Modularization of Graph-Structured Ontology with Semantic Similarity

Soudabeh Ghafourian, Amin Rezaeian, and Mahmoud Naghibzadeh

Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
so.ghafourian@stu-mail.um.ac.ir
amin.rezaeian@stu-mail.um.ac.ir
naghibzadeh@um.ac.ir

Abstract. Modularization is a key requirement to manage the size and complexity of large ontologies by replacing each one by a set of smaller ontologies. Two reasons for this requirement are that current ontology languages such as OWL do not allow partial reuse of ontologies and ontologies are ever growing to cover more knowledge in a specific domain. Many existing modularization methods focus on either semantics or structural aspects of ontologies while both of them are important. In this paper, we consider both semantic and structure by combining these aspects using random walk algorithms to achieve a balance between them. We also define weights for different relations to take semantic into account. The proposed method is designed using two algorithms: a greedy algorithm and a heuristic one to reduce run-time and time-complexity. Our goal is to produce reusable modules of high quality and support large ontologies. The results of the experiments show that our algorithms perform well in comparison with existing golden standard.

Keywords: Ontology Modularization, Partitioning, Ontology Reuse, Semantic Similarity

1 Introduction

Today, we use ontologies in the context of information processing, semantic web, and many other contexts [1]. Large ontologies usually contain many terms. These large ontologies are confronted with challenges in their life cycle such as processing and maintenance [2].

Modularization is an approach to tackle challenges of large ontologies. An ontology module is a small ontology that can have inter-module links with other small ontologies [1] and the union of the produced modules is semantically equal to the first ontology [3].

Ontology modularization is used to achieve different goals such as scalability for querying data and reasoning on ontologies, scalability for evolution and maintenance, complexity management, understandability, context-awareness and personalization and reusability. The goals of ontology modularization can affect the understandability, the advantages, and disadvantages of the resulting modules [1].

Reusability is one of the important goals of modularization of ontologies that apply when designing ontologies or when developing new applications based on ontologies. Current ontology languages such as OWL do not allow importing only parts of an ontology. This is a problem because if an ontology developer needs to reuse only parts of an ontology, they must import the whole ontology, which takes more space [4]. For example, we have home appliances ontology and an ontology developer who is interested in small home appliances; the developer would like to import the part of the ontology, which concerns only small home appliances. Therefore, he is not interested in loading the whole ontology but rather an extracted module of the ontology [2].

In this paper, we present modularization algorithms that assign weights to the different relations that are used in the formalization of the ontology. The goal of our modularization is reusability. The proposed modularization algorithms are performed after the weighing phase of different relations is completed. Until now, some of these relations such as inverseOf, unionOf, intersectionOf, disjointWith, have not been considered. Since different kinds of the relations represent different semantic aspects of the ontology, the weighing is determined according to the importance of these relations. The proposed algorithms consider both structural and semantic criteria.

Our goal is to produce modules in which concepts are both structurally and semantically closely related; hence, the resulting subontologies can be reused in developing new applications.

Modularization is done based on an agglomerative hierarchical algorithm on which we perform some optimization to achieve a better result and use new scoring function. We also propose another algorithm to reduce time-complexity.

The remainder of the paper is organized as follows: in the next section, we briefly present some of the related work. In section 3, we define weights for different relations; describe our modularization algorithms and criteria for modularization. In Section 4, we evaluate the proposed method and report our results. Finally, Section 5 concludes this paper with future work suggestions.

2 Related work

In this section, we review related work to ontology modularization. Recently, many different approaches have been proposed for this purpose. There are several categories for modularization. We categorize modularization based on types of representation of ontology into two sets: graph-based approaches and logic-based approaches [2-3]. Our work is based on graph approaches. In logic-based approaches, the modules are produced based on logical representation of ontologies. The work in [5] extracts modules from OWL-DL ontology based on user's semantic query. Graph-based approaches use graph-theoretic algorithms to traverse the hierarchy of ontologies and some heuristics to get relevant modules [6]. In general, the works [7-10] use agglomerative algorithms to modularize ontologies. An agglomerative approach is an iterative bottom up one in which, in each iteration, two modules with the highest similarity are merged to produce a new module. The work in [7] defines structural and linguistic

similarities. The first criterion is based on hierarchy of classes and the other one is based on similarities between the local descriptions of the classes. They present partition algorithms that combine criteria as a scoring function. Their goal is ontology matching. In matching ontologies, one tries to find the most similar ontology amongst a set of ontologies to the one for which a match is requested. The approach performs modularization of each two ontologies whose similarity is needed. In the modularization, the hierarchical subclassOf relation and a linguistic similarity measure are used.

In [8], A weighted graph is constructed from rdfs:subClassOf and rdfs:subPropertyOf relationships. owl:equivalentClass or owl:equivalentProperty are identified in a preprocessing stage. They present a structure-oriented partitioning algorithm and add RDF sentences to construct blocks from the modules. The goal of this work is ontology matching. This work only considers limited relationships. Relations such as inverseOf, unionOf, intersectionOf, disjointWith, are not considered.

Paper [11] describes a structure-based ontology partitioning. They construct a directed weighted graph based on structure similarities. Five types of criteria between concepts, i.e., subclass, domain/range, definition, substring, and distance relation, are defined. The weighted matrix is constructed according to the number of connections between nodes; then they use Line Island Method [12] to partition and finally perform optimization to improve the partitioning.

The work in [13] proposes an ontology partitioning method, which produces overlapped modules, i.e., final modules may have common concepts. They use semantic similarity between concepts, so their generated graph is conceptual. This work considers structure and semantic, but it does not distinguish between different relationships.

In [9], different graph representations for ontologies are developed. There are three basic representations and two different extensions of these representations. They use several methods to convert ontologies into graphs, and apply community detection algorithms to partition the graphs. Their experiments show that the algorithms work much better when subject, object and predicate are represented as different nodes. Paper [10] further develops the findings of paper [9], but the main difference is that they define a weight function for different relations of the ontology as shown in Table 1; this is the first steps in a semantic approach. They apply three community detection algorithms (a type of an agglomerative algorithm) on different graph representations.

The general topic of papers [7-8] is ontology matching based on structural aspects of ontologies. However, we are interested in ontology modularization with the goal of reusability of resulting modules. The works described in [11] and [13] mostly use structure in order to make a graph representation of an ontology and use classic graph clustering methods to modularize ontologies, while our method uses a hierarchical clustering method. We use different graph clustering methods than [9-10], in addition we consider more relations. These relations are mentioned in section 3.2.

3 Proposed Approach

In this section, we first describe how we represent ontologies. Then we present how we construct the weighted matrix. Next, we normalize the weights of edges, and then Neighborhood Random-Walk Distance is introduced [15]. Finally, the modularization algorithms are described.

Our goal is to bring together into one module the most related concepts that have highest semantic similarities and presumably describe one subdomain. This agrees with the concept of domain specific ontologies [10]. If a good modularization algorithm such as agglomerative algorithms and a suitable scoring function is used this goal is reachable.

Agglomerative algorithms [16] are algorithms where the modules with the highest similarity are iteratively merged. They are bottom-up, this means, initially every node is considered as an independent module and in the end there is only one module.

3.1 Different Graph Representations of Ontology

Ontology web language[1] (OWL) is a semantic web language. It is based on the Resource Description Framework[2] (RDF). RDF represents information as triples of the form (Subject, Predicate, and Object).RDF triples can be mapped to a graph where subject and object are nodes and each predicate is a directed edge from a subject to an object. It displays a simple mental model for RDF that is frequently used [14].

We also use other graph models; since the predicate of one triple is a subject or an object in some other triples, we represent every subject, object, and predicate as separate nodes [10].

There are various representations of ontologies [10].We represent each one of subjects, objects and predicates as a separate node in which we have two types of edges: one type of edge is from subject to predicate and another is from predicate to object. The predicate node contains object or datatype properties. We also consider every individual as a node.

3.2 Constructing Weighted Matrix

We define a weight function to give weighs to different relationships of the ontology. This function assigns an integer number to existing relationships as shown in Table 1. The main reason for weighing relationships is that different relationships have different semantics and show different aspects of the ontology. We would like to distinguish between these aspects and their importance by their weights. This is not meant that a relation with a higher weight is more valuable than a relation with a lower weight, but sometimes it means that a relation with a higher weight has existential precedence over a relation with a lower weight. Therefore, the weights can be changed for different applications and/or in different contexts. For example, if sub-

[1] OWL - http://www.w3.org/OWL
[2] RDF - http://www.w3.org/RDF

ClassOf relation exists because it is part of ontology language, then domain and range relations are meaningful. Therefore, we assign weight 10 to subClassOf and weight 5 (i.e., one half the weight) to domain/range relation. In some other situations, the weights are assigned based on the wideness or narrowness of their meanings. Examples are given below.

List of Different Relationships and Weights. The weights represented in Table 1 are based on previous research and also our assessments of relations which are not studied by previous research. The base of weights is what is mentioned in [10]. For new relations, the weight assignment logic is explained in the previous subsection.

The equivalent relation denotes that the classes have the same meaning, so the classes that have equivalent relation are put into one module. Considering the meaning of the equivalent relation, they are given the highest weight. Furthermore, the subclass relations give some important information about classes; hence, these relationships have high semantic contribution to ontology modularization. As mentioned before, the weight of this relationship is higher than that of domain/range relationship but not as much as the equivalent relationship has [10].

Table 1. List of relationships and weights

Property	Weight	Property	Weight
equivalentClass	20 [10]	unionOf	10
subClassOf	10 [10]	intersectionOf	10
subPropertyOf	10 [10]	disjointWith	0-10
domain	5 [10]	complementOf	10
range	5 [10]	inverseOf	20
comment	0.2 [10]	FunctionalProperty	5
seeAlso	0.2 [10]	InverseFunctionalProperty	5
isDefinedBy	0.2 [10]	Other relations	1
label	0.2 [10]		

The union and intersection relations are the same as subclass relations because if for example class A is the union of C, B and D then each class C, B and D is a subClassOf A.

If disjoint relation exists between highest-level concepts, the weight is considered to be zero because they are really disjoint, however if it occurs in lowest-level concepts, the weight is considered 10. If it occurs somewhere in between, the weight is assigned accordingly.

When two concepts have a complement relationship, it means they are strongly connected. We consider that at first, they have a subclass relationship (their super class is the universal set) and then they have a complement relationship.

We put the inverseOf relations in the modules of the property that is related to, so its weight should be high.

For object property, when the properties have an inverse functional attribute, it means this property implies unique value and on the other hand, domain and range

edges represent subdomains of ontologies. We add the restrictions such as cardinality after modularization because they contain literal values.

Unifying the Weight of Edges. If there is more than one relationship between two nodes, we add up the weights of all the existing relationships between the nodes. Thus, all weights are taken into consideration in our modularization algorithms.

3.3 Normalization

In this phase, the weights of the edges in the graph are normalized to be between zero and one. Thus, the weight of the edge outgoing from a node v is divided by the sum of the weights of outgoing edges from node v. This is needed for input matrix of random walk in which every element must be between zero and one.

$$W_{i,v}^{normal} = \frac{\mathrm{w}_{i,v}}{\Sigma_{j \in out\_edges(V)} W_{j,v}} \quad (1)$$

Where $W_{i,v}$ is the weight of the edge outgoing from a node v that is normalizing and the denominator is the sum of the weights of outgoing edges from node v.

3.4 Neighborhood RandomWalk Distance

We use the neighborhood random walk method [15] to measure vertex closeness. A random walk is a mathematical representation of the path one may navigate through multiple random steps.

$$d(v_i, v_j) = \sum_{T:v_i \to v_j} P(T) c(1-c)^{Length(T)} \quad (2)$$

Where P is transition probability matrix, $Length(T)$ is length of random walk where $Length(T) \leq l$, l is the length that a random walk can go, c is restart probability where $c \in (0,1)$, $d(v_i, v_j)$ is the neighborhood random walk distance from v_i to v_j. It measures vertex closeness and T is a path from v_i to v_j whose length is $Length(T)$ with transition probability $P(T)$.

We perform matrix multiplication on a transition probability matrix of the ontology graph to use the neighborhood random walk model [15].

$$R^l = \sum_{\gamma=0}^{l} c(1-c)^{\gamma} P^{\gamma} \quad (3)$$

In this equation, R is the neighborhood random walk distance matrix. Intuitively every element at row i, column j in R, captures the probability of navigating from node i to j with at most l steps in graph. l is the length that a random walk can go and it comes from the previous formula, and γ is the random walk step. The neighborhood random walk distance matrix is constructed in the following steps:

1. Assigning weights to different relationships
2. Normalization of weight of step 1
3. Constructing transition probability and neighborhood random walk distance matrix

3.5 Modularization Algorithm

The proposed modularization algorithm in this section is an agglomerative algorithm. The main difference between our algorithm and other similar algorithms in [7-8] and [9-10] is that we use a new scoring function that calculates both intra-and inter-connectivity. We apply scoring function for every concept node in our algorithm in order to improve the efficiency. It means that the distance between the node and every other node is measured. Another difference is that we consider more relations than other approaches. We use an agglomerative algorithm, such that in each iteration, we select two modules that have the highest positive impact on the score of modularization. Then those modules are merged. From the scoring function perspective, the scores of other modules may not change. This way the required computation for computing modularization score is not high. The process is shown in Algorithm 1.

The advantages of agglomerative algorithms are that we don't have to know the size and the number of modules and the result of these algorithms depend on the chosen similarity criterion [16].The input to our algorithm is the neighborhood random walk distance matrix.

Our criterion function is the silhouette coefficient $s(i)$ [17] where $-1 \leq s(i) \leq 1$. If $s(i)$ is close to one it means that the node will be appropriately grouped. The average $s(i)$ of a module is a measure that shows how appropriately the nodes have been grouped into modules.

$$s(i) = \frac{a(i)-b(i)}{max\{a(i),b(i)\}} \tag{4}$$

Where i is node, $a(i)$ is the average similarity of i to all other nodes within the same module, $b(i)$ is the highest average similarity of i to nodes of other modules, and $s(i)$ should be computed for each node i. For the module c, s_c which is the average of all $s(i)$ for all nodes i in module c, is defined as follows:

$$s_c = \frac{\sum_{i \in c} s(i)}{n_c} \tag{5}$$

In this equation n_c is the number of nodes in module c. To score the modularization C, we define the scoring function as follows:

$$score(C) = average_{c \in C}(s_c) \tag{6}$$

Equation (6) is used to compute efficiency of the modularization. It comes from the average of scores of each module. Each module's score is computed by the average of scores of its nodes. As the score of every node is calculated using both intra- and inter-module connections, the resulting efficiency of modularization is effected by both intra- and inter-module connections of all nodes in graph.

```
Algorithm 1. Modularization (adjacencyMatrix)
 //C represents whole modularization
 C = put every node in a separate module
 for K=N down to 2   // K shows number of modules
```

```
// for all the pairs of nodes that could be merged
  maxS= -2
  for i=1 to K-1
    for j=i+1 to K
      c=Union(i,j);//merge modules i and j into module C
      C=C∪{c}\{i,j};
       S=Score(C); //according to equation (6)
       //if this new score is better, save it
       if S>maxS
         maxS = S;
         modularization=C;
       end if
    end for
  end for
  bestModularization=modularization;
end for
```

Heuristic Algorithm. We propose the heuristic algorithm to support large ontologies and reduce time-complexity. It is shown in Algorithm 2. The input of this algorithm is incidence matrix that is constructed from adjacency matrix R that is calculated using equation (3). Each row of incidence matrix represents an edge, which consists of first node, second node and weight. It is useful for large data sets. This matrix is ordered descending by weight column.

The time-complexity of this method is $O(n^2)$ where n is the number of concepts in the ontology, whereas the complexity of Algorithm 1 is $O(n^3.complexity_{scoring})$.

```
Algorithm 2. Modularization (adjacencyMatrix)
//C represents whole modularization
  C = put every node in a separate module
  for i=1 to N  // N shows size of Adjacency Matrix
    for j=1 to N
      incidenceMatrix =constructIncidenceMatrix (adjacen-
      cyMatrix);
    end for
  end for
  incidenceMatrix = Sort(incidenceMatrix);
  for i=1 to K // K shows length of adjacency matrix
    if Numberofmodules ==1
      break;
    end if
    //module(i,1) is the module for start node of edge(i)
    //module(i,2) is the module for end node of edge(i)
    c=Union(module(i,1),module(i,2));
    //if size of merged modules are more than ϵ which is
    //number of concepts/3, don't combine modules.
```

```
    if |c|>ϵ
      continue;
    else
      C=C∪{c}\{module(i,1),module(i,2)};
    end if
    //Checks the score of modularization if it is fixed
    //do not merged and terminate the algorithm.
    sc=Score(C);
    if (sc_old=sc)
      break;
    end if
    sc_old=sc;
  end for
```

4 Experimental Results

We have implemented the proposed modularization algorithms in Matlab and use Java to process the ontologies. We use F-measure [10] to evaluate our methods. F-measure is a value between zero and one and higher values show better performance. This metric compares produced modules to reference modules that are already available for tested ontologies. F-measure is computed for pairs of modules in which one module is selected from reference modularization and another is one from the generated modules. Finally, the average F-measures of all modules is computed as the F-measure of whole modularization.

$$F - measure = \frac{2*precision*recall}{precision+recall} \qquad (7)$$

Where *precision* is the number of common concepts between two modules, divide by number of concepts in generated module. On the other hand, *recall* is calculated by dividing number of common concepts by number of reference concepts.

4.1 Dataset

We use FOAF[3], AAIR[4], BIO[5] and SWCO[6] ontologies as introduced in [10] and compared our results to concept grouping of these ontologies that are provided in their websites, and in [10]. Table 2 shows the details of these ontologies. The following provides a brief description of them:

- Friend of a Friend (FOAF) Ontology: FOAF is an ontology that describe persons, their activities and other personal information.

[3] http://xmlns.com/foaf/spec/20100101.html
[4] http://xmlns.notu.be/aair
[5] http://vocab.org/bio/0.1/.html
[6] http://data.semanticweb.org/ns/swc/ontology

- Biographical Information Ontology (BIO): BIO is an ontology to represent biographical information about people, both living and dead.
- Semantic Web Conference Ontology (SWCO): The SWCO defines concepts about academic conferences.
- Atom Activity Streams Vocabulary ontology (AAIR): This ontology is a vocabulary for describing social networking sites activities.

Table 2. Details of ontologies

Ontology	Number of modules as stated in the reference	Number of classes	Number of property
FOAF	5	13	61
BIO	5	42	33
SWCO	5	29	16
AAIR	4	41	26

In [10], they apply three algorithms: Fast Greedy Community (FGC), Walk Community (WTC) and Spin Glass Community (SGC). Because FGC and WTC are agglomerative algorithms, we compare our algorithms with them. As there are several ways introduced in [10] to represent an ontology as a graph, we choose a type of their representation of ontology in that every subject, object and predicate is represented as a separate node.

Our results are shown in Table 3. Column 1 and 2 show F-measure of our algorithms and column 4 and 5 show [10] algorithms the F-measure. The F-measure result is multiplied by 100 as it is done in [10].

Table 3. F-measure comparison of different Algorithms on different ontologies

Ontology	Alg 1	Alg 2	FGC [10]	WTC [10]
FOAF	40	30	32	34
BIO	43	33	83	79
SWCO	47	38	28	31
AAIR	50	41	51	51

Analysis of Result. The distribution of classes and properties within their concept grouping affect F-measure**.** For example for the FOAF ontology, one group just contains properties but we consider both classes and properties to modularize. In concept grouping of AAIR and SWCO, the distribution between classes and property is balanced and subclass relation is defined as the main concept. The groups of the BIO ontology contain one group for classes and four groups for properties, so our score is low. When the concept grouping contains the groups that have classes and property,

our score is good because the main objective of ontology modularization is that the modules describe subdomains.

A Simple Case Study. We also use a small ontology given in [8], which consists of six classes, and one property. In [8], they produce three modules, i.e. modules {Reference, Inproceedings, Book, Monograph}, {Author, Person}, and {hasAuthor}. However in our work, we produce two modules {Reference, Inproceedings, Book, Monograph}, and {has Author, Author, Person}. The reason for these modularizations is that while the work in [8] only considers subClassOf relations, we have considered domain/range and subClassOf relations. The modularization presented by [8] is assumed as reference, and the calculation of F-measure is shown in Table 4. F-measures comparing modules 1 and 2 are consequently 1.0 and 0.5, which are computed using equation (7). As we have only two modules, the F-measure for third module becomes zero. The average of these three gives 0.5 as the F-measure for whole modularization.

Table 4. Experimental result on sample dataset

	Module1	Module2	Module3
Precision	1.0	0.3	0
Recall	1	1	0
F-measure	1.0	0.5	0

5 Conclusion

We have proposed modularization algorithms based on semantic and structure of ontology. Semantic is considered based on assigning weight to different relationships. Furthermore, we have considered more relationships than other approaches that consider only hierarchical relation of classes. Considering more relationships from an ontology leads to making more edges in graph representation of that ontology. Thus, one can make a better decision on whether two nodes are similar.

We used neighborhood random walk distance matrix to combine semantic and structural aspects of an ontology. Each element of this matrix is calculated considering weights of almost all elements of the transition probability matrix, thus weights used in proposed method are more precise than methods, which only use weight matrix.

We have introduced a new scoring function to merge modules. The objectives of this function are to maximize the intra-module similarity and to minimize inter-module similarity. The scoring function shows how appropriately nodes have been grouped in their modules according to its objectives.

As a result, we have produced meaningful modules as we consider more relations than similar methods, and process these relations such that each edge weight has an impact on every module selection.

In future work, we plan to evaluate our experiments with other evaluation methods and other datasets to determine the efficiency of our algorithms. Furthermore, we would like to further investigate the weight of edges to improve our approach.

References

1. Parent, C., Spaccapietra, S.: An Overview of Modularity. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.) Modular Ontologies. LNCS, vol. 5445, pp. 5-23. Springer, Heidelberg(2009)
2. Özacar, T., Öztürk, O., Ünalır, M.O.: ANEMONE: An Environment for Modular Ontology Development. Data & Knowledge Engineering. 70, 504-526 (2011)
3. Doran, P., Tamma, V., Payne, T.R., Palmisano, I.: An Entropy Inspired Measure for Evaluating Ontology Modularization. In: 5thInternationalConference on Knowledge Capture (KCAP'09), pp. 73-80. ACM, New York (2009)
4. Pathak, J., Johnson, T., Chute, C.: Survey of Modular Ontology Techniques and their Applications in the Biomedical Domain. Integrated Computer-Aided Engineering. 16, 225-242 (2009)
5. Zhangl, L., Liul, K., Qinl, X., Tangl, SH.: Extracting Module from OWL-DL Ontology. In: 2011International Conference on System Science, pp. 176-179.IEEE Press (2011)
6. Abadi, M.J.S, Zamanifar, K.: Producing Complete Modules in Ontology Partitioning. 2011 International Conference on Semantic Technology and Information Retrieval, pp. 137-143. IEEE Press (2011)
7. Hu, W., Zhao, Y., Qu, Y.: Partition-Based Block Matching of Large Class Hierarchies. In: Mizoguchi, R., Shi, Zh., Giunchiglia, F. (eds.) The Semantic Web – ASWC 2006. LNCS, vol. 4185, pp. 72–83. Springer, Heidelberg (2006)
8. Hu, W., Qu, Y., Cheng, G.: Matching Large Ontologies: A divide-and-conquer approach. Data & Knowledge Engineering, 67, pp. 140–160 (2008)
9. Coskun, G., Rothe, M., Teymourian, K., Paschke, A.: Applying Community Detection Algorithms on Ontologies for Identifying Concept Groups. In: Proceeding of the 5th International Workshop on Modular Ontologies (WoMO 2011), pp. 12-24. IOS Press (2011)
10. Coskun, G., Rothe, M., Paschke, A.: Ontology Content "At a Glance". In: 7th International Conference on Formal Ontology in Information Systems (FOIS 2012), pp. 147-159. IOS Press (2012)
11. Stuckenschmidt, H., Schlicht, A.: Structure-Based Partitioning of Large Ontologies. In: Stuckenschmidt, H., Parent, Ch., Spaccapietra, S. (eds.) Modular Ontologies. LNCS, vol. 5445, pp. 187-210. Springer, Heidelberg (2009)
12. Batagelj, V.: Analysis of large networks - islands. In: Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks (2003)
13. Etminani, K., Rezaeian-Delui, A., Naghibzadeh, M.: Overlapped ontology partitioning based on semantic similarity measures. In: 2010 5th International Symposium on Telecommunications (IST), pp. 1013–1018. IEEE (2010)
14. Resource Description Framework (RDF), http://www.w3.org/RDF/
15. Cheng, H., Zhou, Y., Xu, Yu, J.: Clustering Large Attributed Graphs: A Balance between Structural and Attribute Similarities. ACM Transactions on Knowledge Discovery from Data. 5, 1-33 (2011)
16. Fortunato, S.: Community detection in graphs. Physics Reports. 486, 75-174 (2010)
17. Rousseeuw, P.: Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis. Journal of Computational and Applied Mathematics. 20, 53–65 (1987)