Partition-Based Block Matching of Large Class Hierarchies

Wei Hu, Yuanyuan Zhao, and Yuzhong Qu

School of Computer Science and Engineering, Southeast University,
Nanjing 210096, P.R. China
{whu, yyzhao, yzqu}@seu.edu.cn

Abstract. Ontology matching is a crucial task of enabling interoperation between Web applications using different but related ontologies. Due to the size and the monolithic nature, large-scale ontologies regarding real world domains cause a new challenge to current ontology matching techniques. In this paper, we propose a method for partition-based block matching that is practically applicable to large class hierarchies, which are one of the most common kinds of large-scale ontologies. Based on both structural affinities and linguistic similarities, two large class hierarchies are partitioned into small blocks respectively, and then blocks from different hierarchies are matched by combining the two kinds of relatedness found via predefined anchors as well as virtual documents between them. Preliminary experiments demonstrate that the partition-based block matching method performs well on our test cases derived from Web directory structures.

1 Introduction

Large-scale ontologies are a kind of ontologies created to describe complex real world domains. Large class hierarchies are one of the most common kinds of large-scale ontologies. Due to the decentralized nature of the Web, these large ontologies or class hierarchies for the same domain aren't unique. Examples can be found in: (a) Web directory structures, e.g., Google and Yahoo [1]; (b) product description standards, e.g., NAICS[1] and UNSPSC[2]; and (c) medicine or biology, e.g., GALEN[3] and FMA[4]. In order to achieve interoperation among Semantic Web applications using these large ontologies or class hierarchies, ontology matching is necessary. However, the size and the monolithic nature of these large ontologies or class hierarchies cause a new challenge to current ontology matching techniques. Therefore, some novel solutions are required.

In this paper, we propose a method for partition-based block matching that is practically applicable to large class hierarchies. Based on both structural affinities and linguistic similarities, two large class hierarchies are partitioned into

[1] http://www.naics.com
[2] http://www.unspsc.org
[3] http://www.opengalen.org
[4] http://sig.biostr.washington.edu/projects/fm

R. Mizoguchi, Z. Shi, and F. Giunchiglia (Eds.): ASWC 2006, LNCS 4185, pp. 72–83, 2006.

small blocks respectively, and then blocks from different hierarchies are matched by combining the two kinds of relatedness found via predefined anchors as well as virtual documents between them. Usually, structural affinities are computed by how closely they are related in the hierarchies, and linguistic similarities are computed by examining the similarities between the descriptions of the classes. The combinations of structural affinities and linguistic similarities are used to reflect the weighted links between the classes. Thus, large class hierarchies can be divided into small blocks base on an efficient linkage-based partitioning algorithm, e.g., ROCK [7]. Thereafter, two kinds of relatedness between blocks are found: one is via anchors which can be predefined by some simple methods or by experts; the other is via virtual documents [9]. These two kinds of relatedness are combined to match blocks in the end. The overview of the matching process is illustrated in Figure 1.

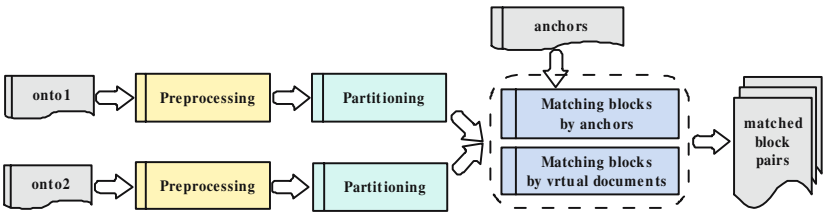

Fig. 1. The overview of the matching process

The rest of the paper is organized as follows: in the next section, some related works are introduced. In Section 3, we propose an efficient algorithm to partition large class hierarchies into small blocks. In Section 4, we present an approach to matching blocks. In Section 5, we show some preliminary experimental results to demonstrate the effectiveness of the method. In Section 6, we conclude with some directions for future work.

2 Related Work

Today, quite a lot of ontology matching or aligning approaches exist in literature, such as QOM [4], OLA [5], and V-Doc [9]. Please see [11] for a good survey about more representative approaches. However, most of these approaches have been developed for small-scale ontologies. For example, in V-Doc, when the ontologies to be matched have thousands of concepts, the matching process will take insufferably long time, and sometimes even cannot work. Another limit in current approaches is that most of them aim at 1:1 matching, not *block matching* (the relationship cardinality of the matching is many-to-many). Even in the field of schema matching, there are only a few works addressing the block matching issue, such as Artemis [2] and iMAP [3]. Artemis firstly computes 1:1 matching by using WordNet, and then generates block matching from the 1:1 matching by a hierarchical clustering algorithm. It is clear that this method is not targeted

to large-scale ontologies because of its computational complexity for computing the 1:1 matching. iMAP semi-automatically discovers both 1:1 and complex mappings (e.g., room-price = room-rate $*$ (1 + tax-rate)). It exploits two new kinds of domain knowledge, i.e., overlap data and external data, to discover complex mappings. However, iMAP may be not a universal solution because it's not easy to specify the domain knowledge in some special cases.

The issue of partitioning large-scale ontologies (including large class hierarchies) has been recently addressed in [6,13,14], etc. In [6], an efficient solution for partitioning ontologies is provided by using ε-Connections. It guarantees that all concepts which have subsumption relations can be partitioned into one block, which becomes a limitation for ontology matching. In [13], large class hierarchies are automatically partitioned into small blocks. The background techniques are dependency graph and "island" algorithm. Although the main contribution of [14] is for ontology visualization, it also presents a method for ontology partitioning by Force Directed Placement algorithm. The main problem of these work is that they do not much concentrate on the sizes of blocks, so they do not well support ontology matching. For example, by applying ε-Connections to GALEN, we can gain only one block including nearly 10,000 concepts, which is not an appropriate size for ontology matching.

Compared with them, our method for partition-based block matching has three features: (a) it is efficient for large class hierarchies. In particular, the time complexity of the partitioning algorithm is $O(n^2)$; (b) it aims at block matching, because we believe it seems more useful for large class hierarchies than the current 1:1 matching; and (c) the sizes of most blocks are small enough to apply current ontology matching techniques to them.

3 Ontology Partitioning

In this section, we firstly introduce the notion of weighted links which are generated by combining two kinds of partitioning features extracted from large class hierarchies. Then, we present an efficient partitioning algorithm based on these weighted links.

3.1 Partitioning Features

In our investigation, large class hierarchies usually have two distinguishing characteristics: (a) they are often represented in DAG (Directed Acyclic Graph) structures and *is-a* relations are the most important built-in relations in large class hierarchies. An example is UNSPSC, it has 16500 classes, and the number of *rdfs:subClassOf* relations is 16500; and (b) linguistic similarities can be found between the local descriptions (e.g., local names, labels, comments) of the classes in these hierarchies. Therefore, two kinds of partitioning features can be extracted from large class hierarchies: one is structural affinities, which are based on (a); the other is linguistic similarities, which are based on (b).

Structural affinities between classes are defined by how closely they are related in the hierarchies, i.e., their structural closeness.

Definition 1 (Structural Affinities between Classes). *Let c_i, c_j be two classes. c_{ij} is the common superclass of c_i and c_j. $depthOf(c_k)$ returns the depth of class c_k in the class hierarchy. The structural affinity between c_i and c_j is defined as follows:*

$$aff_s(c_i, c_j) = \frac{2 \cdot depthOf(c_{ij})}{depthOf(c_i) + depthOf(c_j)}. \tag{1}$$

This equation has also been proposed in [14]. Please note that computing structural affinities between all the classes is time-consuming. Usually, only computing the affinities between the classes with adjacent depthes can obtain moderate results. In our experiments, we only compute structural affinities between the classes which satisfy $|depthOf(c_i) - depthOf(c_j)| \leq 1$.

Linguistic similarities are computed by examining the similarities between the local descriptions of the classes. Here, we adopt the string comparison method proposed in [12]. It considers that the similarity between two descriptions of two classes is related to their commonalities as well as to their differences.

Definition 2 (Linguistic Similarities between Classes). *Let d_i be the description of class c_i, d_j be the description of class c_j. The linguistic similarity between c_i and c_j is defined as follows:*

$$sim_l(c_i, c_j) = comm(d_i, d_j) - diff(d_i, d_j) + winkler(d_i, d_j), \tag{2}$$

where $comm(d_i, d_j)$ stands for the commonality between d_i and d_j, $diff(d_i, d_j)$ for the difference, and $winkler(d_i, d_j)$ for the improvement of the result using the method introduced by Winkler in [15].

The experimental results shown in [12] indicate that 0.65 is a good threshold, i.e., when $sim_l(c_1, c_2) < 0.65$, c_1 and c_2 would not be considered similar. In our experiments, we also find this threshold performs well in most scenarios, so we still take this threshold.

Finally, weighted links between classes are generated by combining the structural affinities and the linguistic similarities.

Definition 3 (Links between Classes). *Let c_i, c_j be two classes. ϵ_1 is a given threshold which satisfies $\epsilon_1 \in [0, 1)$. The weighted link between c_i and c_j is defined as follows:*

$$link(c_i, c_j) = \begin{cases} aff(c_i, c_j) & if\ aff(c_i, c_j) > \epsilon_1 \\ 0 & otherwise \end{cases}, \tag{3}$$

$$aff(c_i, c_j) = \alpha \cdot aff_s(c_i, c_j) + (1 - \alpha) \cdot sim_l(c_i, c_j), \tag{4}$$

where $\alpha \in [0, 1]$, and the selection of the parameter α depends on the structural and linguistic characteristics of the large class hierarchies.

We choose a small ϵ_1 for link filtering in our experiments, because the linkage among the classes is sparse, using a large ϵ_1 may cause many small "island" blocks, i.e., each block only contains several classes.

3.2 Partitioning Algorithm

Our partitioning algorithm is an agglomerative hierarchical partitioning algorithm mainly inspired by ROCK [7], which is a famous agglomerative clustering algorithm in the field of Data Mining. The main difference between ROCK and ours is that ROCK assumes that all the links between classes are the same; while we import the notion of weighted links, which reflect the information about the closeness between classes. Our algorithm accepts as input the set of n blocks to be clustered, which is denoted by B, and the desired number of blocks k, which is initially determined by application requirement. In each partitioning iteration, it selects the block having the maximum cohesiveness firstly, then choose the block having the maximum coupling with it, and finally merge these two blocks into a new block. The pseudo code of the algorithm is presented in Table 1.

Table 1. The partitioning algorithm

```
procedure(B, k)
   for each block B_i in B, do begin
      initialize the internal sum of links within B_i, called cohesiveness;
      initialize the sum of links between B_i and others, called coupling;
   end
   while the number of current blocks m > k do begin
      choose the best block B_i, which has the maximum cohesiveness;
      choose one block from the rest, which has the maximum coupling;
      merge block B_i and B_j named B_p;
      update B_p's cohesiveness and coupling;
      remove B_i and B_j;
      for each block other than B_p, update it's coupling;
      m := m − 1;
   end
end
```

The time complexity of this algorithm is $O(n^2)$. Compared with most other clustering or partitioning algorithms, it is quite efficient. Though k-means method is faster, it is worthy of noting that the means of the blocks are virtual entities, and if we change the means to the real entities (called k-medoids method), the time complexity also becomes $O(n^2)$ (e.g., PAM [8]). In addition, the centroid-based clustering algorithms aren't suitable for blocks of widely different sizes.

The most important point of the partitioning algorithm shown above is the computation of cohesiveness and coupling. Here, *goodness*() is used to compute the cohesiveness and coupling, and it measures the distance of two clusters by comparing the aggregate inter-connectivity of them.

Definition 4 (Goodness). *Let B_i, B_j be two blocks. $sizeOf(B_k)$ returns the number of the classes in B_k. The goodness between B_i and B_j is computed as follows:*

$$goodness(B_i, B_j) = \frac{\sum_{c_i \in B_i, c_j \in B_j} link(c_i, c_j)}{sizeOf(B_i) \cdot sizeOf(B_j)}, \quad (5)$$

when B_i, B_j are the same block, it computes the cohesiveness of the block; when B_i, B_j are two different blocks, it computes the coupling between them.

As pointed out in [7], choosing the denominator as $sizeOf(B_i) + sizeOf(B_j)$ is ill-considered. Though it may work well on well-separated blocks, in case of outliers or blocks with the classes that are neighbors, a large block may swallow other blocks and thus, the classes from different blocks may be merged into a single block. This is because a larger block typically would have a larger number of cross links with other blocks.

4 Block Matching

In this section, we present an approach to matching blocks. As shown in Figure 1, after partitioning pairwise large class hierarchies into two sets of small blocks respectively, we can find two kinds of relatedness between blocks from different sets: one is via predefined anchors; the other is via virtual documents [9]. The two kinds of relatedness are combined to match blocks.

Please note that we only match blocks here, because: (a) blocks give a sketch of large class hierarchies, matching them is helpful for users to understand the correspondence between two large class hierarchies; and (b) the sizes of matched block pairs are usually small enough to take current ontology matching techniques to generate accurate 1:1 matching.

4.1 Relatedness Between Blocks Via Anchors

Predefined matched class pairs, called *anchors*, are utilized to find relatedness between blocks. The anchors can be defined by some simple approaches or by experts. For example, the following steps are taken to gain the anchors in our experiments. Please note that the trade-off between the correctness and the number of the anchors should be considered.

1. Find a set of high precision matched class pairs as starting points. This could be done with some string comparison techniques, e.g., [12].
2. Manually remove some incorrect matched class pairs.
3. Manually add some omissions.

Then, the relatedness between blocks can be computed via the anchors gained above. The background idea is that the more anchors we have found between the two blocks, the more related the two blocks are.

Definition 5 (Relatedness between Blocks via Anchors). *Let B_i be a block in class hierarchy H while $B_j^{'}$ be a block in another class hierarchy $H^{'}$. k denotes the number of the blocks in H, and $k^{'}$ denotes the number of the blocks*

in $H^{'}$. $anchors(B_u, B_v^{'})$ returns the number of predefined anchors between B_u and $B_v^{'}$. The relatedness between B_i and $B_j^{'}$ is defined as follows:

$$rel_a(B_i, B_j^{'}) = \frac{2 \cdot anchors(B_i, B_j^{'})}{\sum_{u=1}^{k} anchors(B_u, B_j^{'}) + \sum_{v=1}^{k^{'}} anchors(B_i, B_v^{'})}. \quad (6)$$

4.2 Relatedness Between Blocks Via Virtual Documents

Relatedness between blocks are also computed via virtual documents, together with the prevalent TF/IDF [10] technique. In [9], the virtual documents are constructed for concepts (classes, properties or instances) from two ontologies. In this paper, the virtual document of a block is an aggregation of the virtual documents of the classes contained in the block.

The virtual document of a block consists of a collection of weighted tokens, which originate from the local descriptions (e.g., local names) of all the classes it contains and incorporate a weighting scheme to reflect the importance of information appeared in different categories (e.g., tokens appeared in *rdfs:label* are more important than those appeared in *rdfs:comment*). These weighted tokens can be used to reflect the intended meaning of the block.

Then, the virtual document of each block can be represented as a vector in the vector space. The components of the vector are the scores from corresponding tokens, which reflect the relatedness between tokens and the block. The higher the score is, the more the token is related to the block. In addition to the selection of tokens to represent the block, it is common to associate a weight to each token in a block to reflect the importance of that token. Thus, TF/IDF technique is adopted to optimize the vector representation.

Definition 6 (Relatedness between Blocks via Virtual Documents). *Let B_i be a block in class hierarchy H while $B_j^{'}$ be a block in another class hierarchy $H^{'}$. s_{ik} denotes the score of a unique token t_{ik} in B_i, and $s_{jk}^{'}$ denotes the score of a unique token $t_{jk}^{'}$ in $B_j^{'}$. D is the dimension of the vector space. The relatedness between B_i and $B_j^{'}$ is measured by the cosine value between two vectors:*

$$rel_v(B_i, B_j^{'}) = \frac{\sum_{k=1}^{D} s_{ik} s_{jk}^{'}}{\sqrt{\sum_{k=1}^{D}(s_{ik})^2 \cdot \sum_{k=1}^{D}(s_{jk}^{'})^2}}. \quad (7)$$

If the vectors of B_i, $B_j^{'}$ don't share any tokens, the relatedness will be 0.0; if all the token scores equal completely, it will be 1.0. The score of a unique token t_k in a specific block is defined as follows:

$$\begin{aligned} score(t_k) &= tf \cdot idf \\ &= \frac{t}{T} \cdot \frac{1}{2}(1 + \log_2 \frac{N}{n}), \end{aligned} \quad (8)$$

where t denotes the refined token occurrence, T denotes the total refined occurrence among all the tokens in a specific block, n denotes the number of the blocks containing this token, and N denotes the number of all the blocks.

4.3 Combination

The two kinds of relatedness between blocks computed above are combined to form the overall relatedness. The background assumption is the overall relatedness between two blocks is higher than the one between two other blocks, then it means that the former matched block pairs have more in common than the latter ones.

Definition 7 (Overall Relatedness between Blocks). *Let B_i be a block in class hierarchy H while $B_j^{'}$ be a block in another class hierarchy $H^{'}$. The overall relatedness between B_i and $B_j^{'}$ is defined as follows:*

$$rel(B_i, B_j^{'}) = \beta \cdot rel_a(B_i, B_j^{'}) + (1 - \beta) \cdot rel_v(B_i, B_j^{'}), \tag{9}$$

where $\beta \in [0, 1]$.

Finally, after combining the relatedness between all the blocks from two hierarchies, we select the matched block pairs whose overall relatedness are larger than a given threshold ϵ_2.

5 Experimental Results

In this section, we present some preliminary experimental results in order to evaluate the performance of the partition-based block matching method. To the best of our knowledge, no existing work has shown the experimental results on matching the blocks of large class hierarchies, so we cannot make an objective comparison and measurement. Although manually observing these results is tedious and error-prone, we still believe the evaluation is essential to make progress in this difficult problem.

We apply a pairwise large class hierarchies available in OWL – two Web directory structures proposed in [1] (the data set can be downloaded from OAEI 2005[5]) – to evaluate the performance of our method. Due to lack of space, we don't list the classes contained in each block, the complete results of all the experiments can be found at our Web site that accompanies this paper[6].

5.1 Labeling Blocks

Before introducing the experimental results, it is helpful to label the blocks by representative classes to assist the evaluation. We derive from the descriptions of the most important classes to display the blocks for human observation and understanding. Here, we simply compute the importance of each class based on the size of its children in the block as well as its relative depth.

Definition 8 (Importance). *Let B_i be a block. c_i is a class in B_i, and C_i^{child} is the set of c_i's children which are also in B_i. $|C_i^{child}|$ returns the number of*

[5] http://oaei.inrialpes.fr/2005/directory/
[6] http://xobjects.seu.edu.cn/project/falcon/matching/

classes in C_i^{child}. $depthOf(c_i)$ *returns the depth of class* c_i *in the class hierarchy. The importance of* c_i *in* B_i *is defined as follows:*

$$importance(c_i) = |C_i^{child}| - 2^{relativeDepth(c_i)}, \tag{10}$$

$$relativeDepth(c_i) = depthOf(c_i) - min_{c_k \in B_i}(depthOf(c_k)), \tag{11}$$

where the first part of $importance(c_i)$ *gives more importance to the classes with more children; while the second part gives less importance to the classes deeper in the class hierarchy.*

After computing the importance of all the classes in a specific block, the descriptions of the most important class is selected for the purpose of displaying the block.

5.2 Web Directory Structures

The data set are constructed from Google, Yahoo and Looksmart Web directories as described in [1]. The data set contains 2265 pairwise ontology files and each file contains a path of *rdfs:subClassOf* relations from the leaf class to the root. In our experiments, we firstly merge these 2265 pairwise files to two large class hierarchies, namely the source ontology and the target ontology, and then we apply the partition-based block matching method to them. Experiments are carried out on a 2.8GHz Pentium 4 with 512MB of RAM running on Windows XP Service Pack 2. The parameters used in the experiments are as follows: α in Equation (4) is 0.5, β in Equation (9) is 0.5, ϵ_1 for links filtering is 0, and ϵ_2 for selecting matched block pairs is 0.15.

As depicted in Figure 1, the process of the partition-based block matching method can be divided into three steps. The first step is preprocessing, including loading two large class hierarchies, parsing them and generating links. The next involves partitioning the two hierarchies into blocks, which are then saved to disk. The final step is matching blocks by combining the two kinds of relatedness found via anchors as well as virtual documents. Table 2 gives a breakdown of how long various steps of the matching process take.

Table 2. The runtime per step

	preprocessing	partitioning	matching blocks
time	22s	4s	18s

By preprocessing the two large class hierarchies, the source ontology contains 1067 classes, the number of *rdfs:subClassOf* relations is 1313, the maximum depth is 10, and the number of links is 4063; the target ontology contains 1560 classes, the number of *rdfs:subClassOf* relations is 2331, the maximum depth is 9, and the number of links is 6921. In the partitioning step, the source ontology is partitioned into 6 blocks, the maximum size of the blocks is 204, the minimum size is 142 and the average size is 178; while the target ontology is partitioned

Table 3. The summary of the experimental results

name	classes	subClassOf	depth	links	blocks	anchors	pairs
source	1067	1313	10	4063	6	521	9
target	1560	2331	9	6921	7		

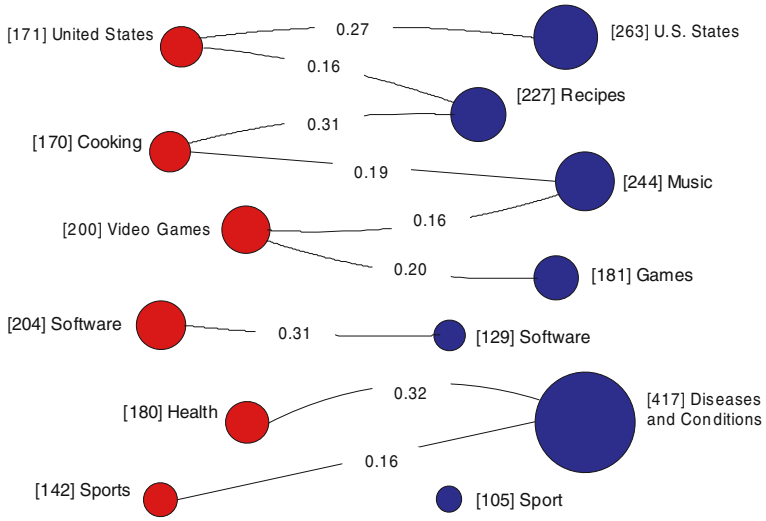

Fig. 2. The details of the experimental results

into 7 blocks, the maximum size of the blocks is 417, the minimum size is 105 and the average size is 223. Finally, in the step of matching blocks, 512 anchors are found by [12], the two kinds of relatedness between blocks are computed via the found anchors as well as virtual documents, and they are combined to gain 9 matched block pairs. The summary of the experimental results on Web directory structures is shown in Table 3.

The graph depicted in Figure 2 shows some useful details of the experimental results. The cycles at the left side represent the blocks of the source ontology and the cycles at the right side represent the blocks of the target ontology. The size of each cycle reflects the number of classes the block contains. The value on each arc shows the overall relatedness between the two matched block pairs.

Discussion. (a) The complete process of the partition-based block matching method takes 44s to complete. Half of the time is spent for preprocessing the two large class hierarchies. It can also be observed that blocks from large class hierarchies can be partitioned with good computational efficiency; (b) 9 matched block pairs are found, 5 matched block pairs are exactly correct by manually evaluating; while 1 potential matched block pairs is missing (Sports vs. Sport), because the number of the anchors from these two blocks is few and the relatedness found via virtual documents is also low due to lack of common tokens. So the approximate precision of our results is 0.56 (5/9) and the recall is 0.83 (5/6); and

(c) current ontology matching techniques can be applied to the matched block pairs for generating 1:1 matched class pairs, for example, we apply V-Doc [9] to the 9 matched block pairs, and then find 798 1:1 matched class pairs.

6 Conclusion and Future Work

In this paper, we propose a method for partition-based block matching that is practically applicable to large class hierarchies. The main contributions of this paper are as follows:

- We present a partitioning algorithm based on both structural affinities and linguistic similarities. The partitioning algorithm is efficient for large class hierarchies, and the time complexity is $O(n^2)$.
- We introduce an approach to matching blocks, which selects matched block pairs by combining the two kinds of relatedness found via predefined anchors as well as virtual documents.
- We describe some preliminary experiments to demonstrate that the partition-based block matching method performs well on our test cases derived from Web directory structures.

As the next step, we are planning to make a comparison between our partitioning algorithm and some others, and the comparison includes both effectiveness and efficiency. Another direction of future research is extending the scope of our method to large-scale ontologies, including both classes and properties. The third direction is how to co-partition (co-clustering) two ontologies, this issue has not yet been touched in the field of ontology matching.

Acknowledgements

The work is supported in part by the NSFC under Grant 60573083, and in part by the JSNSF under Grant BK2003001. The third author of this paper is also supported by NCET (New Century Excellent Talents in University) Program under Grant NCET-04-0472. We would like to thank Ningsheng Jian and Dongdong Zheng for their work in the experiments related to this paper. We would also like to thank anonymous reviewers for their helpful suggestions.

References

1. Avesani, P., Giunchiglia, F., and Yatskevich, M.: A Large Scale Taxonomy Mapping Evaluation. Proceedings of the 4th International Semantic Web Conference. (2005) 67–81
2. Castano, S., De Antonellis, V., and De Capitani Di Vimercati, S.: Global Viewing of Heterogeneous Data Sources. IEEE Transactions on Knowledge and Data Engineering. **13(2)** (2001) 277–297

3. Dhamankar, R., Lee, Y., Doan, A. H., Halevy, A., and Domingos, P.: iMAP: Discovering Complex Semantic Matches between Database Schemas. Proceedings of the 23th ACM SIGMOD International Conference on Management of Data. (2004) 383–394
4. Ehrig, M., and Staab, S.: QOM - Quick Ontology Mapping. Proceedings of the 3rd International Semantic Web Conference. (2004) 683–696
5. Euzenat, J., and Valtchev, P.: Similarity-Based Ontology Alignment in OWL-Lite. Proceedings of the 16th European Conference on Artificial Intelligence. (2004) 333–337
6. Grau, B., Parsia, B., Sirin, E., and Kalyanpur, A.: Automatic Partitioning of OWL Ontologies Using ε-Connections. Proceedings of the 2005 International Workshop on Description Logics. (2005)
7. Guha, S., Rastogi, R., and Shim, K.: ROCK: A Robust Clustering Algorithm for Categorical Attributes. Proceedings of the 15th International Conference on Data Engineering. (1999) 512–521
8. Kaufman, L., and Rousseeuw, P.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons. (1990)
9. Qu, Y. Z., Hu, W., and Cheng, G.: Constructing Virtual Documents for Ontology Matching. Proceedings of the 15th International World Wide Web Conference. (2006) 23–31
10. Salton, G., and McGill, M. H.: Introduction to Modern Information Retrieval. McGraw-Hill. (1983)
11. Shvaiko, P., and Euzenat, J.: A Survey of Schema-Based Matching Approaches. Journal on Data Semantics (IV). (2005) 146–171
12. Stoilos, G., Stamou, G., and Kollias, S.: A String Metric for Ontology Alignment. Proceedings of the 4th International Semantic Web Conference. (2005) 623–637
13. Stuckenschmidt, H., and Klein, M.: Structure-Based Partitioning of Large Concept Hierarchies. Proceedings of the 3rd International Semantic Web Conference. (2004) 289–303
14. Tu, K., Xiong, M., Zhang, L., Zhu, H., Zhang, J., and Yu, Y.: Towards Imaging Large-Scale Ontologies for Quick Understanding and Analysis. Proceedings of the 4th International Semantic Web Conference. (2005) 702–715
15. Winkler, W.: The State Record Linkage and Current Research Problems. Technical Report, Statistics of Income Division, Internal Revenue Service Publication. (1999)