid`, the RDF triple $\langle$`_:genid`, *owl : minCardinality*, `1`''$\rangle$ would never be assigned. Hence, the restriction structure about a `Reference` at least `has` one `Author` is destroyed. But, by using RDF Sentences, such RDF triple can be well preserved. Fig. 6 also illustrates the process of constructing a block from a cluster (**Phase 1.3**): b_1 is the block generated from the cluster g_1 (in the left part of the figure), since the subjects of s_1, s_2, s_3, s_4 are all included in g_1 (recall Definition 4 in Section 2).

To the best of our knowledge, there is no distance-based clustering technique for directly grouping RDF triples (or RDF Sentences), because: (1) the number of RDF triples in an ontology is much larger than the number of entities. For example, `GALEN` has about 60,000 RDF triples, which is six times larger than the number of its entities; and (2) the distance between two RDF triples is difficult to measure, since an RDF triple has not only three nodes but also an order. Computing the distances between RDF Sentences is even harder.

5. Matching blocks

In the previous section, two large ontologies to be matched have been partitioned into two groups of small blocks. A brute force solution for discovering alignments is by directly matching every pair of blocks in the two ontologies. But, it is not necessary to do so, because it is likely that large portions of one or both of them have no matching counterparts [13].

In this section, we will present a heuristic way of finding matched block pairs, i.e., block mappings. Therefore, only block mappings are further inputted to V-Doc [46] and Gmo [27] for alignment discovery. More specifically, a light-weight string comparison technique, I-Sub [55], is firstly employed to exploit anchors between two full ontologies (**Phase 2.1** in Fig. 1), and then the blocks from the two ontologies are matched based on the distribution of the anchors (**Phase 2.2**).

5.1. Finding anchors

We refer to the alignments with high similarities as *anchors*. In our approach, anchors are unnecessarily to be numerous or completely correct, so they can be found by some simple and fast approaches. We employ the following steps to gain anchors between large ontologies.

(1) Use I-Sub [55] to automatically generate a set of candidate anchors.
(2) Manually remove incorrect ones from the candidate anchors (optional).
(3) Manually add some omissions to the candidate anchors (optional).

Please note that the tradeoff between the correctness of anchors and the effort of human interaction should be considered. With the help of some visualization tools such as [1,17], the productivity can be significantly improved. However, manually revising anchors for large ontologies is still time-consuming. In this paper, we do not involve any human interaction during anchor generation.

We briefly introduce I-Sub here. Entities in ontologies contain human readable descriptions in their local names, *rdfs:labels*, *rdfs:comments* and so on. I-Sub improves the traditional edit-distance [37] based string comparison approaches by considering not only the commonalities between the descriptions of entities but also their differences, hence it is more robust to small diverges from the optimal cutoff taking place. Let p, q be two strings, the similarity between p and q is defined as follows:

$$\mathrm{sim}(p,q) = \mathrm{comm}(p,q) - \mathrm{diff}(p,q) + \mathrm{winkler}(p,q), \tag{5}$$

where $\mathrm{comm}(p,q)$ stands for the commonality between p and q, $\mathrm{diff}(p,q)$ for the difference and $\mathrm{winkler}(p,q)$ for the improvement of the result by using a correction coefficient introduced in [62].

If the similarity between two descriptions is greater than a pre-defined value μ ($\mu \in [0,1)$), then I-Sub would consider the two entities containing those two descriptions as a candidate anchor. In [55], it suggests $\mu \geqslant 0.75$.

Although computing anchors between two whole ontologies takes quite a lot of time in our approach (nearly half of the run time), it compensates for the possibility of missing alignments due to only finding alignments within block mappings. (The anchors generated by I-Sub are re-used as a portion of alignments.) According to our experiment, 10–20% alignments are only found by I-Sub, which demonstrates the effectiveness of calculating anchors between two full ontologies. Besides, based on the OAEI 2007 report [15], at least 50% alignments can be directly found by simple string comparison techniques, so it also provides evidence to support the use of I-Sub. In addition, as compared to other matching techniques [16], I-Sub is very efficient and scalable.

5.2. Generating block mappings

Similarities between blocks can be computed based on the distribution of the anchors gained above. The background idea is that the more anchors can be found between two blocks, the more similar the two blocks are.

Let B, B' be two sets of blocks from two ontologies O, O' and Q be a set of anchors found between O and O'. $anchor(b, b', Q)$ returns the number of the anchors in Q between two blocks b and b' ($b \in B$, $b' \in B'$). The similarity between b and b' is defined as follows:

$$\mathrm{sim}(b,b') = \frac{2 \cdot \mathrm{anchor}(b,b',Q)}{\sum_{b_k \in B}\mathrm{anchor}(b_k,b',Q) + \sum_{b'_k \in B'}\mathrm{anchor}(b,b'_k,Q)}. \tag{6}$$

By setting a cutoff η ($\eta \in [0,1)$), any two blocks whose similarity is greater than η is selected to make up a block mapping. Our experiment shows that η is stable with various values, because the number of correct anchors is considerably larger than the number of wrong ones, i.e., a few wrong anchors cannot influence the overall distribution. Please note that we permit a block to appear in more than one block mappings because of the difference in large ontology modeling. However, we would see that, in our experiment, the number of block mappings is much less than $(|B| \cdot |B'|)$, so the computational cost of further alignment discovery would be largely decreased.

Let us see the example in Fig. 2. Some anchors like $\langle 3, \texttt{Book}, \texttt{Book}, =, 1.0\rangle$ and $\langle 4, \texttt{hasAuthor}, \texttt{hasAuthor}, =, 1.0\rangle$ might be easily discovered, because the entities in each anchor have the same local names. Next, some block mappings like $\langle 1, b_1, b'_1, 1.0\rangle$ can be detected.

6. Discovering alignments

Each block in a block mapping is just a small sub-ontology. Now, we can use various existing ontology matching techniques to discover alignments between the blocks in each block mapping. In this paper, we specifically integrate two powerful matchers, V-Doc [46] and Gmo [27], to support alignment discovery, because they are representatives of two kinds of popular ontology matching technology, i.e., the linguistic matching and the structural matching. Our previous experiment showed that they perform better than other similar approaches such as [37,41]. However, we should keep in mind that some other methods [16,19] can also be freely chosen as alternatives. Due to the technical details of V-Doc and Gmo have been published in [46,27], in this section we only generally summarize their distinguishing features.

6.1. Linguistic matching

V-Doc adopts a linguistic approach to ontology matching, whose novelty lies in the construction of virtual documents. Basically, as a collection of weighted words, the virtual document of an entity in an ontology contains not only the local descriptions (e.g., *rdfs:label*), but also some neighboring information to reflect the intended meaning of the entity. Document similarities can be calculated by traditional vector space techniques (e.g., TF/IDF [47,57]) and further be used in similarity-based approaches to ontology matching. Technically, the RDF graph structure is exploited to extract the description information from three sorts of neighboring entities, subject neighbors, predicate neighbors and object neighbors, based on their positions in RDF triples.

6.2. Structural matching

Gmo is an iterative structural matcher, which uses RDF Bipartite Graphs [24] to represent ontologies (there exist two kinds of vertices in an RDF Bipartite Graph: entity vertices and triple vertices) and computes structural similarities between entities by recursively propagating their similarities in the bipartite graphs. The basic idea is as follows. The similarity of two entities from two ontologies comes from the accumulation of similarities from the involved RDF triples taking the two entities as the same role (subject, predicate or object) in the triples, while the similarity of two triples comes from the accumulation of similarities from the involved entities as the same role in these two triples being compared. Gmo takes a set of external alignments as input, which are typically found beforehand by other matchers (in our current implementation, the external alignments come from V-Doc) and incrementally generates new additional alignments as output. The performance of Gmo improves with the accuracy of external alignments increases.

6.3. Similarity combination strategy

Similarity combination is an important and difficult issue in building ontology matching systems. We propose a heuristic strategy to tune the thresholds (or cutoffs) of our matchers gradually based on the measures of linguistic comparability and structural comparability, which makes the combination robust in a variety of ontology matching scenarios.

Our combination strategy follows the two-step method [4,18], which is a sequential composition of matchers [16,54]. (C oma [13] and RiMOM [58] use parallel compositions.) First, we calculate the linguistic comparability by measuring the proportion of candidate alignments against the minimum number of entities in two ontologies. The intuition is as follows. If the number of alignments is close to the number of entities in the smallest ontology, we are nearly done with matching, and it is unnecessary to execute Gmo any more. Then, we determine the threshold of V-Doc based on the linguistic comparability in order to select the strong (i.e., more reliable) alignments.

If the alignments are not enough (e.g., for low overlapped ontologies, the number of alignments might be considerably smaller than the number of entities in the smallest ontology), then we start the second step. The structural comparability is measured by the cosine similarity, which compares which built-in properties are used between two ontologies, and how often. It is used to determine the threshold of Gmo to obtain the weak (i.e., less reliable) alignments.

We divide these two kinds of comparability into three levels (i.e., low, medium and high) to automatically determine the similarity combination strategy. For example, if the linguistic comparability is high, our approach would lower the threshold of V-Doc, then more alignments from V-Doc can be combined to the final alignments. For another example, if the linguistic comparability and the structural comparability are both low, our approach would combine less alignments from V-Doc and Gmo to the final result but make the alignments from V-Doc more than the ones from Gmo, because the alignments from V-D oc are assumed to be more reliable than the ones from Gmo.

We adopt a greedy alignment selection algorithm [25] to select the final alignments. Here, the anchors generated by I-Sub are also considered. More specifically, we choose alignments with multiple iterations. In an iteration, the candidate alignment with the highest similarity is selected, and all its conflicting alignments are removed. (We only permit each entity to appear in one alignment. Therefore, if an alignment that contains the entity d is chosen, then all the other candidates that include d would be deleted.) The process terminates until no candidate is left.

Please consider the example in Fig. 2 again. Although by using I-Sub, we can find three alignments (i.e., anchors), each one contains the entities having the same names. The alignment between `Reference` and `Entry` and the alignment between `Inproceedings` and `ConferencePaper` cannot be directly found, because they do not have similar names. However, a human may still recognize that they should be alignments because of the similar contexts of the ontologies. V-Doc can utilize neighboring information, hence, regarding `Reference` for instance, it imports the names of `hasAuthor`, `Book` and `Inproceedings`. By using the neighboring information, V-Doc detects `Reference` and `Entry` should be an alignment. Next, Gmo finds that the two ontologies has similar graph structures, hence it propagates similarities in the two graphs. As a result, the alignment between `Inproceedings` and `ConferencePaper` can be discovered.

Supposing that V-Doc can also find the three alignments containing the entities with the same names, the number of alignments found by V-Doc is 4, and the minimum number of the entities in O, O' is 5. Hence, the linguistic comparability of O and O' is 0.8. The two ontologies O, O' share five built-in properties: *rdfs:subClassOf*, *rdfs:domain*, *rdfs:range*, *owl:minCardinality* and *owl:onProperty*. As an example, the number of *rdfs:subClassOf* in O, O' is 5 and 3, respectively. The number of the other built-in properties is 1, respectively. The cosine similarity between those built-in properties is $19/\sqrt{29 \cdot 13} = 0.98$, where $19 = 5 \cdot 3 + 1 + 1 + 1 + 1, 29 = 5^2 + 1 + 1 + 1 + 1, 13 = 3^2 + 1 + 1 + 1 + 1$. So, the structural comparability is 0.98. Since both the linguistic comparability and the structural comparability are in the high level, the five alignments are all combined as the final result for output.

7. Evaluation

We have implemented the proposed approach in Java, called Pbm Partition-based *B*lock *M*atching) and integrated it into our ontology matching system Falcon-AO [29], which participated in the last OAEI 2007 campaign [15] and was recognized by the organizers as one of the best matching systems. In this section, we will report the results of an experimental study on synthetic and real world data sets. The test cases and experimental results are all available at our web site.[6] All the tests are carried out on an Intel Core 2 Duo 2.13 GHz desktop machine with 2 GB DDR2 memory under Windows XP Professional operating system (SP2) and Java 1.6 compiler. Some parts of evaluation results regarding the real world data sets are quoted from [15].

[6] http://iws.seu.edu.cn/projects/matching/res/pbm_new.zip.

7.1. Synthetic test

We will measure the performance of PBM on partitioning ontologies as well as on finding block mappings.

7.1.1. Data sets

In our evaluation, we choose two pairs of ontologies: `Russia12` and `TourismAB`, which can be downloaded from the web site.[7] The reasons for selecting them as the test cases are: (1) they come from real world domains and widely used in the field of ontology matching, (2) their sizes are moderate. If the sizes of ontologies are too small, it is unnecessary to partition them into blocks; while if the sizes are too large, they are not appropriate for human observation; and (3) they have opened reference files that contain alignments between entities, which are useful for evaluating block mappings. Short descriptions of the two pairs of ontologies are given below.

- `Russia12`. The two ontologies are created independently by different people from the content of two travel web sites about Russia. `Russia1` contains 151 classes and 76 properties. `Russia2` contains 162 classes and 81 properties. The reference alignment file contains 85 alignments.
- `TourismAB`. The two ontologies are created separately by different communities describing the tourism domain of Mecklenburg–Vorpommern (a federal state in the northeast of Germany). `TourismA` contains 340 classes and 97 properties. `TourismB` contains 474 classes and 100 properties. The reference alignment file contains 226 alignments.

7.1.2. Experimental methodology and evaluation metrics

The ideal block mappings should provide both good partitioning quality and good mapping quality. The partitioning quality reflects the effectiveness of the partitioning algorithm, while the mapping quality indicates the possibility of discovering alignments further. In order to measure these two kinds of quality, three experiments are designed for evaluation.

In the first experiment, three volunteers are trained to set up blocks as gold standards manually for evaluating the partitioning quality of ontologies. However, constructing reference blocks is not an easy job, everyone has his/her own opinion on partitioning, so the reference blocks built here are only generally agreed by all the volunteers. We assess the partitioning quality by comparing to the reference ones. In the experiment, the number of the reference blocks is as follows: 22 ones for `Russia1`, 22 ones for `Russia2`, 18 ones for `TourismA` and 23 ones for `TourismB`. In order to make the number of the blocks from any ontology constructed by PBM the same as the number of the reference ones, we temporarily change the termination condition of our partitioning algorithm (in the real world test, we strictly obey the termination condition in Fig. 4). Technically, we set ϵ large enough ($\epsilon = 1000$). During partitioning, once the number of the computed blocks equals to the number of the reference ones (it can be satisfied in the test), the partitioning algorithm stops immediately. This kind of approach is widely adopted in data clustering [23].

We use a well-known metric *entropy* to compare the automatically generated blocks with the manually established references. Before introducing the metric, we firstly describe a basic operation (*prec*), which measures the precision of a computed block with respect to a reference one. Let B be a set of computed blocks ($|B| = n$) and R be a set of manual ones ($|R| = m$). b_i denotes a block in B, while r_j denotes a block in R. $|b_i|$ returns the number of entities in b_i, and $|r_j|$ is defined analogously. $b_i \bigcap r_j$ calculates the common entities in both b_i and r_j. The prec of a computed block b_i referring to r_j is defined as follows:

$$\mathrm{prec}(b_i, r_j) = \frac{|b_i \bigcap r_j|}{|b_i|}. \tag{7}$$

The *entropy* measures the distribution (or disorder) of entities in blocks and reflects the overall partitioning quality. The *entropy* of each block indicates its precision dispersal over the global partitioning. A smaller score indicates a better partitioning quality. The best possible entropy score is 0, and the worst is 1. The *entropy* of a set of computed blocks B is defined as follows:

$$\mathrm{entropy}(B) = \frac{1}{\sum_{i=1}^{n}|b_i|} \cdot \sum_{i=1}^{n} \mathrm{entropy}(b_i) \cdot |b_i|, \tag{8}$$

$$\mathrm{entropy}(b_i) = -\frac{1}{\log m} \cdot \sum_{j=1}^{m} \mathrm{prec}(b_i, r_j) \cdot \log(\mathrm{prec}(b_i, r_j)). \tag{9}$$

For the criterion function that we propose in the partitioning algorithm (see Formula (3)), there are several possible variations about the denominator. So, we also compare two other popular choices, $|g_i| + |g_j|$ and $log(|g_i| \cdot |g_j|)$, with our $cut()$ function.

In the second experiment, we evaluate the mapping quality of computed block mappings by measuring the *correctness* with different number of those block mappings. The background idea is as follows. The higher the quality of those block

[7] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/.

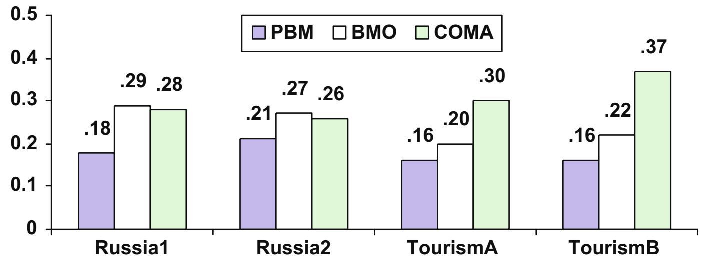


Fig. 7. *Entropy* of Pbm, Bmo and Coma.

mappings is, the more potential alignments could be found in such block mappings. Furthermore, the *correctness* also implies the upper bound if only matching the blocks in block mappings. Let O, O' be two ontologies and BM be a set of computed block mappings ($|\mathrm{BM}| = l$). bm_k denotes a block mapping in BM. $bm_k.b, bm_k.b'$ are two blocks in bm_k from O, O', respectively. Let A be a set of alignments in a reference alignment file ($|A| = n$). a_i denotes an alignment in A. $a_i.d, a_i.d'$ are two entities in a_i from O, O', respectively. The *correctness* of BM is defined as follows:

$$correctness(\mathrm{BM}) = \frac{1}{n} \cdot \sum_{k=1}^{l} |\{a_i | a_i \in A, a_i \cdot d \in bm_k.b, a_i \cdot d' \in bm_k \cdot b'\}|. \tag{10}$$

The *correctness* is influenced by two parameters: the cutoff μ of anchors and the cutoff η of block mappings. We firstly use reference alignments as anchors to determine the best η, and then test various μ to find whether Pbm is stable with a variety of cutoffs for anchors. Intuitively, the *correctness* of BM increases when the number of the block mappings increases, i.e., when the cutoff of the block mappings decreases. But, the computational cost of alignment discovery increases as well. Furthermore, an optimal parameter setting obtained in one domain is often reusable for other domains [7], so we choose the best μ and η here and apply them to the real world cases further.

We choose two other block matching approaches for the above two tests. The first one is Bmo [28], which uses a completely contrary process as compared to Pbm. The second one is based on Coma [13], which is a level-based method.[8] Please see Section 3 for more details.

In the third experiment, we make a comparison between the alignments from Pbm and the ones directly generated from Falcon-AO without partitioning. Because the sizes of the ontologies in `Russia12` and `TourismAB` are normal, we can execute I-Sub, V-Doc and Gmo on the whole ontologies. (Currently, it is impossible for us to do this experiment on large ontologies.) This metric serves as the truly cost of partitioning and block matching in Falcon-AO.

Two standard information retrieval metrics (*precision* and *recall*) are applied to measure the alignments. Let A be a set of computed alignments and A' be a set of reference alignments, the *precision* and *recall* of A referring to A' are defined as follows:

$$precision(A, A') = \frac{|A \cap A'|}{|A|}, \quad recall(A, A') = \frac{|A \cap A'|}{|A'|}. \tag{11}$$

7.1.3. Experimental results

Firstly, the experimental results of the partitioning quality (*entropy*) of Pbm, Bmo and Coma are shown in Fig. 7. They indicate that Pbm is dominant in all the four test cases (i.e., the blocks constructed by Pbm are most consistent to human understanding), and Bmo performs better than Coma.

The *entropy* of Pbm and two other variations to the denominator in Formula (3) is shown in Fig. 8. It indicates that our criterion function is slightly better than the other two denominators. Actually, we find that, although the other two functions may perform well on fully-separated clusters, in case of outliers or clusters including lots of entities that are neighbors, a large cluster tends to swallow all the other clusters and thus, the entities from all the other clusters would be merged into the large one. This is because a larger cluster typically might have a larger number of cross links with other clusters.

Secondly, we evaluate the mapping quality (*correctness*) of Pbm as compared to Bmo. Before making a comparison, we need to determine two key parameters: the cutoff μ of anchors and the cutoff η of block mappings. By using the reference alignments as anchors, with the variation of η, the *correctness* of Pbm is shown in Fig. 9, and the number of block mappings is shown in Fig. 10. They indicate that, in the two test cases, when η increases, the *correctness* of the block mappings decreases, and the number of block mappings decreases as well. We can also see that, in most values of η, the *correctness* is fine. For instance, in `TourismAB`, when $\eta = 0.225$, the *correctness* is still larger than 0.70. It demonstrates that the *correctness* does

[8] Because Coma does not grant users to adjust the desired number of blocks, we have to implement the partitioning algorithm by ourselves according to the paper. However, it is still not clear about how it matches blocks, so we cannot compare its mapping quality with our approach.

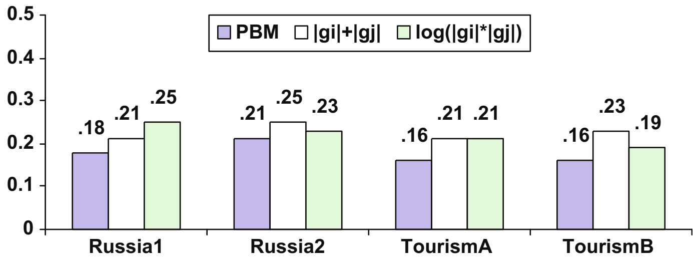


Fig. 8. *Entropy* of PBM and two other denominators.

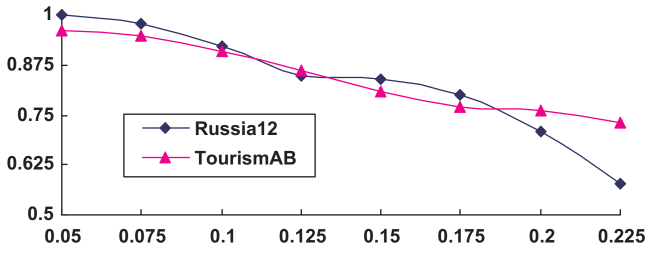


Fig. 9. *Correctness* of PBM with the variation of η.

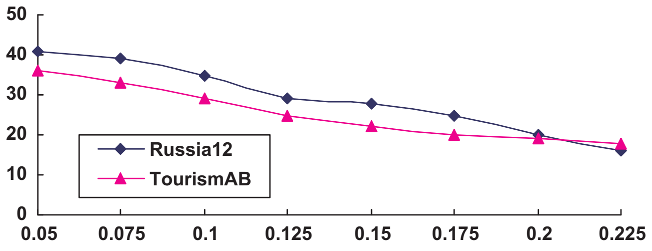


Fig. 10. Number of block mappings with the variation of η.

not decrease drastically as η increases. In the following experiments, we choose $\eta = 0.075$ as the cutoff of block mappings, because the *correctness* is fairly large, while the number of block mappings is fairly small.

Then, we test a number of μ to see whether PBM is sensitive to various cutoffs of anchors. With the variation of μ and $\eta = 0.05, 0.075, 0.1$, the *correctness* of PBM is shown in Fig. 11, and the number of block mappings is depicted in Fig. 12. We can observe that, with μ increases, i.e., the number of anchors decreases, the *correctness* of the block mappings keeps high. It implies that PBM is stable with a pretty good accuracy. In the following experiments, we set $\mu = 0.85$ as the cutoff of anchors, because a higher cutoff usually yields to more accurate alignments. Besides, Fig. 11 and Fig. 12 also convince that PBM is robust with slight changes on η (e.g., $\eta = 0.05, 0.1$).

By adopting $\mu = 0.85, \eta = 0.075$, we compare the *correctness* of PBM with that of BMO. The comparison results are exhibited in Fig. 13. It shows that the mapping quality of the two approaches is good, and BMO is a little better than PBM. Even though

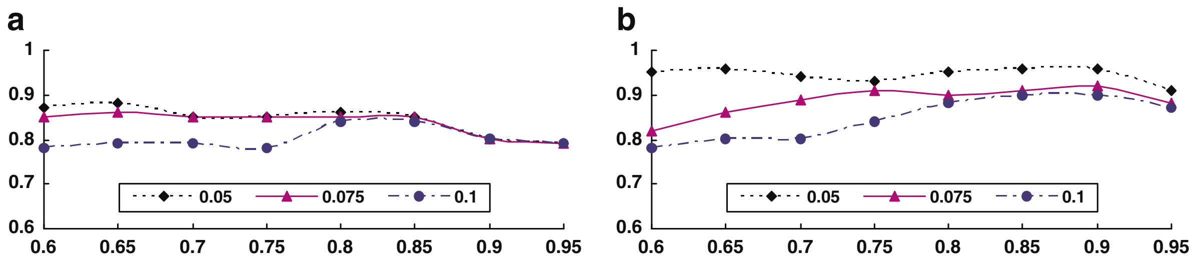


Fig. 11. *Correctness* of PBM with the variation of μ ($\eta = 0.05, 0.075, 0.1$) (a) Russian12, and (b) tourismAB.

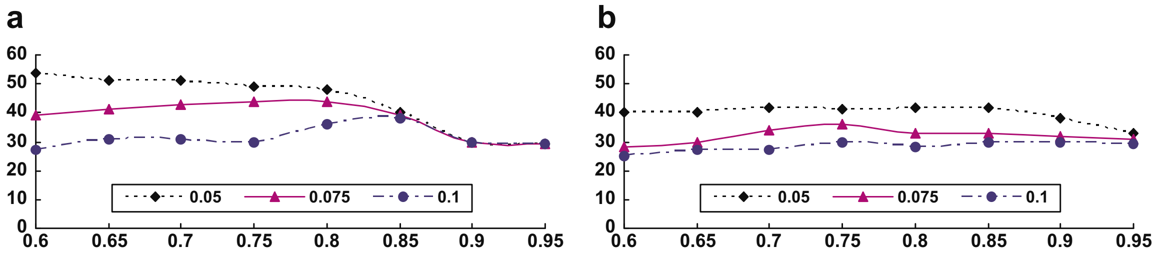


Fig. 12. Number of block mappings with the variation of μ ($\eta = 0.05, 0.075, 0.1$): (a) Russian12, and (b) tourismAB.

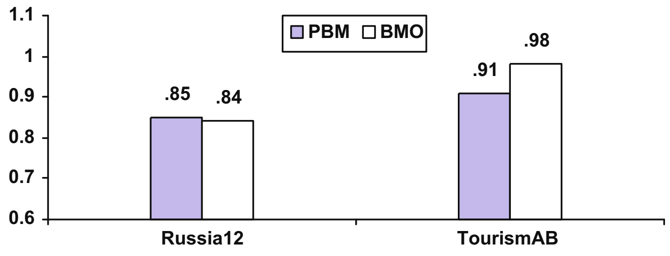


Fig. 13. *Correctness* of Pbm and Bmo.

the two parameters are not the best ones (e.g., $\mu = 0.8, 0.9$ and $\eta = 0.05, 0.1$), the correctness of Pbm is still comparable (see Fig. 11).

More specifically, Pbm exploits 39 block mappings for `Russia12` and 33 ones for `TourismAB`, while Bmo constructs 25 block mappings for `Russia12` and 26 ones for `TourismAB`. Therefore, the potential computational cost of Pbm on discovering alignments is higher than that of Bmo. However, the number of block mappings is still much smaller than the product of the number of two sets of blocks ($22 \cdot 22 = 484$ for `Russia12` and $18 \cdot 23 = 414$ for `TourismAB`). It demonstrates that each block only corresponds to a small number of other ones. Besides, Fig. 13 also implies that, if we only find alignments within block mappings, 15% alignments for `Russia12` and 9% ones for `TourismAB` would never be found by Pbm.

If we go further to observe the run time spent by Pbm and Bmo during partitioning ontologies and matching blocks, we would immediately find that Pbm is more efficient than Bmo. Based on the current implementation, Pbm takes only 5 seconds and 8 seconds to complete the `Russia12` and `TourismAB` cases respectively (including the parsing time), while Bmo spends nearly 15.5 min and 30 min. More importantly, Bmo would suffer from the scalability problem, because it matches two full ontologies firstly. Therefore, Bmo is not feasible for large ontologies.

Thirdly, we directly apply Falcon-AO without partitioning to the whole ontologies in `Russia12` and `TourismAB` and compare the results with the alignments found by Pbm. The experimental results are depicted in Fig. 14. Basically, some alignments would be lost because of partitioning. For example, in `Russia12`, Falcon-AO without partitioning outperforms 6% in *recall*, which means that 6% alignments are not found by Pbm. But, in `TourismAB`, we interestingly find that Pbm discovers more alignments than Falcon-AO without partitioning does. The reason for this situation is that more alignments found by Gmo are combined into the final alignments, since the structures of blocks tend to be more similar after partitioning. But, Gmo also involves some wrong alignments, thus the *precision* of Pbm is always worse than that of Falcon-AO without partitioning. However, the experimental results demonstrate that Pbm does not lose too much *precision* or *recall*.

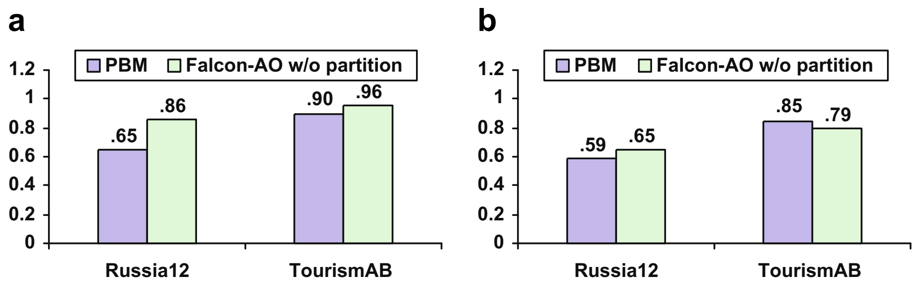


Fig. 14. *Precision* and *recall* of Pbm and Falcon-AO without partitioning: (a) precision, and (b) recall.

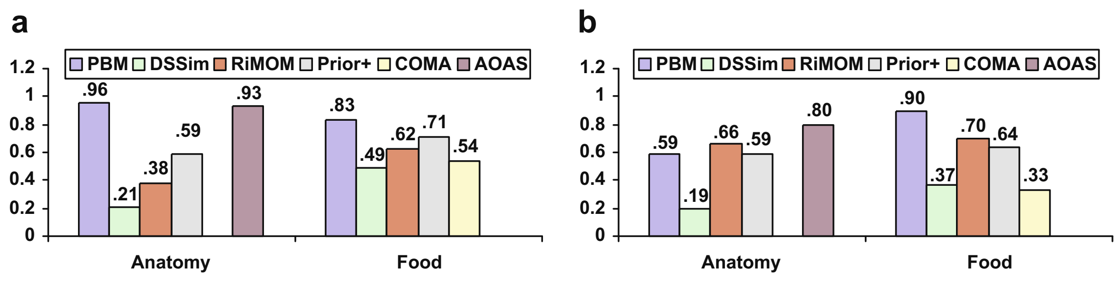


Fig. 15. *Precision* and *recall* of PBM, DSSIM, RIMOM, PRIOR+, COMA and AOAS: (a) precision, and (b) recall.

Based on the experimental results above, we conclude that PBM is efficient to achieve both good partitioning quality and good mapping quality.

7.2. Real world test

We will report the results of PBM on discovering alignments in two large real world ontology matching tasks: `Anatomy` and `Food`. It is also worthy of noting that our approach has been used and tested in several real-life applications, such as book integration [61] and query rewriting for art thesauri [26].

7.2.1. Data sets

In OAEI 2007, the organizers provided three large ontology matching tracks: `Anatomy`, `Food` and `Library`.[9] All of them are blind tests, i.e., participants do not know the references beforehand, and organizers help evaluate the results. Since the ontologies in the `Library` task are written in Dutch, most ontology matching tools cannot handle this language and do not present any results on this test, we ignore it here. Short descriptions of the `Anatomy` and `Food` tasks are given below.

- `Anatomy`. The ontologies of the `Anatomy` track are the NCI Thesaurus describing the human anatomy (denoted by `Human`), published by the National Cancer Institute (NCI), and the Adult Mouse Anatomical Dictionary (denoted by `Mouse`), which is developed as part of the Mouse Gene Expression Database project. `Human` contains 3304 entities, while `Mouse` contains 2743 entities. The two ontologies are represented as complex graphs.
- `Food`. The test case is a taxonomy task in which the hierarchies come from thesauri, and it has a lot of multi-lingual texts. It includes two ontologies: `AGROVOC`, which is developed by Agriculture Organization (FAO), and `NALT`, which is developed by United Nations Food Organization. `AGROVOC` contains 28,439 entities, while `NALT` contains 42,326 entities. The two ontologies are represented as simple tree structures.

7.2.2. Experimental methodology and evaluation metrics

We compare the alignments generated by PBM to the ones found by the other three ontology matching tools, DSSIM [43], RIMOM [58] and PRIOR+ [39], which also participated in OAEI 2007. Besides, COMA [13] attended the `Food` task in OAEI 2006, which is exactly the same as the one in OAEI 2007, and AOAS [63] is a domain-specific tool, which typically aims at matching life-science ontologies and achieved the best performance in the `Anatomy` track in OAEI 2007. Hence, we also compare COMA and AOAS with PBM. For more details, please see Section 3.

In OAEI 2007, the organizers also provided several matching tasks with small ontologies, such as `Benchmarks` and `Directory`, and many ontology matching tools participated in these tracks. Although our tool, FALCON-AO, is one of the best matching tools, there exist several tools that performed slightly better than ours in these tracks. For instance, OLA2 [33] and PRIOR+ outperformed FALCON-AO in the `Directory` track, which involves about 4500 cases, each containing two small ontologies. Because we focus on making a comparison on matching large ontologies rather than matching a large amount of ontologies, we do not present the results of these tracks in this paper. For more details, we refer the reader to the OAEI 2007 report [15].

The performance of PBM, DSSIM, RIMOM, PRIOR+, COMA and AOAS are measured based on *precision* and *recall*. Also, the execution time is evaluated.

In this experiment, the parameters of PBM are configured as follows: (1) $\epsilon = 500$, i.e., the number of entities in any block is less than 500. This value is set based on the memory requirements of V-DOC and GMO; and (2) $\mu = 0.85$ and $\eta = 0.075$, which are gained from the synthetic test.

[9] http://oaei.ontologymatching.org/2007/.

Table 1
Execution time of PBM, DSSIM, RIMOM, PRIOR+ and AOAS

	PBM	DSSIM	RIMOM	PRIOR+	AOAS
`Anatomy`	12 min	4 h	75 min	23 min	2 h
`Food`	6 h	1 week	4 h	1.5 h	–

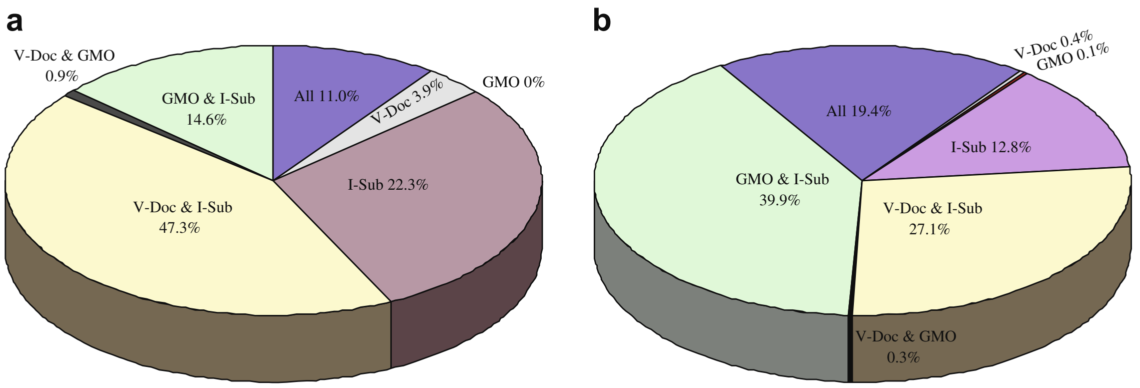


Fig. 16. Percentages of the alignments found by I-SUB, V-DOC and GMO.

7.2.3. *Experimental results*

The comparison results of the *precision* and *recall* of PBM, DSSIM, RIMOM, PRIOR+, COMA and AOAS are illustrated in Fig. 15. The figure indicates that AOAS performs best in the `Anatomy` track as compared to the other five tools, which demonstrates the effectiveness of using domain knowledge in ontology matching. In fact, AOAS is the best tool in the `Anatomy` track in OAEI 2007. PBM outperforms the other four generic tools, DSSIM, RIMOM, PRIOR+ and COMA, in average *precision* and *recall* and is only slightly behind AOAS, which means that our divide-and-conquer approach can solve the large ontology matching problem. It also proves that, even without using domain knowledge, PBM can generate accurate alignments and guarantee a good coverage.

The execution time of PBM, DSSIM, RIMOM, PRIOR+ and AOAS is shown in Table 1. (COMA does not provide its complete execution time yet.) It shows that PBM is the most efficient one. Please note that, in the `Food` track, RIMOM and PRIOR+ do not execute all the matching strategies, but only use the edit-distance based strategies as alternatives, and the run time of PRIOR+ excludes the preprocessing stage. DSSIM manually partitions the ontologies into several small pieces. Besides, it also shows that, because of using domain knowledge, AOAS takes more time to achieve better precision and recall.

Based upon the experimental results, we conclude that PBM achieves good precision and recall with significant reduction of run time.

It is also interesting to analyze the portions of the alignments found by each matcher (i.e., I-SUB, V-DOC and GMO) within the final alignments. We collect the alignments generated by each matcher and examine how many alignments are discovered by exactly one, two or three matchers as compared to the final alignments. The analysis is illustrated in Fig. 16. It shows that every matcher contributes quite a large number of alignments to the final results, so it proves that each matcher takes effect in matching large ontologies. Many alignments are combined from more than one matchers, so the accuracy is well guaranteed. Besides, we could also observe that more alignments come from I-SUB and V-DOC than GMO, which demonstrates that matching large ontologies from the linguistic perspective is more reliable than matching from the structural perspective, because large ontologies tend to contain well annotated linguistic descriptions. Furthermore, the portion of the alignments discovered by V-DOC and GMO reflects the additional computation due to the overlapping between blocks, while the portion of the alignments only generated by I-SUB indicates the number of alignments PBM would omit if we only find alignments within block mappings, so using I-SUB to generate anchors over two full ontologies compensates for the possibility of missing alignments due to block matching.

8. Conclusion

In summary, the main contributions of this paper are listed as follows:

- We have introduced a divide-and-conquer approach to matching large ontologies. Not only does it solve the scalability problem, but it also achieves good precision and recall with significant reduction of execution time.

- We have designed a structure-oriented partitioning algorithm to partition the entities of each RDFS or OWL ontology into a set of small clusters and use RDF Sentences as basic units to construct blocks from the clusters. It both controls the sizes of blocks and guarantees the completeness of blank nodes. Furthermore, it is practical for current ontology matching technology.
- We have proposed a heuristic strategy to block matching in terms of the distribution of anchors, which avoids matching each pair of blocks. Furthermore, the anchors can be combined into the final alignments as an important compensation for block matching.
- We have integrated two powerful matchers, V-Doc and GMO, to discover alignments in each block mapping. These two matchers discover alignments from different perspectives, and our combination strategy is flexible.
- We have implemented our prototype in Java and experimentally evaluated our approach on both synthetic and real world data sets. In the synthetic test, we define the partitioning quality and the mapping quality to measure our approach on partitioning ontologies and finding block mappings. In the real world test, the results from OAEI 2007 demonstrate that our approach can achieve good precision and recall in short time.

In the future work, we look forward to implementing a public Web interface to make the technology widely accessible and effectively interacted with users. We hope to consider certain random picking or sorting techniques to improve our approach further, especially to decrease the run time of finding anchors. We also want to apply some ontology summarization approaches to blocks so that block mappings can be discovered quickly only based on the summaries. Finally, we would like to go beyond matching large ontologies and cope with large database schemas and XML schemas in order to support data integration between different data models.

Acknowledgements

This work is funded in part by the NSFC under Grant 60573083 and also in part by the 973 Program of China under Grant 2003CB317004. We appreciate Christian Meilicke and Willem van Hage for providing the `Anatomy` and `Food` data sets, respectively. We also thank Miklos Nagy, Jie Tang and Ming Mao for useful discussion on DSSim, RiMOM and Prior+, respectively. Finally, we are very grateful to Christophe Guéret, Zhisheng Huang, Shenghui Wang and anonymous reviewers for their valuable comments.

References

[1] B. Alexe, L. Chiticariu, R.J. Miller, W. Tan, Muse: mapping understanding and design by example, in: Proceedings of the 24th International Conference on Data Engineering, 2008, pp. 10–19.
[2] D. Aumueller, H.-H. Do, S. Massmann, E. Rahm, Schema and ontology matching with COMA++, in: Proceedings of the 24th ACM International Conference on Management of Data, ACM Press, 2005, pp. 906–908.
[3] F. Bobillo, M. Delgado, J. Gómez-Romero, An ontology design pattern for representing relevance in OWL, in: Proceedings of the 6th International Semantic Web Conference, LNCS, vol. 4825, Springer, 2007, pp. 72–85.
[4] N. Bozovic, V. Vassalos, Two-phase schema matching in real world relational databases, in: Proceedings of the ICDE Workshop on Information Integration Methods, Architectures, and Systems, 2008, pp. 290–296.
[5] D. Brickley, R.V. Guha (Eds.), RDF vocabulary description language 1.0: RDF schema, W3C Recommendation, 10 February 2004, <http://www.w3.org/TR/rdf-schema/>.
[6] S. Castano, V. De Antonellis, S. De Capitani di Vimercati, Global viewing of heterogeneous data sources, IEEE Transactions on Knowledge and Data Engineering 13 (2) (2001) 277–297.
[7] K.C.-C. Chang, B. He, C. Li, M. Patel, Z. Zhang, Structured databases on the web: observations and implications, SIGMOD Record 33 (3) (2004) 61–70.
[8] G. Cheng, W. Ge, H. Wu, Y. Qu, Searching semantic web objects based on class hierarchies, in: Proceedings of the WWW Workshop on Linked Data on the Web, 2008.
[9] I.F. Cruz, W. Sunna, N. Makar, S. Bathala, A visual tool for ontology alignment to enable geospatial interoperability, Journal of Visual Languages and Computing 18 (3) (2007) 230–254.
[10] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, iMAP: discovering complex semantic matches between database schemas, in: Proceedings of the 23th ACM International Conference on Management of Data, ACM Press, 2004, pp. 383–394.
[11] C.H.Q. Ding, X. He, H. Zha, M. Gu, H.D. Simon, A min–max cut algorithm for graph partitioning and data clustering, in: Proceedings of the IEEE International Conference on Data Mining, 2001, pp. 107–114.
[12] L. Ding, R. Pan, T. Finin, A. Joshi, Y. Peng, P. Kolari, Finding and ranking knowledge on the semantic web, in: Proceedings of the 4th International Semantic Web Conference, LNCS, vol. 3729, Springer, 2005, pp. 156–170.
[13] H.-H. Do, E. Rahm, Matching large schemas: approaches and evaluation, Information Systems 32 (6) (2007) 857–885.
[14] A. Doan, A. Halevy, Semantic integration research in the database community: a brief survey, AI Magazine 26 (1) (2005) 83–94.
[15] J. Euzenat, A. Isaac, C. Meilicke, P. Shvaiko, H. Stuckenschmidt, O. Sváb, V. Svátek, W.R. van Hage, M. Yatskevich, Results of the ontology alignment evaluation initiative 2007, in: Proceedings of the ISWC + ASWC Workshop on Ontology Matching, 2007, pp. 96–132.
[16] J. Euzenat, P. Shvaiko, Ontology Matching, Springer, 2007.
[17] S.M. Falconer, M. Storey, A cognitive support framework for ontology mapping, in: Proceedings of the 6th International Semantic Web Conference, LNCS, vol. 4825, Springer, 2007, pp. 114–127.
[18] A. Gal, Managing uncertainty in schema matching with top-k schema mappings, in: Journal on Data Semantics VI, LNCS, vol. 4090, Springer, 2006, pp. 90–114.
[19] A. Gal, G. Modica, H. Jamil, A. Eyal, Automatic ontology matching using application semantics, AI Magazine 26 (1) (2005) 21–31.
[20] B.C. Grau, I. Horrocks, Y. Kazakov, U. Sattler, Just the right amount: extracting modules from ontologies, in: Proceedings of the 16th International World Wide Web Conference, ACM Press, 2007, pp. 717–726.
[21] T.R. Gruber, A translation approach to portable ontology specifications, Knowledge Acquisition 5 (2) (1993) 199–220.

[22] S. Guha, R. Rastogi, K. Shim, ROCK: a robust clustering algorithm for categorical attributes, in: Proceedings of the 15th International Conference on Data Engineering, 1999, pp. 512–521.

[23] J. Han, M. Kamber, Data Mining: Concepts and Techniques, second ed., Morgan Kaufman Publishers, 2006.

[24] J. Hayes, C. Gutiérrez, Bipartite graphs as intermediate model for RDF, in: Proceedings of the 3rd International Semantic Web Conference, LNCS, vol. 3298, Springer, 2004, pp. 47–61.

[25] B. He, K.C.-C. Chang, Automatic complex schema matching across web query interfaces: a correlation mining approach, ACM Transactions on Database Systems 31 (1) (2006) 346–395.

[26] L. Hollink, M. van Assem, S. Wang, A. Isaac, G. Schreiber, Two variations on ontology alignment evaluation: methodological issues, in: Proceedings of the 5th European Semantic Web Conference, LNCS, vol. 5021, Springer, 2008, pp. 388–401.

[27] W. Hu, N. Jian, Y. Qu, Y. Wang, GMO: a graph matching for ontologies, in: Proceedings of the K-CAP Workshop on Integrating Ontologies, 2005, pp. 41–48.

[28] W. Hu, Y. Qu, Block matching for ontologies, in: Proceedings of the 5th International Semantic Web Conference, LNCS, vol. 4273, Springer, 2006, pp. 300–313.

[29] W. Hu, Y. Qu, Falcon-AO: a practical ontology matching system, Journal of Web Semantics (system paper), 2008.

[30] W. Hu, Y. Zhao, Y. Qu, Partition-based block matching of large class hierarchies, in: Proceedings of the 1st Asian Semantic Web Conference, LNCS, vol. 4185, Springer, 2006, pp. 72–83.

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

[32] R. Kannan, S. Vempala, A. Vetta, On clustering: good bad and spectral, Journal of the ACM 51 (3) (2004) 497–515.

[33] J.F.D. Kengue, J. Euzenat, P. Valtchev, OLA in the OAEI 2007 evaluation contest, in: Proceedings of ISWC + ASWC Workshop on Ontology Matching, 2007, pp. 188–195.

[34] T. Kirsten, A. Thor, E. Rahm, Instance-based matching of large life science ontologies, in: Proceedings of the 4th International Workshop on Data Integration in the Life Sciences, LNCS, vol. 4544, Springer, 2007, pp. 172–187.

[35] G. Klyne, J. Carroll, Resource description framework (RDF): concepts and abstract syntax, W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/rdf-concepts/>.

[36] J. Lacasta, J. Nogueras-Iso, R. Béjar, P.R. Muro-Medrano, F.J. Zarazaga-Soria, A web ontology service to facilitate interoperability within a spatial data infrastructure: applicability to discovery, Data and Knowledge Engineering 63 (3) (2007) 947–971.

[37] I.V. Levenshtein, Binary codes capable of correcting deletions insertions and reversals, Soviet Physics – Doklady 10 (8) (1966) 707–710.

[38] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.

[39] M. Mao, Y. Peng, The Prior+: results for OAEI campaign 2007, in: Proceedings of ISWC + ASWC Workshop on Ontology Matching, 2007, pp. 219–226.

[40] R. McCann, W. Shen, A. Doan, Matching schemas in online communities: a web 2.0 approach, in: Proceedings of the 24th International Conference on Data Engineering, 2008, pp. 110–119.

[41] S. Melnik, H. Garcia-Molina, E. Rahm, Similarity flooding: a versatile graph matching algorithm and its application to schema matching, in: Proceedings of the 18th International Conference on Data Engineering, 2002, pp. 117–128.

[42] P. Mork, P. Bernstein, Adapting a generic match algorithm to align ontologies of human anatomy, in: Proceedings of the 20th International Conference on Data Engineering, 2004, pp. 787–790.

[43] M. Nagy, M. Vargas-Vera, E. Motta, DSSim – managing uncertainty on the semantic web, in: Proceedings of ISWC + ASWC Workshop on Ontology Matching, 2007, pp. 160–169.

[44] N.F. Noy, Semantic integration – a survey of ontology-based approaches, SIGMOD Record 33 (4) (2004) 65–70.

[45] P.F. Patel-Schneider, P. Hayes, I. Horrocks (Eds.), OWL web ontology language semantics and abstract syntax, W3C Recommendation, 10 February 2004. <http://www.w3.org/TR/owl-semantics/>.

[46] Y. Qu, W. Hu, G. Cheng, Constructing virtual documents for ontology matching, in: Proceedings of the 15th International World Wide Web Conference, ACM Press, 2006, pp. 23–31.

[47] V.V. Raghavan, S.K.M. Wong, A critical analysis of vector space model for information retrieval, Journal of the American Society for Information Science 37 (5) (1986) 279–287.

[48] E. Rahm, P. Bernstein, A survey of approaches to automatic schema matching, VLDB Journal 10 (4) (2001) 334–350.

[49] E. Rahm, H.-H. Do, S. Massmann, Matching large xml schemas, SIGMOD Record 33 (4) (2004) 26–31.

[50] C. Rosse, J.L.V. Mejino, A reference ontology for biomedical informatics: the foundational model of anatomy, Journal of Biomedical Informatics 36 (6) (2003) 478–500.

[51] P. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of Computational and Applied Mathematics 20 (1) (1987) 53–65.

[52] J. Seidenberg, A. Rector, Web ontology segmentation: analysis classification and use, in: Proceedings of the 15th International World Wide Web Conference, ACM Press, 2006, pp. 13–22.

[53] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (8) (2000) 888–905.

[54] P. Shvaiko, J. Euzenat, A survey of schema-based matching approaches, in: Journal on Data Semantics IV, LNCS, vol. 3730, Springer, 2005, pp. 146–171.

[55] G. Stoilos, G. Stamou, S. Kollias, A string metric for ontology alignment, in: Proceedings of the 4th International Semantic Web Conference, LNCS, vol. 3729, Springer, 2005, pp. 624–637.

[56] H. Stuckenschmidt, M. Klein, Reasoning and change management in modular ontologies, Data and Knowledge Engineering 63 (2) (2007) 200–223.

[57] X. Su, J.A. Gulla, An information retrieval approach to ontology mapping, Data and Knowledge Engineering 58 (1) (2006) 47–69.

[58] J. Tang, J. Li, B. Liang, X. Huang, Y. Li, K. Wang, Using Bayesian decision for ontology mapping, Journal of Web Semantics 4 (4) (2006) 243–262.

[59] K. Tu, M. Xiong, L. Zhang, H. Zhu, J. Zhang, Y. Yu, Towards imaging large-scale ontologies for quick understanding and analysis, in: Proceedings of the 4th International Semantic Web Conference, LNCS, vol. 3729, Springer, 2005, pp. 702–715.

[60] G. Tummarello, C. Morbidoni, R. Bachmann-Gmür, O. Erling, RDFSync: efficient remote synchronization of RDF models, in: Proceedings of the 6th International Semantic Web Conference, LNCS, vol. 4825, Springer, 2007, pp. 537–551.

[61] M. van Gendt, A. Isaac, L. van der Meij, S. Schlobach, Semantic web techniques for multiple views on heterogeneous collections: a case study, in: Proceedings of the 18th Belgium–Netherlands Conference on Artificial Intelligence, LNCS, vol. 4172, Springer, 2006, pp. 426–437.

[62] W. Winkler, The state record linkage and current research problems, Technical Report, Statistics of Income Division, Internal Revenue Service Publication, 1999.

[63] S. Zhang, O. Bodenreider, Hybrid alignment strategy for anatomical ontologies: results of the 2007 ontology alignment contest, in: Proceedings of the ISWC + ASWC Workshop on Ontology Matching, 2007, pp. 139–149.

[64] X. Zhang, G. Cheng, Y. Qu, Ontology summarization based on RDF sentence graph, in: Proceedings of the 16th International World Wide Web Conference, ACM Press, 2007, pp. 707–715.

[65] A.V. Zhdanova, P. Shvaiko, Community-driven ontology matching, in: Proceedings of the 3rd European Semantic Web Conference, LNCS, vol. 4011, Springer, 2006, pp. 34–49.

Wei Hu received his B.Sc. degree in Computer Science and Technology from the Southeast University, China, in 2005. He is currently a Ph.D. student at the School of Computer Science and Engineering, Southeast University, China. From November 2007 to April 2008, he was a guest researcher in the Knowledge Representation and Reasoning Group, Vrije Universiteit Amsterdam, the Netherlands. His research interests include ontology matching, data integration and Semantic Web.

Yuzhong Qu is a full professor at the School of Computer Science and Engineering, Southeast University, China. He is currently leading the Institute of Web Science at the Southeast University, China. He received his B.Sc. and M.Sc. degrees in Mathematics from the Fudan University, China, in 1985 and 1988, respectively. He received his Ph.D. degree in Computer Software from the Nanjing University, China, in 1995. His research interests include Software Engineering, Semantic Web and Web Science.

Gong Cheng received his B.Sc. degree in Computer Science and Technology from the Southeast University, China, in 2006. He is currently a Ph.D. student at the School of Computer Science and Engineering, Southeast University, China. His research interests include semantic search, data management and Semantic Web.